



2021/2022

IRRIGATION CONNECTÉE

Rapport de projet

Rédigé par :

Sarah BARRABE

Encadré par :

Joël MAUQUIE

Thierry BOYER

Sébastien BORD





Remerciements



Avant toutes choses, je trouve approprié de présenter mes remerciements à tous ceux qui ont pu m'accompagner durant ce projet. Je remercie donc :

- M. Thomas VOGEL, directeur adjoint de l'EPL FPA de Tarn-et-Garonne dans le domaine de Capou, pour la confiance qu'il nous a apporté tout au long du projet ainsi que pour le temps qu'il a pu nous accorder afin de nous présenter au mieux ce qu'il attendait.
- M. Joël MAUQUIE, responsable projet, pour l'aide apportée lors de nos questionnements. Mais aussi pour nous avoir largement épaulé lors de l'ébauche de la conception de la base de données.
- M. Thierry BOYER, responsable projet, pour le savoir-faire apporté lors de la conception des divers diagrammes UML ou encore pour nous avoir guidé vers divers problèmes et diverses optimisations à prendre en compte dans notre développement.
- M. Sébastien BORD, responsable projet, pour nous avoir emmené sur le site de Capou où nous avons pu rencontrer notre partenaire professionnel M. VOGEL. Et aussi pour l'aide apportée afin de pouvoir présenter un site fonctionnel à M. VOGEL.
- Salim EL BAKRAOUI, Frédéric MACABIAU et enfin Mélissa SUEUR pour leur implication dans le projet. Aussi pour leur patience, leurs travaux ou encore leur capacité à travailler en équipe.
- Enfin je me permets de remercier Tristan LA PAGLIA, Jean-Philippe LOUBEJAC COMBALBERT et Lucas MAURIE pour l'entraide au sein du projet malgré un développement sur des outils différents.

Je remercie donc chaque personne qui a pu contribuer de loin comme de près à ce projet et à son bon déroulement. Enfin je remercie également ceux qui ont pu m'aider dans l'élaboration de ce rapport.

Table des matières

I. Introduction.....	4
II. Fonctionnalités de connexion.....	6
1. Formulaire de connexion.....	6
2. Oubli de mot de passe.....	9
3. Envoi de mail.....	11
4. Tests unitaires.....	13
5. Problèmes rencontrés.....	13
III. Interface d'administration.....	14
1. Mise en place d'une interface d'administration.....	14
2. Mise en place d'un formulaire d'inscription.....	16
3. Carte OpenLayers.....	19
4. Graphiques.....	20
5. Problèmes rencontrés.....	21
IV. Conclusion.....	22
V. Annexes.....	23
1. Annexe 1 – security.yaml.....	23
2. Annexe 2 – .env.....	24
3. Annexe 3 – AdminController.php.....	25
4. Annexe 4 – Méthode persistEntity() du UserCrudController.php.....	26
5. Annexe 5 – ParcelleB.json.....	27

Table des figures

« Diagramme de l'exigence 1.3 ».....	5
« Diagramme de l'exigence 1.4 ».....	6
« Diagramme de l'exigence 1.5 ».....	6
« Diagramme des cas d'utilisation de connexion et d'oubli de mot de passe ».....	7
« Interface primaire du formulaire de connexion ».....	8
« Diagramme des divers rôles ».....	8
« Implémentation des droits d'accès ».....	9
« Message d'erreur identifiants erronés ».....	10
« Interface d'oubli de mot de passe ».....	10
« Représentation schématique du système de réinitialisation de mot de passe ».....	11
« Interface de réinitialisation du mot de passe ».....	11
« Modèle du mail de réinitialisation du mot de passe ».....	12
« Configuration du MAILER_DSN ».....	13
« Création d'un e-mail en PHP ».....	13
« Récupération d'un utilisateur en fonction de l'e-mail renseigné ».....	14
« Test unitaire Admin ».....	14
« Résultat du test Admin ».....	14
« Test unitaire Technicien ».....	14
« Résultat du test Technicien ».....	14
« Diagramme des cas d'utilisation de l'administrateur ».....	15
« Page d'accueil du bundle EasyAdmin ».....	16

« Formulaire d'inscription EasyAdmin ».....	17
« Menu utilisateur ».....	18
« Modèle du mail d'inscription ».....	19
« Carte OpenLayers d'origine ».....	20
« Carte OpenLayers implémentée ».....	21
« Graphique avec Zingchart ».....	22

I. Introduction

Dans le cadre de notre deuxième année de BTS Systèmes Numériques (SN) option Informatique et Réseaux (IR), il est impératif de réaliser un projet professionnel associant les connaissances et compétences acquises durant ces deux années à l'élaboration d'un système fonctionnel suivant un cahier des charges préalablement rédigé avec le client. Ce projet professionnel aura pour finalité une soutenance présentant les diverses fonctionnalités techniques mises en place pour répondre aux besoins du projet.

Afin de mener à bien cet impératif nécessaire à l'obtention du BTS, il nous a été demandé de réaliser un système d'irrigation connecté pour le lycée agricole de Capou. Pour conduire ce projet nous avons donc été divisée en deux équipes, la première élaborera son projet sur le framework¹ PHP Symfony et la seconde sur le framework Python Django.

Pour ma part je suis concernée par la première équipe développant sur Symfony et constitué de Salim EL BAKRAOUI, Frédéric MACABIAU, Mélissa SUEUR et moi-même. Le but de ce projet est donc l'élaboration d'un site web dynamique permettant de contrôler à distance un système d'irrigation ainsi que de pouvoir consulter sur divers supports des données tel que l'humidité des parcelles de l'exploitation.

En se reportant au diagramme des exigences présenté dans le rapport commun, je peux lister de manière générale les principales fonctionnalités que j'ai implémenté. Les exigences dont j'ai la charge sont colorées en teinte brune-orangée.

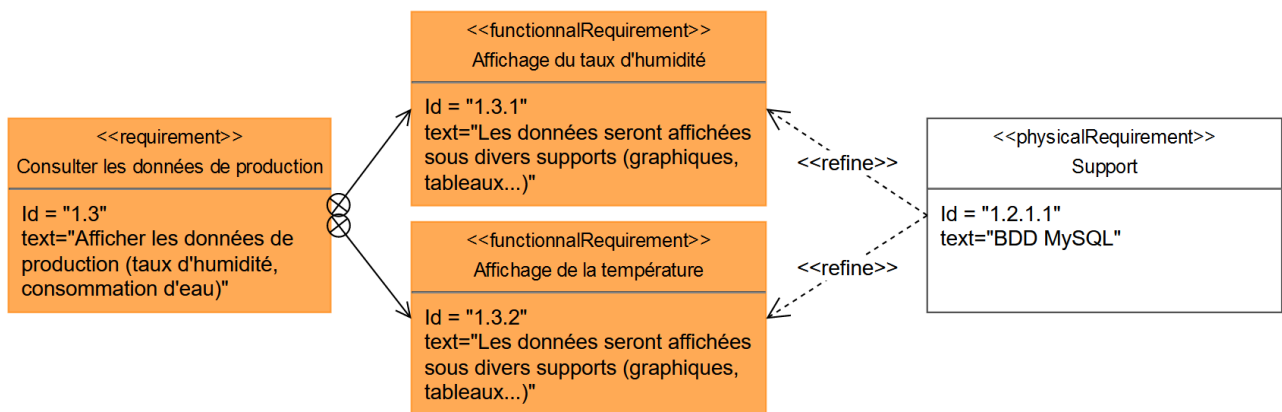
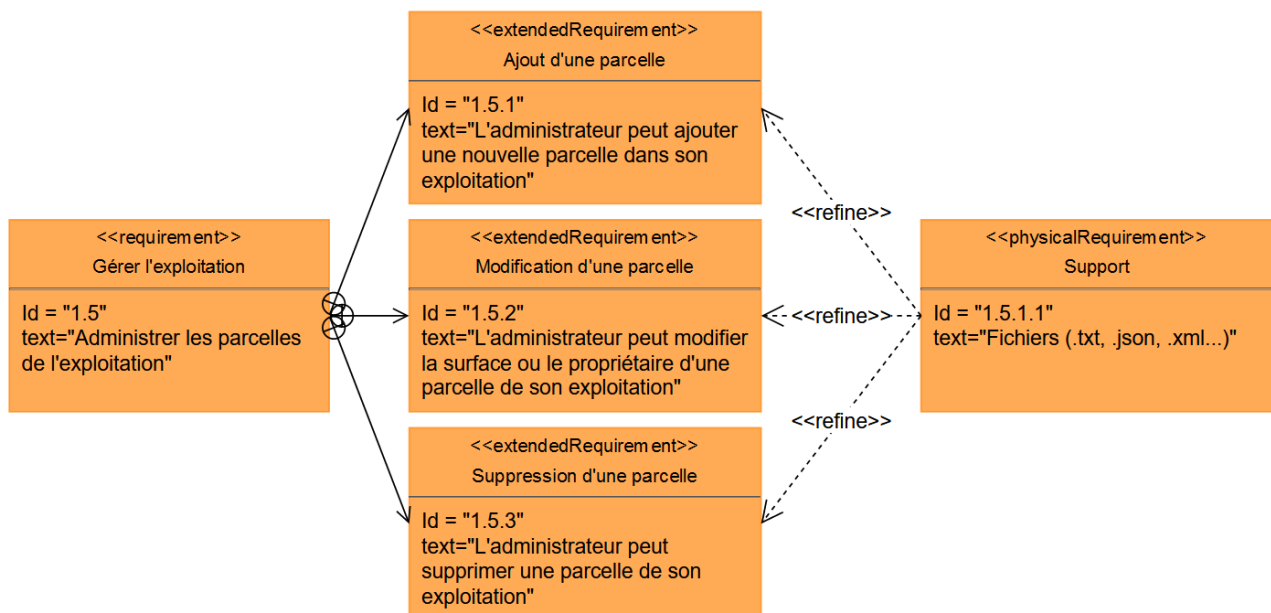
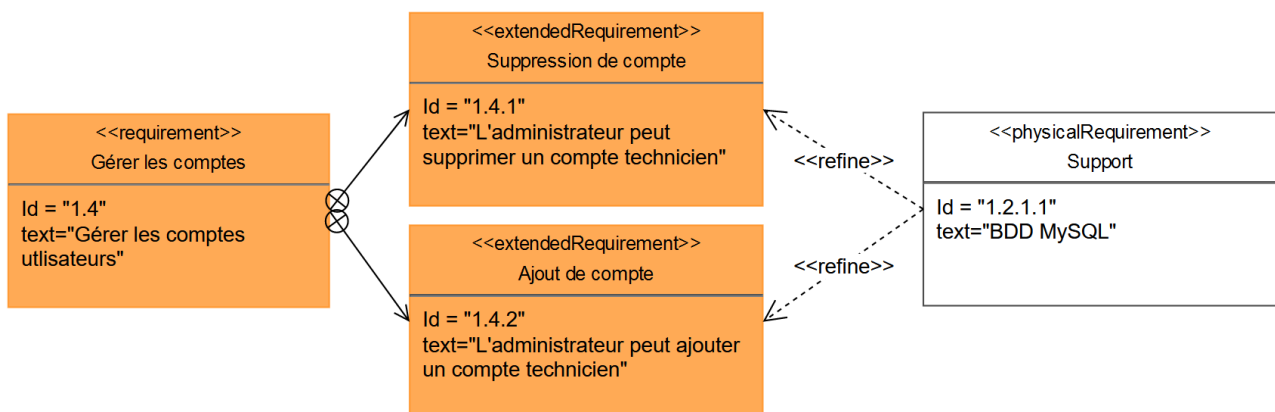


Figure 1: « Diagramme de l'exigence 1.3 »

Ci-dessus, l'une de mes exigences réalisées (et encore actuellement en optimisation de développement) en parallèle avec Frédéric, lui sur l'interface technicien et moi sur l'interface administrateur, était de pouvoir construire des graphiques à partir de relevés archivés dans la base de données. Ces graphes permettent à terme d'afficher l'humidité de telle ou telle parcelle de l'exploitation ainsi que la température.

¹ Framework : ensemble d'outils et de composants logiciels permettant de créer l'architecture d'un site

Ensuite deux principales exigences viennent s'ajouter à celle présentée précédemment qui sont d'après le diagramme d'exigences le service de gestion des comptes utilisateurs ainsi que l'administration des parcelles de l'exploitation.



À présent commençons la description des diverses tâches que j'ai pu réaliser.

II. Fonctionnalités de connexion

Afin de permettre aux utilisateurs de l'application de pouvoir s'identifier, j'ai dû mettre en place des formulaires, l'un permettant l'inscription et le second la connexion. On peut représenter de manière intuitive ces deux fonctionnalités avec le diagramme suivant.

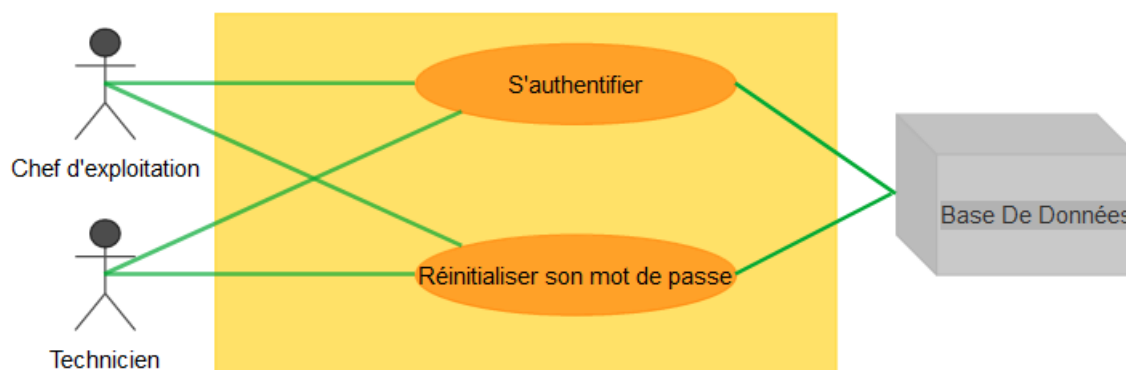


Figure 4: « Diagramme des cas d'utilisation de connexion et d'oubli de mot de passe »

1. Formulaire de connexion

Quels sont les bundles² permettant de construire une interface de connexion ?

Tout d'abord pour mettre en place une interface de connexion sur Symfony, il existe deux principaux bundles qui sont le FOSUserBundle (Friends Of Symfony User Bundle) et le MakerBundle. J'ai donc étudié les différentes fonctionnalités qu'ils proposent afin de choisir celui qui correspondrait le mieux à notre application.

Bundles \ Fonctions	Entité User	Personnalisation	Connexion	Oubli de mot de passe
FOSUser	Intégrée	Fastidieuse	Via nom d'utilisateur	Oui
Maker	Intégrée	Simple	Via e-mail/nom d'utilisateur	Non

Après étude de ces deux bundles, je me suis tournée vers le bundle Maker car il me permet en premier temps d'éviter de trop lourdes modifications dans le code et en second temps de pouvoir proposer aux utilisateurs une connexion via leur adresse mail sans nécessité de nom d'utilisateur. L'unique défaut du bundle Maker que j'ai pu constater est l'absence d'une

² Bundle : module contenant une multitude d'éléments permettant de façonner votre application web

fonctionnalité permettant de gérer l'oubli de mot de passe. J'ai donc réglé cet inconvénient avec l'implémentation d'un second bundle.

Comment mettre en place une interface de connexion ?

Afin de réaliser l'interface de connexion à l'aide du MakerBundle, la première étape fût de créer la table User (Utilisateur) dans la base de données à l'aide de la fonctionnalité proposée par le bundle. Cette implémentation a été réalisée par Mélissa, camarade en charge de la base de données, sur ma demande. Ensuite s'imbrique la création du formulaire avec les personnalisations qui s'y imposent. Le formulaire standard à sa création est le suivant :

Please sign in

Email Password

Figure 5: « Interface primaire du formulaire de connexion »

À terme, la partie design de l'interface a été implémentée de manière à suivre la charte graphique du site web par Mélissa.

Quels sont les différents rôles ?

Il existe plusieurs rôles permettant de représenter tous les utilisateurs du site web. En effet un simple visiteur sur le site n'aura pas les mêmes droits qu'un technicien ayant accès à l'interface d'irrigation connectée ou même qu'un administrateur qui possèdent tous les droits.

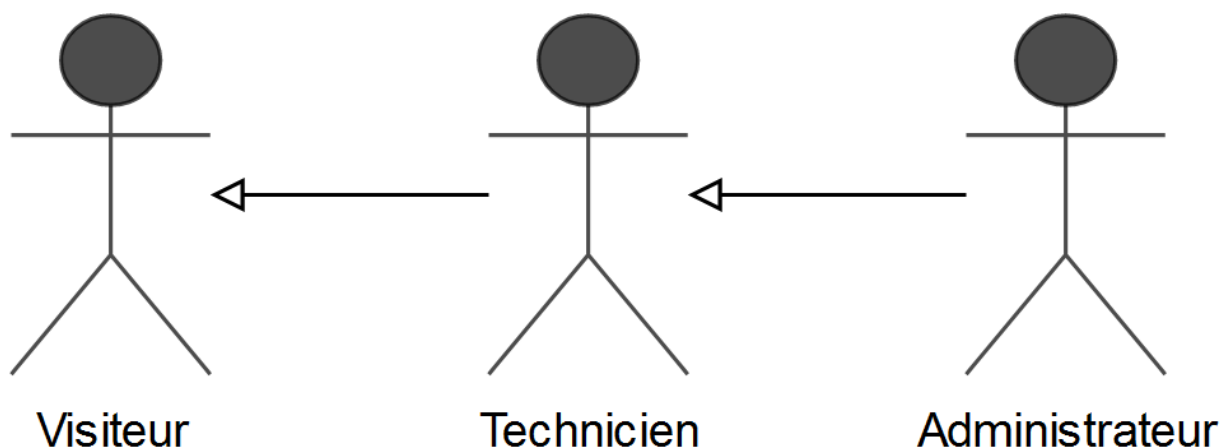


Figure 6: « Diagramme des divers rôles »

Ci-dessous, j'ai récapitulé au sein d'un tableau les diverses fonctionnalités et caractéristiques que doit remplir chaque rôle.

	Visiteur ←	Technicien ←	Administrateur
Personne en charge	Tout individu	Exécutant(s) en charge de l'exploitation	Chef d'exploitation
Fonctionnalités	<ul style="list-style-type: none"> • S'authentifier • Réinitialiser son mot de passe • Consulter la page d'accueil 	<ul style="list-style-type: none"> • Consulter les données de production de ses parcelles • Contrôler l'arrosage de ses parcelles • Télécharger un historique de relevés 	<ul style="list-style-type: none"> • Gérer les comptes utilisateurs • Gérer l'exploitation • Contrôler l'arrosage de l'exploitation

En résumé, il existe entre les trois rôles une relation d'héritage ainsi le rôle administrateur hérite du rôle de technicien qui lui-même hérite du rôle visiteur correspondant au rôle par défaut. Par extension, le rôle administrateur possède plus de fonctionnalités que le technicien ou que le visiteur. La différence majeure entre le technicien et l'administrateur est la consultation des données, en effet le technicien ne peut voir que les données de ces parcelles alors que l'administrateur peut visualiser les données de toutes les parcelles.

Comment définir les rôles au sein du projet ?

Afin d'assurer la sécurité des pages en limitant leurs accès aux utilisateurs concernés, il m'a fallu définir différents droits dans le fichier `security.yaml` (cf. [Annexe 1 – security.yaml](#)). Ce fichier utilise deux paramètres afin de configurer les éléments nécessaires aux divers services permettant une authentification simple et sécurisée. D'abord le `firewall` permettant de gérer l'authentification puis l'`access_control` qui gère les autorisations liées aux rôles. Ci-dessous se trouve donc le code que j'ai pu rajouter afin de mettre en place ce système de hiérarchie des rôles.

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

  role_hierarchy:
    ROLE_TECHNICIAN: ROLE_USER
    ROLE_ADMIN: ROLE_TECHNICIAN
```

Figure 7: « Implémentation des droits d'accès »

En premier lieu, ici, l'`access_control` définit que deux routes sont protégées par des rôles par exemple si un utilisateur souhaite accéder à une page du site comportant le chemin « `/admin` » il devra forcément avoir le rôle administrateur, si ce n'est pas le cas l'utilisateur sera redirigé vers une page d'erreur. De même pour la route « `/technician` ». Ensuite le `role_hierarchy` définit l'héritage entre les rôles, ici l'héritage que j'ai implémenté est identique au schéma précédent on a donc le technicien (`ROLE_TECHNICIAN`) qui hérite du rôle visiteur (ici `ROLE_USER`) et l'administrateur (`ROLE_ADMIN`) qui hérite du rôle technicien.

Quelles sont les différentes fonctionnalités du système de connexion ?

Pour pouvoir se connecter, l'utilisateur doit renseigner son adresse e-mail et son mot de passe puis cliquer sur le bouton « se connecter ». L'utilisateur sera alors redirigé sur l'interface qui lui est attribuée. Associé au formulaire, un système d'envoi de message d'erreur lorsque les informations de connexion sont erronées a également été implémenté.

Vos identifiants de connexion sont incorrects !

Figure 8: « Message d'erreur identifiants erronés »

2. Oubli de mot de passe

Que faire pour l'oubli de mot de passe ?

Afin de pallier l'absence de la fonctionnalité d'oubli de mot de passe, j'ai donc pu trouver le bundle ResetPassword permettant d'implémenter celle-ci. À l'aide d'une simple commande, le strict nécessaire est créé comme par exemple une nouvelle table dans la base de données pour enregistrer les requêtes d'oubli de mot de passe nommée reset_password_request et ainsi garder en mémoire la disponibilité du lien de réinitialisation.

Une intervention en CSS ?

L'interface permettant de réinitialiser son mot de passe est parfaitement fonctionnelle dû à sa simplicité de mise en œuvre. J'ai tout de même pu intervenir sur ce système afin de modifier le style CSS de la page pour que celui-ci respecte la charte graphique à l'image du formulaire de connexion réalisé par Mélissa.

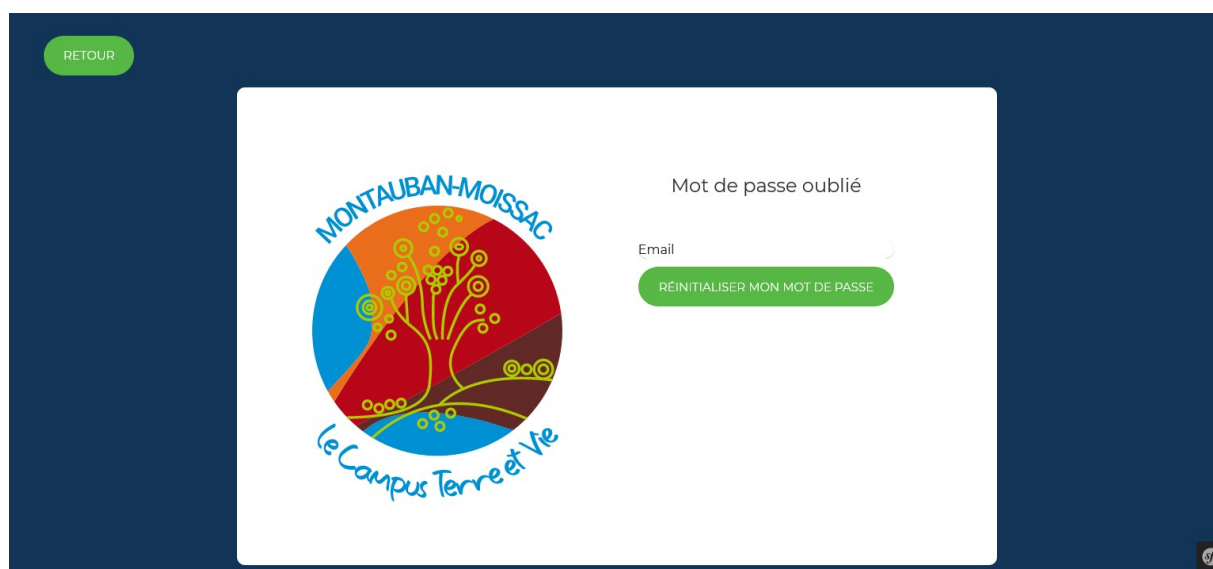


Figure 9: « Interface d'oubli de mot de passe »

Afin de m'accorder aux autres travaux, j'ai, pour réaliser le style de cette page, repris le template utilisé pour le formulaire de connexion pour que ces deux espaces suivent la même charte. Une fois repris, je l'ai simplement adapté à mon formulaire.

Comment fonctionne l'oubli de mot de passe ?

Le système permettant de réinitialiser son mot de passe peut être représenté par le schéma suivant :

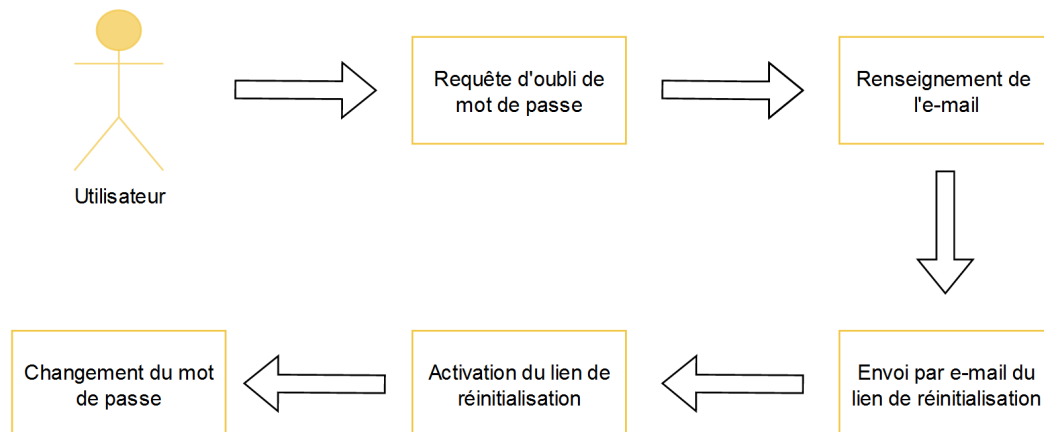


Figure 10: « Représentation schématique du système de réinitialisation de mot de passe »

En définitif pour pouvoir réinitialiser son mot de passe, l'utilisateur doit donc cliquer sur le lien « Mot de passe oublié ? » présent sur le formulaire de connexion puis il doit renseigner son adresse mail dans le champ adéquat du formulaire d'oubli de mot de passe. Une fois cela fait, l'utilisateur sera redirigé vers une page d'information précisant qu'un mail lui a été envoyé. À l'ouverture du mail, une indication menant à cliquer sur un lien renvoi l'utilisateur vers le formulaire du site lui permettant de changer son mot de passe.



Figure 11: « Interface de réinitialisation du mot de passe »

3. Envoi de mail

Pourquoi cette fonctionnalité ?

Le service de réinitialisation de mot de passe est associé au formulaire de connexion et l'envoi de mail est une fonctionnalité indissociable de ce service. En effet comme expliqué précédemment à chaque requête d'oubli de mot de passe, un mail (cf. [Figure 12](#)) est envoyé à l'utilisateur qui en fait la demande afin de pouvoir lui envoyer un lien de réinitialisation.



Figure 12: « Modèle du mail de réinitialisation du mot de passe »

Comment intégrer le service d'envoi de mail ?

Avec Symfony l'envoi de mail est d'ores et déjà intégré, il suffit simplement de l'activer et de créer les fonctions nécessitant l'envoi d'un mail. Il existe deux librairies permettant de réaliser cela, la librairie SwiftMailer et la librairie Mailer. Dans le tableau de comparaison ci-dessous, les deux librairies sont confrontées sur trois catégories que j'ai jugé importantes pour déterminer laquelle est la plus adaptée.

Fonctions Librairies	Poids Message	Personnalisation	Création d'un e-mail
Mailer	2ko	Ensemble du mail	Code moderne et simplifié
SwiftMailer	16ko	Corps du mail	Code plus ancien => moins simplifié

Comme on peut le constater sur le tableau de confrontation, j'ai donc choisi d'utiliser la librairie Mailer associée au composant Mime afin de pouvoir créer des mails de type TemplatedEmail(). Ce type me permet d'associer un fichier .html.twig au TemplatedEmail et d'ainsi donner une structure à celui-ci.

Pour l'intégration du module Mailer il suffit, en ajout de la commande d'installation, de préciser dans le fichier de configuration du projet nommé .env (cf. [Annexe 2 – .env](#)) la valeur de la variable

MAILER_DSN. Cette variable correspond au serveur SMTP en charge de transporter les e-mails d'un bout à un autre. Dans notre cas je vais utiliser une autre fonctionnalité de Symfony me permettant d'envoyer mes e-mails via un fournisseur tiers. Pour l'intégration du module Mime, seul l'entrée de la commande d'installation est requise.

Le fournisseur tiers que j'ai choisi est Gmail, je l'ai donc installé par le biais d'une commande en plus des composants Mailer & Mime puis j'ai modifié la variable MAILER_DSN. À l'origine, c'est-à-dire sans fournisseur tiers, cette variable aurait été de la forme :

```
MAILER_DSN=smtp://user:password@smtp.example.com:port
```

Maintenant avec l'ajout du service Gmail la variable doit s'adapter à celui-ci et être mise sous la forme :

```
MAILER_DSN=gmail://user:password@default
```

En conclusion sur l'intégration du service d'envoi de mail, il m'a donc fallu installer les composants Mailer & Mime & Google-mailer. Puis j'ai dû définir la variable permettant d'effectuer le transport de mes e-mails. Par exemple, ci-dessous AddrTest666@gmail.com correspond au paramètre user donc à l'adresse mail d'envoi et Q5qcd9yNbuE7D3m correspond au mot de passe d'accès de l'expéditeur.

```
###> symfony/mailer ###  
MAILER_DSN=gmail://AddrTest666@gmail.com:Q5qcd9yNbuE7D3m@default  
###< symfony/mailer ###
```

Figure 13: « Configuration du MAILER_DSN »

Comment créer un mail ?

À présent que le service de mail est mis en place, on peut créer un e-mail afin de pouvoir l'envoyer à l'utilisateur ciblé. Pour cela j'ai pu créer une instance de la classe TemplatedEmail elle-même héritée de la classe Email du composant Mime. Ci-dessous un exemple de création d'e-mail :

```
$email = (new TemplatedEmail())  
->from(new Address('AddrTest666@gmail.com', 'no-reply-reset-password'))  
->to($user->getEmail())  
->subject('Oubli de mot de passe')  
->htmlTemplate('reset_password/email.html.twig')  
->context([  
    'url'=> $url  
]);  
  
return $email;
```

Figure 14: « Création d'un e-mail en PHP »

On commence par instancier un objet TemplatedEmail puis on y implémente toutes les fonctionnalités souhaitées et nécessaires.

D'abord il faut ajouter une adresse d'expéditeur ici l'adresse est AddrTest666@gmail.com et son nom sera affiché comme étant no-reply-reset-password.

Ensuite on ajoute l'adresse du destinataire, donc en ce qui concerne l'interface d'oubli de mot de passe, on récupère l'e-mail renseignée dans le champ puis avec la base de données on récupère

l'utilisateur correspondant à cette adresse mail (car en effet l'adresse mail est unique ainsi il est impossible d'avoir deux fois la même adresse mail d'enregistrée) afin d'éviter d'avoir à enregistrer l'adresse mail alors que celle-ci est déjà sauvegardée dans la base de données.

```
$user = $this->entityManager->getRepository(User::class)->findOneBy([
    'email' => $emailFormData,
]);
```

Figure 15: « Récupération d'un utilisateur en fonction de l'e-mail renseigné »

Puis vient le « subject » qui correspond à l'objet du mail ici « Oublie de mot de passe ». Et enfin viennent l'« htmlTemplate », permettant d'attribuer un fichier twig à l'e-mail afin que celui-ci respecte une organisation d'écriture, et la fonctionnalité « context » permettant de renseigner des variables aux templates twig.

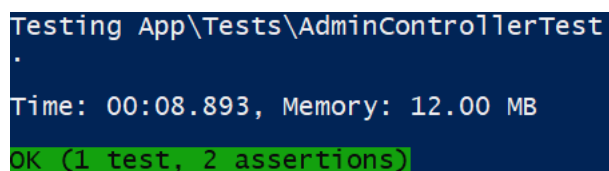
4. Tests unitaires

Les deux tests unitaires ci-dessous exposent le bon fonctionnement de la redirection des utilisateurs souhaitant accéder à des ressources protégées vers la page login si ceux-ci ne sont pas connectés.

```
public function testDisplayLoginForm(): void
{
    $client = static::createClient();
    $client->request('GET', '/admin');

    $this->assertResponseRedirects('/login');
}
```

Figure 16: « Test unitaire Admin »

A screenshot of a terminal window showing the results of a PHPUnit test. The text is as follows: "Testing App\Tests\AdminControllerTest", "Time: 00:08.893, Memory: 12.00 MB", and "OK (1 test, 2 assertions)". The last line is highlighted in green.

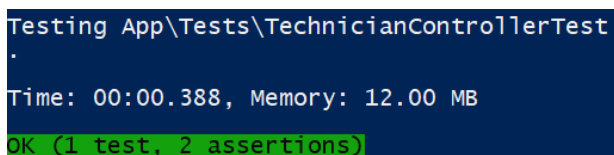
```
Testing App\Tests\AdminControllerTest
Time: 00:08.893, Memory: 12.00 MB
OK (1 test, 2 assertions)
```

Figure 17: « Résultat du test Admin »

```
public function testDisplayLoginForm(): void
{
    $client = static::createClient();
    $client->request('GET', '/technician');

    $this->assertResponseRedirects('/login');
}
```

Figure 18: « Test unitaire Technicien »

A screenshot of a terminal window showing the results of a PHPUnit test. The text is as follows: "Testing App\Tests\TechnicianControllerTest", "Time: 00:00.388, Memory: 12.00 MB", and "OK (1 test, 2 assertions)". The last line is highlighted in green.

```
Testing App\Tests\TechnicianControllerTest
Time: 00:00.388, Memory: 12.00 MB
OK (1 test, 2 assertions)
```

Figure 19: « Résultat du test Technicien »

5. Problèmes rencontrés

Au cours du développement du formulaire de connexion j'ai dû faire face à certains problèmes comme par exemple la mise en place du mail envoyé avec le lien de réinitialisation du mot de passe. En effet je n'arrivais pas à connecter le fichier twig à mon mail afin que celui-ci respecte l'aspect choisi. Finalement à force de recherche et de test, j'ai trouvé une énième manière d'appeler le fichier twig et de l'associer à mon mail.

III. Interface d'administration

L'interface d'administration est le service personnel du chef d'exploitation soit de l'administrateur. Cette interface lui permettra à terme de visualiser toutes les données relevées des parcelles de l'ensemble de son exploitation ainsi que de gérer tous ce qui concerne son exploitation. C'est par exemple l'administrateur qui a la charge d'inscrire des utilisateurs ou de supprimer des comptes, mais aussi d'ajouter, de modifier ou même de supprimer une parcelle...etc

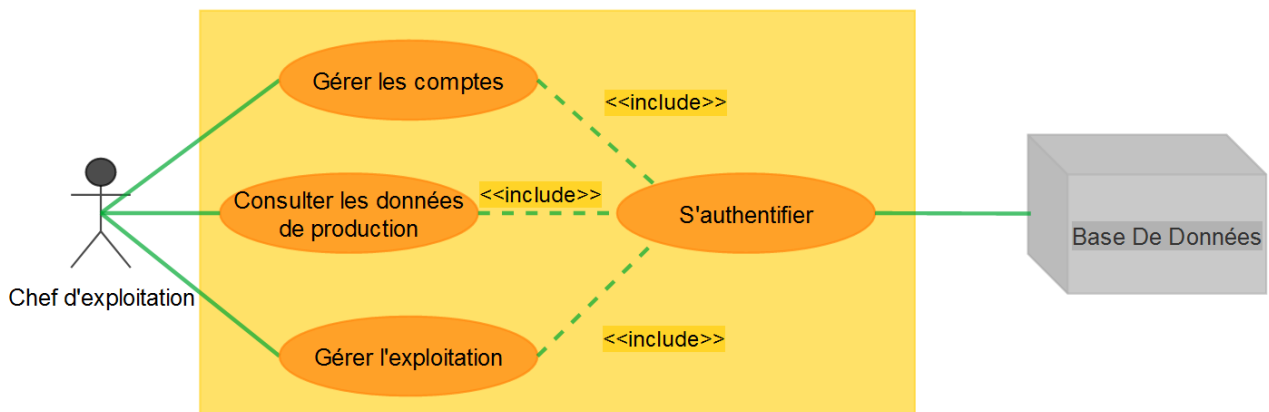


Figure 20: « Diagramme des cas d'utilisation de l'administrateur »

1. Mise en place d'une interface d'administration

Quel bundle choisir afin de construire une interface d'administration ?

Tout d'abord pour mettre en place une interface d'administration sur Symfony, il existe deux principaux bundles, d'abord SonataAdmin puis EasyAdmin.

Bundles \ Fonctions	Prise en main	Utilisation	Charge du code
SonataAdmin	Contraignante	Back-office complexe	Dense
EasyAdmin	Aisée	Fonctionnalités spécifiques / interface rapide	Légère

Lorsque l'on m'a attribué le projet, j'ai, au vu des différentes fonctionnalités que notre site doit proposer, pu constater que celui-ci ne demandait pas d'implémentations complexes liées à l'interface d'administration en elle-même. C'est pour cela que je me suis portée sur le bundle EasyAdmin car il propose des fonctionnalités spécifiques tout en étant simple à mettre en place, j'ai donc trouvé que cela était un bon compromis afin d'éviter d'avoir une trop lourde charge de code pour des fonctionnalités complexes dont nous n'aurions sûrement pas besoin.

Comment mettre en place une interface d'administration ?

Le bundle EasyAdmin est relativement simple à implémenter car l'entrée d'une simple commande suffit à l'installer au sein du projet. Ensuite j'ai créé ce que l'on appelle un Dashboard (cf. [Annexe 3 – AdminController.php](#)) qui permet de visualiser cette interface d'administration. Une fois créé on tombe alors sur la page d'accueil du bundle qu'il m'a fallu modifier afin de l'approprier à mon interface web.

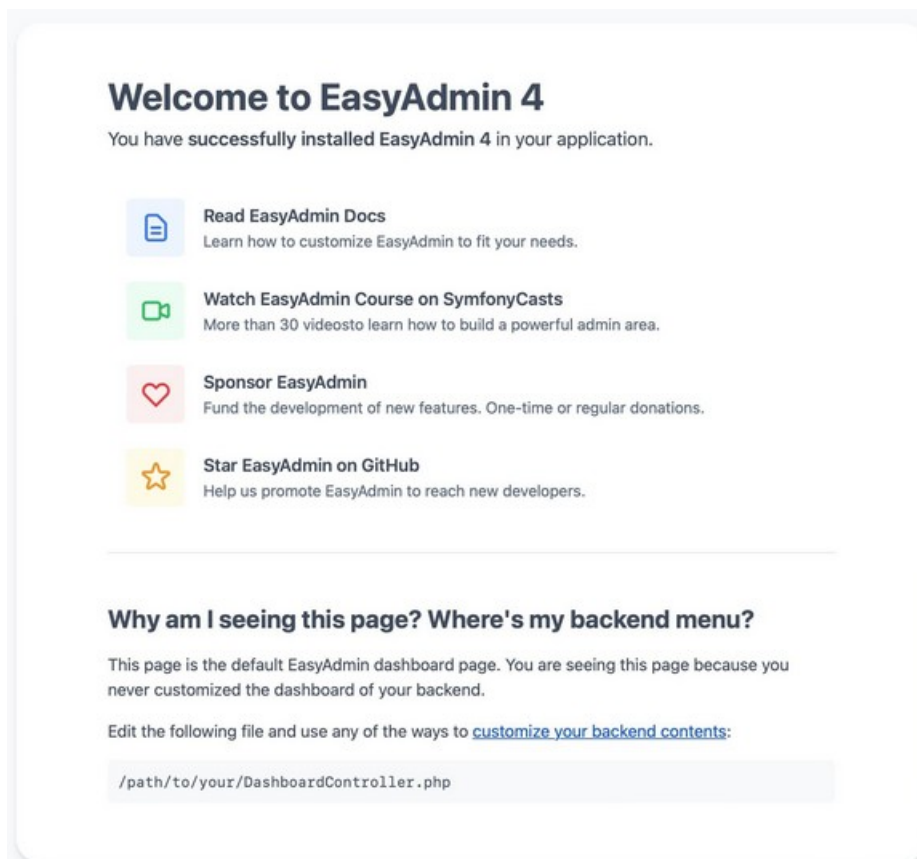


Figure 21: « Page d'accueil du bundle EasyAdmin »

Pour modifier cette page j'ai créé des fichiers nommés Crud qui me permettent de créer des pages pouvant gérer les différentes entités que j'aurais sélectionné comme devant faire partie de l'interface d'administration. Par exemple l'administrateur peut gérer les comptes il a donc accès aux utilisateur, j'ai donc créé un fichier Crud permettant de gérer l'entité User. En revanche la table `reset_password_request` évoquée dans la partie [II.2.Oubli de mot de passe](#) ne doit pas apparaître dans l'interface administrateur car il n'y aurait aucun intérêt à manipuler les requêtes d'oubli de mot de passe de ses utilisateurs qui sont automatiquement gérées par le système de l'interface web.

2. Mise en place d'un formulaire d'inscription

L'interface d'inscription est un service à part entière de l'interface d'administration puisqu'en effet seul l'administrateur peut inscrire des utilisateurs. Il aurait été inapproprié de laisser la main à n'importe quel visiteur du site de pouvoir s'inscrire. Ainsi nous avons défini que seul l'administrateur posséderait cette fonctionnalité d'où la présence du cas d'utilisation « Gérer les comptes ».

Afin de réaliser l'interface d'inscription, j'ai donc mis en place un Crud lié à l'entité User qui va donc me permettre d'ajouter, de modifier ou encore de supprimer des utilisateurs. Pour réaliser le formulaire permettant l'inscription, j'ai utilisé des fonctionnalités d'EasyAdmin couplées à mes besoins.

Ajouter un utilisateur

Create and add another

Create

Prénom*

Nom*

Adresse e-mail*

ex: jean-dupont@gmail.com

Téléphone*

Adresse

ex: 8 rue Laplace

Ville

Code postal

Fonction(s)*

☒ Technician

☐ Administrator

Couleur*



Figure 22: « Formulaire d'inscription EasyAdmin »

Par exemple l'implémentation du champ Adresse e-mail est réalisé par le code :

```
EmailField::new('email', 'Adresse e-mail')->setHelp('ex: jean-dupont@gmail.com')->setRequired(1)
```

EmailField::new('email', 'Adresse e-mail') correspond à la création d'un nouvel objet de type e-mail dont le nom de l'encadré est affiché comme étant Adresse e-mail et dont la sauvegarde s'effectue en base de données dans le champ email de la table User. La précision du type EmailField permet de protéger les inscriptions contre un éventuel oubli de l'arobase de la part de l'administrateur. En effet, en cas d'oubli le champ s'affiche alors en rouge et l'inscription de l'utilisateur est impossible tant que celui-ci n'a pas été modifié.

La fonctionnalité setHelp() permet d'afficher à proximité du champ une information complémentaire fournissant des indications sur le champ lui-même ou sur le manière dont le remplir.

Enfin setRequired() est une fonctionnalité attendant un booléen en paramètre afin de définir si oui ou non le champ doit être rempli avant de pouvoir valider l'inscription. Si oui alors on passe 1 ou true en paramètre pour que l'inscription soit validée uniquement si le champ a été rempli et à l'inverse on passe 0 ou false pour indiquer que ce champ n'est pas obligatoire à l'inscription d'un utilisateur.

Comment le fichier Crud est-il lié au Dashboard de l'interface d'administration ?

Pour afficher le formulaire d'inscription comme sur la [Figure 22](#), j'ai donc dû associer le fichier Crud au fichier Dashboard évoqué précédemment.

```
yield MenuItem::section('Utilisateurs');
yield MenuItem::subMenu('Actions', 'fas fa-list')->setSubItems([
    MenuItem::linkToCrud('Ajouter utilisateur', 'fas fa-plus', User::class)
        ->setAction(Crud::PAGE_NEW),
    MenuItem::linkToCrud('Liste utilisateurs', 'fas fa-eye', User::class)
]);
```

Le lien est effectué avec le linkToCrud ayant en paramètre la classe de l'entité concernée par le Crud souhaité, j'ai également passé une icône bootstrap ainsi qu'une chaîne de caractère servant de label au lien cliquable. Le rendu de cette implémentation

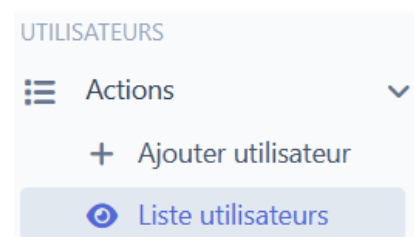


Figure 23: « Menu utilisateur »

Pour la création des autres Cruds utiles tel que celui associé à l'entité des parcelles, l'implémentation suit la même trame que précédemment.

Comment le mot de passe est-il attribué à l'utilisateur lors de l'inscription par l'administrateur ?

Pour qu'un utilisateur puisse se connecter il lui faut son adresse mail ainsi que son mot de passe. Cependant lors de l'inscription d'un utilisateur l'administrateur ne doit pas connaître le mot de passe d'accès du nouvel inscrit. C'est pourquoi j'ai caché dans le formulaire d'inscription le champ du mot de passe pour qu'il ne puisse pas le remplir. Afin de créer ce mot de passe j'ai mis en place, lors de l'envoi dans la base de données de l'utilisateur créé réalisé par la méthode `persistEntity()` (cf. [Annexe 4 – Méthode `persistEntity\(\)` du `UserCrudController.php`](#)), un système de génération de mot de passe aléatoire. J'ai utilisé la librairie Faker de php afin de générer ce mot de passe. J'ai choisi un nombre minimal et maximal de 8 caractères afin d'assurer une protection certaine à l'accès de son espace personnel. Ce mot de passe est par cohérence crypté dans la base de données. Afin de générer ce mot de passe aléatoire, Faker me propose ces deux lignes de commandes :

```
$faker = Factory::create('fr_FR');  
$faker->password(8,8)
```

La première ligne me permet de créer et d'initialiser un générateur faker. Le paramètre passé correspond à la langue choisi en cas de demande d'adresse, de prénom...etc. En effet le générateur s'adapte en fonction de la langue choisie, pour le mot de passe aléatoire cette langue n'influe pas puisque les caractères sont indépendants les uns des autres.

Enfin la seconde ligne crée un mot de passe aléatoire de 8 caractères.

Afin de transmettre ce mot de passe à l'utilisateur inscrit, j'ai, de la même manière que l'envoi du mail de réinitialisation du mot de passe (cf. [II.3.Envoi de mail](#)), créé un mail.



Figure 24: « Modèle du mail d'inscription »

Le mot de passe n'étant pas sauvegardé dans la base de données, j'ai dû créer une variable permettant de stocker le mot de passe non crypté. Cette variable n'est alors jamais découverte si ce n'est pour le mail ci-dessus ou l'utilisateur a besoin de connaître son mot de passe.

3. Carte OpenLayers

Sur l'interface d'administration, j'ai implémenté une carte web afin de pouvoir visualiser les parcelles de l'exploitation ainsi que les divers piquets présents.

Quelle librairie choisir pour l'implémentation d'une cartographie web ?

Afin de mettre en place une cartographie web, il existe de nombreuses librairies. Ci-dessous j'ai choisi de comparer Google Maps Platform à OpenLayers à Leaflet.

Fonctions Librairies	Prix	Licence	Poids de la librairie	Documentation
Google Maps Platform	Payant	Propriétaire	?	?
OpenLayers	Gratuit	Open Source	1Mo	Complète
Leaflet	Gratuit	Open Source	123Ko	Moins complète

Pour réaliser notre cartographie web, j'ai donc choisi d'utiliser la librairie JavaScript OpenLayers car, malgré un poids conséquent pour la librairie, je serais certaine de pouvoir réaliser toutes les fonctionnalités dont j'aurais besoin au contraire de Leaflet ou je ne peux en être certaine. De plus je connaissais déjà OpenLayers donc je pouvais m'assurer de la présence de fonctionnalités nécessaires à notre projet.

Quelle fonctionnalité implémentée avec OpenLayers ?

À la création de mon interface d'administration, j'ai implémenté une simple carte sur la page d'accueil de l'espace administrateur. Cette carte est celle proposée par OpenLayers pour démarrer.

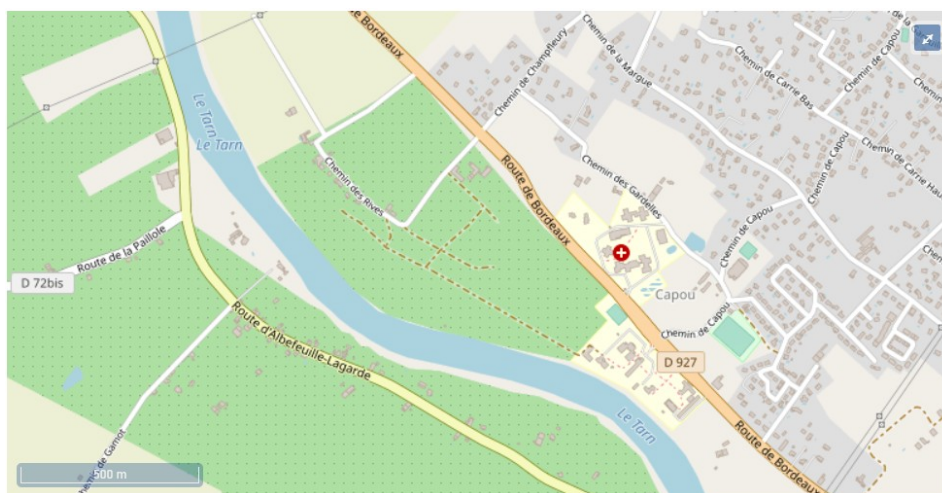


Figure 25: « Carte OpenLayers d'origine »

Puis Mélissa a pris la suite de l'implémentation de la carte afin dans un premier temps de donner une vue satellite à celle-ci et dans un second temps d'y ajouter les divers points factices des piquets. Une fois cela fait je suis repassée sur la carte dans le but de délimiter les parcelles de l'exploitation. À l'heure actuelle l'aspect de la carte est le suivant :



Figure 26: « Carte OpenLayers implémentée »

On peut voir sur la carte la délimitation de deux parcelles, la n°141 et n°131 (ces deux nombres correspondent à leur identifiant au sein de la base de données). Afin de pouvoir délimiter les parcelles, j'ai récupéré de manière rudimentaire (sur Google Maps) des coordonnées approximatives permettant cela. Une fois récupérées, j'ai rentré ces coordonnées dans un fichier un Json (cf. Annexe 5 – ParcelleB.json).

Au sein de la table Plot (Parcelle) de la base de données, il existe un champ nommé filepath permettant d'associer une parcelle à un fichier. Ainsi chaque parcelle peut posséder son fichier délimitant ses coordonnées. Pour les délimiter sur la carte, je récupère donc pour chaque parcelle le chemin du fichier qui lui est associé puis en traitant les données JSON, je trace sur la carte un polygone liant chacune des coordonnées enregistrées dans le document.

L'administrateur pourra changer les coordonnées des parcelles à sa guise, ainsi, afin de mettre à jour la carte, j'ai utilisé des requêtes Ajax liées à JQuery. Ajax est une fonctionnalité JavaScript permettant de rafraîchir des éléments d'une page web sans nécessité de rafraîchir toute la page. Associée à Ajax, la bibliothèque JavaScript JQuery permet de simplifier l'écriture des requêtes Ajax parfois complexes.

4. Graphiques

Encore sur l'interface d'administration, des graphiques sont présents afin de pouvoir visualiser les dernières données relevées par les piquets sur chaque parcelle. Les graphiques

affichent bien les derniers relevés factices des piquets, cependant ils n'affichent pas encore les relevés en fonction des parcelles.

J'ai choisi deux librairies différentes proposant des graphiques différents, la première est chart.js et la seconde est zingchart. Ci-dessous un exemple de graphique avec zingchart :

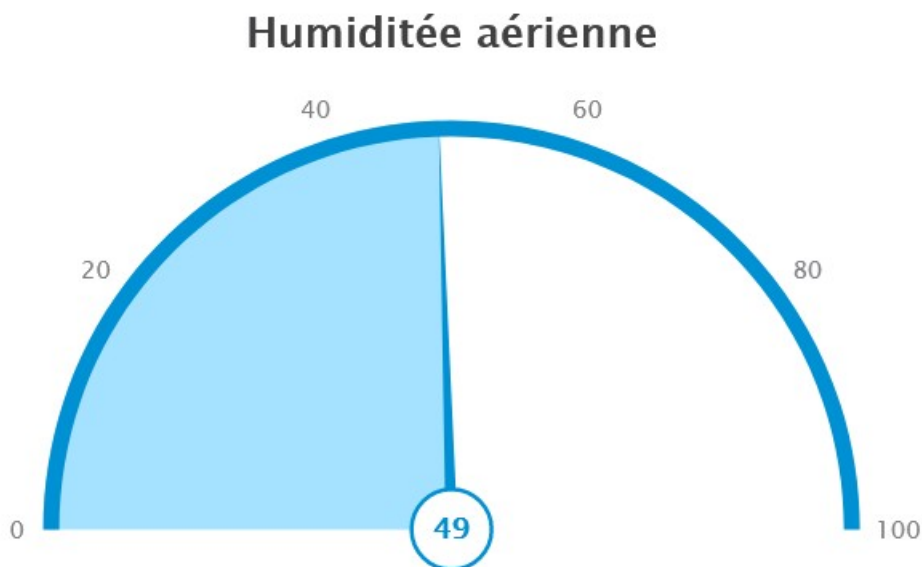


Figure 27: « Graphique avec Zingchart »

5. Problèmes rencontrés

Au cours du développement de l'interface d'administration, j'ai rencontré plusieurs problèmes notamment sur la partie permettant de délimiter les parcelles. Il m'a été compliqué de mettre en place ce système mais à force de persévérance, le travail a payé et la délimitation fonctionne parfaitement.



IV. Conclusion



En résumé de ce rapport, j'ai donc pu mettre en place divers services nécessaires au bon fonctionnement de notre site tel que l'implémentation d'un espace administrateur ou encore le développement de fonctionnalités utiles sur la cartographie web OpenLayers.

L'ensemble du développement du projet s'est déroulé au sein d'une équipe agréable et impliquée. Cela m'a donc permis de découvrir et d'approfondir des concepts au sein d'un environnement favorable à cet apprentissage.

J'ai trouvé et trouve encore vraiment passionnant le développement web avec tous les outils qu'il me permet de prendre en main et toutes les fonctionnalités que cela implique de mettre en place.

En ce qui concerne les problèmes, j'en ai naturellement rencontré un nombre important mais cela m'a permis d'apprendre d'eux notamment en trouvant comment les résoudre et en sachant remettre en question mon code.

Pour la poursuite du projet durant le temps qu'il nous reste, il faudra pour ma part que je puisse identifier qu'un piquet appartient à telle ou telle parcelle afin de pouvoir afficher les bons relevés de chaque parcelle. En général, il faut également que l'on termine le lien entre le matériel et le logiciel. Car ce lien est actuellement existant et réalisé par Mélissa et Salim cependant malgré la bonne réception des données, nous n'arrivons pas encore à les afficher sur notre site afin de pouvoir les envoyer dans la base de données.

1. Annexe 1 – security.yaml

```

1 security:
2   enable_authenticator_manager: true
3   # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
4   password_hashers:
5     Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
6     App\Entity\User:
7       algorithm: auto
8
9   # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
10  providers:
11    # used to reload user from session & other features (e.g. switch_user)
12    app_user_provider:
13      entity:
14        class: App\Entity\User
15        property: email
16  firewalls:
17    dev:
18      pattern: ^/(_(profiler|wdt)|css|images|js)/
19      security: false
20    main:
21      lazy: true
22      provider: app_user_provider
23      custom_authenticator: App\Security\AppAuthenticator
24      logout:
25        path: app_logout
26        # where to redirect after logout
27        target: app_home
28
29      # activate different ways to authenticate
30      # https://symfony.com/doc/current/security.html#the-firewall
31
32      # https://symfony.com/doc/current/security/impersonating_user.html
33      # switch_user: true
34
35      # Easy way to control access for large sections of your site
36      # Note: Only the *first* access control that matches will be used
37      access_control:
38        - { path: ^/admin, roles: ROLE_ADMIN }
39        - { path: ^/technician, roles: ROLE_TECHNICIAN }
40
41      role_hierarchy:
42        ROLE_TECHNICIAN: ROLE_USER
43        ROLE_ADMIN: ROLE_TECHNICIAN
44
45  when@test:
46    security:
47      password_hashers:
48        # By default, password hashers are resource intensive and take time. This is
49        # important to generate secure password hashes. In tests however, secure hashes
50        # are not important, waste resources and increase test times. The following
51        # reduces the work factor to the lowest possible values.
52        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
53          algorithm: auto
54          cost: 4 # Lowest possible value for bcrypt
55          time_cost: 3 # Lowest possible value for argon
56          memory_cost: 10 # Lowest possible value for argon

```


2. Annexe 2 – .env

```
# In all environments, the following files are loaded if they exist,
# the latter taking precedence over the former:
#
# * .env                contains default values for the environment variables needed by the app
# * .env.local          uncommitted file with local overrides
# * .env.$APP_ENV       committed environment-specific defaults
# * .env.$APP_ENV.local uncommitted environment-specific overrides
#
# Real environment variables win over .env files.
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
#
# Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex >=1.2).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=20241180510a88839e027f27d6865079
###< symfony/framework-bundle ###

###> symfony/mailer ###
MAILER_DSN=gmail://AddrTest666@gmail.com:Q5qcd9yNbuE7D3m@default
###< symfony/mailer ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
# DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://btssn_c1:b0urdelle@10.255.7.85:3306/capou1?serverVersion=10.4.24-MariaDB&charset=utf8mb4"
# DATABASE_URL="mysql://btssn_c1:b0urdelle@90.63.226.129:3306/capou1?serverVersion=10.3.22-MariaDB&charset=utf8mb4"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###
```

3. Annexe 3 – AdminController.php

```
1 <?php
2
3 /*=====
4 Name      : AdminController.php
5 Path      : src/Controller/Administrator
6 Author    : BTS SNIR, Lycée Antoine Bourdelle
7 Description : Administrator's interface control
8 Date      : 2022
9 =====*/
10
11 namespace App\Controller\Administrator;
12
13 use App\Entity\Card;
14
15 class AdminController extends AbstractDashboardController
16 {
17     private $doctrine;
18
19     public function __construct(ManagerRegistry $doctrine)
20     {
21         $this->doctrine = $doctrine;
22     }
23
24     #[Route('/admin', name: 'app_admin')]
25     #[IsGranted('ROLE_ADMIN')]
26     public function index(): Response
27     {
28         $users = $this->doctrine->getRepository(User::class)->count([]);
29         $plots = $this->doctrine->getRepository(Plot::class)->count([]);
30         $openValve = $this->doctrine->getRepository(Valve::class)->count(['state' => 1]);
31         if(($this->doctrine->getRepository(Card::class)->count([]))!=0){
32             $cards = $this->doctrine->getRepository(Card::class)->count([]);
33         }
34         else{
35             $cards = 0;
36         }
37         return $this->render('roles/administrator/index.html.twig', ['users'=>$users, 'plots'=>$plots, 'openValve'=>$openValve, 'cards'=>$cards]);
38     }
39
40     public function configureDashboard(): Dashboard
41     {
42         return Dashboard::new()
43             ->setTitle('Irrigation Connectée');
44     }
45
46     public function configureMenuItems(): iterable
47     {
48         yield MenuItem::linkToUrl('Accueil', 'fas fa-home', $this->generateUrl('app_admin'));
49
50         yield MenuItem::section('Utilisateurs');
51         yield MenuItem::subMenu('Actions', 'fas fa-list')->setSubItems([
52             MenuItem::linkToCrud('Ajouter utilisateur', 'fas fa-plus', User::class)->setAction(Crud::PAGE_NEW),
53             MenuItem::linkToCrud('Liste utilisateurs', 'fas fa-eye', User::class)
54         ]);
55
56         yield MenuItem::section('Parcelles');
57         yield MenuItem::subMenu('Actions', 'fas fa-list')->setSubItems([
58             MenuItem::linkToCrud('Ajouter parcelle', 'fas fa-plus', Plot::class)->setAction(Crud::PAGE_NEW),
59             MenuItem::linkToCrud('Liste parcelles', 'fas fa-eye', Plot::class)
60         ]);
61
62         yield MenuItem::section('Vannes');
63         yield MenuItem::subMenu('Actions', 'fas fa-list')->setSubItems([
64             MenuItem::linkToCrud('Ajouter vanne', 'fas fa-plus', Valve::class)->setAction(Crud::PAGE_NEW),
65             MenuItem::linkToCrud('Liste vannes', 'fas fa-eye', Valve::class)
66         ]);
67
68         yield MenuItem::section('Piquets');
69         yield MenuItem::subMenu('Actions', 'fas fa-list')->setSubItems([
70             MenuItem::linkToCrud('Ajouter piquet', 'fas fa-plus', Card::class)->setAction(Crud::PAGE_NEW),
71             MenuItem::linkToCrud('Liste piquets', 'fas fa-eye', Card::class)
72         ]);
73
74         yield MenuItem::linkToLogout('Se déconnecter', 'fas fa-sign-out-alt');
75     }
76 }
```

4. Annexe 4 – Méthode persistEntity() du UserCrudController.php

```
public function persistEntity(EntityManagerInterface $entityManager, $entityInstance): void
{
    if(!$entityInstance instanceof User) return;

    $faker = Factory::create('fr_FR');

    $entityInstance->setPassword($faker->password(8,8));
    $entityInstance->setPlainPassword($entityInstance->getPassword());
    $password = $this->encoder->hashPassword($entityInstance, $entityInstance->getPassword());
    $entityInstance->setPassword($password);

    $email = $this->mailerController->emailRegistration($entityInstance);
    $loader = new FilesystemLoader('..\templates');
    $twig = new Environment($loader);

    $renderer = new BodyRenderer($twig);
    $renderer->render($email);
    $this->mailerController->emailSend($email);

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

    $this->logger->info("Un administrateur vient d'ajouter un nouvel utilisateur ");
}
```

5. Annexe 5 – ParcelleB.json

```
1 {
2   "type": "FeatureCollection",
3   "features": [
4     {
5       "type": "Feature",
6       "id": "01",
7       "geometry": {
8         "type": "Polygon",
9         "coordinates": [
10          [
11            [1.312298932433445, 44.03402375604839],
12            [1.3108437306703056, 44.03288209300098],
13            [1.3096669217090324, 44.03337694963541],
14            [1.3102424672808006, 44.0340701152547],
15            [1.311367889079467, 44.03477240289098]
16          ]
17        ]
18      }
19    ]
20  }
```