| Direction | CNPE | Service | Equipe | Date de dernière modification |
|---|---|---|---|---|
| **DPNT** | **Golfech** | **A2P** | **GSI** | **19/07/2024** |

# Documentation Technique (DT) de l'application CHARLIE.

| | |
|---|---|
| Projet | X |
| Définitif | |

Résumé

Ce document décrit l'architecture de l'application CHARLIE. Pour développer cette application, nous utilisons le langage PHP, le Framework Symfony, ainsi que PostgreSQL pour la gestion de notre base de données.

**Version PostgreSQL** : 10.21
**Version du projet Symfony :** 5.4

**Accessibilité :**

| | |
|---|---|
| Libre | |
| EDF | X |
| Diffusion restreinte | |
| Confidentiel | |

## Version du document

| **Indice** | **Commentaire** |
|---|---|
| Indice 001 | Version Initiale – 02/05/2022 – Moustapha KEBE moustapha.kebe@edf.fr |
| Indice 002 | Version 2 – 19/07/2024 – Moustapha KEBE moustapha.kebe@edf.fr |
| | |
| | |

| **Rédacteur(s)** | **Vérificateur(s)** | **Approbateur** |
|---|---|---|
| Moustapha KEBE (Alternant) | Yassine LAKHAL(Alternant)<br>Thibault RENAUDIN (RSIT/RSSI) | Thibault RENAUDIN (RSIT/RSSI) |
| | | |

# Table des matières

## Tables des figures

# Introduction

## 1. Présentation du projet

CHARLIE est une application web développée en PHP/Symfony, qui permet :

- De localiser et de suivre les dossiers de réalisation des travaux (DRT) : Grâce à CHARLIE, la localisation des DRT est facilitée par des codes-barres à scanner. Lors de la transmission d'un DRT, les codes-barres du DRT et du destinataire sont scannés. Ces informations sont stockées dans une base de données, permettant de dresser l'historique des positions d'un DRT et de savoir qui le détient à un instant T.
- De créer et d'imprimer des bordereaux de transmission des DRT pour BPOIS (Prestataire de la section Doc)

Avec CHARLIE, une traçabilité complète des DRT jusqu'à leur destination finale est assurée.

L'application dispose également d'une interface d'administration permettant de :

- ✔ Gérer la suppression ou la modification des DRT
- ✔ Créer, modifier, supprimer et attribuer des rôles aux différents utilisateurs
- ✔ Créer, modifier et supprimer des étapes
- ✔ Créer, modifier et supprimer des services

Ce document technique nous permettra de comprendre l'architecture globale de l'application CHARLIE, les différentes entités (tables pour la base de données) et les relations entre ces entités, ainsi que de comprendre les algorithmes de certaines fonctions clés.

Pour développer cette application, nous avons utilisé les technologies suivantes :

- PHP 7.4 (mis à jour à la version 8.2.18 le 19/07/2024)
- Symfony 5.4
- Twig
- JavaScript
- DataTables pour gérer les listes des DRT, services, étapes et utilisateurs
- Ajax pour les requêtes entre le front et le back
- HTML et CSS
- PostgreSQL 10.21 pour la gestion de notre base de données

## 2. Contexte du projet

### Historique du projet :

Les DRT, scannés et enregistrés dans un fichier Excel, suivent un processus comportant plusieurs étapes spécifiques à chaque service et sont ensuite transférés à BPOIS, prestataire de la section Doc pour regroupement et traitement, mais la gestion via Excel devient problématique en raison de la lenteur et du manque de visibilité. L'application CHARLIE a été développée pour centraliser le suivi et la localisation des DRT, résolvant ainsi ces difficultés.

### Principaux contributeurs

- Développeurs : Moustapha KEBE, Yassine LAKHAL et Thibault RENAUDIN, responsables du développement complet de CHARLIE, de la conception à la mise en production.
- Scrum Master : Thibault RENAUDIN, qui a supervisé le projet, assurant la liaison entre les développeurs et le Product Owner tout en contribuant au développement.
- Product Owner : Myriam ZAWISLAK, la commanditaire du projet.

### Utilisateurs cibles

L'application est destinée principalement aux métiers et à BPOIS, le prestataire de la section Doc.

# I. Configuration et Installation

## 1. Prérequis

### Environnement de Développement

Pour développer et exécuter l'application, il est recommandé d'utiliser l'environnement suivant :

- **Système d'exploitation** : Ubuntu version 22.04.4 LTS
- **Serveur Web** : Apache version 2.4
- **PHP** : 8.2.18
- **Base de données** : PostgreSQL version 10.21

### Logiciels et Outils Nécessaires

Pour configurer et développer l'application CHARLIE, les outils suivants sont nécessaires :

- **Composer** : Gestionnaire de dépendances pour PHP, version 2.0. Utilisé pour installer et gérer les bibliothèques PHP nécessaires au projet.

- **Visual Studio Code** : Environnement de développement intégré (IDE) recommandé, avec extensions pour PHP Symfony et Twig.

## 2. Installation du projet

Installez composer si ce n'est pas déjà fait :

- Téléchargez le programme d'installation de Composer depuis le site officiel de Composer.
- Exécutez le programme d'installation et suivez les instructions pour installer Composer sur votre système.

Utilisez la commande suivante pour créer un nouveau projet Symfony :
[redacted] (ici charlie)
Une fois le projet créé, naviguez dans le dossier du projet :
[redacted] (ici charlie)
Pour installer les dépendances nécessaires à une application web, utilisez la commande :
[redacted]
Vous pouvez démarrer le serveur intégré avec la commande :
[redacted]
Ou utilisez le CLI Symfony si vous l'avez installé, avec la commande :
[redacted]
NB : Vous pouvez installer la Symfony CLI globalement avec Composer en utilisant la commande suivante :
[redacted]

## 3. Mise en place de la base de données

Connectez-vous à PostgreSQL avec l'utilisateur postgres et créez une nouvelle base de données ainsi qu'un utilisateur :
[redacted] (ici charliedb)
Configurer Symfony : Dans votre projet Symfony, configurez l'accès à la base de données en modifiant le fichier **.env** à la racine de votre projet avec les informations de connexion à PostgreSQL :
[redacted]

Installer Doctrine : Si ce n'est pas déjà fait, installez Doctrine, l'ORM utilisé par Symfony :
[redacted]

Créer les entités (les tables de la base de données) et la migration : Utilisez la console Symfony pour générer une entité et créer une migration :
[redacted]

[redacted] ou [redacted]

Vérifier la connexion : Pour vérifier que la connexion à la base de données fonctionne, vous pouvez utiliser la commande suivante :
[redacted]

- bin/ : Contient le script console, le principal point d'entrée pour les commandes en ligne de commande.
- config/ : Inclut les fichiers de configuration, souvent initialisés avec des valeurs par défaut pour divers aspects de l'application.
- public/ : Sert de répertoire racine pour le site web, avec index.php agissant comme le point d'entrée principal pour les requêtes HTTP dynamiques.
- src/ : Héberge le code source que vous écrivez pour votre application.
- templates/ : Regroupe les vues de l'application, c'est-à-dire les fichiers HTML qui seront rendus et affichés aux utilisateurs.
- var/ : Contient les fichiers générés par l'application pendant son exécution, tels que les caches et les logs.
- vendor/ : Comprend tous les paquets installés via Composer, y compris Symfony. Il est essentiel à la productivité et ne doit pas être modifié manuellement.
- .env : Un fichier utilisé pour définir les variables d'environnement nécessaires au fonctionnement de l'application dans différents environnements (développement, production, etc.).

Voici comment changer l'environnement de l'application dans le **.env** de Symfony (dev ou prod)

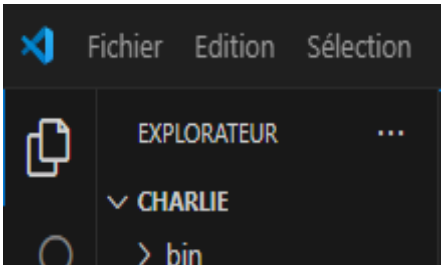
Voici comment ajouter le serveur SMTP dans le **.env** de Symfony

Voici comment ajouter la BDD postgres dans le **.env** de Symfony

# II. Architecture du Projet

## 1. Structure des dossiers

*Figure 1 : Structure des dossiers*



## 2. Conception de l'application

*Figure SEQ Figure * ARABIC 3 : Diagramme d'activité des fonctionnalités pour la partie BPOIS (Docapost)*

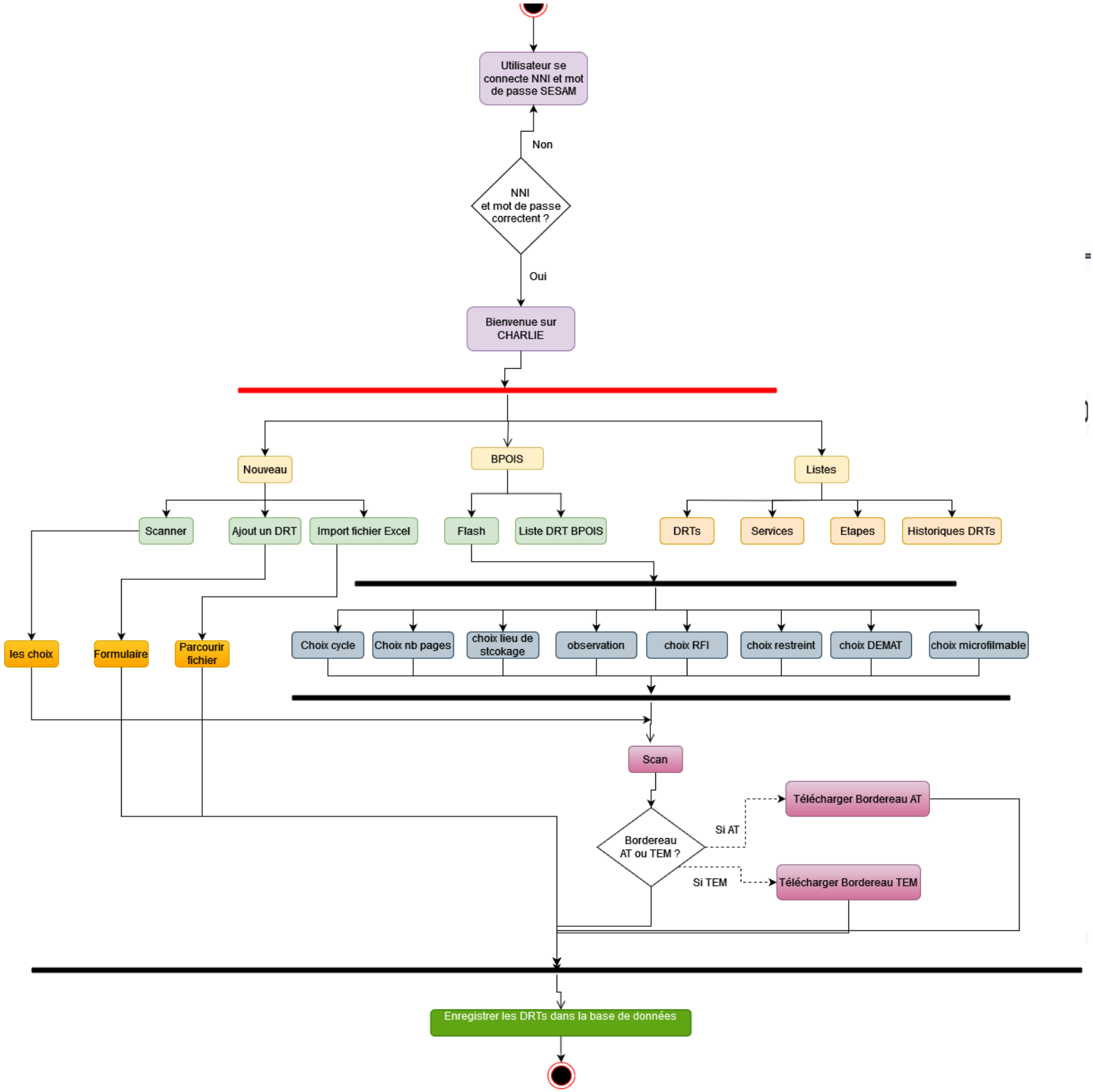
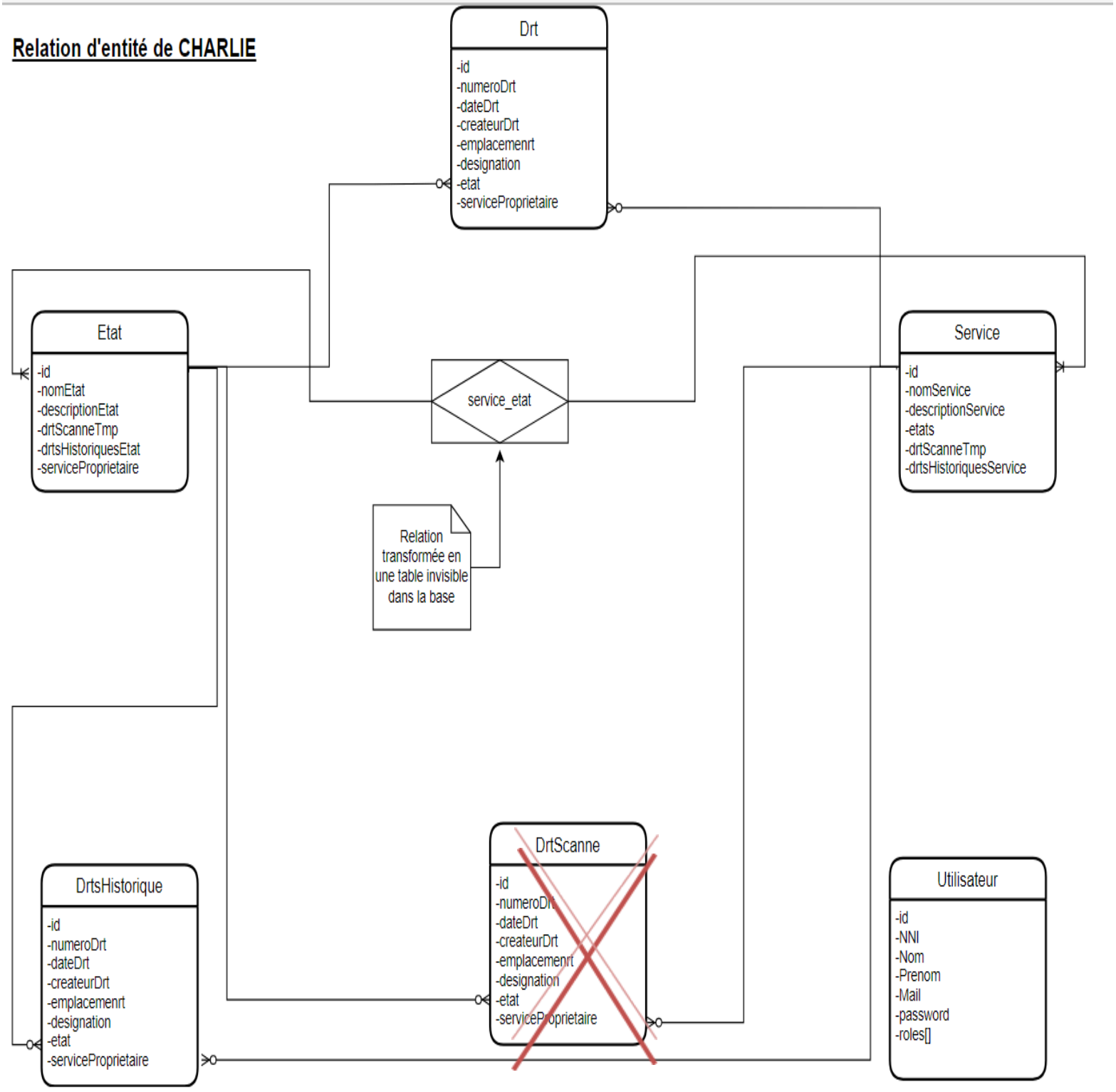


*Figure SEQ Figure * ARABIC 4 : Diagramme d'entité (classe)*



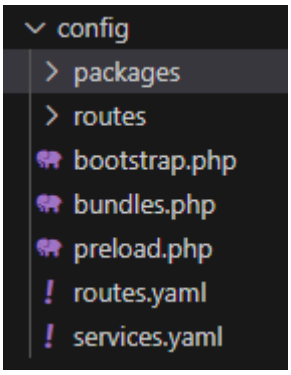
Cette entité n'existe plus

# III. Développement Backend

## a. Symfony Framework

### Configuration et utilisation des bundles principaux :

Notre application a été configurée avec les fichiers stockés dans le répertoire **config/**, qui a cette structure par défaut :




- Le répertoire **packages/** stocke la configuration de chaque paquet installé dans l'application
- Les fichier **bundles.php** active/désactive les paquets dans l'application. Ce fichier est généré automatiquement lors de l'installation de Symfony et est mis à jour lorsque vous installez ou désinstallez des bundles via composer.
- Le fichier **routes.yaml** définit la configuration du routage
- 
- Le fichier **services.yaml** permet de définir les services utilisés dans CHARLIE et de configurer leur injection de dépendances.

## b. Modèles et Entités

### Définition des entités

*Dans src\Entity*

#### a. Drt.php

Cette entité représente un DRT. Ses attributs incluent :

- Numéro
- Date de création
- Personne créatrice (relation avec Utilisateur)
- Emplacement
- Désignation
- État (relation avec Etat)
- Service propriétaire (relation avec Service)

  etc.

### b. DrtsHistorique.php

Cette entité est similaire à Drt.php, mais elle est utilisée pour maintenir un historique des modifications et suppressions de DRTs. Elle a les mêmes attributs que Drt et enregistre les changements pour traçabilité.

### c. Etat.php

Cette entité représente l'état d'un DRT. Ses attributs incluent :

- Nom
- Description
- Service propriétaire (relation avec Service)

### d. Service.php

Cette entité décrit un service. Ses attributs incluent :

- Nom
- Description
- États associés (relation avec Etat via une collection)

### e. Utilisateur.php

Cette entité représente un utilisateur. Ses attributs incluent :

- NNI (identifiant unique)
- Nom
- Prénom
- Email (format : prenom.nom@edf.fr)
- Mot de passe (pour l'authentification)
- Rôles associés

## ImportFichier.php

Cette classe permet d'importer un fichier (Excel ou PDF). Ses attributs et méthodes incluent :

- Fichier (le fichier à importer)
- Méthodes :
  - getFichier() : récupère le fichier.
  - setFichier($fichier) : associe un fichier donné en paramètre.

### f. Recherche.php

Cette classe représente les critères de recherche pour les DRTs. Ses attributs incluent :

- Numéro DRT
- Créateur (relation avec Utilisateur)
- État d'un DRT (relation avec Etat)
- Service propriétaire (relation avec Service)
  etc.
- Elle inclut également les getters et setters pour ces attributs.

## Relations entre les entités

### Drt - DrtsHistorique

- Relié à Utilisateur (créateur)
- Relié à Etat
- Relié à Service

### Etat

- Relié à Service (plusieurs états peuvent être associés à un service)

### Service

- Contient une collection d'Etat
- Peut être associé à plusieurs Drt

### Utilisateur

- Peut créer plusieurs Drt
- Peut être associé à plusieurs Recherche

### ImportFichier

- Pas de relation directe avec les autres entités, mais associé à des opérations d'importation de fichiers

### Recherche

- Relié à Utilisateur (créateur d'un Drt)
- Peut concerner un Drt spécifique
- Peut concerner un Etat spécifique
- Peut concerner un Service spécifique

## c. Contrôleurs

Les contrôleurs sont des classes qui gèrent les actions relatives aux déférentes entités de CHARLIE (gèrent les requêtes HTTP et retournent des réponses HTTP).
Voici une explication des deux principaux contrôleurs : **ScanetteController** et **ListeVueController.**

*Dans src\Controller*

### ScanetteController

Il gère la fonctionnalité de scan de et le traitement des DRT. Voici une explication détaillée de certaines de ses fonctions qui permettent de gérer le cycle de vie des DRT depuis le scan jusqu'à leur enregistrement et génération de bordereaux sous forme de PDF, en s'appuyant sur les services (`LdapService` et `NumBordereauService`).

*Propriétés*

**$username** : Stocke le nom d'utilisateur récupéré via LDAP.
**$sessionInterface** : Gère les sessions pour stocker temporairement des données.
**$numBordereauService** : Service pour gérer les numéros de bordereaux.

*Constructeur*

```
public function __construct(SessionInterface $sessionInterface, NumBordereauService $numBordereauService)
```

Le `constructeur` initialise les propriétés *`username`*, *`sessionInterface`* et *`numBordereauService`*.

*Fonctions*

- Scanner

```
public function Scanner(Request $request, LdapService $ldap): Response
```

*Description :*

- La fonction `Scanner` gère la soumission d'un formulaire de scan, la vérification des doublons, la création et la sauvegarde des nouveaux DRTs dans la base de données, ainsi que la génération de fichiers PDF et la finalisation des scans.

*Étapes principales :*

a. Création et gestion du formulaire :
- Création d'un formulaire `ScanDrtType` et traitement de la requête $request.
- Extraction du nom d'utilisateur via le service LDAP pour l'attribuer à username.

b. Initialisation :
- Récupération du gestionnaire de doctrine pour les opérations de base de données.
- Récupération d'un tableau temporaire `tatbDrtsTmp` depuis la session pour stocker les DRTs scannés.

c. Traitement du formulaire soumis :
- Vérification si le formulaire a été soumis et est valide.
- Extraction et conversion des données du formulaire (numéro DRT, état, service propriétaire, etc.).
- Vérification de la non-existence du DRT dans la base de données (drtDoublon).
- Création d'un nouvel objet Drt, remplissage de ses attributs, ajout au tableau temporaire et persistance en base de données.
- Mise à jour du tableau des DRTs dans la session.

d. Gestion des actions post-formulaire :
- Si le bouton **Imprimer Bordereau** est pressé, génération d'un fichier PDF contenant les DRTs scannés.
- Si un bouton **Terminer** est pressé, sauvegarde des DRTs scannés, affichage d'un message de succès, et redirection vers la liste des DRTs.

e. Rendu de la vue :
- Affichage du formulaire de scan, des DRTs scannés et d'autres informations utiles dans la vue `Scanette.html.twig`.

- DOCAPOSTEScanner

```
public function DOCAPOSTEScanner(Request $request, LdapService $ldap,  $numBordereau = null, $cycle = null): Response
```

*Description*

- La fonction DOCAPOSTEScanner est similaire à la fonction Scanner mais spécifique aux utilisateurs ayant le rôle ROLE_DOCAPOSTE. Elle permet de gérer les scans de

DRT pour BPOIS (Docapost), incluant la modification et l'ajout de DRTs existants, ainsi que la génération de bordereaux spécifiques.

- saveDRT

```
private function saveDRT($lesDRTs, $ldap, $numBordereau, $cycle)
```

Cette fonction sauvegarde les DRTs scannés dans la base de données et met à jour la session.

### ListeVueController

Le contrôleur `ListeVueController` gère différentes fonctionnalités pour la gestion et la visualisation des DRT. Voici une explication détaillée des principales méthodes de ce contrôleur :

- *ajaxData*

```
public function ajaxData(Request $request)
```

  - **Route** : `/ajax_data`
  - **Méthodes HTTP** : GET
  - **Fonction** : Retourne des données en JSON pour une requête AJAX.
  - **Fonctionnement** :
    - Récupère les paramètres de requête pour filtrer les DRTs.
    - Crée un objet `Recherche` et le remplit avec les filtres obtenus.
    - Utilise le repository `Drt` pour obtenir les DRTs filtrés.
    - Formate les données et les retourne en JSON et elles seront traitées côté javascript.

- *ajaxDataDOCAPOSTE*

```
public function ajaxDataDOCAPOSTE(Request $request)
```

  - **Route** : `/ajax_data_docaposte`
  - **Méthodes HTTP** : GET
  - **Fonction** : Retourne des données spécifiques à BPOIS (DOCAPOST) en JSON pour une requête AJAX.
  - **Fonctionnement** :
    - Similaire à `ajaxData`, mais avec des filtres spécifiques à BPOIS.
    - Utilise des dates et d'autres filtres spécifiques pour les DRTs liés à BPOIS.
    - Les données seront retournées en JSON et traitées côté javascript.

- *ListeDrts*

```
public function ListeDRTs(Request $request)
```

- **Route** : `/ListeDrts`
- **Fonction** : Affiche une liste des DRTs scannés.
- **Fonctionnement** :
  - Crée et gère un formulaire de recherche.
  - Récupère les DRTs via le repository `Drt`.
  - Passe les DRTs et le formulaire à la vue `ListeDRTs.html.twig`.

Les autres fonctions : `ListeDrtsDOCAPOSTE`, `ListeDrtsByService`, `CycleVie`, `RecapitulatifByService et ListeEtats ListeServices` sont similaires à la fonction ListeDrts décrit précédemment.

### d. Formulaires

*Dans src\Form*

Dans ce dossier se trouvent les fichiers relatifs aux différents formulaires de CHARLIE. Notamment, le fichier `DrtForm.php` se distingue par sa fonctionnalité spécifique suivante :

```php
$builder->get('serviceProprietaire')->addEventListener(
    FormEvents::POST_SUBMIT,
    function (FormEvent $event) {
        $form = $event->getForm();
        $this->addEtatField($form->getParent(), $form->getData());
    }
);

$builder->addEventListener(
    FormEvents::POST_SET_DATA,
    function (FormEvent $event) {
        $data = $event->getData();
        $form = $event->getForm();
        if ($data && $etat = $data->getEtat()) {
            $service = $form->get('serviceProprietaire')->getData();
            $this->addEtatField($form, $service);
            $form->get('serviceProprietaire')->setData($service);
        } else {
            $this->addEtatField($form, null);
        }
    }
);
```

Il dynamise l'affichage des `états` disponibles en fonction du `service` choisi.
L'événement `FormEvents::POST_SET_DATA` est utilisé pour récupérer les données une fois que le formulaire a été pré-rempli, c'est-à-dire après la sélection d'un service. Quant à l'événement `FormEvents::POST_SUBMIT`, il est déclenché suite à l'appel de `Form::submit()`, permettant ainsi de synchroniser les données entre le modèle et la vue après leur traitement. La fonctionnalité en question fait appel à la méthode `addEtatField(Form`

`$form, ?Service $service)`, qui ajoute dynamiquement au `ComboBox` les états correspondant au service sélectionné dans le formulaire. L'événement `FormEvents::POST_SET_DATA` est utilisé pour récupérer les données une fois que le formulaire a été pré-rempli, c'est-à-dire après la sélection d'un service.

### e. Repository

*Dans src\Repository*

Ce répertoire contient des méthodes pour interroger la base de données.
Par exemple, dans le fichier `DrtRepository.php` nous pouvons noter les méthodes suivantes :

- findAllDrtsDocaposte

```
public function findAllDrtsDocaposte(Recherche $filtre, $orderDir = null, $orderColumn = null, $length = null, $start = null, $docaposte = null)
```

Cette fonction permet de trouver tous les `DRTs` en fonction de plusieurs filtres et de paramètres de tri. Elle prend en entrée un objet `Recherche` contenant les critères de recherche, ainsi que d'autres paramètres optionnels comme la direction de tri, la colonne de tri, la longueur (nombre de résultats) et le point de départ (pour la pagination), ainsi qu'un identifiant pour filtrer les DRT pour BPOSI (Docapost).

*Paramètres* :

- `Recherche $filtre` : Objet contenant les critères de recherche.
- `string|null $orderDir` : Direction de tri (ASC ou DESC).
- `int|null $orderColumn` : Colonne à utiliser pour le tri.
- `int|null $length` : Nombre de résultats à retourner.
- `int|null $start` : Point de départ pour la pagination.
- `int|null $docaposte` : Identifiant pour filtrer les DRT pour BPOIS.

*Logique* :

- Définit les valeurs par défaut pour le tri (`DESC` pour `d.dateDrt` ).
- Modifie la colonne de tri en fonction de `orderColumn`.
- Crée une requête de base pour récupérer les DRTs depuis l'entité `Drt`.
- Ajoute des conditions à la requête en fonction des critères présents dans l'objet `Recherche`.
- Joint d'autres entités si nécessaire (`serviceProprietaire`, `etat`).
- Limite les résultats et gère la pagination.
- Retourne les résultats de la requête.

- findAllDrts

```
public function findAllDrts(Recherche $filtre, $orderDir = null, $orderColumn = null, $length = null, $start = null)
```

Cette fonction est similaire à `findAllDrtsDocaposte`, mais sans le filtre spécifique docaposte.

- countDrts

```
public function countDrts($filtre)
```

Cette fonction compte le nombre de DRTs en fonction des critères de recherche fournis.

*Paramètres* :

- `Recherche $filtre` : Objet contenant les critères de recherche.

*Logique* :

- Crée une requête pour compter les DRTs.
- Ajoute des conditions en fonction des critères présents dans l'objet `Recherche`.
- Retourne le nombre total de DRTs correspondant aux critères.

- countDrtsDocaposte

Cette fonction est similaire à countDrts, mais avec le filtre spécifique docaposte.

- countDifferentEtat

Cette fonction compte le nombre de DRTs pour chaque état distinct, filtré par un service propriétaire spécifique.

*Paramètres* :

- `int $id` : Identifiant du service propriétaire.

Logique :

- Crée une requête pour compter les DRTs groupés par état.
- Joint les entités `serviceProprietaire` et `etat`.
- Retourne une requête qui peut être exécutée pour obtenir les résultats.

- findDrtsBetweenDates

Cette fonction trouve tous les DRTs créés entre deux dates spécifiques.

*Paramètres* :

- `DateTime` $`dateDebut` : Date de début.
- `DateTime` $`dateFin` : Date de fin.

*Logique* :

- Crée une requête pour récupérer les DRTs dont la date se situe entre `dateDebut` et `dateFin`.
- Retourne les résultats de la requête.

- getNumBordereau

Cette fonction récupère le dernier numéro de bordereau pour un cycle spécifique dans l'année courante.

*Paramètres* :

- `string $cycle` : Cycle pour lequel récupérer le numéro de bordereau.

*Logique* :

- Crée une requête SQL native pour récupérer le dernier numéro de bordereau non dématérialisé pour le cycle donné dans l'année courante.
- Exécute la requête et retourne le numéro de bordereau, ou 0 s'il n'y a aucun résultat.

### f. Service

Un service Symfony est tout simplement une classe PHP.

*Dans src\Service*

Dans ce dossier, nous avons les deux services suivants :

### LdapService :

Elle a deux attributs :

- `$pramas` : Cet attribut représente un ensemble de paramètres contenant les informations de connexion à l'annuaire LDAP.
- `$ldap` : une instance de Ldap

Un constructeur : qui permet d'initialiser les attributs de la classe.
Une fonction :

```
public function getInfosByUid($uid)
```

Cette permet de récupérer les informations d'un utilisateur depuis un annuaire LDAP en utilisant son UID (User Identifier).

- La méthode construit une requête LDAP pour rechercher un utilisateur dont l'UID correspond à la valeur passée en paramètre.
- La requête est exécutée sur l'annuaire LDAP.
- Si aucun résultat n'est trouvé, une exception est lancée.
- Si l'utilisateur est trouvé, les informations sont extraites des attributs LDAP et retournées sous forme de tableau associatif. Les attributs couramment extraits incluent l'UID, le nom, le prénom et le NNI.

### NumBordereauService

Cette classe permet d'incrémenter le numéro de bordereau pour un cycle donné. Elle utilise un `EntityManager` pour interagir avec `DrtRepository` afin d'appeler la fonction `getNumBordereau()`.

### Méthode getNumBordereau :

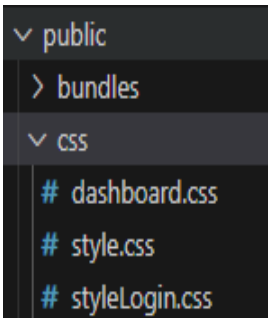
Cette méthode renvoie le ` `next` ` numéro de bordereau pour un cycle donné. La méthode `getNumBordereau` de DrtRepository est appelée pour obtenir le dernier numéro de bordereau pour le cycle donné, puis on incrémente ce numéro de 1.

# IV. Développement Frontend

## 1. Twig Templates

Les vues de notre application se trouvent ici. Cela signifie que toutes les pages affichées à l'écran (la partie HTML) seront situées dans le dossier template.
Nous pouvons noter que ce dossier templates contient les différents fichiers avec l'extension .twig utilisés par les méthodes de nos contrôleurs pour les vues de notre application.

## 2. CSS




`style.css` et `styleLogin.css` sont les deux feuilles de style utilisées. style.css contient le style général de l'application, tandis que styleLogin.css est spécifiquement chargé pour styliser la page de connexion.

De plus, nous intégrons Bootstrap pour une mise en page réactive et cohérente. Nous utilisons les versions suivantes de Bootstrap :

- CSS Bootstrap 5.0.2
- Bootstrap Icons 1.4.1
- Bootstrap JS 5.1.3

*Figure : Architecture dossier template*

## 3. JavaScript

```
13:33 [ webadm @ noeyyqed.adam.adroot.edf.fr ][ /appli/projects/Charlie/apache_2.4/htdocs/Charlie ] > ls public
/js/
dashboard.js         jquery-3.6.0.min.js         recherche.js  sortTable.js
jquery-3.5.1.min.js  jquery.dataTables-2.0.8.js  scanette.js   target.js
```

Voici une explication des deux principales fonctions JavaScript `chargerTableau` et la fonction ajax pour l'initialisation de `DataTables` pour `liste_drts_docaposte`. Ces fonctions prennent le relais une fois que nos fonctions `ajaxData` et `ajaxDataDOCAPOSTE` de nos contrôleurs retournent les données en JSON. Elles envoient également des données au `back-end` (contrôleur pour les fonctions `ajaxData` et `ajaxDataDOCAPOSTE`) lors de l'application d'un filtre :

- chargerTableau

La fonction chargerTableau est utilisée pour initialiser une table `DataTables` avec des données provenant du serveur via une requête `AJAX`. Elle prend deux paramètres : l'ID de la table (`tableauId`) et l'URL à partir de laquelle les données doivent être récupérées (`url`).

*Paramètres*

- `tableauId` : ID de la table HTML à laquelle DataTable doit être appliqué.
- `url` : URL de l'endpoint AJAX pour récupérer les données.

*Fonctionnalités*

**Définition des colonnes et des options**

- columns : Définition des colonnes de la table avec des données spécifiques et des options de rendu personnalisées.
- columnDefs : Définition des attributs supplémentaires pour les colonnes, comme la largeur.

**Initialisation de DataTable**

- Configure des options comme le défilement horizontal et vertical, le tri, la recherche, et le chargement des données côté serveur.
- ajax : Configure la requête AJAX pour récupérer les données. Les filtres sont envoyés en tant que paramètres de la requête.
- initComplete: Masque certaines colonnes et éléments pour les utilisateurs qui n'ont pas le rôle `ROLE_ADMIN`.

**Application des filtres**

- Recharge les données de la table chaque fois qu'un filtre est modifié.

- Ajax pour Initialisation de DataTable pour liste_drts_docaposte

Cette fonction est similaire à chargerTableau, mais spécifique à la table `liste_drts_docaposte`.

# V. Intégration de DataTables

## 1. Configuration de DataTables

### a. Installation

Inclure les fichiers `JavaScript` et `CSS` de `DataTables` et `jQuery` dans le fichier `base.html.twig`.

```
<script  type="text/javascript" src="{{ asset('js/jquery-3.6.0.min.js') }}"></script>
<script  type="text/javascript" src="{{ asset('js/jquery.dataTables-2.0.8.js') }}"></script>
```

```
<link rel="stylesheet" type="text/css" href=" {{ asset('css/jquery.dataTables-2.0.8.css') }}">
```

### b. Configuration

Après avoir inclus les fichiers nécessaires, initialisez DataTables sur une table HTML existante en utilisant jQuery.

```
$(document).ready(function() {
        $('#liste_drts_docaposte').DataTable();
});
```

Assurez-vous que l'ID de votre table HTML correspond à celui utilisé dans le script d'initialisation (`#liste_drts_docaposte` dans cet exemple).

*Options de configuration :*
DataTables offre de nombreuses options pour personnaliser son comportement, comme la pagination, le tri, le filtrage, etc.

```
$('#liste_drts_docaposte').DataTable({
    searching: false,
    scrollX: true,
    scrollY: 500,
    scrollCollapse: true,
    responsive: true,
    serverSide: true,
    processing: true,
    order: [1, 'desc'],
```

## 2. Utilisation avec Symfony

### a. Intégration des données backend avec DataTables

*Logique* :

#### Configuration du DataTable :

- **Colonnes** : Vous définissez les colonnes du tableau dans la configuration de DataTables. Chaque colonne correspond à une donnée spécifique à afficher. Par exemple, une colonne pour le `cycle`, une autre pour le numéro DRT (`num`), etc.

#### Configuration de l'AJAX :

- **URL** : Vous spécifiez l'URL du contrôleur Symfony qui fournira les données via une requête AJAX. Par exemple, `url: 'ajax_data_docaposte'`
- **Méthode HTTP** : En général, GET est utilisé pour récupérer des données.
- **Data** : Vous pouvez envoyer des paramètres supplémentaires avec la requête AJAX, tels que des filtres ou des critères de recherche. Ces paramètres sont récupérés des éléments de formulaire ou des champs de recherche sur la page.
- **DataSrc** : DataTables attend une réponse JSON structurée d'une certaine manière. `dataSrc` est une fonction qui indique à DataTables où trouver les données dans la réponse JSON.

#### Formatage des Données :

- **Réponse JSON** : Le backend (Symfony) renvoie des données au format JSON. Cela inclut un tableau de données et des informations sur le nombre total d'enregistrements et le nombre d'enregistrements filtrés.
- **Formatage** : Les données sont formatées dans le backend pour correspondre aux attentes de DataTables. Par exemple, les dates.

### b. Gestion des requêtes AJAX pour DataTables

DataTables propose l'option **server-side processing**, qui permet de mettre en place un chargement différé des données du tableau via AJAX. Lorsque cette option est activée, DataTables envoie automatiquement lors d'un changement sur les données à afficher (changement de page, changement du nombre d'élément à afficher ou changement des tris) une requête AJAX pour récupérer les données à afficher. Cette option permet de réduire la charge imposée à DataTables pour le rendu de nombreuses données, en ne lui envoyant que les données utiles à l'affichage. L'utilisation d'AJAX permet de mettre à jour les données du tableau sans recharger la page.

DataTables utilise AJAX pour charger les données de manière asynchrone, ce qui permet une expérience utilisateur fluide sans recharger la page. La gestion des requêtes AJAX implique la configuration du frontend et la gestion de ces requêtes côté serveur.

*Logique* :

Envoi de la Requête AJAX :

- **DataTables** envoie des requêtes AJAX à l'URL spécifiée chaque fois qu'une nouvelle page est demandée, que les colonnes sont triées, ou que des filtres sont appliqués.
- **Paramètres** : Par défaut, 5 paramètres sont envoyés par DataTables:
    - Start (indice de départ des données correspondant à la page demandée dans le tableau)
    - Length (nombre d'éléments demandés)
    - OrderColumn (indice de la colonne utilisée pour le tri)
    - OrderDri (ordre du tri, ASC ou DESC pour croissant ou décroissant)

  Une fois ces paramètres récupérés côté serveur, il suffit de les utiliser dans une requête puis de renvoyer les données demandées.
- Il est également possible d'ajouter des paramètres à ces requêtes envoyées automatiquement. Nous avons ajouté 6 paramètres, qui correspondent aux filtres demandés pour le tableau :

```
data: function (d) {
        d.numero = $('#numeroDrt').val();
        d.rfi = $('#rfi').val();
        d.dateDebut = $('#dateDebut').val();
        d.dateFin = $('#dateFin').val();
        d.dateCaptive = $('#dateCaptive').val();
        d.lieuDeStockage = $('#lieuStockage').val();
        d.cycle = $('#cycle').val();
      },
```

Ce code est situé dans la configuration DataTables : A chaque requête envoyée, le contenu des champs de filtre est récupéré et est envoyé comme paramètre pour chacun des filtres.

```
$('#numeroDrt, #dateCaptive,  #recherche, #dateDebut, #dateFin, #rfi,
#lieuStockage, #cycle').on('input', function () {
    $('#liste_drts_docaposte').DataTable().ajax.reload();
});
```

Ce code permet dynamiser le chargement du tableau, en rechargeant les données à chaque modification des champs filtres.

### Traitement des Requêtes Côté Serveur :

- **Contrôleur Symfony** : Le contrôleur `ListeVueController` récupère les paramètres envoyés par DataTables, effectue les filtrages et les tris nécessaires sur les données, puis renvoie les résultats au format JSON.
- **Filtrage et Tri** : Les paramètres reçus sont utilisés pour filtrer et trier les données avant de les envoyer. Par exemple, si un utilisateur filtre par `date` ou trie par une `colonne spécifique`, ces informations sont prises en compte pour formater les résultats.

### Réponse AJAX :

- **Structure des Données** : La réponse JSON doit inclure :
  - data : Le tableau des données à afficher.
  - recordsTotal : Le nombre total des données dans la base de données (sans filtrage).
  - recordsFiltered : Le nombre total des données après application des filtres (si applicable).
- **Utilisation de dataSrc** : Dans le frontend, dataSrc est utilisé pour extraire les données correctes du JSON renvoyé par le serveur et les injecter dans le tableau.

### Mise à Jour du Tableau :

- DataTables met automatiquement à jour le tableau avec les nouvelles données reçues. Cela inclut la gestion des pages, le tri et le filtrage.

# VI. Déploiement

## 1. Environnements de déploiement

### Configuration des environnements

Il faut se référer au fichier **`PTP-Eastreinte.docx`**. Vous pouvez le trouver dans le dossier `\\atlas.edf.fr\CO\82gol-dpn\Outils-Metiers.005\SLS.010\SLS_GSI\1 - Bible GSI\01_VM\Doc_Technique_Appli_Locale`

### Stratégies de déploiement

Nous effectuons un déploiement manuel, c'est-à-dire que nous copions le projet sur une clé USB sécurisée depuis notre PC de développement, puis nous le transférons sur nos ordinateurs EDF.

## 2. Outils de déploiement

### Outils utilisés pour le déploiement

Une fois le projet transféré sur nos ordinateurs EDF, nous utilisons WinSCP pour le transférer sur la VM et Solar-PuTTY pour accéder aux VM.

### Commandes et étapes de déploiement

Il faut se référer au fichier **`PTP-Eastreinte.docx`**. Vous pouvez le trouver dans le dossier `\\atlas.edf.fr\CO\82gol-dpn\Outils-Metiers.005\SLS.010\SLS_GSI\1-Bible GSI\01_VM\Doc_Technique_Appli_Locale`

# VII. Ressources et Références

## Documentation externe

- Vous trouverez les guides d'utilisation (utilisateurs et admin) de CHARLIE dans ce dossier :

```
\\atlas.edf.fr\CO\82gol-dpn\Outils-Metiers.005\SLS.010\SLS_GSI\1 - Bible GSI\01_VM\Doc_Technique_Appli_Locale\Charlie
```

## Liens vers la documentation des Framework et des bibliothèques utilisées :

https://symfony.com/doc/current/configuration.html#configuration-files

## Tutoriels et guides recommandés

https://symfony.com/doc/5.4/the-fast-track/fr/3-zero.html

# VIII. Retour d'expérience

## 1. Mise à Jour de DataTables pour résoudre les problèmes de pagination et de tri sur Chrome et Edge

*Problème :*

Nos tableaux utilisant DataTables sur l'application CHARLIE fonctionnent parfaitement sous Firefox. Sous Chrome ou Edge, c'est également le cas lors du premier affichage, cependant lorsque nous essayons de paginer ou de trier la table ou même d'agrandir la page du navigateur, la requête XHR reste bloquée en statut "Pending" indéfiniment, ce qui provoque un rechargement infini du tableau.

*Solution :*

Pour résoudre ce problème, nous avons mis à jour les versions CSS et JS de DataTables.

*Ancienne version :*

- Pour DataTable js : 1.13.4
- Pour DataTable CSS : 1.13.4

*Nouvelle version :*

- Pour DataTable js : 2.0.8
- Pour DataTable CSS : 2.0.8

## 2. Problème d'affichage de DataTables sous Chrome et solution de contournement

Problème :

Sous Chrome, avec l'application CHARLIE, nous avons constaté que certaines pages ne s'affichent pas entièrement ou que la table n'arrive pas à afficher les données. Cela est dû au fait que Chrome restreint l'accès aux CDN (Content Delivery Network, ou réseau de diffusion de contenu) distants de DataTables JS et CSS , de même pour le CDN de jQuery.

Solution :

Pour résoudre ce problème, nous avons importé les fichiers JS et CSS de DataTables et celui de jQuery en local au lieu de les charger depuis les CDN via Internet. Nous avons ensuite mis à jour les chemins d'appel de ces fichiers.

*Au lieu de :*

```
<link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/1.13.4/css/jquery.dataTables.css">

<script type="text/javascript" charset="utf8"
src="https://cdn.datatables.net/1.13.4/js/jquery.dataTables.js"></script>

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

*Nous les remplaçons par :*

```
<link href="{{ asset('css/ jquery.dataTables-2.0.8.css) }}" rel="stylesheet">

<script  type="text/javascript" src="{{ asset('js/ jquery.dataTables-2.0.8.js')
}}"></script>

<script  type="text/javascript" src="{{ asset('js/ jquery-3.6.0.min.js')
}}"></script>
```

### 3. Problème de Connexion et de Rôles dans l'Application CHARLIE

Problème :

Un utilisateur se connecte à l'application **CHARLIE** avec son NNI et le mot de passe SESAM. Il a été enregistré dans l'application avec le rôle : `DOCAPOSTE`. Cependant, il n'arrive pas à voir les fonctionnalités associées à ce rôle. Cela signifie que cet utilisateur n'a pas le bon profil sur CHARLIE. Ce problème est dû au fait que l'utilisateur s'est connecté avec son NNI en minuscule.

Si cela vous arrive avec l'application Transport, voici la solution :

*Solution* :

Il faut entrer le **NNI en MAJUSCULE**.

## Annexes

### Mise en place de la pagination des pages.

NB : Cette méthodologie n'est plus utilisée, elle a été remplacée par DataTables

## Installation de KNP Paginator :

Afin d'intégrer **KNP Paginator** à notre projet, nous allons faire appel à ***composer*** au moyen de la commande suivante :

***composer require knplabs/knp-paginator-bundle***

Insérez les lignes ci-dessous dans " ***config\packages\twig.yaml*** "

```
paths:
    '%kernel.project_dir%/vendor/knplabs/knp-paginator-bundle/templates': KnpPaginator
```

Dans ***config\packages***, créez le fichier ***knp_paginator.yaml*** et insérerez les lignes suivantes :

```
knp_paginator:
    page_range: 3 # number of links shown in the pagination menu (e.g: you have
10 pages, a page_range of 3, on the 5th page you'll see links to page 4, 5, 6)
    default_options:
        page_name: page                # page query parameter name
        sort_field_name: sort          # sort field query parameter name
        sort_direction_name: direction  # sort direction query parameter name
         distinct: true                  # ensure distinct results, useful when
ORM queries are using GROUP BY statements
        #filter_field_name: filterField  # filter field query parameter name
        #filter_value_name: filterValue  # filter value query parameter name
    template:
                                                                    pagination:
'@KnpPaginator/Pagination/twitter_bootstrap_v4_pagination.html.twig'   #   sliding
pagination controls template
         sortable: '@KnpPaginator/Pagination/sortable_link.html.twig' # sort link
template
          filtration: '@KnpPaginator/Pagination/filtration.html.twig'   # filters
template
```

## Activation de la pagination dans le contrôleur

Pour **activer la pagination** dans le contrôleur, il nécessite dans un premier temps de **récupérer toutes les données devant être paginées**. Dans notre exemple, il s'agit de **tous les DRTs**.

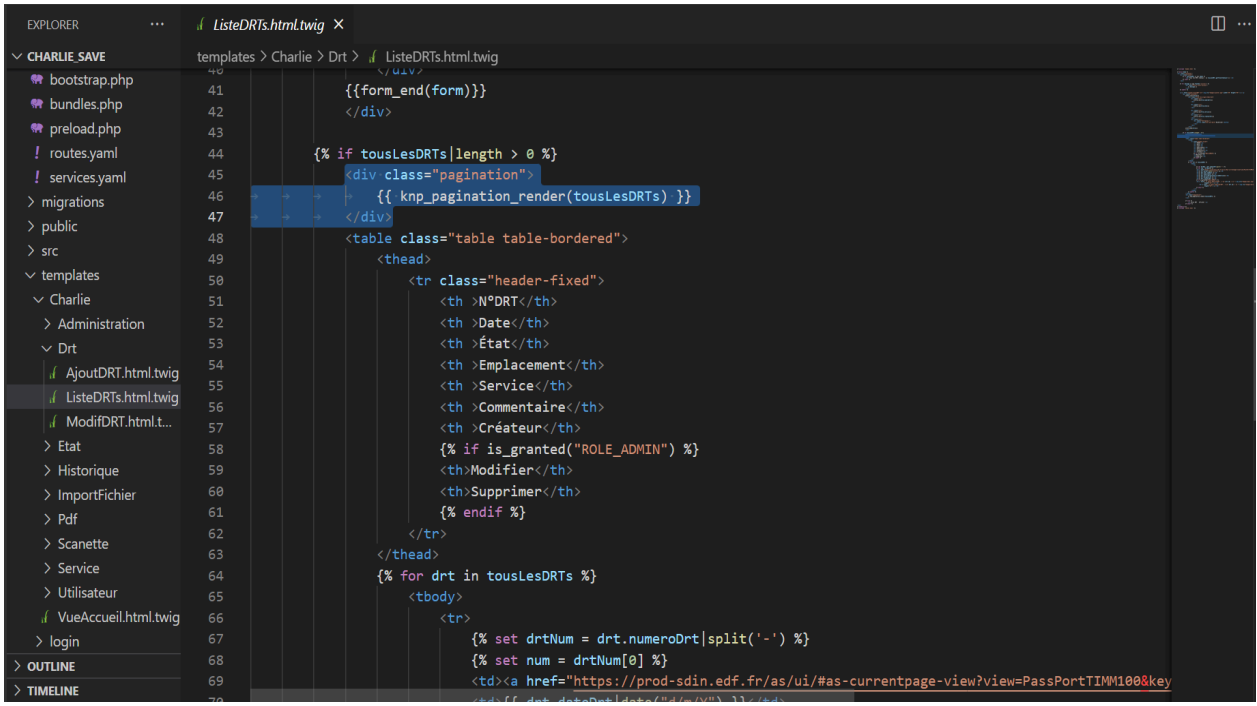
**KNP Paginator** se chargera de "**filtrer**" cette liste et de conserver uniquement les données correspondant à la page courante.

Nous utiliserons la méthode "**paginate**" qui se chargera de tout ça. Cette méthode prend 3 paramètres comme dans notre cas ci-dessous.

```
   // Nombre de résultats par page
    $limite = 100;
    // Numéro de la page en cours, passé dans l'URL, 1 si aucune page
    $page = $request->query->getInt('page', 1);
   // Requête contenant les données à paginer (ici nos Drts)
    $requete = $this->getDoctrine()->getRepository(Drt::class)->findBy(['serviceProprietaire' => $id,]);
    //la pagination
    $lesDRTs = $paginator->paginate(
      $requete,
      $page,
      $limite
    );
```

## Affichage de la pagination dans la vue

Une fois les données envoyées à la vue, nous pouvons y **afficher le sélecteur de pages** au moyen de la méthode "**knp_pagination_render**" comme dans le code ci-dessous, dans le fichier "***Charlie/Drt/ListeDRTs.html.twig***" du dossier "**template**"




Dans le dossier translation, créez un fichier par exemple ***KnpPaginator.yaml*** et insérez les lignes suivantes :

```
label_next: suivant
label_previous: précédent
```