

**CHARTON Dylan**

# Dossier projet

**Titre professionnel : Développeur  
Web et Web Mobile**

Stage effectué du 10/01/2022 au 04/03/2022

Création d'une solution de  
gestion de contenu à destination  
des mairies



<u>1. Introduction</u>	4
<u>1.1 Remerciements</u>	4
<u>1.2 Compétences du référentiel</u>	5
<u>2. Résumé</u>	6
<u>2.1 L'entreprise</u>	6
<u>2.2 Le projet</u>	6
<u>3. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet</u>	7
<u>3.1 Nomenclature et définition des objets</u>	7
<u>3.2 Back-Office</u>	7
<u>3.3 Front</u>	9
<u>4. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile</u>	11
<u>4.1 Modalité de réalisation</u>	11
<u>4.2 La stack technologique</u>	13
<u>5. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile</u>	16
<u>5.1 Le Workflow</u>	16
<u>5.2 Initialisation du projet</u>	17
<u>5.3 Gestion des utilisateurs</u>	19
<u>5.4 Gestion des entités</u>	28
<u>5.5 Upload des images</u>	45
<u>5.6 Gestion du Front</u>	47
<u>5.7 Le déploiement avec Gitlab et Heroku</u>	53
<u>6. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)</u>	55
<u>6.1 Les tests Cypress</u>	55

<u>7. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité</u>	58
<u>8. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone</u>	60
<u>9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes).</u>	61
<u>Annexes</u>	63
<u>Eléments intégrés sans template</u>	68

# 1. Introduction

## 1.1 Remerciements

Je pense qu'il est opportun de commencer ce dossier par remercier les personnes qui m'ont accompagnées tout au long de la formation. Ma première pensée va à Alain Merucci, coach formateur de Lons-le-Saunier qui par sa pédagogie, sa bonne humeur et son désir de nous tirer vers le haut, a été un moteur essentiel de l'aboutissement de mon travail. Il a toujours su avoir les mots justes, mêlant un sérieux à toute épreuve lors des moments les plus graves à une bonhomie franchouillarde qui a permis de faire sortir de leur cocons les plus introvertis d'entre nous.

Merci à mes camarades de formation, grâce à qui la notion d'apprentissage par les pairs a pris tout son sens. Ils ont su construire une atmosphère extrêmement studieuse et j'ai eu un plaisir infini à bénéficier de leur talent pour façonner ma manière de développer. Merci à tous d'avoir également su attendrir la complexité de certains problèmes auxquels nous avons été confrontés en ayant toujours un mot pour détendre l'atmosphère.

Enfin, je remercie très chaleureusement mon tuteur de stage, Jérôme Gavignet, qui m'a donné un formidable avant-goût du monde professionnel du développement web. Bien que télé-travaillé il a su m'accompagner sans trop me donner la main grâce à nos visios et a été très à l'écoute de mes suggestions d'humble aspirant développeur. Dire que ce stage ne m'a pas mis en difficulté serait un mensonge, mais c'est ce que j'ai le plus aimé, car c'est grâce à cette difficulté, grâce à ce challenge que j'ai su aller plus loin.

Merci à tous. Je vous souhaite une très bonne lecture.

Dylan

## 1.2 Compétences du référentiel

Activités types	Compétences professionnelles	
<b>Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité</b>	Maquetter une application Réaliser une interface utilisateur web statique et adaptable Développer une interface utilisateur web dynamique Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce	✓ ✓ ✓
<b>Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité</b>	Créer une base de données Développer les composants d'accès aux données Développer la partie back-end d'une application web ou web mobile Elaborer et mettre en oeuvre des composants dans une application de gestion de contenu ou e-commerce	✓ ✓ ✓ ✓

## 2. Résumé

### 2.1 L'entreprise

J'ai effectué mon stage au sein de l'entreprise JGAWeb fondée en 2015 sous le régime de l'auto-entreprise par Jérôme Gavignet. Il exerce dans sa propre entreprise en parallèle de son poste de développeur back-end au sein d'une entreprise Lyonnaise.



### 2.2 Le projet

L'objectif du projet est d'être une solution de gestion de contenu à l'attention des municipalités. L'intérêt est donc de regrouper les points d'attractivités des communes, les salariés, les services mais aussi la vie communale au travers de la publication de documents divers et variés comme les rapports de conseils municipaux, les arrêtés et les informations des associations affiliées par exemple. Il y a aussi une place donnée aux actualités des communes. Ainsi, le but est de favoriser la communication des municipalités envers leurs administrés et centraliser les services en un seul point. Il y a d'ailleurs une volonté du gouvernement de décentraliser une partie de ses services (type carte d'identité, passeport, acte de naissances etc...) vers les municipalités. Le but est donc de faciliter ce lien avec ces services pour faciliter la vie des usagers.

Proposer une solution distribuable permet d'uniformiser et éventuellement de proposer de nouveaux services petit à petit en fonction de l'ampleur du client, nous évoquerons dans quelle mesure un peu plus tard. Bien sûr l'objectif est également de proposer un produit conçu pour mobile, moderne, intuitif et surtout parfaitement fonctionnel.

## **3. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet**

Le projet a été alimenté petit à petit par les idées de base que mon tuteur avait du projet, mais aussi dans une certaine mesure par ma contribution. En clair, ce cahier des charges n'est pas la version finale du produit mais la version d'aujourd'hui seulement, car il sera amené à avoir de nouvelles fonctionnalités régulièrement tellement l'éventail de compétences nécessaire peut varier selon l'ampleur du client.

### **3.1 Nomenclature et définition des objets**

- ❖ Le terme de « Client » désignera dans l'énorme majorité des cas les municipalités, ils auront un rôle d'administrateur.
- ❖ Le terme de « Super Admin » désignera l'administrateur du portail, celui qui crée les clients ainsi que les entités fermées.

### **3.2 Back-Office**

#### **○ La navigation**

- ❖ S'effectuera via un menu latéral présentant toutes les entités accessibles par le client.

#### **○ User**

- ❖ Sera créée par le(s) SuperAdmin(s) et se connectera grâce à son adresse mail et son mot de passe.
- ❖ Se voit attribuer un identifiant de client (*i.e de municipalité*).
- ❖ Devra obligatoirement définir un nouveau mot de passe à sa première connexion (automatiquement redirigé, page inaccessible autrement).
- ❖ Pourra demander la réinitialisation de son mot passe sur la page de connexion.

#### **○ Client**

- ❖ Crée par le(s) SuperAdmin(s), associé à un utilisateur de façon manuelle à la création de celui-ci.
- ❖ Champs : Nom, adresse, code postal, ville, latitude et longitude.
- ❖ Relations : OneToMany avec Employee, Actuality, Document, Structure, User, Equipment, Contact.
- ❖ Le champ d'adresse doit être autocomplété. Les champs de code postale, ville, latitude et longitude sont remplis automatiquement en base en fonction de la sélection de l'adresse.

#### **○ Presentation**

- ❖ Crée par le client pour décrire sa municipalité, possibilité d'ajouter des images.

- ❖ Champs : Titre, description, images *OneToMany*, client *OneToOne*.
- ❖ L'ID du client est passé automatiquement à l'objet lors de sa création.

## ● **Actuality**

- ❖ Crée par le client pour faire part des dernières informations. Une date de début doit être choisie pour définir à partir de quand elle s'affiche. La date de fin peut ne pas être remplie, l'actualité sera alors toujours affichée. Pareil si le champ *always\_display* est coché.
- ❖ Champs : Titre, image *OneToMany*, date de début, date de fin, affichage permanent, description courte, description, slug, client *ManyToOne*.

## ● **Structure & StructureType**

- ❖ StructureType crée par SuperAdmin, Structure créée par le client.
- ❖ Structure permet de créer les organismes partenaires tels que les clubs sportifs, les associations, les amicales...
- ❖ Champs StructureType : Titre
- ❖ Champs Structure : Titre, description courte, description, images *OneToMany*, contacts *OneToMany*, client *ManyToOne*, type de structure *ManyToOne*

## ● **Image**

- ❖ Permet de stocker le nom des images des entités qui doivent en enregistrer plusieurs.
- ❖ Les images elles mêmes sont stockées sur le serveur
- ❖ Champs : Image, presentation *ManyToOne*, actuality *ManyToOne*, structure *ManyToOne*

## ● **Employee**

- ❖ Crée par le client, il s'agit de son équipe administrative.
- ❖ Champs : Prénom, nom, genre, titre (poste), description courte, description, image, contacts *OneToMany*, client *ManyToOne*,

## ● **Contact & ContactType**

- ❖ ContactType crée par SuperAdmin et défini les moyens de contact (réseaux sociaux, mail, téléphone, site web).
- ❖ Contact crée par le client et forcément associé à un objet.
- ❖ Champs Contact: Valeur, type de contact *OneToOne*, employee *ManyToOne*, structure *ManyToOne*, équipement *ManyToOne*
- ❖ Champs ContactType : Titre, pictogramme, préfixe, type

## ● **Document & DocumentType**

- ❖ DocumentType crée par le SuperAdmin.
- ❖ Document crée par le client et permet de rendre accessibles et téléchargeables les documents de la vie municipale.
- ❖ Champs Document : Titre, description courte, description, document, type *ManyToOne*, client *ManyToOne*.
- ❖ Champs DocumentType : Titre

## ● **Equipment & EquipmentType**

- ❖ EquipmentType crée par le SuperAdmin

- ❖ Equipment crée par le client, permet de définir les établissements utiles à la vie quotidienne des administrés : Commerces, transports, lieux culturels etc...
- ❖ Champs Equipment : Titre, image, description, latitude, longitude, type d'équipement
- ❖ Champs EquipmentType : Titre

### ● **Customization**

- ❖ Permet au client de personnaliser les couleurs de son site.
- ❖ Champs : Couleur primaire, couleur secondaire, couleur de fond, client *ManyToOne*

### ● **Shortcut**

- ❖ Gérée par le Super Admin exclusivement
- ❖ Champs : Picto, titre, description courte, lien

## **3.3 Front**

### ● **Menu de navigation**

- ❖ Menu en haut de page qui devient un menu burger latéral en mobile. Il contient les liens suivants : Accueil, Vie Municipale, Vie Citoyenne, Actualités, Équipements et Démarches
- ❖ Le menu de démarches est un menu déroulant qui présente toutes les démarches disponible ou la page avec toutes les démarches.

### ● **Page de présentation**

- ❖ Page d'accueil définie par l'ID du client mais dont l'URL sera transparent. C'est en fait la page d'accueil.

### ● **Page « Actualités »**

- ❖ Page sur laquelle figure la liste des actualités ordonnées par date de fin croissante. Le format choisi est celui de cartes bootstrap avec une image en thumbnail, le titre, la description courte et la date de publication dans le footer.
- ❖ Cliquer sur une actualité dirige vers l'actualité en question, avec l'image principale en haut d'une carte bootstrap puis la description courte, puis la description, mise en page grâce à un TextEditorField

### ● **Page « Vie Municipale »**

- ❖ Page sur laquelle figure la liste des employés et plus bas la liste des trois derniers compte-rendus de conseil municipal
- ❖ Cliquer sur un document dirige sur la page du document, permettant de le télécharger et lire la description plus en détail. Lorsque plus d'un document est lié au client, un menu latéral s'affiche à droite avec les trois derniers documents ajoutés.

### ● **Page « Vie Citoyenne »**

- ❖ Page sur laquelle figure la liste des structures sous forme de cartes bootstrap présentant une thumbnail, le nom des structures, une description courte ainsi que les icônes de contact dans le footer

- ❖ Cliquer sur une structure dirige sur sa page où il est intégré le nom de la structure en titre de page, puis une photo, la description courte et la description mise en forme par un TextEditorField.

## ● **Page « Equipements »**

- ❖ Page sur laquelle figure la liste des équipements sous forme de cartes bootstrap présentant une thumbnail, le nom des équipements, la description tronquée et les icônes de contact dans le footer
- ❖ Cliquer sur une structure dirige sur sa page où il est intégré le nom de l'équipement en titre de page, puis une photo, et la description mise en forme par un TextEditorField.
- ❖ Une carte référence tous les points d'intérêts

## ● **Page « Documents »**

- ❖ Page sur laquelle figure la liste des documents liés au client.
- ❖ Cliquer sur un document dirige sur la page du document, permettant de le télécharger et lire la description plus en détail. Lorsque plus d'un document est lié au client, un menu latéral s'affiche à droite avec les trois derniers documents ajoutés.

## ● **Page « Démarches »**

- ❖ Page sur laquelle figure sous forme de cartes Bootstrap toutes les démarches accessibles depuis le site de la mairie. Il y est renseigné les informations basiques mais aussi si les démarches sont faisables en ligne ou bien en mairie.
- ❖ En ligne : Cliquer sur la carte Bootstrap redirige directement sur le site du gouvernement permettant de lancer
- ❖ En mairie : Cliquer sur la carte Bootstrap redirige sur la page de contact de la mairie.

# 4. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile

## 4.1 Modalité de réalisation

Ce projet a été entièrement réalisé en télétravail, avec des points réguliers via Slack et Google Hangout pour le partage d'écrans. Nous avons utilisé Jira Software pour travailler en mode Agile.

Jira est un outil de gestion de projets édité par Atlassian et sorti pour la première fois en 2002. Via la mise en place de tickets permettant de fractionner le travail, il est possible d'avoir un suivi permanent et de catégoriser les tâches.

Les tickets peuvent ensuite être détaillés, permettant l'ajout de commentaires, de liens et d'images ou de tout autre pièce. Jira peut aussi être relié à d'autres services comme Slack par exemple.

A FAIRE 6 TICKETS	EN COURS 1 TICKET	IN REVIEW 1 TICKET	FINI 13 TICKETS ✓
Admin: Saisir les coordonnées GPS via API address MAIR-91	Customisation de la colorimétrie du front MAIR-85	Correction du test pour le rendre autonome MAIR-94	Bonne pratique : Rendre le nom de variable plus explicite MAIR-95
Ajouter des liens de raccourcis MAIR-83			Bugs : Les bugs trouvés en préparant la démo MAIR-89
Ajouter carte avec tous les points d'intérêts MAIR-84			Modifier la redirection MAIR-86
Ajouter la notion de communication citoyenne MAIR-90			Tests cypress MAIR-79
Présentation: Ajout logo MAIR-92			Hydrater le BO pour avoir un exemple cohérent MAIR-87
Présentation: Ajout photo épuisée			Ajout de ext-intl

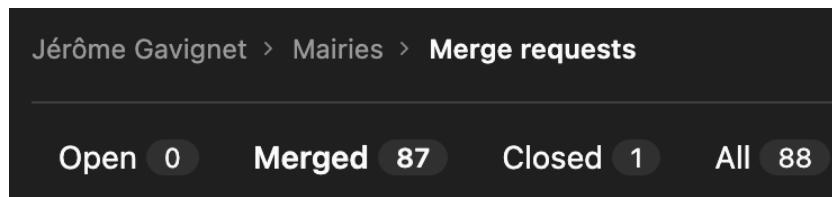
Un exemple de l'utilisation de Jira

Utiliser Jira a grandement favorisé ma compréhension du projet. En voyant quelles étaient les prochaines tâches, j'ai pu optimiser et anticiper. Cela permet aussi une grande flexibilité, un ticket peut permettre d'en corriger un précédent. Nous avons en effet une

trace constante de ce qui a été fait et en ce sens, c'est très agréable de toujours savoir où nous en sommes.

Le stockage des versions du code a été fait avec Gitlab, sur un repository privé. Gitlab est une plateforme assez similaire à GitHub à la différence qu'elle est open-source. Le code est effectivement disponible ici : <https://gitlab.com/gitlab-org/gitlab> et c'est assez impressionnant.

Gitlab permet l'utilisation de Merge Requests (MR), ce qui favorise beaucoup la revue de code et dans le cadre de ce stage a permis à mon tuteur de voir et commenter mon code.



Mais avant tout ça, il a fallut faire un point sur Git. Même si j'ai utilisé Git pendant ma formation de façon assez sérieuse, j'étais loin d'imaginer que je n'avais fait qu'effleurer la surface (et c'est encore le cas aujourd'hui). J'ai par exemple découvert le fonctionnement d'un « rebase » et comment bien l'exploiter pour avoir un arbre propre. J'ai également découvert les « squashes », pour fusionner des commits multiples en un seul, toujours dans un but de lisibilité et de propreté maximum. J'ai aussi découvert la commande « git stash », pour mettre son travail de côté sans nécessairement faire de commit avant de changer de branche.

Au delà des commandes git elles même, j'ai surtout découvert une méthodologie et une rigueur propre au monde professionnel. Par exemple, le nom des branches est codifié par rapport au nom du ticket. La branche reliée au ticket « MAIR-05 » avec pour intitulé « Créer l'entité Client » s'appellera donc MAIR-05-ClientEntity.



Aussi, lorsque j'ai soumis une MR à revue de mon tuteur et qu'il y a des corrections à effectuer, les faire sur un commit séparé est bien plus pratique que d'effectuer un squash qui reset les commits précédents et demande donc au reviewer d'éplucher à nouveau tout le code.

Merged | Created 1 week ago by Dylan Charton | Developer | Edit

### feat : MAIR-102-ContactPage: Added the contact page and the contact form

Overview 14 | Commits 4 | Pipelines 4 | Changes 24 | All threads resolved

04 Mar, 2022 1 commit

fix : MAIR-102-ContactPage : Added a verification if the client does not have... Dylan Charton authored 6 days ago 8ac21cca

03 Mar, 2022 1 commit

feat : MAIR-102-ContactPage : updated how I get the mail of the client Dylan Charton authored 6 days ago 3670c94c

02 Mar, 2022 2 commits

fix : MAIR-102-ContactPage : updated how I fetch the client mail to be sure it is a mail Dylan Charton authored 1 week ago 18da3435

feat : MAIR-102-ContactPage: Added the contact page and the contact form Dylan Charton authored 1 week ago f01785f6

Add previously merged commits

Exemples de commits

En clair, tout cela m'a appris avant tout le travail collaboratif et l'organisation qu'il implique.

## 4.2 La stack technologique

### ○ PHP 7.4

PHP est un langage de développement interprété côté serveur très utilisé en développement web, j'utilise sa version 7.4

### ○ SYMFONY 5.4

Symfony est un framework PHP développé depuis 2005 par une entreprise française : Sensiolabs (Cocorico !) et dont l'architecture repose sur le modèle MVC (Modèle Vue Contrôleur) qui permet une bonne segmentation du code et optimise le travail collaboratif.

Symfony à l'avantage de proposer des composants pré-développés permettant de démarrer un projet très rapidement. Nous en verrons beaucoup l'efficacité dans la partie 5 de ce dossier, j'ai pris beaucoup de plaisir à constater la puissance de ce framework. Symfony a aussi une durée de maintenabilité très importante, dont un résumé se trouve ici : <https://symfony.com/releases> — En somme, Symfony 5.4 est soutenu jusqu'à 2024 puis basculera dans une régime de corrections de sécurité uniquement pendant un an minimum.

Symfony embarque également une armada de commandes qui seront utiles tout au long du développement, nous allons en voir beaucoup pendant ce dossier, car j'en ai

avalé un paquet et je commence à les connaître par cœur (s'en est presque effrayant, croyez moi).

## ○ COMPOSER POUR LA GESTION DES BUNDLES

Composer est un gestionnaire de dépendances développé en PHP qui permet de gérer les librairies utilisées par un projet. Tout comme Symfony, il est disponible en open-source et est donc alimenté par la communauté. Dans un second temps, le site <https://packagist.org/> permet de répertorier tous les paquets (ou bundles, ou dépendances comme nommé plus haut). Composer fonctionne à l'aide d'un bon nombre de fichiers comme composer.json, composer.lock et surtout le dossier vendor.

Les dépendances les plus notables utilisées :

## ○ EASYADMIN POUR LE BACK OFFICE ( ABRÉGÉ BO )

EasyAdmin est la solution de mise en place d'une administration plébiscitée par Symfony. Nous nous cantonnerons à la version 3 pour travailler en PHP 7.4 car la version 4 du bundle exige au minimum de passer sous PHP 8, ce qui ne fait pas partie des exigences du projet.

EasyAdmin permet la mise en place de CRUDs, de tableaux de bords (dashboards) qui permettent de gérer des contrôleurs liés à des entités. Il offre une grande liberté dans la gestion de ses templates et s'accorde très bien à Symfony dans sa compatibilité avec les services de Symfony. Les services sont fondamentaux pour le bon fonctionnement d'une application Symfony, toutes les actions effectuées viennent très probablement d'un service, comme l'envoi de mail, la réinitialisation du mot de passe, le stockage de données en base, ou bien l'upload de fichiers.

## ○ VICHUPLOADERBUNDLE

Quelle belle transition, car en ce qui concerne l'upload de fichiers et d'images, c'est la dépendance VichUploaderBundle que j'ai utilisé et qui facilite grandement la mise en place de champs adéquat ainsi que le lien avec la base de données. VichUploaderBundle est également disponible en open-source sur Github : <https://github.com/dustin10/VichUploaderBundle>

## ○ WEBPACK ENCORE

Webpack Encore est géré par Symfony et permet « d'empaqueter » les assets comme les images et typographies, les fichiers CSS et JavaScript. Il génère des fichiers compilés à partir de ressources définies au préalable et simplifie donc leur gestion. Dans

mon cas, c'est notamment pour le thème utilisé que Webpack Encore était nécessaire, car il utilise un bon nombre de librairies JavaScript.

## ● **CYPRESS**

Cypress est utilisé pour mettre en place les tests End-to-End, c'est à dire les tests qui permettent de vérifier le bon comportement de l'application lorsqu'elle est entre les mains d'un utilisateurs. Ou d'un programme, en l'occurrence. Cypress à l'énorme avantage d'avoir une interface graphique qui permet de voir les tests se dérouler ainsi qu'un Dashboard qui permet d'en avoir une traçabilité.

Dans mon cas, Cypress est arrivé plutôt dans le dernier tiers du projet, je n'ai donc pas pu en développer pour toutes les fonctionnalités mais il a été adopté dès son arrivée.

Voici pour les dépendances les plus importantes. Il y en a bien sûr d'autres que j'évoquerai au moment venu.

## ● **LE THÈME SANDBOX**

Ce thème permet d'éviter d'avoir à maquetter mais m'a permis d'avoir une grande souplesse au niveau du front.

## **5. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile**

### **5.1 Le Workflow**

Avant de traiter de ce sur quoi j'ai travaillé, il pourrait être intéressant de comprendre comment j'ai travaillé. Comme mentionné plus haut, j'ai travaillé seul sur ce projet avec simplement mon tuteur pour me donner des conseils lorsque j'en avais besoin. Voici donc comment s'est organisé mon travail :

- ❖ Tout est en anglais dans le code
- ❖ Les tâches sont organisées en tickets sur Jira, que je suis dans l'ordre.
- ❖ Chaque ticket devait représenter une branche. La convention pour nommer cette branche est de prendre le nom de l'espace de travail, le numéro du ticket et le thème du ticket. Par exemple, le ticket 56 qui porte sur l'entité Structure s'appelle MAIR-56-Structure. MAIR est un diminutif de l'espace de travail qui est « Mairie »
- ❖ Lorsque j'ai terminé de coder et si je n'ai pas d'autres commits que celui que je m'apprête à faire, je passe aux étapes qui arrivent. Si j'ai déjà fait plusieurs commits sur la même branche sans avoir push (pour pouvoir potentiellement revenir en arrière par exemple) je vais effectuer un « squash ». C'est à dire que je prends tous mes commits, j'en extrait toutes les modifications qui ont été faites et je les place en changements à indexer. De cette manière, tous mes commits n'en feront plus qu'un, ce qui permettra d'avoir une structure de projet bien plus propre.
- ❖ J'effectue la commande `git add *` pour indexer les changements effectués sur la branche.
- ❖ Puis `git commit -am 'feat : MAIR-56-Structure : le commentaire'`. Pour « empaqueter » les changements dans un conteneur virtuel (là, il s'agit de faire fonctionner sa capacité d'abstraction pour bien comprendre)
- ❖ Enfin je fais un `git push origin main` pour envoyer ce paquet vers le dépôt distant.

Là aussi on remarque une convention de nommage pour le commentaire, cela permet de faire des messages de commit clairs et compréhensibles par tous, ce qui est indispensable pour une bonne organisation.

Après le push, étant donné que je travaille sur Gitlab, mon terminal de commande m'indique que je peux soumettre une MR et m'indique un lien.

```
Acs@PORT201910-092 MINGW64 ~/Desktop/Stage/mairies (MAIR-104-CorrectColors)
$ git push origin MAIR-104-CorrectColors
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 479 bytes | 239.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for MAIR-104-CorrectColors, visit:
remote:   https://gitlab.com/jerome-gavignet/mairies/-/merge_requests/new?merge_request%5Bsource_branch%5D=MAIR-104-CorrectColors
remote:
To gitlab.com:jerome-gavignet/mairies.git
 * [new branch]      MAIR-104-CorrectColors -> MAIR-104-CorrectColors
```

Autrement dit, une demande de merge sur la branche principale effectuée auprès d'un supérieur qui va vérifier le code et faire des retours. C'est la fonctionnalité qui m'a fait le plus progresser, car c'est grâce à ce fonctionnement asynchrone que j'ai pu absorber toutes les remarques et corrections faites sur mon code et tâcher de ne pas les reproduire. D'autant plus qu'il est possible de retrouver les anciennes MR (d'où l'intérêt de bien nommer ses branches) pour les consulter à nouveau.

## 5.2 Initialisation du projet

### 5.2.1 L'environnement

Pour commencer, j'ai installé la CLI de Symfony, puis Composer. Après avoir vérifié leur bonne mise en marche grâce à quelques commandes, je suis allé me placer dans mon dossier cible et j'ai généré l'architecture de base de l'application. Puis, je l'ai relié au repository Gitlab créé par mon tuteur. Puis, j'ai effectué la commande pour installer EasyAdmin.

```
$ composer require easycorp/easyadmin-bundle
```

Je suis donc allé sur la documentation de Symfony qui, encourageant son utilisation, comporte également la documentation d'EasyAdmin. Je vérifie aussi que mon fichier composer.json a bien été mis à jour

```
// . . .
"require": {
    "php": ">=7.2.5",
    "ext-ctype": "*",
    "ext-gd": "*",
    "ext-iconv": "*",
    "ext-intl": "*",
    "composer/package-versions-deprecated": "1.11.99.4",
    "doctrine/annotations": "^1.0",
    "doctrine/doctrine-bundle": "^2.5",
    "doctrine/doctrine-migrations-bundle": "^3.2",
    "doctrine/orm": "^2.10",
    "easycorp/easyadmin-bundle": "^3.5",
// . . .
},  
// . . .
```

composer.json

## 5.2.2 La base de données

Pour la base de données, étant donné que je suis en 5.4.4 je n'ai pas besoin d'implémenter le bundle Doctrine car il est nativement présent. En revanche, je vais aller configurer mon fichier .env.local à la racine du projet pour initialiser les accès à la base de données. C'est un fichier qui définit les variables locales et est donc ignoré par git.

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/mairies"
```

.env.local

Maintenant, il ne me reste plus qu'à appeler une commande pour créer la base de données sur phpMyAdmin.

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

## 5.2.3 Sécurisation de l'administration

Enfin, pour finir l'initialisation, il m'a été demandé de sécuriser les routes d'administration. Pour cela, j'ai utilisé les fichiers de configuration de Symfony et notamment le fichier security.yaml.

Le format YAML est ici utilisé comme format de configuration. Il y est stocké des informations utilisées par les différents services de Symfony. Ici, il m'a suffit de me rendre à la ligne `access_control` pour rajouter une contrainte d'accès aux utilisateurs ayant le rôle `ROLE_ADMIN`.

```
access_control:  
- { path: ^/admin, roles: ROLE_ADMIN }
```

config/package/security.yaml

La notion de rôle est très importante dans Symfony, car cela va avec la notion d'autorisations qui garantie la sécurité des applications. Nous y reviendront lorsque nous aborderons l'utilisateur en lui-même. Dans le cas présent, le rôle nous permet de filtrer l'accès au Back-Office grâce à cette ligne.

Ensuite, il est possible de mettre en place une hiérarchie de rôles, de sortes à ce que certains incluent d'autres naturellement. Par exemple, le rôle ROLE\_SUPER\_ADMIN aura également par défaut les droits du rôle ROLE\_ADMIN qui a lui même les rôles du ROLE\_USER. Ici c'est une hiérarchie assez simple, mais les possibilités sont très nombreuses.

```
role_hierarchy:  
ROLE_ADMIN: ROLE_USER  
ROLE_SUPER_ADMIN: ROLE_ADMIN
```

config/package/security.yaml

## 5.3 Gestion des utilisateurs

Suite logique de la création des rôles et de la sécurisation du B.O, j'ai crée les premiers utilisateurs. Tout commence par la création de l'entité User. Une entité, c'est le nom dénominateur commun des objets qu'on veut regrouper dans une table de base de données. De façon un peu barbare, l'entité User représentera tous les objets User présents dans la base et ils auront en commun non seulement leur appartenance à cette entité, mais cela leur confère leurs propriétés. Ici, nous voulons que les utilisateurs aient les propriétés suivantes :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	▾ Plus
<input type="checkbox"/>	2 firstname	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	3 lastname	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	4 email	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	5 password	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	6 roles	json			Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	7 change_password	tinyint(1)			Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	8 active	tinyint(1)			Non	Aucun(e)			▾ Plus
<input type="checkbox"/>	9 client_id	int(11)			Oui	NULL			▾ Plus

Les propriétés un peu étranges seront expliquées au fil de ce dossier.

Pour créer l'entité, j'utilise la commande et je me laisse guider par la console pour créer les champs nécessaires.

```
PS C:\Users\Acs\Desktop\Stage\mairies> php bin/console make:entity Users
created: src/Entity/Users.php
created: src/Repository/UsersRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> firstname
```

La console est un outil formidable, elle parle beaucoup, ici elle indique qu'elle a crée des fichiers PHP et qu'elle m'invite à effectuer une autre commande, lorsque je serais prêt.

Intéressons nous à nos fichiers PHP. D'abord, nous retrouvons User.php qui synthétise tout ce qu'on a demandé à la console et même plus. En annotations au dessus des attributs, nous retrouvons une indication sur le mapping de l'entité dans la base. Cette annotation est interprétée et permet la génération de la migration.

```
//. . .
/**
 * @ORM\Column(type="string", length=255)
 */
private $firstname;
//. . .
```

src/Entity/User.php

Ensuite nous avons le fichier UserRepository.php, qui nous permet d'aller chercher des objets en appelant des méthodes établies ici dans un contrôleur par exemple. Il est aussi possible de créer ses propres méthodes de filtres, nous verrons cela un peu plus tard.

Si je reviens à ma console, je vois qu'il m'est proposé d'effectuer une migration. Ce sont des fichiers php qui stockent les commandes SQL à effectuer pour créer la structure de base de données nécessaire au bon fonctionnement du projet. C'est un peu comme une notice d'utilisation... Sauf qu'elle fait le montage à votre place. En effet, une commande suffit, j'ai nommé:

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

Notons que depuis le début de cette partie je vous parle d'exécuter des commandes mais hypothétiquement, si je connaissais sur le bout des doigts les attributs, les annotations ORM et ses propriétés, ainsi que les getters et setters, je pourrais écrire ces fichiers à la main. Mais la puissance de Symfony et de ses librairies est là, beaucoup de code est générable automatiquement. C'est un gain de temps formidable. Ce qui ne nous dispense pas de comprendre le code sous-jacent.

J'ai généré ma migration, et je peux aller la voir dans le dossier dédié. On peut également les modifier à la main si nécessaire, mais il faut bien penser à modifier les entités concernées, sinon il risque d'y avoir des incohérences entre ce qu'il y a en base et ce qui est interprété par l'ORM Doctrine. Auquel cas, la migration suivante apportera son lot de rectifications qui ne risque pas de vous plaire. Autant gérer le problème à la source en cas de migration incorrecte donc.

```
use Doctrine\DBAL\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20220110163306 extends AbstractMigration
{

    ...

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your
        // needs
        $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL,
firstname VARCHAR(255) NOT NULL, lastname VARCHAR(255) NOT NULL, email
VARCHAR(255) NOT NULL, password VARCHAR(255) NOT NULL, roles JSON NOT NULL,
change_password TINYINT(1) NOT NULL, active TINYINT(1) NOT NULL, PRIMARY
KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE =
InnoDB');

    }

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

migrations/Version20220110163306.php

Puis, j'effectue la commande suivante :

```
$ php bin/console doctrine:migrations:migrate
```

Elle va exécuter dans l'ordre la ou les migrations enregistrées et donc créer notre table User dans le cas présent.

Maintenant que la table est créée, il faut y mettre des données. Mettre des données comme ça, à la main, c'est bien une fois, mais je vais utiliser des Fixtures pour générer des jeux de données en un instant. Grâce à composer je n'ai qu'une commande à effectuer :

```
$ composer require --dev orm-fixtures
```

Puis je crée le fichier `UserFixtures.php` dans lequel je vais créer la fonction `load()` qui va permettre de charger les fixtures. Il s'agit ici d'un fonctionnement simple de création d'objet et de définition d'attributs.

```
namespace App\DataFixtures;

use App\Entity\User;
use App\Entity\Client;
use Doctrine\Persistence\ObjectManager;
use Doctrine\Bundle\FixturesBundle\Fixture;

class UserFixtures extends Fixture
{

    public function load(ObjectManager $manager): void
    {
        $admin = new User();
        $admin->setFirstname("Dylan")
            ->setLastname("Charton")
            ->setEmail("dylan.charton@gmail.com")
            ->setPassword('Admin123456!')
            ->setRoles(['ROLE_ADMIN'])
            ->setChangePassword(false)
            ->setActive(true);

        $superAdmin = new User();
        $superAdmin->setFirstname("Jérôme")
            ->setLastname("Gavignet")
            ->setEmail("jerome.gavignet@gmail.com")
            ->setPassword('Superadmin123456!')
            ->setRoles(['ROLE_SUPER_ADMIN'])
            ->setChangePassword(false)
            ->setActive(true);

        $manager->persist($admin);
        $manager->persist($superAdmin);
        $manager->flush();
    }
}
```

DataFixtures/UserFixtures.php

Puis je n'ai plus qu'à exécuter cette commande pour charger les fixtures en base de données :

```
$ php bin/console doctrine:fixtures:load
```

Nous avons ainsi deux utilisateurs prêts à se connecter ! Mais sans formulaire c'est un peu compromis.

### 5.3.1 Le login

Je vais créer le formulaire de connexion et appliquant les recommandations de sécurité de Symfony (détailé dans la partie dédiée à la veille sur la sécurité). Je commence donc par aller définir l'entité qui devra être vérifiée ainsi que le critère de vérification.

Ensuite, il s'agit de mettre en place la route pour me connecter. Je commence donc par créer un contrôleur appelé SecurityController.php. La plupart de mes contrôleurs vont étendre la classe AbstractController qui donne accès à des méthodes très pratiques, notamment en terme de sécurité (isGranted par exemple, nous en reparlerons).

Puis, je crée ma fonction login() en prenant soin d'indiquer la route sous forme d'annotation. Ici il faut prendre soin de nommer la route de la même façon que je l'ai nommée dans security.yaml, car tout est lié.

```
namespace App\Controller;

use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

class SecurityController extends AbstractController
{
    /**
     * @Route("/login", name="login")
     */
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        if($this->getUser()){
            return $this->redirectToRoute('admin');
        }

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

src/Controller/SecurityController.php

Dans la fonction, je peux gérer des conditions supplémentaires, comme rediriger l'utilisateur sur la page d'admin par exemple. Ensuite, je retourne la vue de mon login. Le formulaire de connexion est un peu particulier car c'est un des seuls qui devra être écrit à la main, les autres pouvant être générés automatiquement. Ma vue doit comporter le retour du message d'erreur dans le cas où il y en a un, ainsi que mes champs. La spécificité est que les champs d'identifiant et de mot de passe doivent absolument avoir pour attribut name les valeurs « \_username » (même s'il s'agit d'un email) et « \_password ». Je rajoute le bouton de connexion, et voilà, l'identification est en place.

```
//. . .

<div class="form-control my-5 d-flex">
    <label for="username" class="col-2">Email :</label>
    <input type="text" id="username" data-cy="username" name="_username"
class="border-0 col-10" value="{{ last_username }}" />
</div>

<div class="form-control d-flex">
    <label for="password" class="col-5">Mot de passe :</label>
    <input type="password" id="password" data-cy="password" class="border-0 col-7"
name="_password" />
</div>
<input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">
<a href="{{ path('password_request') }}" class="mb-5 fs-13 hover">Mot de passe
oublié ?</a>
<button type="submit" class="btn btn-primary rounded-pill >>Connexion</button>
//. . .
```

template/security/login.html.twig

En revanche, ce n'est pas sécurisé du tout, mes mots de passes sont en clair dans ma base de données. Pour les hasher, il me suffit de spécifier un algorithme (ou de laisser le choix à Symfony) et le travail se fera seul. Afin de limiter les possibilités d'erreurs dans le code, les composants de sécurité de base ne nécessitent que très peu de code pour être opérationnels.

```
//. . .

password_hashers:
    App\Entity\User: 'auto'

//. . .
```

config/package/security.yaml

Je reviens donc à mes deux utilisateurs fonctionnels mais dont le mot de passe est en clair dans ma base, et je vais apporter les modifications nécessaires pour que le mot de passe soit automatiquement hashé pour eux aussi.

```
namespace App\DataFixtures;  
//...  
  
class UserFixtures extends Fixture  
{  
  
    public function load(ObjectManager $manager): void  
    {  
        $admin = new User();  
        //...  
        =>setPassword($this->hasher->hashPassword($admin, 'Admin123456!'))  
    }  
    //...  
  
    $superAdmin = new User();  
    //...  
    =>setPassword($this->hasher->hashPassword($superAdmin,  
'Superadmin123456!'))  
    //...  
  
    $manager->persist($admin);  
    $manager->persist($superAdmin);  
    $manager->flush();
```

DataFixtures/UserFixtures.php

### 5.3.2 La gestion du mot de passe

Etant donné qu'il n'y a pas de formulaire d'inscription, c'est le « Super Admin » qui doit créer les utilisateurs et leur affilier un client à la main (ce qui ne pose pas de problème en terme de confiance, le « Super Administrateur » étant lié à l'entité qui crée l'application, à savoir moi ou bien JGAWeb). À la création de l'utilisateur, le « Super Admin » choisit de façon arbitraire un mot de passe qu'il communique au client. Cependant, il ne faut pas que le gestionnaire de l'application aie connaissance des mots de passe, pour des raisons de sécurité évidentes. Ainsi, j'ai mis en place une propriété dans l'utilisateur qui permet de gérer ce qu'il se passe à la première connexion. Il s'agit de la propriété changePassword.

```
//...  
  
/**  
 * @ORM\Column(type="boolean")  
 */  
private $changePassword;  
//...
```

Pour gérer cela, j'ai choisi d'utiliser le service d'EventSubscriber de Symfony afin de provoquer des changements selon un événement défini. Je génère donc le fichier selon la documentation. EasyAdmin dispose d'événements personnalisés que je vais pouvoir utiliser. Ici ce sera BeforeEntityPersistedEvent qui comme son intitulé l'indique se déclenchera juste avant la persistance des données en base. Ensuite je définit les méthodes qui vont réagir en fonction des écouteurs d'événements dans la méthode `getSubscribedEvent()`. Ne prêtons attention qu'à la seconde méthode : `setUserPassword()`

```
class EasyAdminSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        // return the subscribed events, their methods and priorities
        return [
            BeforeEntityPersistedEvent::class => [['setClientId', 10],
            ['setUserPassword', 0]],
            BeforeEntityUpdatedEvent::class => ['updateSlug'],
        ];
    }

    // . . .

    public function setUserPassword(BeforeEntityPersistedEvent $event)
    {
        $entity = $event->getEntityInstance();

        if(!$entity instanceof User){
            return;
        }
        $entity->setChangePassword(true);
        $this->encodePassword($entity);
    }

    private function encodePassword(User $entity)
    {
        $password = $entity->getPassword();
        $entity->setPassword(
            $this->passwordEncoder->hashPassword($entity, $password)
        );
        $this->entityManager->persist($entity);
        $this->entityManager->flush();
    }
}
```

src/EventSubscriber/EasyAdminSubscriber.php

La première méthode assigne la valeur `true` à la propriété `changePassword` puis appelle la seconde méthode qui va hasher le mot de passe et le persister en base. Maintenant, la valeur `changePassword` est `true` systématiquement mais peut être changée, ce qui n'aurait pas été le cas si j'avais défini cette propriété comme `true` directement dans la classe `User`. Il faut désormais que je fasse quelque chose avec cette

information, par exemple que je force l'utilisateur à se rendre sur une page de changement de mot de passe à sa première connexion. Je commence d'abord par créer la route dans mon `SecurityController.php` puis je crée un type de formulaire que je nomme `ChangePasswordFormType.php` et qui n'est lié à aucune entité en particulier. Il ne comporte qu'un champ de type `RepeatedType` qui permet de vérifier automatiquement que la valeur dans les deux champs est similaire.

Puis, mon contrôleur me permet de récupérer le contenu du champ et de l'assigner au mot de passe de l'utilisateur courant après l'avoir hashé.

```
/** * @Route("/change-password", name="change_password") */
public function changePassword(Request $request, ManagerRegistry $doctrine, UserPasswordHasherInterface $passwordHasher)
{
    $this->denyAccessUnlessGranted('ROLE_USER');
    $manager = $doctrine->getManager();
    $user = $this->getUser();
    $form = $this->createForm('App\Form\ChangePasswordFormType');
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $newpwd = $form->get('plainPassword')['first']->getData();
        $newEncodedPassword = $passwordHasher->hashPassword($user, $newpwd);
        $user->setPassword($newEncodedPassword);
        $user->setChangePassword(0);

        $manager->flush();
        $this->addFlash('notice', 'Votre mot de passe a bien été
changé !');

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

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

src/Controller/SecurityController.php

Désormais, l'utilisateur dispose d'un mot de passe sécurisé. Je n'ai pas eu le temps de proposer un système où l'utilisateur pourrait changer à sa guise son mot de passe, cependant, en cas d'oubli ou de perte du mot de passe j'ai priorisé la mise en place d'un système de réinitialisation du mot de passe. Pour cela, j'ai utilisé le bundle de SymfonyCasts : `composer require symfonicasts/reset-password-bundle`

Puis j'exécute la commande `:bin/console make:reset-password` qui génère plusieurs fichiers dont une entité qui nécessite d'être envoyée en base de données

puisqu'elle va stocker le « jeton » (plus couramment appelé token) qui sera envoyé par mail dans l'URL de l'utilisateur qui a réclamé la réinitialisation. Le fichier le plus intéressant est le [ResetPasswordController.php](#) (Annexe 1 p.63-64) qui gère toutes les fonctionnalités de la réinitialisation.

De mon côté, tout ce que j'ai à faire c'est régler le paramètre MAILER\_DSN dans le fichier [.env](#) à la racine du projet.

### 5.3.3 Désactiver un utilisateur

Si un client ne souhaite plus bénéficier de l'application il faut pouvoir désactiver les utilisateurs liés. Pour cela, j'ai découvert le service UserChecker qui est une vérification supplémentaire pour vérifier si l'utilisateur a le droit de se connecter. Je vais créer un UserChecker personnalisé et remplir la fonction [checkPostAuth\(\)](#) pour renvoyer une erreur lorsque la propriété Active de l'utilisateur est réglée sur *false*.

```
<?php
namespace App\Security;

use App\Entity\User as AppUser;
use Symfony\Component\Security\Core\Exception\CustomUserMessageAccountStatusException;
use Symfony\Component\Security\Core\User\UserCheckerInterface;
use Symfony\Component\Security\Core\User\UserInterface;

// . . .

class UserChecker implements UserCheckerInterface
{
    public function checkPostAuth(UserInterface $user): void
    {
        if (!$user instanceof AppUser) {
            return;
        }

        // user account is expired, the user may be notified
        if ($user->getActive() == false) {
            throw new CustomUserMessageAccountStatusException('Votre compte a été
désactivé', ["Votre compte a été désactivé"], 401);
        }
    }
}
```

src/Security/UserChecker.php

Ensuite, il ne me reste plus qu'à aller dans [security.yaml](#) pour spécifier auprès du pare-feu qu'il doit utiliser cet UserChecker.

## 5.4 Gestion des entités

C'est un projet conséquent, avec beaucoup d'entités. Je ne vais donc pas détailler la création de chacune d'entre elle mais plutôt les fonctionnalités intéressantes qui leur

sont liées et ce par soucis de brièveté. D'ailleurs, les relations entre les entités sont détaillées dans le cahier des charges qui est lui aussi conséquent.

L'entité Client est l'entité la plus importante et c'est la seule que je détaillerai autant, car c'est une entité parfaitement pédagogique. Je vais d'ailleurs très certainement évoquer les entités à laquelle elle est liée dans le même temps.

Comme nous l'avons vu précédemment, je serais plus bref sur la création de l'entité. Je commence par définir mon entité grâce à la commande suivante :

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

Puis, comme vu précédemment, j'effectue la migration et j'effectue la persistance en base. Voilà la structure que je dois avoir :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	
<input type="checkbox"/>	2 name	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			
<input type="checkbox"/>	3 address	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			
<input type="checkbox"/>	4 zipcode	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			
<input type="checkbox"/>	5 town	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			
<input type="checkbox"/>	6 latitude	double			Non	Aucun(e)			
<input type="checkbox"/>	7 longitude	double			Non	Aucun(e)			
<input type="checkbox"/>	8 active	tinyint(1)			Non	Aucun(e)			

Maintenant que j'ai mon entité, je vais pouvoir réfléchir à comment créer des objets qui lui sont liés. Pour cela, le bundle EasyAdmin va m'être d'une aide très précieuse. Il intègre un système de « Dashboards » qui sont en fait des contrôleurs à partir desquels je peux relier d'autres contrôleurs qui vont gérer mes CRUDs. Dans mon cas, j'ai choisi de travailler avec un seul Dashboard qui sera relié à tous les prochains CRUDs. Ce sera `AdminController.php` qui jouera ce rôle. Il va étendre `AbstractDashboardController` qui fonctionne de la même façon qu'`AbstractController` qu'on a vu plus haut. Il va aussi appeler `MenuItems` d'EasyAdmin qui va me servir à paramétriser le menu latéral. Grâce à une méthode prévue à cet effet, je vais répertorier mes boutons de navigation au fil du développement. Le premier à paramétrier est celui qui mène au Dashboard, la méthode `::linkToDashboard` est prévue à cet effet.

```

class AdminController extends AbstractDashboardController
{
    // . .

    public function configureMenuItems(): iterable
    {
        yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');

        if($this->isGranted('ROLE_SUPER_ADMIN'))
        {
            yield MenuItem::linkToCrud('Clients', 'fa fa-users',
Client::class);
        }
    }
    // . .
}

```

src/Controller/Admin/AdminController.php

Pour les liens de CRUD ce sera le même fonctionnement, sauf que ce sera la méthode ::linkToCrud qui sera utilisée et qu'on spécifie en troisième argument l'entité qui est concernée.

J'ai donc paramétré le menu mais pour le moment il ne mène à rien. Pour remédier à cela, je vais effectuer la commande suivante :

```
$ php bin/console make:admin:crud
```

Cela va générer un contrôleur de CRUD. Ici, il sera lié à l'entité Client et s'appellera donc [ClientCrudController.php](#).

Les CRUDs d'EasyAdmin tournent autour des actions exécutables. Il y a la page d'index, présentée sous la forme d'un tableau répertoriant les objets liés à l'entité. Il y a également la page de création, la page de modification et la page de détail. La suppression quant à elle s'effectue via une modale. EasyAdmin permet de paramétrier l'accès et la présence des actions liées à ces pages, nous le verront un peu plus tard.

Les champs sont générés automatiquement en fonction de l'entité, mais je peux les customiser. La seule chose que je vais faire, c'est modifier le nom de l'entité au pluriel pour que le titre de ma page soit plus cohérent. [configureCrud\(\)](#) prend en argument le service Crud d'EasyAdmin qui gère comme son nom l'indique les paramètres généraux des CRUDs. Je vais aussi en profiter pour autoriser l'accès à ce CRUD seulement aux personnes disposants du rôle Super Admin.

```
// . .
public function configureCrud(Crud $crud): Crud
{
    return $crud
        ->setEntityLabelInPlural('Clients')
        ->setEntityPermission('ROLE_SUPER_ADMIN')
        ->setPageTitle('new', 'Créer un client')
        ->setPageTitle('edit', 'Modifier un client')
    ;
}
// . . .
```

src/Controller/Admin/ClientCrudController.php

Maintenant que cela est fait, je vais retirer la page par défaut d'EasyAdmin au profit d'une page personnalisée. Je vais créer la route de base dans mon Dashboard et puis je vais créer le template qui va étendre un des templates de base d'EasyAdmin, sans quoi je ne retrouverai pas la barre latérale.

```
class AdminController extends AbstractDashboardController
{
    /**
     * @Route("/admin/", name="admin")
     */
    public function index(): Response
    {
        return $this->render('admin/home.html.twig');
    }
}
```

src/Controller/Admin/AdminController.php

```
{% extends '@EasyAdmin/page/content.html.twig' %}

{% block content_title %}Administration{% endblock %}

{% block main %}
    <h3>Bonjour, {{ app.user.firstname }}.</h3>
{% endblock %}
```

templates/admin/home.html.twig

Maintenant, je vais créer ma première route qui prend un paramètre : l'ID du client. Notons que la méthode 'compact' me permet de synthétiser l'usage de variables qui sont similaires dans le contrôleur et dans l'appel qui en sera fait dans le template.

```

/**
 * @Route("admin/client/{id}", name="show_client")
 */
public function showClient(Client $client) : Response
{
    $this->denyAccessUnlessGranted('access', $client);
    return $this->render('admin/client.html.twig', compact('client'));
}

```

src/Controller/Admin/AdminController.php

Ces informations sont cependant privées, les autres clients ne sont pas supposés y accéder. Je mets donc en place un Voter. Lorsque je fais appel à la méthode `denyAccessUnlessGranted()` (ou `isGranted`) cela va déclencher un Voter qui déterminera les autorisations d'accès à l'utilisateur. (Annexe 2 p.65). Les deux cas qui permettront un accès à ces données sont le rôle SUPER\_ADMIN et la relation avec l'object Client concerné. Je vais donc commencer par vérifier la première condition. Puis via la méthode `canAccess()` je vérifie la présence de la relation entre l'utilisateur courant et le client visé.

La seconde entité créée va me permettre d'aborder la personnalisation des champs d'EasyAdmin. L'entité Employee comporte un certain nombre de champs différents que je peux paramétrier grâce à la méthode `configureFields()`. Le fonctionnement que j'ai choisi repose sur la valeur de retour de la fonction, c'est un tableau qui comprend les champs assignés auxquels je peux attribuer des paramètres. D'abord, je peux choisir d'appliquer un type de champ (TextField, ChoiceField par exemple) ou rester global avec simplement la dénomination Field pour laisser EasyAdmin choisir en fonction du type de champ. Puis, certains champs ont leur propres méthodes pré-définies. Le ChoiceField demande de définir des choix qui peuvent être soit inscrits en durs soit être le résultat d'une méthode qui va chercher des données dans une entité par exemple.

```

public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('firstname', 'Prénom'),
        TextField::new('lastname', 'Nom'),
        ChoiceField::new('gender', 'Genre')->setChoices(['Masculin' =>
'Masculin', 'Féminin' => 'Féminin']),
        TextField::new('title', 'Titre'),
        TextField::new('short_description', 'Description courte'),
        TextEditorField::new('description', 'Description'),
        TextareaField::new('imageFile', 'Avatar')-
>setFormType(VichImageType::class),
        CollectionField::new('contacts', 'Contacts')
            ->setEntryType(ContactValueType::class)
            ->setFormTypeOption('by_reference', false)
            ->onlyOnForms(),
    ];
}

```

src/Controller/Admin/EmployeeCrudController.php

Ensuite, je vais créer la relation entre Client et Employee. Pour cela je relance la commande `php bin/console make:entity`, la console m'avertit que l'entité existe déjà mais que je peux rajouter des champs. Cette fois je vais rajouter un champ de type relation et me laisser guider pour choisir la bonne relation. Ici, je veux qu'un Employee soit relié à un et un seul Client, mais qu'un Client puisse potentiellement être relié à de nombreux Employee.

```

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> client

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> Client

What type of relationship is this?

-----

| Type       | Description                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------------------|
| ManyToOne  | Each Users relates to (has) one Client.<br>Each Client can relate to (can have) many Users objects                            |
| OneToMany  | Each Users can relate to (can have) many Client objects.<br>Each Client relates to (has) one Users                            |
| ManyToMany | Each Users can relate to (can have) many Client objects.<br>Each Client can also relate to (can also have) many Users objects |
| OneToOne   | Each Users relates to (has) exactly one Client.<br>Each Client also relates to (has) exactly one Users.                       |


-----  

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> 

```

On va juste remplacer Users par Employee...

Lorsque j'ai terminé, de nouveaux getters et setters devraient être créés, ils permettent de définir et d'accéder aux propriétés d'un objet depuis celui qui lui est relié. Cependant, je vais ici favoriser une autre méthode pour l'étape suivante.

En effet, je dois pouvoir trier l'affichage des employés en fonction du client auquel ils appartiennent. Pour cela je vais aller définir une méthode dans le fichier `EmployeeRepository.php`. Cela me permet de développer un peu sur le langage DQL (Doctrine Query Language) qui est un langage de requêtes basé sur l'objet, pas sur la relation entre les tables. C'est un langage qui demande d'abstraire les requêtes, ce qui les rend facilement transposables. Ainsi, on ne raisonne pas en colonne mais en propriété d'objet. Un exemple sera plus facile à appréhender.

```
class EmployeeRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Employee::class);
    }

    /**
     * @return Employee[]
     */
    public function findByClientId($value)
    {
        return $this->createQueryBuilder('e')
            ->andWhere('e.client = :val')
            ->setParameter('val', $value)
            ->orderBy('e.id', 'ASC')
            ->setMaxResults(10)
            ->getQuery()
            ->getResult();
    }
}
```

src/Repository/EmployeeRepository.php

J'utilise le `QueryBuilder` qui est un outil incluant des méthodes permettant de simplifier l'écriture de requêtes. Ici c'est une requête très simple où j'utilise un alias 'e' qui par voie d'usage est la première lettre de mon objet (`Employee`) et le représente. Ainsi, lorsque j'indique `e.client = :val` je crée une condition sur la propriété `client` de l'objet `Employee`. De cette façon, je peux désormais appeler cette méthode du Repository afin d'affiner ma recherche d'objets.

```

/**
 * @Route("client/{id}/employees", name="client_employees")
 */
public function clientEmployees(EmployeeRepository $repository, Client
$client)
{
    $employeeList = $repository->findByClientId($client);
    $classNamePlural = "employés" ;

    return $this->render('front/employees.html.twig',
compact('employeeList', 'client', 'classNamePlural'));
}

```

src/Repository/EmployeeRepository.php

Pour développer un peu plus la puissance du QueryBuilder, je vais utiliser l'entité Actuality. Le client va pouvoir déterminer la date à laquelle est publié l'article mais aussi la date à laquelle il est retiré de façon automatique. Cependant, il y a certains actualités que le client peut vouloir afficher en permanence, dans ce cas il a deux possibilités :

- Laisser le champ de date de fin vide
- Provoquer une permutation de la propriété always\_display de false à true.

La nuance est que le client peut choisir dans un premier temps de publier un article entre deux dates données, mais ensuite il peut décider de le laisser toujours affiché d'un simple clique sur le bouton de switch.

La requête va être construite de façon différente. Je veux les actualités dont la date de début est antérieure à aujourd'hui et où soit :

- La date de fin est indéfinie
- La date de fin est postérieure à aujourd'hui
- La propriété always\_display est définie sur « true »

```

/**
 * @return Actuality[] Returns an array of Actuality objects
 */
public function findByDateOrDisplay($value)
{
    $qb = $this->createQueryBuilder('a');
    return $qb
        ->andWhere('a.client = :val')
        ->andWhere($qb->expr()->gte(':now', 'a.start_date'))
        ->andWhere($qb->expr()->orX(
            $qb->expr()->isNull('a.end_date'),
            $qb->expr()->lte(':now', 'a.end_date'),
            $qb->expr()->eq('a.always_display', ':true')
        ))
        ->setParameters(new ArrayCollection([
            new Parameter('val', $value),
            new Parameter('now', new \DateTimeImmutable(),
Types::DATE_IMMUTABLE),
            new Parameter('true', 1),
        ]))
        ->orderBy('a.end_date', 'ASC')
        ->getQuery()
        ->getResult()
    ;
}

```

src/Repository/ActualityRepository.php

J'ai utilisé la classe Expr qui rajoute des méthodes très utiles pour la construction des requêtes.

Pour rester sur l'entité Actuality, elle m'a permis d'expérimenter une fonctionnalité intéressante et va me permettre de parler des routes. Les routes sont définies comme je l'ai montré brièvement par des annotations dans les contrôleurs. En revanche, quid des routes qui doivent être définies en fonction d'un paramètre comme l'identifiant d'un client ou le slug d'une page ? À partir du moment où nous avançons sur le site d'une mairie il y a un identifiant de client qui est transmis:

```

/**
 * @Route("client/{id}/actualities", name="client_actualities")
 */
public function clientActualities(ActualityRepository $repository, Client
$client)
{
    $actualityList = $repository->findByDateOrDisplay($client);
    $classNamePlural = "actualités" ;

    return $this->render('front/actualities.html.twig',
compact('actualityList', 'client', 'classNamePlural'));
}

```

src/Controller/FrontController.php

Il permet aux requêtes d'aller cibler le bon objet.

Maintenant, je dois faire une requête sur un article spécifique appartenant à un client. Pour que ma requête s'effectue bien, je vais devoir fournir l'identifiant du client et un élément distinctif de l'article. J'aurais pu choisir un ID en valeur numérique, cela aurait été relativement simple à mettre en place, mais pour rappel la partie « /client/{id} » devra être transparente pour le client, donc invisible. En revanche, l'URL final comportera le paramètre de l'actualité, alors autant le rendre lisible et compréhensible par l'utilisateur. Pour cela je vais utiliser un slug, un élément distinctif d'une URL.

Je vais donc commencer par créer une propriété slug dans ma classe Actuality, puis dans mon formulaire de création je rajoute un champ de type SlugField qui est déjà prévu à cet effet et qui me permet d'indiquer sur quelle propriété de l'objet le slug doit être basé.

```
/**  
 * @ORM\Column(type="string", length=255)  
 */  
private $slug;
```

src/Entity/Actuality.php

```
public function configureFields(string $pageName) : iterable  
{  
    return [  
        TextField::new('title', 'Titre'),  
        SlugField::new('slug', 'Slug')  
            ->setTargetFieldName('title')  
            ->addCssClass('visually-hidden'),  
        DateField::new('start_date', 'Date de début'),  
        DateField::new('end_date', 'Date de fin'),  
        BooleanField::new('always_display', 'Affichage permanent'),  
        TextField::new('short_description', 'Description courte'),  
        TextEditorField::new('description', 'Description'),  
        CollectionField::new('images', "Images")  
            ->setEntryType(ImageType::class)  
            ->setFormTypeOption('by_reference', false)  
            ->onlyOnForms()  
    ];  
}
```

src/Entity/Actuality.php

Cela fonctionne très bien... Tant qu'on ne modifie pas le titre de l'article. Le slug n'est effectivement pas mis à jour dans ces conditions.

Dans ce cas ce n'est pas très grave, je vais utiliser un EventSubscriber avec l'événement BeforeEntityUpdatedEvent et redéfinir le slug à partir d'ici en allant chercher le titre et en le « sluggifiant » .

```

public function updateSlug(BeforeEntityUpdatedEvent $event)
{
    $entity = $event->getEntityInstance();

    if (is_callable([$entity, 'setSlug'])) {
        $slug = $this->slugger->slug(strtolower($entity->getTitle()));
        $entity->setSlug($slug);
    }
    return;
}

```

src/EventSubscriber/EasyAdminSubscriber.php

Maintenant que j'ai mon slug, je peux l'utiliser dans ma route pour aller cibler le bon objet.

```

/**
 * @Route("client/{id}/actualities/{slug}", name="show_actuality")
 * @ParamConverter("actuality", options={"mapping": {"id": "client_id",
 "slug": "slug"}})
 */
public function showActuality(Client $client, Actuality $actuality)
{
    return $this->render('front/show.actuality.html.twig',
compact('client', 'actuality'));
}

```

src/Controller/FrontController.php

Depuis le début, j'utilise le ParamConverter pour transformer mes paramètres en objets et faciliter l'injection dans mes méthodes. Ici, je dois définir clairement de quelle entité les paramètres empruntent la propriété et la propriété en question.

La quatrième entité qu'il me paraît intéressant à développer est l'entité Contact. C'est une entité qui regroupe les clés étrangères de toutes les tables qui ont besoin de contacts, à savoir : Client, Structure, Equipment, Employee et ContactType.

Au niveau de la création il n'y a pas de différence, seulement c'est son lien avec l'entité ContactType et l'usage que j'en fais qui est important, car je le refais plusieurs fois pour d'autres entités.

Autrement dit, c'est une table qui permet de définir les moyens permettant d'entrer en contact avec les objets qui lui sont liés. Dans les faits, c'est une table qui a cette structure :

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	
2	contact_type_id	int(11)			Non	Aucun(e)			
3	employee_id	int(11)			Oui	NULL			
4	value	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			
5	structure_id	int(11)			Oui	NULL			
6	equipment_id	int(11)			Oui	NULL			
7	client_id	int(11)			Oui	NULL			

J'ai une relation OneToOne entre Contact et ContactType, ce qui signifie qu'un objet Contact (une adresse mail) n'est caractérisé que par un seul type de contact (email). Pour rajouter des contacts, il faut mettre à jour les CRUD des entités qui auront besoin de contacts, prenons l'exemple de Equipment:

```
public function configureFields(string $pageName): iterable
{
    return [
        // ...
        CollectionField::new('contacts', 'Contacts')
            ->setEntryType(ContactValueType::class)
            ->setFormTypeOption('by_reference', false)
            ->onlyOnForms(),
        // ...
    ];
}
```

src/Controller/Admin/EquipmentCrudController.php

J'ai créé un FormType dans lequel je spécifie deux champs : La valeur et le type de contact. Le type de champs EntityType permet d'aller chercher une Entité définie juste après et ensuite de choisir la clé utilisée pour l'affichage dans le champ de sélection.

```
class ContactValueType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options):
void
    {
        $builder
            ->add('value', TextType::class)
            ->add('contactType', EntityType::class, [
                'class' => ContactType::class,
                'choice_label' => 'title'
            ])
        ;
    }
    // ...
}
```

src/Controller/Admin/EquipmentCrudController.php

Il ne me reste plus qu'à créer quelques types :

## Types de contact

[Créer un type de contact](#)

<input type="checkbox"/>	Titre	Pictogramme	Préfixe	Type	<a href="#">...</a>
<input type="checkbox"/>	Email	uil uil-envelope	mailto:	Email	<a href="#">...</a>
<input type="checkbox"/>	Téléphone	uil uil-phone	tel:	Téléphone	<a href="#">...</a>

2 résultats

[Précédent](#) [1](#) [Suivant](#)

Ainsi notre modification du CrudController prend sens. Le CollectionField permet de choisir un FormType personnalisé qui sera itérable et permettra dans le cas présent d'ajouter autant d'objets que nécessaire.

Contacts

Value	0612345678	
Contact type	Téléphone	
Value	mail-ex@mp.le	
Contact type	Email	

[+ Ajouter un nouvel élément](#)

Pouvoir créer des objets est intéressant mais dans la mesure où cette plateforme est destinée à mutualiser plusieurs clients il faut trouver un moyen pour que les clients n'aient pas besoin de remplir eux même leur ID de client pour faire l'association entre les objets créés et eux. Je vais donc faire en sorte que l'ID de client de l'utilisateur courant (i.e celui qui crée les objets) s'applique automatiquement aux objets qui nécessitent un ID de client.

Je vais donc de nouveau utiliser un EventSubscriber. Étant donné que c'est une classe tout à fait classique, je vais instancier le service Security qui comprend les données de l'utilisateur et qui nous seront utiles, et définir ma méthode pour attribuer l'ID de Client de l'utilisateur courant à l'objet qu'il est en train de créer.

```

class EasyAdminSubscriber implements EventSubscriberInterface
{
    private $security;
    private $slugger;
    private $entityManager;
    private $passwordEncoder;

    public function __construct(Security $security, SluggerInterface $slugger,
        EntityManagerInterface $entityManager, UserPasswordHasherInterface
        $passwordEncoder)
    {
        $this->security = $security;
        $this->slugger = $slugger;
        $this->entityManager = $entityManager;
        $this->passwordEncoder = $passwordEncoder;
    }

    ...

    public function setClientId(BeforeEntityPersistedEvent $event)
    {
        $entity = $event->getEntityInstance();

        if (is_callable([$entity, 'setClient'])) {
            $client = $this->security->getUser()->getClient();
            $entity->setClient($client);
        }
        return;
    }

    ...
}

```

src/EventSubscriber/EasyAdminSubscriber.php

Par la suite, la nécessité de localiser automatiquement le client et ses équipements s'est fait ressentir. En revanche, il n'était pas concevable de demander au client d'aller chercher sa latitude et longitude ou celle des équipements car c'est beaucoup trop chronophage et le but est de leur faciliter les choses. Dans un premier temps, j'ai donc mis en place un système d'autocomplétion en AJAX du champ d'adresse.

J'ai utilisé l'API du gouvernement disponible ici : <https://adresse.data.gouv.fr/>.

Pour faire la requête et traiter le retour, j'ai utilisé la librairie autocomplete.js disponible ici : <https://tarekraafat.github.io/autoComplete.js/#/>

La requête est très transparente sur ce qu'elle renvoie (Annexe 3 p.66) et on comprend que le script va manipuler les données et les replacer dans un tableau plus pratique à exploiter. (Annexe 4 p.67)

Ainsi, lorsque le client commence la saisie, des adresses lui sont proposées.



Lorsqu'il choisit une adresse, des champs cachés sont automatiquement remplis : Code postal, ville, latitude et longitude. Les deux derniers champs sont prévus pour être réutilisés avec une carte Leaflet (en cours de développement, un petit échantillon en Annexe 5 p.68).

J'ai également permis à l'utilisateur de personnaliser les couleurs du front. Pour cela, j'ai crée une entité Customization (car par la suite il n'y aura pas que la couleur de modifiable) dans laquelle j'ai défini trois propriétés pour trois couleurs.

Puis, je me suis intéressé à la façon de proposer différentes couleurs sans les inscrire en dur. Il faut pouvoir permettre à l'utilisateur de choisir les couleurs qu'il veut avec un sélecteur de couleur, mais il faut aussi proposer des couleurs prédéfinies dans le cas où l'utilisateur n'a pas d'idées.

Pour le sélecteur de couleur, c'est très simple :

```
public function configureFields(string $pageName) : iterable
{
    return [
        AssociationField::new('client', 'Client')->onlyOnIndex(),
        ColorField::new('mainColor', 'Couleur principale')->showValue(),
        ColorField::new('secondaryColor', 'Couleur secondaire')-
>showValue(),
        ColorField::new('backgroundColor', 'Couleur de fond')-
>showValue(),
    ];
}
```

src/EventSubscriber/EasyAdminSubscriber.php

En revanche, lorsqu'il s'agit des couleurs prédéfinies, la tâche se complexifie. Je pourrais hypothétiquement renseigner les couleurs en dur dans le CrudController, mais cela n'est pas maintenable.

## Modifier les couleurs personnalisées

Sauvegarder et continuer l'édition Sauvegarder les modifications

Couleur principale  
  
 Utilisée pour les interactions, les boutons et les liens par exemple.

Couleur secondaire  
  
 Utilisée pour les éléments esthétiques.

Couleur de fond  
  
 Utilisée comme couleur de fond.

J'ai découvert que je pouvais stocker des données dans un fichier yaml qui me sert à configurer des Services. Les Services de Symfony sont ce qui permet à l'application de fonctionner, l'exemple le plus basique est Mailer. J'ai donc commencé par créer un fichier colors.yaml où recenser mes configurations possibles.

```
services:
  _defaults:
    autowire: true
    autoconfigure: true

  App\Service\ColorChoice:
    arguments:
      $primaryColors:
        - '<span class="text-black-50">Ignorez ce champ si vous préférez remplir les valeurs personnalisées <i class="fas fa-arrow-up"></i></span>':
          null
        - '<input type="color" value="#5eb9f0"> #5eb9f0'
        - '<input type="color" value="#3f78e0"> #3f78e0'
        - '<input type="color" value="#605dba"> #605dba'
        - '<input type="color" value="#747ed1"> #747ed1'
        - '<input type="color" value="#a07cc5"> #a07cc5'
        - '<input type="color" value="#d16b86"> #d16b86'
        - '<input type="color" value="#e668b3"> #e668b3'
        - '<input type="color" value="#e2626b"> #e2626b'
        - '<input type="color" value="#f78b77"> #f78b77'
        - '<input type="color" value="#fab758"> #fab758'
        - '<input type="color" value="#45c4a0"> #45c4a0'
        - '<input type="color" value="#7cb798"> #7cb798'
```

config/packages/services/colors.yaml

Puis, j'ai crée une classe ColorChoice.php dans mon dossier Service qui me permet d'aller chercher les configurations du fichier yaml liées par le nom des arguments.

```

<?php

namespace App\Service;

class ColorChoice
{
    /**
     * @var array
     */
    private $primaryColors;
// . .

    public function __construct(array $primaryColors, array $secondaryColors,
array $backgroundColors)
    {
        $this->primaryColors = $primaryColors;
// . .
    }

    public function choicePrimary()
    {
        return $this->primaryColors;
    }
}

```

config/packages/services/colors.yaml

J'importe `colors.yaml` dans `services.yaml` sans oublier d'exclure ma classe de service afin d'éviter un double appel du service qui ne fonctionnerait pas.

```

imports:
- { resource: 'services/colors.yaml' }

services:
# default configuration for services in *this* file
    _defaults:
        autowire: true      # Automatically injects dependencies in your
services.
        autoconfigure: true # Automatically registers your services as
commands, event subscribers, etc.

# makes classes in src/ available to be used as services
# this creates a service per class whose id is the fully-qualified class
name
App\:
    resource: '../src/'
    exclude:
        - '../src/Service/ColorChoice.php'
        - '../src/DependencyInjection/'
        - '../src/Entity/'
        - '../src/Kernel.php'
        - '../src/Tests/'

```

config/services.yaml

Je reviens dans mon CrudController pour ajouter les champs

```

public function configureFields(string $pageName) : iterable
{
    return [
        //...
        ChoiceField::new('presetMainColor', 'Couleur prédefinie principale')
            ->onlyOnForms()
            ->escapeHTML(false)
            ->setChoices($this->colors->choicePrimary()),
        ChoiceField::new('presetSecondaryColor', 'Couleur prédefinie secondaire')
            ->onlyOnForms()
            ->escapeHTML(false)
            ->setChoices($this->colors->choiceSecondary()),
        ChoiceField::new('presetBackgroundColor', 'Couleur de fond prédefinie')
            ->onlyOnForms()
            ->escapeHTML(false)
            ->setChoices($this->colors->choiceBackground()),
    ];
}

```

src/Controller/Admin/CustomizationCrudController.php

En l'état, ils ne sont reliés à rien, alors je vais m'amuser un peu dans ma classe de customisation et rajouter des propriétés qui ne correspondent à rien en base de données. En revanche ils font le lien entre le formulaire et l'entité, alors ils sont essentiels. Le principe c'est que les valeurs de ces propriétés remplissent quand même mes champs en base. Je peux le faire de plusieurs façons, mais voici comment j'ai préféré le faire :

```

public function setPresetMainColor(string $presetMainColor)
{
    $this->presetMainColor = $presetMainColor;
    $this->setMainColor($presetMainColor);

    return $this;
}

```

src/Controller/Admin/CustomizationCrudController.php

Notons que je place les champs de couleur prédefinies après les sélecteurs de couleur non pas parce que chronologiquement je les ai fait après mais parce qu'ils peuvent garder une valeur vide, alors que les sélecteurs sont par défaut en hexadécimal #000000.

## 5.5 Upload des images

Pour la gestion des images, j'ai utilisé le bundle VichUploaderBundle dont la documentation se trouve sur Github et sur le site de Symfony. De la même façon que pour le bundle des Fixtures, je vais l'installer grâce à composer. Ce bundle va

considérablement faciliter l'upload d'images. Il ne va pas stocker d'images en base mais sur le serveur. En revanche, en base je vais stocker le nom du fichier qui sera ensuite traité. Dans le fichier `vich_uploader.yaml` je peux gérer différents chemins où les images vont être stockées.

```
vich_uploader:
    db_driver: orm

    mappings:
        employees_images:
            uri_prefix: /upload/images/employees
            upload_destination: '%kernel.project_dir%/public/upload/images/
employees'
            namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
            inject_on_load: false
            delete_on_update: true
            delete_on_remove: true
```

src/Controller/Admin/CustomizationCrudController.php

Ensuite, je dois relier ce chemin à une entité et une propriété. Dans le cas où je suis dans une entité qui ne nécessite qu'une seule image, je peux créer le champ d'upload d'image dans l'entité directement concernée. Ce champ n'est pas persisté en base, il permet simplement d'upload le fichier sur le serveur. En parallèle, il faut créer un champ qui stocke le nom du fichier et un champ `updatedAt` qui prend automatiquement le moment d'envoi de l'image et permet de provoquer les écouteurs d'événements.

```
use Vich\UploaderBundle\Mapping\Annotation as Vich;

/**
 * @ORM\Entity(repositoryClass=EmployeeRepository::class)
 * @Vich\Uploadable
 */
class Employee
{
    // ...

    /**
     * @Vich\UploadableField(mapping="employees_images", fileNameProperty="image")
     * @var File|null
     */
    private $imageFile;

    /**
     * @ORM\Column(type="datetime", nullable=true)
     * @var \DateTimeInterface|null
     */
    private $updatedAt;

    // ...

    public function setImageFile(?File $imageFile = null): void
    {
        $this->imageFile = $imageFile;

        if (null !== $imageFile) {
            $this->updatedAt = new \DateTimeImmutable();
        }
    }

    public function getImageFile(): ?File
    {
        return $this->imageFile;
    }
}
```

src/Entity/Employee.php

Autrement, pour les cas où on veut mettre plusieurs images, il est mieux de créer une table dédiée qui sera reliée aux entités qui ont besoin d'images.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier  Supprimer  Plus
<input type="checkbox"/>	2 presentation_id	int(11)			Oui	NULL			Modifier  Supprimer  Plus
<input type="checkbox"/>	3 actuality_id	int(11)			Oui	NULL			Modifier  Supprimer  Plus
<input type="checkbox"/>	4 image	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier  Supprimer  Plus
<input type="checkbox"/>	5 updated_at	datetime			Oui	NULL			Modifier  Supprimer  Plus
<input type="checkbox"/>	6 structure_id	int(11)			Oui	NULL			Modifier  Supprimer  Plus

Pour exploiter les champs et pouvoir uploader plusieurs images ou documents (car oui VichUploaderBundle permet l'upload de fichiers, pas seulement d'images) j'ai crée un FormType que j'ai intégré aux CrudController des entités liés à la table image.

```
class ImageType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options):
void
{
    $builder
        ->add('imageFile', VichFileType::class)
;
}
```

src/Form/ImageType.php

```
public function configureFields(string $pageName) : iterable
{
    return [
        TextField::new('short_title', 'Description courte'),
        TextEditorField::new('description', 'Description'),
        CollectionField::new('images', "Images")
            ->setEntryType(ImageType::class)
            ->setFormTypeOption('by_reference', false)
            ->onlyOnForms()
    ];
}
```

src/Controller/Admin/PresentationCrudController.php

## 5.6 Gestion du Front

### 5.6.1 Twig

Twig est un générateur de gabarit utilisé par Symfony afin de dynamiser les pages et permet de créer des variables, des boucles, des conditions mais surtout des blocks réutilisables et ce de façon concise. Cela permet de séparer de façon plus distincte la logique et la vue, les deux ayant été gérées par PHP à l'origine. La syntaxe de Twig est

son grand atout, facilement discernable et très facile à comprendre il devient très aisément de récupérer les données envoyées par les contrôleurs et donc de les exploiter.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="{% block meta_description %}{% endblock %}">
    <title>{% block title %}{% endblock %}</title>

  {# . . .
  </head>
```

template/base.html.twig

Si on observe cette partie de code provenant de mon template de base, on remarque que la syntaxe Twig permet de substituer des informations qui seront générées dynamiquement par la suite selon les pages. De fait, pour le cas présent, tous mes autres fichiers de vue auront cette structure :

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

{% block title 'Actualités' | title %}
{% block meta_description %}Retrouvez nos actualités...{% endblock %}

{% block body %}

{# . . .

{% endblock body %}
```

template/front/actualities.html.twig

## 5.6.2 Le thème et Webpack

Le front a été mis en place à l'aide d'un thème Bootstrap appelé « Sandbox » et dont un exemple en ligne est disponible ici : <https://sandbox.elemisthemes.com/index.html>. Le thème venant avec beaucoup de fichiers CSS et de JavaScript, j'ai utilisé le bundle Webpack Encore pour gérer le contenu du dossier assets. En effet, cette librairie inclut Webpack qui « emballe » les fichiers CSS et JavaScript, les regroupe et les minifie. Ainsi, une structure classique qui devrait potentiellement nécessiter une dizaine de librairies, permet de tout regrouper dans un seul fichier. Pareillement pour les fichiers CSS. Le fonctionnement de Webpack Encore repose sur un seul fichier : [webpack.config.js](#) qui permet de configurer son fonctionnement.

```

Encore
    // directory where compiled assets will be stored
    .setOutputPath('public/build/')
    // public path used by the web server to access the output path
    .setPublicPath('/build')
    // only needed for CDN's or sub-directory deploy
    //.setManifestKeyPrefix('build/')

    /*
     * ENTRY CONFIG
     *
     * Each entry will result in one JavaScript file (e.g. app.js)
     * and one CSS file (e.g. app.css) if your JavaScript imports CSS.
     */
    .addEntry('app', './assets/app.js')

```

template/front/actualities.html.twig

Ici il désigne un fichier app.js qui servira d'entrée des scripts lors du build (nom du dossier où se trouvent les fichiers compilés et action de compilation). Dans ce fichier app.js j'importe les fichiers dont je vais avoir besoin :

```

import './css/plugins.css';
import './css/style.css';
import './css/colors/custom.css';
import './css/fonts/dm.css';

```

assets/app.js

Je peux aussi bien y inclure du CSS comme du JavaScript, dans tous les cas, au final j'obtiendrai un fichier CSS et un fichier JavaScript.

### 5.6.3 Construction des vues

J'ai construit les vues après avoir mis en place un certain nombre d'entités pour avoir des informations à fournir aux templates, et pas seulement des espaces vides. Je vais donc fournir des exemples notables de mon utilisation de Twig et du thème Sandbox. Symfony reposant sur une structure Modèle Vue Contrôleur (MVC).

Le cas le plus basique pour la construction d'un template a été de reprendre un modèle de page pour y insérer les données souhaitées parfois de façon conditionnelles. En amont, le contrôleur fait appel au Repository qui va s'occuper d'aller chercher les données, je stocke le résultat de la recherche dans une variable que je passe à la vue. Ainsi, je peux accéder à mes données avec Twig. En pratique, cela donne un résultat qui ressemble à cela :

```

/**
 * @Route("client/{id}/equipment/{slug}", name="show_equipment")
 * @ParamConverter("equipment", options={"mapping": {"id": "client_id",
 "slug": "slug"}})
 */
public function showEquipment(Client $client, Equipment $equipment,
EquipmentRepository $repo)
{
    $equipmentList = $repo->findThreeExceptCurrent($client, $equipment);
    return $this->render('front/showEquipment.html.twig',
compact('client', 'equipment', 'equipmentList'));
}

```

src/Controller/FrontController.php

```

// . . .

{%
    for equipment in equipmentList %}
        <div class="item col-md-4">
            <div class="item-inner">
                <article>
                    <div class="card custom-border">
                        <figure class="card-img-top overlay overlay-1 hover-scale"><a href="{{path('show_equipment', {'id':client.id, 'slug':equipment.slug})}}> </a>
                        <figcaption>
                            <h5 class="from-top mb-0">Voir plus</h5>
                        </figcaption>
                    </figure>
                    <div class="card-body">
                        <div class="post-header">
                            <div class="post-category text-line text-pale-primary">
                                <span>{{equipment.equipmenttype}}</span>
                            </div>
                        </div>
                    </div>
                </article>
            </div>
        </div>
    {% endfor %}
// . . .

```

src/Controller/FrontController.php

Twig m'a aussi permis de construire des vues conditionnelles. Par exemple, si la variable transmise par le contrôleur est vide, alors j'inclus un fichier partiel.

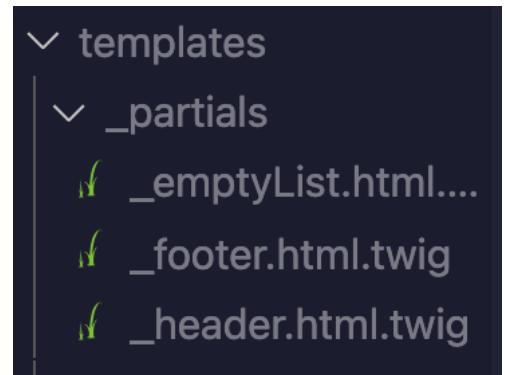
```

//. .
{%
    if equipmentList is empty %}
        {{ include('_partials/_emptyList.html.twig') }}
    {% else %}
//. .

```

src/Controller/FrontController.php

Les fichiers partiels sont des fichiers de template qui comprennent des morceaux de page intégrables dans des templates plus globaux. Ainsi, le header et le footer sont dans des fichiers séparés pour éviter la répétition de code et pour que chaque template ne serve qu'à la fonctionnalité qui lui est liée. Pour les trouver et les reconnaître, ils sont classés dans un dossier \_partials et ont une nomenclature spécifique qui utilise l'underscore « \_ » en début de nom.



Ensuite, bien que le thème soit très riche et très pratique, il ne convenait pas totalement aux besoins et aux idées que j'avais. J'ai donc à de nombreuses reprises retravaillé les templates pour mieux coller au besoin. Cela passait par la suppression ou l'ajout de carrousel, la refonte des marges internes et externes de certains éléments, la modification des classes utilisées ou même régulièrement la permutation de liens pour de simples éléments esthétiques et indicatifs. (Exemple en Annexe 6 p.68-69)

J'ai par ailleurs totalement conçu le design et intégré les pages de contact, de connexion, de réinitialisation de mot de passe et du changement de mot de passe (Annexe 7 p.69-70)

Twig permet l'utilisation de filtres qui permettent d'agir sur le contenu. La syntaxe pour les appeler utilise le symbole de la ligne verticale « | » puis le nom du filtre et parfois des paramètres. Un exemple simple : lors de la mise en place de la vue de la page des équipements, la description était trop longue pour que le rendu soit acceptable. J'ai alors découvert qu'il existait un filtre qui permet de tronquer les chaînes de caractères :

Le filtre raw est très utilisé pour mes descriptions, il permet d'interpréter les balises HTML. De fait, le champ de description du CRUD étant un éditeur de texte, cela permet d'avoir un rendu fidèle pour la description.

J'ai aussi poussé la chose un peu plus loin en créant mon propre filtre Twig. À l'origine j'étais obligé de concaténer deux appels de variables pour mon entité contact. Donc, plutôt que d'avoir ça :

```
//...
{%
    for contact in equipment.contacts %}
        <a href="{{ contact.contactType.prefix }}{{ contact.value }}" class="mx-3">
            <i class="{{ contact.contactType.picto }}></i>
        </a>
    {% endfor %}
//...
```

templates/front/equipments.html.twig

J'ai ça :

```
//. . .
{%
    for contact in equipment.contacts %}
        <a href="{{ contact | contact_href }}" class="mx-3">
            <i class="{{ contact.contactType.picto }}></i>
        </a>
{%
    endfor %}
//. . .
```

templates/front/equipments.html.twig

Grâce à ça :

```
//. . .
class AppExtension extends AbstractExtension
{
    public function getFilters(): array
    {
        return [
            new TwigFilter('contact_href', [$this, 'contactHref']),
            new TwigFilter('full_address', [$this, 'fullAddress'])
        ];
    }

    public function contactHref(Contact $contact)
    {
        return $contact->getContactType()->getPrefix().$contact->getValue();
    }
//. . .
```

templates/front/equipments.html.twig

Une simple fonction `getFilters()` qui répertorie les filtres personnalisés et les associe à une fonction. Je n'ai d'ailleurs même pas besoin d'appeler ce fichier dans mes contrôleurs liés à mes vues, de la même façon que je n'ai pas à appeler Twig pour l'utiliser de base.

Twig permet aussi la définition de variables globales. Dans mon cas, j'ai voulu répertorier un Service pour un système de liens raccourcis vers des démarches administratives du gouvernement. Les démarches sont créées par le Super Admin qui renseigne un pictogramme issu de la librairie IconScout, un titre, une description et un lien. Je n'ai même pas besoin de créer une fonction customisée dans le Repository car je vais utiliser la méthode `findAll`. Pour la page contact, je redirige en fonction de l'ID du Client, j'ai expliqué ça plus tôt avec le ParamConverter.

Ensuite, je crée le service comme vu précédemment avec les couleurs et j'y intègre une fonction qui va appeler mon Repository.

```

<?php

namespace App\Service;
use App\Repository\ShortcutRepository;

class ShortcutList
{
    private $shortcuts;

    public function __construct(ShortcutRepository $shortcuts)
    {
        $this->shortcuts = $shortcuts;
    }

    public function getShortcutList()
    {
        return $this->shortcuts->findAll();
    }
}

```

src/Service/ShortcutList.php

Ensuite, je spécifie dans le fichier twig.yaml que mon service est une variable globale

```

twig:
    default_path: '%kernel.project_dir%/templates'
    globals:
        shortcutList: '@App\Service\ShortcutList'

```

config/packages/twig.yaml

Enfin, dans mon template, afin d'éviter de faire deux appels en base, je stocke le résultat dans une variable Twig, puis je l'appelle lorsque cela est nécessaire.

```

{% set shortcuts = shortcutList.getShortcutList() %}
    {% if shortcuts is not empty %}
        <li class="nav-item dropdown"><a class="nav-link dropdown-toggle"
href="#">Démarches</a>
            <ul class="dropdown-menu">
                <li class="nav-item"><a class="dropdown-item"
href="{{ path('client_shortcuts', {'id' : client.id})}}>Toutes les démarches</a></li>
                    {% for shortcut in shortcuts %}
                        <li class="nav-item d-flex align-items-center"><a
href="{{ shortcut.link is null ? path('client_contact', {'id' : client.id}) : shortcut.link }}"
class="dropdown-item"><i class="{{shortcut.picto}} me-2 fs-25"></i>{{ shortcut.title }}</a></li>
                    {% endfor %}
            </ul>
        </li>
    {% endif %}

```

template/\_partials/\_header.html.twig

## 5.7 Le déploiement avec Gitlab et Heroku

Dans la deuxième partie du stage j'ai découvert l'intégration continue et le déploiement continu (CI/CD) qui permet de livrer le produit en continu et de l'alimenter au fur et à mesure. Cela permet aussi de vérifier qu'une fonctionnalité ne dysfonctionne

pas avec le reste de l'application. Dans mon cas, c'était surtout une façon d'avoir une version en ligne consultable par de potentiels clients et par la partie marketing.

Dans ce projet, je n'ai pas à proprement parlé mis en place le CI/CD car même mon tuteur l'a découvert en cours de projet. Il a donc effectué l'initialisation et j'ai géré l'intégration continue par la suite. Voilà cependant ce que j'en ai compris :

Heroku est une plateforme en tant que service (PaaS) qui permet d'externaliser tout ce qui relève de l'architecture et de l'infrastructure des applications. Le développeur peut donc se concentrer sur son produit.

De mon côté je n'ai eu qu'à gérer un fichier lié au déploiement, le fichier `gitlab-ci.yml` et qui va plutôt gérer les tâches (jobs) et étapes (stages) des pipelines. Une pipeline c'est un composant clé de l'intégration continue avec Gitlab, c'est ce qui fait le lien entre le code que je propose et l'application déployée. C'est donc en quelques sortes un conteneur qui fait la jonction entre les deux interfaces que sont le développement et la production. Mon rôle a donc été de gérer les pipelines et plus particulièrement les « stages ». En effet, j'ai effectué les tests Cypress (qui sont détaillés dans la partie suivante) sur la plateforme en ligne et lorsque j'appliquais des modifications il fallait les déployer avant les tests.

The screenshot shows a GitLab pipeline interface for a feature branch named 'feat : MAIR-102-ContactPage'. The pipeline has four stages: 'fix : MAIR-102-ContactPage : Added a v...', 'feat : MAIR-102-ContactPage : updated ...', 'fix : MAIR-102-ContactPage : updated h...', and 'feat : MAIR-102-ContactPage: Added th...'. Each stage is triggered by a specific commit (e.g., #484377310, #484085983, #482823235, #482569279) and includes a green checkmark indicating successful execution. The pipeline status is shown as 'All threads resolved'.

Des pipelines et la stage de test

```
only:  
  - main
```

`.gitlab-ci.yml`

Cette ligne permet de choisir une condition de déploiement, ici je ne déploie que lorsqu'on merge sur la branche principale. Parfois je devais donc gérer le déploiement avant de merge et je manipulais donc cette ligne.

Plus haut, on retrouve le job de test que je retirais parfois également pour pouvoir déployer avant les tests et éventuellement apporter des correctifs. Enfin, je rétablissais les lignes une fois les tests rédigés et lorsque j'avais bien testé s'ils fonctionnaient je lançais le fichier complet en test puis en déploiement sur la branche principale. Lors de la soumission de la MR, seuls les tests sont effectués et le déploiement ne s'effectue donc que lorsque mon tuteur la valide et permet le merge.

## **6. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)**

### **6.1 Les tests Cypress**

J'ai utilisé Cypress pour effectuer des tests à partir du dernier tiers de mon stage. Cypress permet de réaliser des tests End-To-End, c'est à dire de bout-en-bout qui reproduisent le comportement de l'utilisateur sur le site afin de tester le bon fonctionnement des fonctionnalités.

Au delà de Cypress, pour abstraire un peu plus les tests et les rendre plus compréhensibles j'ai utilisé la syntaxe Gherkin rendue disponible par le pré-processeur Cucumber qui permet d'écrire des scénarios de tests parfaitement compréhensible par un humain qui ne connaît pas le code.

Cypress s'installe via npm (qui n'est pas un acronyme, mais qui se réfère à son utilitaire de commande), le gestionnaire de paquets de Node.

Il intègre une interface visuelle qui émule un navigateur de recherche et que je peux appeler à l'aide de la commande `node_modules/.bin/cypress open` et qui permet de visualiser l'exécution du test en temps réel. Cypress intègre également un système de « snapshots » ou « screenshots » qui permettent de revenir en arrière à chaque étape du test pour vérifier comment il s'est déroulé. Enfin, il enregistre une vidéo du déroulement du test.

Passons à la construction du test maintenant. La structure est assez simple à comprendre. Désormais, Cypress se base sur un fichier avec l'extension .feature dans

lequel je vais créer mes scénarios de tests. Ces scénarios sont dirigés par des clauses qui commencent toutes soit par « Given », « When », « Then » ou « And » et qui témoignent d'un état du test.

Given établi la situation initiale, When représente les actions effectuées et Then attend une assertion, autrement dit une confirmation ou un résultat. And ne sert qu'à concaténer des instructions de même nature et rend la lecture plus naturelle.

Admettons que j'aie écrit un premier Scenario :

```
Feature: The User
  Scenario: Creating a new user
    Given I login with user "jerome.gavignet@gmail.com" and password
    "Superadmin123456!"
      And I visit the page "/admin"
      When I click on "Utilisateurs"
      And I click on "Créer un utilisateur"
      And I fill "Prénom" with value "Toto"
      And I fill "Nom" with value "Dupont"
      And I fill "Adresse email" with value "toto.ch71@gmail.com"
      And I fill "Mot de passe" with value "test"
      And I choose value "ROLE_ADMIN" in select "Rôle"
      And I check the "Actif" input
      And I click on "Créer"
      Then I should see "Toto" in the column "Prénom"
      And I should see "Dupont" in the column "Nom"
```

Cypress/integration/admin/Users.feature

En l'état, il serait assez naïf d'attendre un résultat par ce simple fichier. Maintenant, je vais donc procéder à la liaison de ces clauses avec du JavaScript qui va simuler le comportement de l'utilisateur.

Pour cela, je crée un fichier dans lequel je vais créer des fonctions avec un format très particulier :

```
import { Given, When, Then } from "cypress-cucumber-preprocessor/steps";
const URL = "http://mairies.herokuapp.com"

Given('I login with user {string} and password {string}', (username, password)=>{
  cy.visit(URL + '/login')
  cy.get('[data-cy=username]')
  .type(username)
  cy.get('[data-cy=password]')
  .type(password)
  cy.get('[type=submit]')
  .click()
})

Given('I click on {string}', (buttonName)=>{
  cy.contains(new RegExp(`^${buttonName}$`, 'g'))
  .click()
})

When('I visit the page {string}', (path)=>{
  cy.visit(URL + path)
})
```

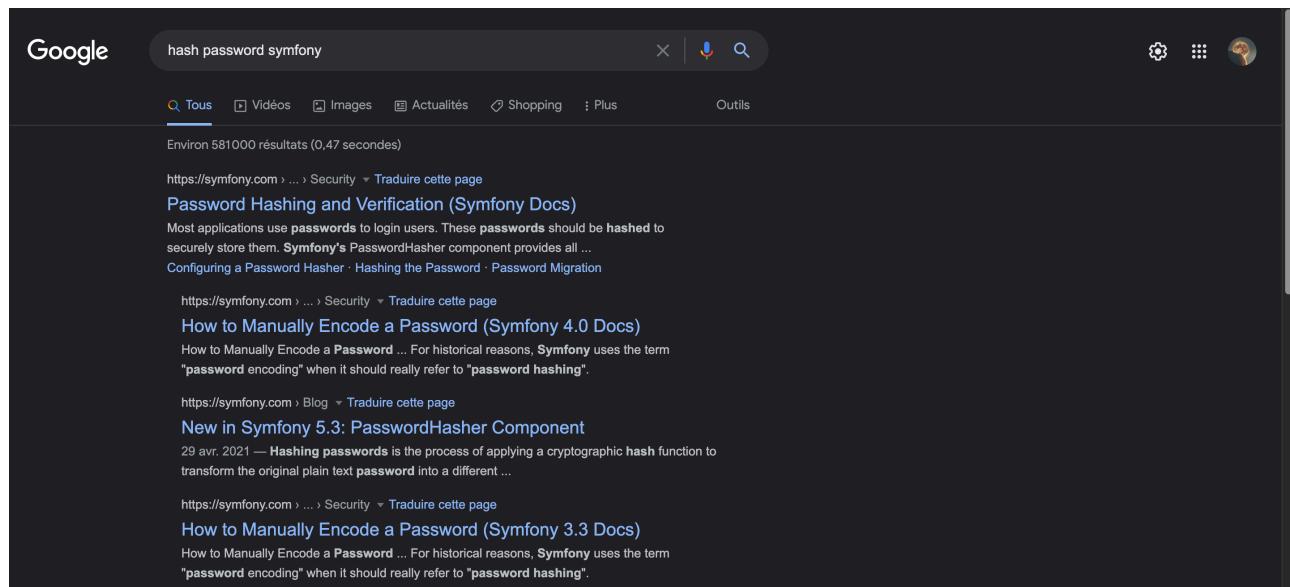
cypress/support/manage-user.js

Cela me permet de relier la fonction qui va suivre avec la clause indiquée. Maintenant pour indiquer au JavaScript d'agir sur l'interface de Cypress, je vais utiliser l'objet « cy » sur lequel je vais systématiquement appliquer des méthodes qui vont me permettre de naviguer. La documentation est extrêmement bien fournie et permet de saisir la puissance incroyable de cet outil. Je n'ai plus qu'à rédiger mes tests et les exécuter sur l'interface de Cypress (Annexe 8 p.71).

Bien sûr, le principe du test automatisé est de ne nécessiter aucune intervention humaine. Je dois donc veiller dans mes scénarios à toujours supprimer mes ajouts. Par exemple, si je crée un utilisateur, je vais tester la connexion, l'existence, mais si je ne le supprime pas le prochain test échouera car j'ai bien évidemment empêché la possibilité d'avoir deux fois le même utilisateur. Par ailleurs, cela évite d'avoir rapidement une centaine d'objets non supprimés dans la base de données et rend le test complet : L'application a le même état avant et après le test. Dans un contexte de test avant déploiement comme cela a été mon cas, ils ne doivent pas être perceptibles par l'utilisateur final mais doivent nous permettre de valider la fonctionnalité.

## 7. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

Sur la question de la sécurité, j'ai effectué ma veille sur la documentation officielle de Symfony qui décrit très bien les principes d'authentification, de vérification de l'utilisateur ainsi que d'autorisation d'accès. Dans la plupart des cas, j'écrivais des mots clés liés à mon problème et le premier résultat était souvent la documentation officielle. Par exemple « hash password symfony»



Le résultat de ma recherche

Je vais donc dans cette partie démontrer tout ce que j'ai utilisé pour sécuriser au mieux l'application.

Dans un premier temps, du côté de l'utilisateur j'ai montré comment bloquer le back-office aux utilisateurs qui ne disposent pas du bon rôle. Pour ce qui est de la connexion, Symfony a une façon presque magique de la gérer. En effet, via la configuration de ses « providers » et « firewall » depuis le fichier security.yaml. Le « provider » est en fait l'utilisateur courant. Ainsi pour accéder aux parties sécurisées de l'application, je peux définir où elle ira chercher cette instance de provider et à partir de quelle donnée qui lui est liée. Ici, je décide que le provider viendra de ma base de données et plus précisément de l'entité User. Pour être encore plus précis, la propriété à vérifier sera l'email.

Le « Firewall » prend en charge chaque requête et vérifie si elle nécessite un l’authentification d’un utilisateur, auquel cas il demandera l’authentification pour y accéder. Ici, je défini le provider de mon firewall, puis je lui défini le nom de la route de déconnexion, et enfin celle de connexion. Le système de connexion est une chose, mais il faut sécuriser les données des utilisateurs, notamment pour tout ce qui est relatif à cette connexion.

Je retourne dans le fichier `security.yaml` pour y ajouter la ligne `password_hasher` et définir un algorithme de hashage pour mon entité User dont je spécifie le namespace. Ici, je règle l’algorithme sur « auto » afin que soit perpétuellement défini l’algorithme le plus performant. Hasher un mot de passe c’est le traduire en une série de caractère qui le masque totalement grâce à un algorithme, c’est une méthode de protection de mot de passe basique. De fait, Symfony propose un service natif pour hasher les mots de passes. De retour dans mes fixtures, il me suffit d’instancier ce service et d’appeler la méthode `hashPassword()` sur les mots de passe à protéger.

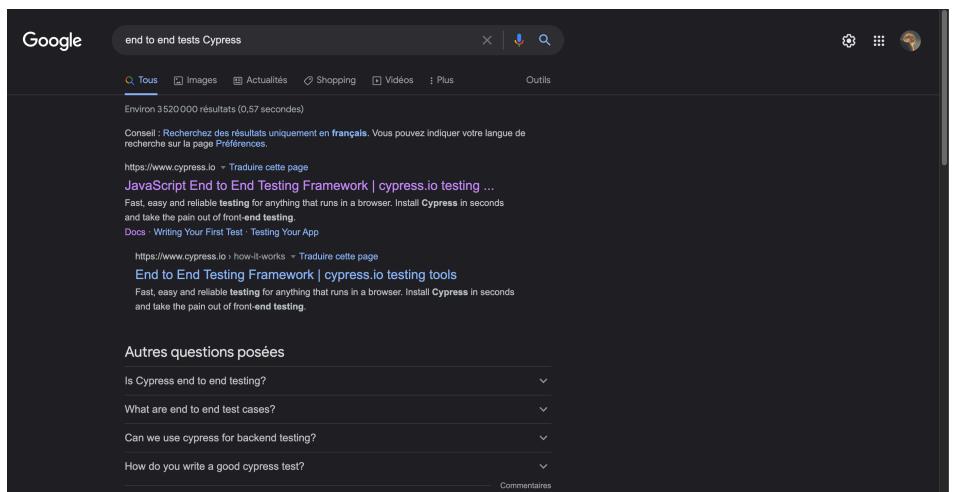
## 8. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone

Pendant le projet, j'ai beaucoup eu besoin de faire de recherches pour m'aider. Voici la description d'une de ces situations. Ici, je vais décrire une situation de recherche lorsque j'ai dû chercher à implémenter des tests end-to-end. Mon tuteur m'a indiqué la technologie à utiliser, et j'ai fait le reste. J'ai donc écrit la recherche suivante dans mon moteur de recherche : end to end tests Cypress.

Je suis bien sûr tombé sur la documentation de Cypress, qui guide très bien lors de l'utilisation, pour peu qu'on connaisse un minimum le fonctionnement des tests. Ce n'était pas mon cas, et si mon stage m'a appris quelque chose, c'est bien que regarder des vidéos YouTube de pontes de la pédagogie du développement comme GrafikArt ça m'ennuie profondément, donc je reste la plupart du temps cantonné à l'écrit lorsqu'il s'agit d'assimiler des notions importantes. Du coup, j'ai dû approfondir ma recherche et sortir de la documentation de Cypress qui ne me satisfaisait pas (pour le moment) pour aller chercher de la vulgarisation.

Parmi les liens que j'ai exploré, celui-ci a été le plus utile pour moi : <https://www.smashingmagazine.com/2021/09/cypress-end-to-end-testing/>

Il est récent, bien formaté et après l'avoir parcouru une première fois en travers pour voir les thèmes abordés, je commence la lecture et il est effectivement très complet.



## **9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes).**

Voici donc la partie que j'ai choisi d'extraire et de traduire :

So, we came to consider building our test suite anew. After visiting an unconference, I discovered Cypress.

Cypress is an all-in-one testing framework that does not use Selenium or WebDriver. The tool uses Node.js to start a browser under special control. The tests in this framework are run at the browser level, not just remote-controlling. That offers several advantages.

In short, here are the reasons why I chose this framework:

- **Excellent debugging capability**

Cypress' test runner can jump back to any state of the application via snapshots. So, we can directly see an error and all of the steps before it. In addition, there is full access to Chrome's developer tools (DevTools), and clicks are fully recorded.

- **Better ways to wait for actions in the test or UI or in the responses from the API**

Cypress brings implicit waiting, so there is no need for appropriate checks. You can also make the test wait for animations and API responses.

- **Tests are written in JavaScript**

This mitigates the learning curve to write tests. Cypress' test runner is open-source, so it fits our product strategy.

However, this article is a guide, so let's stop with this general information and get going.

Traduction :

Donc, nous en sommes venus à reconsidérer totalement notre façon de tester. Lors d'une inconférence (*ndlr : il semblerait que ce soit un néologisme de plus venu de l'autre côté de l'atlantique*) à laquelle j'ai assisté, j'ai découvert Cypress.

Cypress est un framework de test tout-en-un qui n'utilise pas Selenium ou WebDriver. Cet outil utilise Node.js pour démarrer un navigateur sous son contrôle exclusif. Les tests sont donc exécutés au niveau du navigateur, et pas seulement contrôlés à distance. Cela offre certains avantages.

En bref, voici les raisons qui m'ont fait choisir ce framework:

- **Une excellente aide pour résoudre les bugs**

L'interface graphique de Cypress permet de retourner aux états précédents de l'application via des clichés. Donc nous pouvons directement voir l'erreur ainsi que toutes les étapes qui la précèdent. Par ailleurs, elle offre un accès total aux outils de développement de Chrome (DevTools) et les cliques sont tous enregistrés.

- **Une meilleure manière d'attendre des actions lors de tests d'interface d'utilisateur ou lors de l'attente de réponses d'API.**

Cypress implémente un système d'attente implicite, il n'y a donc pas besoin de véritables vérifications. Il est aussi possible d'ajouter une délai d'attente à une étape pour laisser une animation se dérouler ou attendre une réponse d'API.

- **Les tests sont écrits en JavaScript.**

Ce qui permet d'adoucir la courbe de difficulté lors de l'écriture de tests. L'interface graphique de Cypress est en accès libre, ce qui correspond à la stratégie de notre produit.

Cependant, cet article est une guide, alors cessons les informations générales et allons plus en profondeur.

# Annexes

```
<?php

namespace App\Controller;

/**
 * @Route("/reset-password")
 */
class ResetPasswordController extends AbstractController
{
    use ResetPasswordControllerTrait;

    private $resetPasswordHelper;
    private $entityManager;

    public function __construct(ResetPasswordHelperInterface $resetPasswordHelper, EntityManagerInterface $entityManager)
    {
        $this->resetPasswordHelper = $resetPasswordHelper;
        $this->entityManager = $entityManager;
    }

    /**
     * Display & process form to request a password reset.
     *
     * @Route("", name="password_request")
     */
    public function request(Request $request, MailerInterface $mailer): Response
    {
        $form = $this->createForm(ResetPasswordRequestFormType::class);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            return $this->processSendingPasswordResetEmail(
                $form->get('email')->getData(),
                $mailer
            );
        }

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

    /**
     * Confirmation page after a user has requested a password reset.
     *
     * @Route("/check-email", name="check_email")
     */
    public function checkEmail(): Response
    {
        // Generate a fake token if the user does not exist or someone hit this page directly.
        // This prevents exposing whether or not a user was found with the given email address or not
        if (null === ($resetToken = $this->getTokenObjectFromSession())) {
            $resetToken = $this->resetPasswordHelper->generateFakeResetToken();
        }

        return $this->render('security/check_email.html.twig', [
            'resetToken' => $resetToken,
        ]);
    }

    /**
     * Validates and process the reset URL that the user clicked in their email.
     *
     * @Route("/reset/{token}", name="reset_password")
     */
    public function reset(Request $request, UserPasswordHasherInterface $userPasswordHasher, string $token = null): Response
    {
        if ($token) {
            // We store the token in session and remove it from the URL, to avoid the URL being
            // loaded in a browser and potentially leaking the token to 3rd party JavaScript.
            $this->storeTokenInSession($token);

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

        $token = $this->getTokenFromSession();
        if (null === $token) {
            throw $this->createNotFoundException("Aucun token n'a été trouvé dans l'URL ou la session.");
        }
    }
}
```

Annexe 1 (part 1) : src/Controller/ResetPasswordController.php

```

try {
    $user = $this->resetPasswordHelper->validateTokenAndFetchUser($token);
} catch (ResetPasswordExceptionInterface $e) {
    $this->addFlash('reset_password_error', sprintf(
        'Il a eu un problème lors de la validation de votre requête - %s',
        $e->getReason()
    ));
    return $this->redirectToRoute('password_request');
}

// The token is valid; allow the user to change their password.
$form = $this->createForm(ChangePasswordFormType::class);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
    // A password reset token should be used only once, remove it.
    $this->resetPasswordHelper->removeResetRequest($token);

    // Encode(hash) the plain password, and set it.
    $encodedPassword = $userPasswordHasher->hashPassword(
        $user,
        $form->get('plainPassword')->getData()
    );

    $user->setPassword($encodedPassword);
    $this->entityManager->flush();

    // The session is cleaned up after the password has been changed.
    $this->cleanSessionAfterReset();

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

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

private function processSendingPasswordResetEmail(string $emailFormData, MailerInterface $mailer):
RedirectResponse
{
    $user = $this->entityManager->getRepository(User::class)->findOneBy([
        'email' => $emailFormData,
    ]);

    // Do not reveal whether a user account was found or not.
    if (!$user) {
        return $this->redirectToRoute('check_email');
    }

    try {
        $resetToken = $this->resetPasswordHelper->generateResetToken($user);
    } catch (ResetPasswordExceptionInterface $e) {

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

    $email = (new Templatemail())
        ->from(new Address('contact@jgaweb.fr', 'JGAWeb Contact'))
        ->to($user->getEmail())
        ->subject('Votre demande de nouveau mot de passe')
        ->htmlTemplate('security/email.html.twig')
        ->context([
            'resetToken' => $resetToken,
        ])
        ;

    $mailer->send($email);

    // Store the token object in session for retrieval in check-email route.
    $this->setTokenObjectInSession($resetToken);

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

```

Annexe 1 (part 2) : src/Controller/ResetPasswordController.php

```

<?php

namespace App\Security\Voter;

use App\Entity\User;
use App\Entity\Client;
use Symfony\Component\Security\Core\Security;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\Authorization\Voter\Voter;
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;

class ClientVoter extends Voter
{
    const ACCESS = 'access';

    private $security;

    public function __construct(Security $security)
    {
        $this->security = $security;
    }

    protected function supports(string $attribute, $client): bool
    {
        return in_array($attribute, [self::ACCESS])
            && $client instanceof \App\Entity\Client;
    }

    protected function voteOnAttribute(string $attribute, $client,
    TokenInterface $token): bool
    {
        $user = $token->getUser();
        // if the user is anonymous, do not grant access
        if (!$user instanceof UserInterface) {
            return false;
        }
        // if the user is a superAdmin, grant access
        if ($this->security->isGranted('ROLE_SUPER_ADMIN')) {
            return true;
        }

        // otherwise verify if the user has a relation with the client
        switch ($attribute) {
            case self::ACCESS:
                return $this->canAccess($client, $user);
                break;
        }

        throw new \LogicException('This code should not be reached!');
    }

    private function canAccess(Client $client, User $user): bool
    {
        return in_array($user, $client->getUsers()->getValues());
    }
}

```

Annexe 2 : src/Security/Voter/ClientVoter.php

Annonce 3 : Capture de l'inspecteur. On voit clairement l'URL de la requête et les données.

```

$(function(){
    new autoComplete({
        selector: ".js-field-address-autocomplete",
        placeHolder: "Cherchez une adresse...",
        data: {
            src: async () => {
                try {
                    let query = document.querySelector(".js-field-address-autocomplete").value;
                    const source = await fetch("https://api-adresse.data.gouv.fr/search/?q=" + query + "&type=housenumber&autocomplete=1");
                    const data = await source.json();
                    const addresses = data.features;
                    let output = [];
                    addresses.forEach(function(address){
                        output.push(Object.assign({name: address.properties.name + " " + address.properties.city}, address));
                    })
                    return output;
                } catch (error) {
                    return error;
                }
            },
            keys: ["name"],
            cache: false
        },
        trigger: (query) => {
            return query.replace(/\g/, "").length; // Returns "Boolean"
        },
        resultsList: {
            element: (list, data) => {
                if (!data.results.length) {
                    // Create "No Results" message element
                    const message = document.createElement("div");
                    // Add class to the created element
                    message.setAttribute("class", "no_result");
                    // Add message text content
                    message.innerHTML = `<span>Found No Results for "${data.query}"</span>`;
                    // Append message element to the results list
                    list.prepend(message);
                }
            },
            noResults: true,
        },
        resultItem: {
            highlight: {
                render: true
            }
        },
        events: {
            input: {
                selection: (event) => {
                    document.querySelector(".js-field-address-autocomplete").value = event.detail.selection.value.properties.name;
                    document.querySelector(".js zipcode-autocomplete").value = event.detail.selection.value.properties.postcode;
                    document.querySelector(".js-town-autocomplete").value = event.detail.selection.value.properties.city;
                    document.querySelector(".js-latitude-autocomplete").value = event.detail.selection.value.geometry.coordinates[1];
                    document.querySelector(".js-longitude-autocomplete").value = event.detail.selection.value.geometry.coordinates[0];
                }
            }
        }
    });
})

```

# Eléments intégrés sans template



Annexe 5 : C'est un peu vide mais j'ai beaucoup d'idées pour faciliter l'utilisation et la consultation

**Sandbox** Demos ▾ Pages ▾ Projects ▾ Blog ▾ Blocks ▾ Documentation ▾ EN ▾ Contact



— TEAMWORK

## Amet Dolor Bibendum Parturient Cursus

Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit.

Mollis enim risus eget lacinia mollis eros velut. Nulla utinam sit libero, a pharetra augue. Proin

**About Us**  
Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum. Nulla vitae elit libero, a pharetra augue. Donec id elit non mi porta gravida at eget metus.

[Twitter](#) [Facebook](#) [Pinterest](#) [Instagram](#) [YouTube](#)

**Popular Posts**

-  **Magna Mollis Ultricies**  
26 Mar 2021 • 3
-  **Ornare Nullam Risus**  
16 Feb 2021 • 6
-  **Euismod Nullam Fusce**  
8 Jan 2021 • 5

**Categories**

- Teamwork (21)

Annexe 6 (part 1) : Le thème de base sur une page de blog

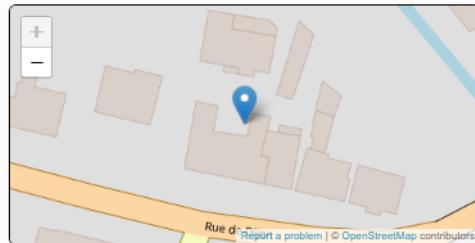


— TYPE 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam dictum, iacus sit amet mattis  
luctus, purus mauris consectetur nibh, sit amet tincidunt enim nunc non dolor. Proin tristique  
lorem eros, sed lacinia tellus dictum eu. Etiam elementum blandit fringilla. Aenean vel bibendum  
nisl, ac blandit metus.

### Contacts

- [Email](#)
- [Téléphone](#)



### Autres équipements



**Lore ipsum**

Lorem ipsum dolor...



**Lore ipsum**

Lorem ipsum dolor...

Annexe 6 (part 2) : Le même template mais retravaillé pour correspondre aux besoins



Si un compte avec votre adresse mail existe, vous recevrez un message de  
réinitialisation de mot de passe. Ce lien expire dans 1 heure.

Si vous ne recevez pas de mail, vérifiez vos spams ou bien [réessayez.](#)



Annexe 7 (part 1) : La page après avoir demandé une réinitialisation de mot de passe

Prénom *	Nom *
Email *	
Message *	



#### Adresse

2 Rue Souvert  
21220  
Gevrey-Chambertin

J'accepte que mes données soient réutilisées dans le but d'être recontacté par Municipalité de Genlis.

**Envoyer**

\* Champs obligatoires

Annexe 7 (part 2) : La page « Contact »

**NOS PARTENAIRES LOCAUX**


**ups !**

**Il semblerait que la liste des partenaires soit vide...**

Si vous êtes administrateur vous pouvez ajouter vos partenaires depuis l'**administration**.

Annexe 7 (part 3) : Le module qui s'affiche si des objets ne sont pas présents

The screenshot shows the mairies application interface. At the top, there is a navigation bar with links for Support, Docs, and Log In. Below the navigation bar, there are tabs for Tests, Runs, and Settings, with Tests being the active tab. A search bar with the placeholder "Press Ctrl + F to search..." is present. On the left, a sidebar lists "INTEGRATION TESTS" under "admin", showing three feature files: Client.feature, Color.feature, and Users.feature. On the right, there are buttons for "New Spec File" and "Run 3 integration specs".

Annexe 8 (part1)

The screenshot shows the mairies application interface. On the left, a Cypress test runner window displays a test script for "cypress/integration/admin/Users.feature". The script contains steps to click on the "Créer un utilisateur" button, fill in the "Prénom" field with "Toto", and fill in the "Adresse email" field with "toto.ch71@gmail.com". On the right, a browser window shows the "EasyAdmin" application's "Créer un utilisateur" form. The form fields include "Prénom" (Toto), "Nom" (Dupont), "Adresse email" (toto.ch71@gmail.com), "Mot de passe" (empty), "Rôle" (ROLE\_USER), and "Client" (empty). A note at the bottom of the browser window says "Cannot show Snapshot while tests are running".

Annexe 8 (part2)