

# Rapport de stage

**Cédric LECLERCQ**  
**Développeur Web-Web mobile**

Stage réalisé du 23/03/2020 au 30/04/2020  
Pour la CAD (communauté d'agglomérations du Douais)  
Sous l'encadrement de mon tuteur : Mr Eremus

**Alerte Citoyens**

**Projet développé avec :**



# Sommaire

1-Listes des compétences couvertes.....	p 3
a - Développer une interface utilisateur web dynamique	
b - Réaliser une interface utilisateur avec une solution de gestion de contenu	
c - Créer une base de données	
d - Développer les composants d'accès aux données	
e - Développer la partie back-end d'une application web	
f - Élaborer et mettre en œuvre des composants dans une application de gestion de contenu	
2-Résumé de l'application.....	p 4
3-Cahier des charges.(en annexe)	
4-Spécifications techniques.....	p 5
5-Réalisations.....	p 6
a - Avant toute chose, installation et préparation de Symfony	
b - Discussions et échanges sur l'application et le cahier des charges	
c - La page d'accueil	
d - Création de la base de donnée	
e - Retour sur Symfony	
f - Le CRUD (pour create, read, update, delete)	
g - En avant pour le formulaire d'inscription	
h -Attaquons la partie administration	
6-Jeu d'essai.....	p 30
a - Je rempli un formulaire d'inscription	
b - Vérification de son entrée en Bdd	
c - Connexion administrateur	
d - Arrivée sur la plate forme d'administration	
e - Validation de la nouvelle demande de souscription	
f - sélection de l'alerte	
g - Envoi du mail	
7-Exemple de difficulté rencontrée, recherche et traduction.....	p 34
8-Pour conclure et à suivre.....	p 35
9-Annexes	

# **1- Les compétences couvertes lors de cette réalisation :**

(les compétences listées ci-dessous feront l'objet d'un développement plus conséquent plus avant dans ce livret)

## **a - Développer une interface utilisateur web dynamique :**

Deux parties distinctes ont été ici réalisées à destinations des particuliers usagers et des administrateurs. La page d'accueil reprend des éléments de dynamisme et d'interaction (slider, menu déroulant,,,) afin de rendre la navigation et l'usage intuitive et « vivante ».

L'application est principalement destinée à un usage sur ordinateur, néanmoins son interface est prévue pour être utilisable et s'adapter aux différents supports (« responsive design »).

## **b - Réaliser une interface utilisateur avec une solution de gestion de contenu :**

Dans le cadre de cette application, le particulier ne sera amené qu'à procéder à son inscription.

Par contre l'administrateur aura à sa disposition de nombreuses possibilités de gestion et pourra facilement générer des alertes dont il aura la maîtrise du contenu.

Un système d'authentification est mis en place avec restrictions d'accès selon les rôles définis.

## **c - Créer une base de données :**

L'application repose sur une base de données réfléchie en amont, qui sera alimentée pour partie de manière automatisée et pour partie par des ajouts/modifications de contenu cadrées.

Les insertions de données font l'objet de mesures de sécurité particulières, les mots de passe sont cryptés.

## **d - Développer les composants d'accès aux données :**

Plusieurs méthodes de recherche, de tri ont été mises en place à destination de l'administrateur (principal utilisateur de cette application).

De même des possibilités d'ajout, de modification (cadrées) et de visualisation lui seront données.

## **e - Développer la partie back-end d'une application web :**

La majeure parti de l'application repose sur l'interaction avec la base de données de manière transparente pour l'utilisateur. Les accès (connexions, requêtes ..) sont sécurisés.

Chaque item de l'application est développé de manière autonome en respectant les interactions.

## **f - Élaborer et mettre en œuvre des composants dans une application de gestion de contenu :**

Chaque composant de l'application est mis en place pour répondre aux besoins précis dans une optique de transparence et d'efficacité.

## **2- Résumé de l'application :**

Cette application nommée provisoirement « Alerte Citoyens » est destinée aux services municipaux et aux entreprises publiques de la communauté d'agglomération du Douaisis.

Elle a pour but de permettre à ces organismes de pouvoir diffuser efficacement et de manière simple, des messages ou alertes à destination des particuliers ayant souscrit à ce programme, les informant d'événements les concernant et pouvant impacter leur quotidien.

Il peut s'agir d'informations événementielles (braderies, défilés..) de travaux dans leur rue (voiries...) de coupures passagères des services de distribution (eau, gaz..) etc..

Cette application contient deux parties :

- Une partie visible par tous où les citoyens pourront s'informer sur les modalités de ce programme et y souscrire via un formulaire d'inscription en ligne et également prendre contact avec les administrateurs du service pour toute question ou demande de désabonnement.
- Une partie réservée aux administrateurs (invisible pour le citoyen), qui permettra la création simplifiée et l'envoi des alertes ainsi que la gestion complète des ressources. Sur ces pages l'administrateur pourra accepter ou refuser les demandes de souscriptions (ce qui enverra un mail automatique de réponse) et créer tout les items nécessaires à la modélisation de son alerte et enfin l'envoyer aux citoyens concernés.

Cette application a été réfléchiée et organisée de manière à être aisément utilisable par l'administrateur concerné et lui permettre d'envoyer une alerte complète et documentée sans difficulté.

Pour ce faire chaque étapes de la création puis de l'envoi de cette alerte ont été hiérarchisées et sont automatiquement enregistrées en base de données pour être réutilisables par la suite.

Pour faciliter également les choses (notamment lors de l'affichage des listes des souscripteurs), diverses méthodes de tri et de recherche ont été mises en place.

Tous les items et messages sont automatiquement enregistrés en base de données avec les informations sur leur date création/modification et auteur (service, administrateur..).

Selon le choix du citoyen, l'alerte sera reçue via mail ou sms (ou les deux) et ce, de manière totalement automatisée.

A ce jour une version Bêta fonctionnelle uniquement sur l'envoi de mails va être présentée au client pour validation. Sa production sera complétée en fonction de son retour.

### **3- Cahier des charges:**

L'intégralité du cahier des charges fourni se trouve dans les annexes:

### **4- Spécifications techniques:**

#### **-Symfony :**

Comme spécifié dans le cahier des charges, l'application sera réalisée sous Symfony 5, framework Php reconnu pour la praticité de mise en place et pour son aspect sécuritaire.

Symphony permet également l'utilisation de « bundle » facilement intégrables. Nous utiliserons notamment « SwiftMailer » pour l'envoi de Mails.

#### **-Front :**

Évidemment HTML5 et Bootstrap v4, FontAwesome, CSS 3, JavaScript (Jquery/ajax).

#### **-Back :**

Php 7.4.

#### **-Base de données :**

La conception de la base à été réalisée sous MySql WorkBench 8.0 CE.  
PhpMyAdmin, MySql via WampServeur 64.

#### **-Sauvegarde et versionning :**

Nous utiliserons GitHub et Github Desktop (GitKraken à également été testé).

#### **-Environnement :**

L'application sera réalisée sous PhpStorm.

#### **-Échanges et compte rendus :**

En cette période particulière Discord nous à été fort utile.

#### **-Recherches :**

Mes recherches se sont faites (pour ce que je ne trouvais pas dans mes notes ou livres) sur les sites dédiés, principalement sur la doc Symfony (<https://symfony.com/doc/current/index.html>) et sur Stack Overflow (<https://stackoverflow.com/>) ou OpenClassroom (<https://info.openclassrooms.com>).

## 5- Réalisations:

C'est parti !

a- Avant toute chose, installation et préparation de Symfony :

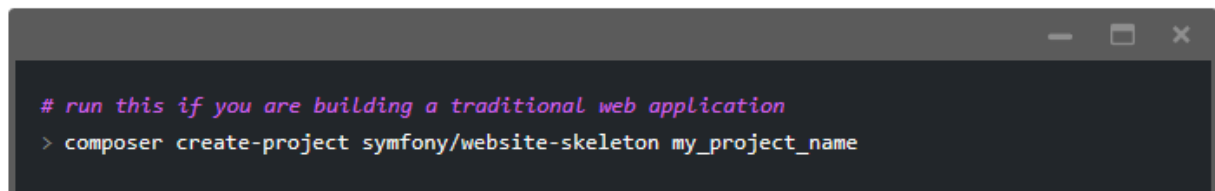
Passage obligé par le site afin de bien respecter les étapes et prendre les liens de téléchargement <https://symfony.com/doc/current/setup.html>

Installation de Composer :

Télécharger le composer (lien : <https://getcomposer.org/doc/00-intro.md>)

Installation de Symfony :

Nous choisissons d'installer la version complète pour application web



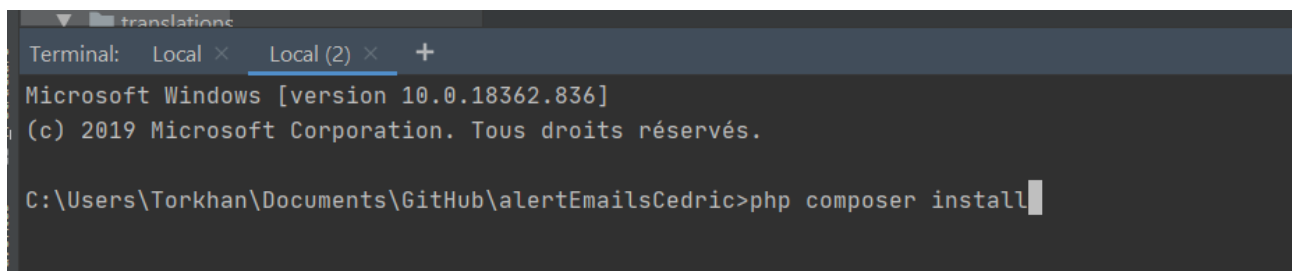
```
# run this if you are building a traditional web application
> composer create-project symfony/website-skeleton my_project_name
```

Le nom de mon dossier sera ici : alertEmailsCedric (il sera placé dans documents/GitHub pour la sauvegarde et le versionning via GitHubDesktop)

Dans le fichier composer.json

- vérifier que version php est supérieure à celle demandée dans le composer.json
- « install » pour installer le composer.phar (avec la version php de l'ordinateur ici 7.4) .

Création du dossier « Vendor » grâce à la ligne de commande « php composer install » tapée dans le terminal de Php Storm comme ceci :



```
Terminal: Local × Local (2) × +
Microsoft Windows [version 10.0.18362.836]
(c) 2019 Microsoft Corporation. Tous droits réservés.

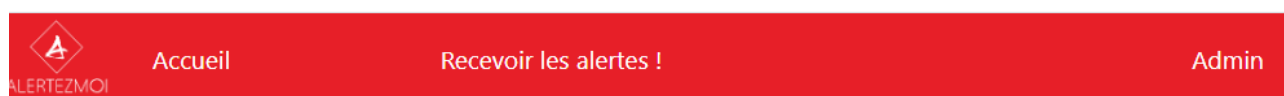
C:\Users\Torkhan\Documents\GitHub>alertEmailsCedric>php composer install
```

Dernier point, le fichier .env qui devra être modifié avec les informations de connexion à la base de donnée un fois celle-ci créée.

## b - Discussions et échanges sur l'application et le cahier des charges :

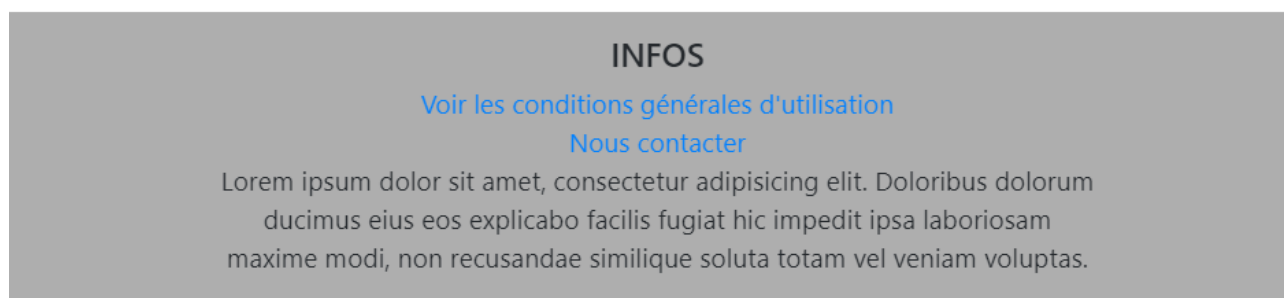
La première étape fut évidemment de discuter de l'application, de bien en saisir le sens et l'orientation à lui donner. Un temps d'étude du cahier des charges a également été pris, afin d'avoir une vue d'ensemble des exigences client et de pouvoir orienter un premier jet sur la future réalisation. D'entrée nous nous sommes mis d'accord sur le changement des couleurs de la page d'accueil et des pages de l'application. En effet les couleurs proposées de base avaient un rendu visuel assez...particulier et ne « collaient » pas à notre sens, à la vocation de l'application.

Nous avons donc opté pour un rouge ([#e51b23](#)) sur la barre de navigation plus en accord avec le côté « alerte », titres sur ce même rouge.



**Alertez - Moi !**

Le footer quant à lui prendra un gris classique ([#AAAAAA](#)) avec texte en noir et liens en bleu selon les conventions d'usage.



### c - La page d'accueil :

Celle-ci est définie très clairement dans le cahier des charges.

Elle comprend donc une barre de navigation classique avec logo (ici logo provisoire créé par ma collègue Noémie Hoffman) en Bootstrap afin d'avoir un affichage cohérent sous tout type de support, notamment mobile, avec apparition d'un bouton type « hamburger » pour le menu).

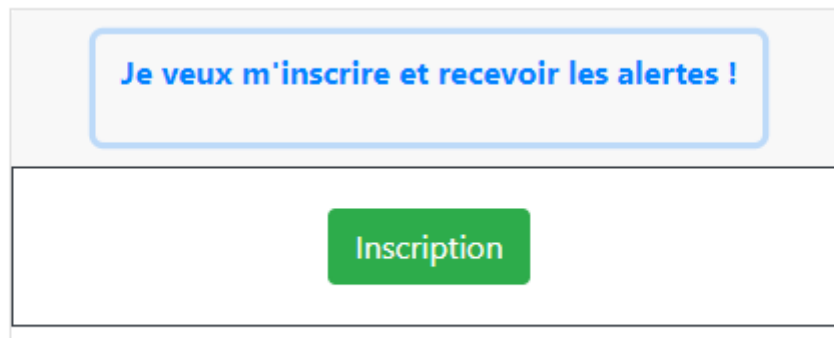


S'en suit un carrousel de trois images. Là encore Bootstrap est notre ami, et une rapide recherche sur le site <https://getbootstrap.com/docs/4.0/components/carousel/>, nous permet de trouver celui qui correspond le mieux à la demande. Les photos utilisées sont libres de droits.

```
<!-- slider -->
<div id="slide" class="my-2">
  <div id="homeCarousel" class="carousel slide" data-ride="carousel">
    <ol class="carousel-indicators">
      <li data-target="#carouselExampleCaptions" data-slide-to="0" class="active"></li>
      <li data-target="#carouselExampleCaptions" data-slide-to="1"></li>
      <li data-target="#carouselExampleCaptions" data-slide-to="2"></li>
    </ol>
    <div class="carousel-inner">
      <div class="carousel-item active">
        
        <div class="carousel-caption d-none d-md-block">
          <h5>First slide label</h5>
          <p>Nulla vitae elit libero, a pharetra augue mollis interdum.</p>
        </div>
      </div>
      <div class="carousel-item">
        
        <div class="carousel-caption d-none d-md-block">
          <h5>Second slide label</h5>
          <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
        </div>
      </div>
    </div>
  </div>
</div>
```



Vient ensuite un menu déroulant type « accordéon » (provisoire) qui contiendra le lien (également présent dans la navbar) vers la page d'inscription.



The image shows a registration form. It consists of a light gray rectangular container. Inside this container, at the top, is a blue rounded rectangle with a thin blue border containing the text "Je veux m'inscrire et recevoir les alertes !". Below this, centered within the gray container, is a green rounded rectangle with a thin green border containing the text "Inscription".

Conformément au cahier des charges un encart contenant des logos de partenaire vient ensuite. Pour l'instant nous avons simulé cet encart avec notre logo, il sera remplacé par les logos partenaires (tels EDF, Véolia etc,,), le but de cet encart est de donner du poids et de rassurer les futurs destinataires des alertes avec des logos d'entreprises publiques qu'ils connaissent.

## Il nous font confiance



Suite à cet encart, vient un texte à caractère informatif qui expliquera ce qu'est cette application, son but et son fonctionnement. Ce texte est pour l'instant en Lorem Ipsum car il sera rédigé par le client .

## "Alertez - moi", en quelques mots

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ab accusantium, at atque blanditiis cum deserunt dicta doloremque eligendi est excepturi id illo, incidunt laborum nobis, numquam odit quam veritatis vitae.

Accusamus adipisci aut fugiat labore molestiae nobis quam qui recusandae saepe voluptatum. Aliquid amet assumenda debitis eos facilis incidunt inventore iste laboriosam mollitia natus neque officiis optio quam, quisquam sapiente. Aliquam cumque incidunt nostrum rem, rerum unde voluptate. Accusantium animi architecto commodi consequatur cum debitis eius eos est hic laboriosam laudantium libero officia officiis quae quaerat, ratione suscipit tempore tenetur?

Pour finir, le footer comprendra deux liens :

Le premier sur les conditions générales d'utilisation envoyant sur une page contenant ces informations légales (le souscripteur devra accepter ces Cgu, il faut donc qu'il puisse en prendre connaissance).

Un deuxième lien vers la classique page contact, car il faut que le souscripteur puisse entrer en contact avec l'administrateur pour toutes question particulière ou demande de désabonnement.

Enfin la possibilité d'ajout de quelques lignes rapides (infos légales ou autres) est donnée et est pour l'instant symbolisée par du Lorem.

### INFOS

[Voir les conditions générales d'utilisation](#)

[Nous contacter](#)

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Doloribus dolorum ducimus eius eos explicabo facilis fugiat hic impedit ipsa laboriosam maxime modi, non recusandae similique soluta totam vel veniam voluptas.



Le fichier sql généré pourra nous permettre de vérifier la bonne tenue de notre base et vérifier qu'il n'y a pas d'erreurs.

Notre table s'appellera « alertecitoyen62141cdd » nom à la fois évocateur mais aussi compliqué à trouver pour d'éventuels pirates. Elle est créée dans PhpMyAdmin.

### e - Retour sur Symfony :

Notre base étant définie nous allons pouvoir la remplir via Symfony.

Tout d'abord il nous faut remplir les informations de connexion dans le fichier .env cité plus haut :

```
###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# For an SQLite database, use: "sqlite:///kernel.project_dir%/var/data.db"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=mysql://nom:mdp@localhost:3308/alertecitoyen62141cdd?serverVersion=5.7
###< doctrine/doctrine-bundle ###
```

Ici nom et mdp seront renseignés avec nos identifications de connexion PhpMyAdmin.  
On vérifie également que le port employé est le bon :



The screenshot shows the WampServer configuration interface. At the top, there's a logo and the text 'WampServer'. Below it, the version 'Version 3.2.0 - 64bit' is displayed, along with language and theme dropdowns set to 'french' and 'classic'. The main section is titled 'Configuration Serveur'. It lists the following details:

- Version Apache :** 2.4.41 - [Documentation](#)
- Server Software :** Apache/2.4.41 (Win64) PHP/7.4.0 - Port défini pour Apache : 80
- Version de PHP :** 7.4.0 - [Documentation](#)
- Extensions Chargées :** A list of 36 extensions including apache2handler, calendar, ctype, dom, filter, gmp, imap, ldap, mysqli, pcre, pdo\_sqlite, Reflection, soap, sqlite3, xdebug, xmlrpc, Zend OPcache, bcmath, com\_dotnet, curl, exif, gd, hash, intl, libxml, mysqlnd, PDO, Phar, session, sockets, standard, xml, xmlwriter, zip, bz2, Core, date, fileinfo, gettext, iconv, json, mbstring, openssl, pdo\_mysql, readline, SimpleXML, SPL, tokenizer, xmlreader, xsl, and zlib.
- Version de MariaDB :** 10.4.10 - Port défini pour MariaDB : 3306 - Default DBMS - [Documentation](#)
- Version de MySQL :** 8.0.18 - Port défini pour MySQL : 3308 - [Documentation](#)

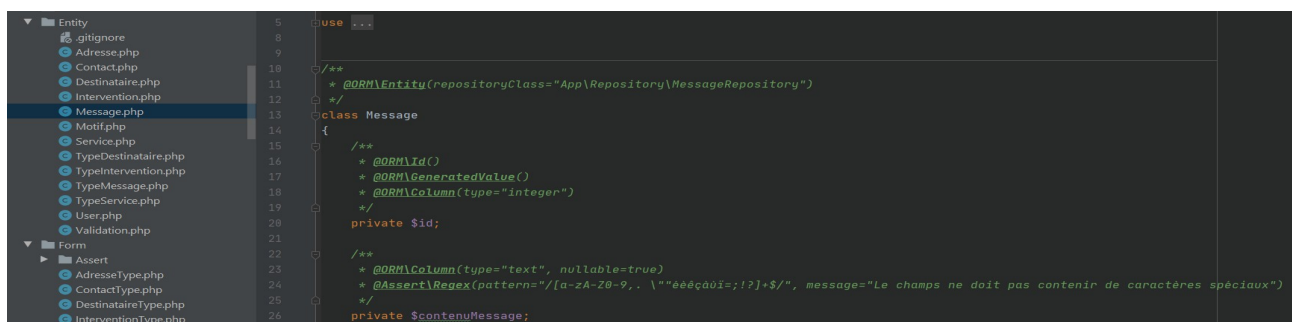
Nous avons donc une connexion établie entre Symfony et notre base de données.

Nous allons pouvoir la créer via la commande `php bin/console doctrine:database:create`, et remplir sa structure en créant les entités :

```
1  $ php bin/console make:entity
2
3  Class name of the entity to create or update:
4  > Product
5
6  New property name (press <return> to stop adding fields):
7  > name
8
9  Field type (enter ? to see all types) [string]:
10 > string
11
12 Field length [255]:
13 > 255
14
15 Can this field be null in the database (nullable) (yes/no) [no]:
16 > no
17
18 New property name (press <return> to stop adding fields):
19 > price
20
21 Field type (enter ? to see all types) [string]:
22 > integer
23
24 Can this field be null in the database (nullable) (yes/no) [no]:
25 > no
26
27 New property name (press <return> to stop adding fields):
28 >
29 (press enter again to finish)
```

Ceci en commençant par les tables simples sans liaisons pour finir par les principales.

Symfony nous a créé automatiquement les entités :



Ainsi que les méthodes get et set qui vont avec.

```
php bin/console make:migration et (si succès)
php bin/console doctrine:migration:migrate
```

Database	Table	Rows	Engine
alertcitoyen62141cdd	Nouvelle table		
	accept	0	InnoDB
	adresse	18	InnoDB
	contact	12	InnoDB
	destinataire	22	InnoDB
	intervention	3	InnoDB
	message	5	InnoDB
	migration_versions	1	InnoDB
	motif	2	InnoDB
	service	1	InnoDB
	type_destinataire	2	InnoDB
	type_intervention	2	InnoDB
	type_message	3	InnoDB
	type_service	2	InnoDB
	user	3	InnoDB
information_schema			
mysql			
performance_schema			
sys			



f-Le **CRUD** (pour create, read, update, delete) :

Ici encore Symfony est là pour nous simplifier la vie !

La commande `php bin/console make:crud` (sur les entités créées) vas générer les Contrôleurs, les repositories, les forms et les vues associées avec le routing, nous permettant l'affichage et la création des items .

Exemple ici pour la table (entity) « motif » :

```
<?php

namespace App\Controller;

use ...

/**
 * @Route("/admin/motif")
 */
class MotifController extends AbstractController
{
    /**
     * @Route("/", name="motif_index", methods={"GET"})
     */
    public function index(MotifRepository $motifRepository): Response
    {
        return $this->render('admin/motif/index.html.twig', [
            'motifs' => $motifRepository->findAll(),
        ]);
    }

    /**
     * @Route("/new", name="motif_new", methods={"GET", "POST"})
     */
    public function new(Request $request): Response
    {
        $motif = new Motif();
        $form = $this->createForm('type: MotifType::class', $motif);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $entityManager = $this->getDoctrine()->getManager();
        }
    }
}
```

## g - En avant pour le formulaire d'inscription !

Comme dit en introduction, notre citoyen, destinataire des alertes, doit s'inscrire. Nous allons donc lui proposer de remplir un formulaire d'inscription.

Pour afficher et entrer notre nouveau destinataire en base de données lorsque celui-ci aura soumis son formulaire d'inscription, nous allons nous servir de l'item « new » de notre MVC destinataire.

Nous avons déjà pour notre entité « destinataires », le controller, le repository et les vues, ainsi qu'un « form » => DestinataireType.

Remplissons ce fichier avec les inputs que nous voulons voir apparaître dans notre formulaire.

```
class DestinataireType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $validator = new EmailValidator();
        $validator->isValid( email: "example@example.com", new RFCValidation());
        $builder

        ->add( child: 'nomDestinataire', type: TextType::class,[

            'label' => 'Nom',
            'constraints' => [
                new Assert\NotBlank([
                    'message' => 'Veuillez renseigner votre nom'
                ]),
                new Assert\Length([
                    'min' => 3,
                    'minMessage' => 'Votre nom doit contenir au minimum 3 caractères',
                    'max' => 50,
                    'maxMessage' => 'votre nom ne peut contenir que 50 caractères au maximum'
                ]),
            ],
        ],
    )
}
```

Un peu d'explications sur cet exemple concernant le nom de notre destinataire :

- nous entrons la « valeur » du champ qui correspond à la variable présente dans notre entité

```
private $adresseMailDestinataire;
```

puis ajoutons le type d'input que l'on souhaite voir entré, ici ce sera un nom donc du texte

```
type: TextType::class,[
```

entre les crochets qui suivent nous allons pouvoir modifier notre champ, par exemple lui donner un label pour l'affichage, (ou écrire un texte d'aide en dessus du champ avec la commande 'help'..) mais aussi ajouter des contraintes afin notamment, d'obliger notre destinataire à remplir ce champs ('NotBlank'), mais aussi de contraindre certains aspects comme la taille du texte. En effet nous



souhaitons obtenir son nom, mais une personne mal intentionnée ou un peu vicieuse pourrait ne rentrer qu'une seule lettre ou au contraire, insérer dans ce champs le contenu d'un page wikipédia ! Nous allons donc contraindre la taille du texte et ajouter un message d'aide :

```
new Assert\Length([
    'min' => 3,
    'minMessage' => 'Votre nom doit contenir au minimum 3 caractères',
    'max' => 50,
    'maxMessage' => 'votre nom ne peut contenir que 50 caractères au maximum'
])
```

(n'oublions pas d'entrer le « use » correspondant à la bonne exécution des ces contraintes :)

```
use Symfony\Component\Validator\Constraints as Assert;
```

Arrêtons-nous ici un instant car nous entrons dans un moment important de la sécurité de notre application et de sa base de données !!!!

Par ce formulaire nous permettons à notre destinataire d'envoyer des informations qui seront enregistrées dans notre base de données, il nous faut donc la protéger !

Pour l'instant nous n'avons fait que cadrer la taille du texte qui sera entré, il nous faut maintenant empêcher un utilisateur mal intentionné (et bien informé) d'entrer, par exemple, du code qui pourrait nous nuire (détruire la base de donnée ou en recueillir les informations) ce serait catastrophique.

Pour nous prémunir contre ce que l'on appelle « l'injection » de code il nous faut interdire la possibilité d'en insérer dans le champ. Le « code » étant composé notamment de sigles particuliers nous allons empêcher leur utilisation.

Direction notre entity destinataire (Destinataire.php)

```
/**
 * @ORM\Column(type="string", length=150, nullable=true)
 * @Assert\Regex(pattern="[a-zA-Z0-9,.\ \"'èèçàùï=;!~]+$/", message="Le champs ne doit pas contenir de caractères spéciaux ")
 */
private $nomDestinataire;
```

Nous ajoutons un 'regex' en @assert afin de n'autoriser que les caractères usuels à la composition d'un nom et par définition interdire tout les autres, Le regex permet également de cadrer la forme (par exemple un numéro de téléphone qui devra comporter 10 chiffres séparés par un espaces ou un tiret ou une barre et commençant obligatoirement par 06 ou 07 pour forcer à bien entrer un numéro de portable...).

S'en suis le message qui s'affichera en cas d'erreur.

Vérifions que cela fonctionne :

Nom

**ERREUR** Le champs ne doit pas contenir de caractères spéciaux

✖

Bien nous voilà protégés !

Reste à faire de même pour la totalité des champs qui composeront notre formulaire en fonction de nos attentes :

- le champs 'numéro dans la rue' ne pourra comporter qu'entre 1 et 5 chiffres suivi éventuellement d'un bis ou ter,
- le champs 'email' devra être au format xxxx@yy.zz(ou zzz)
- le champs 'téléphone portable' comme expliqué ci-dessus, devra être au format 06(ou 07) suivit de quatre blocs de deux chiffres séparés par espace ou - ou / (ou pas séparés du tout).

Toutes ces contraintes et restrictions font que nous aurons un formulaire qui sera à la fois aidant (avec des consignes pour aider notre destinataire à bien le remplir) et nous apportant un certain niveau de sécurité pour notre base de données.

Continuons d'aider et d'accompagner notre destinataire dans son inscription :

Il doit renseigner sa ville, hors dans notre application nous connaissons déjà toutes les villes pouvant être concernées => pourquoi pas lui proposer un menu déroulant ?

Retour sur notre DestinataireType pour un peu de code :

```
->add( child: 'idAdresse', type: EntityType::class,[
    'class' => Adresse::class,
    'label' => 'Ville',
    'query_builder' => function (AdresseRepository $er) {
        return $er->createQueryBuilder( alias: 't')
            ->orderBy( sort: 't.nomVille', order: 'ASC');
    },
    'choice_label' => 'nomVille',
])
```

(bien entendu nous avons préalablement rempli la table « adresse » de notre base avec les villes).

Nous appelons pour cela notre entity Adresse dans notre input et faisons une requête à la base de données pour nous afficher lors du click sur l'input toutes les villes ici rangées par ordre alphabétique.

Bien continuons.

Notre destinataire peut choisir de recevoir son alerte via mail ou sms, pourquoi ne pas pré-cocher la case « je souhaite recevoir par mail » lorsque celui renseigne son adresse mail ?

C'est parti pour un peu de JavaScript (Jquery) !

Tout d'abord il nous faut « linker » nos fichiers Js et css à la vue :

```
{% block stylesheets %}

    <script src="{ asset("js/inscription.js") }"></script>
    <link rel="stylesheet" href="{ asset("css/inscription.css") }">
{% endblock %}
```

Nous créons le fichier inscription.js dans public/js de notre dossier :

```
$(document).ready(function(){ //coche auto sur reception par mail
    $('#destinataire_adresseMailDestinataire').change(function() {
        let $inputs = $('#destinataire_adresseMailDestinataire'), $('#destinataire_okMailDestinataire');

        if ($inputs[0].val() != "") {
            $inputs[1].attr( "checked", "checked" )
        }
    });
});
```

Voilà la case se coche toute seule.

Bien sur il faut essayer de tout prévoir (par exemple si la personne efface ensuite son adresse mail => décoche automatique), lui donner la possibilité de ne fournir que mail ou sms mais l'obliger à en entrer au moins un....

Dernière chose, il faut que notre destinataire accepte les CGU, bien qu'on ne peut s'assurer qu'il les lisent il nous faut son accord. Pour cela nous ajoutons une case et un peu de texte d'aide:

☐ J'accepte les conditions d'utilisation

(Vous devez accepter les conditions générales pour pouvoir valider votre inscription.)

et faisons en sorte que le bouton d'envoi du formulaire reste caché tant que la case n'est pas cochée (ce qui fera également disparaître le texte) :

☒ J'accepte les conditions d'utilisation

Valider

Bien, un peu de style (css) et nous voilà avec un beau formulaire dynamique et sécurisé !

Passons au côté « back » car c'est bien d'avoir un beau formulaire mais il doit nous servir à recueillir les informations souhaitées et les enregistrer dans notre base de données.

Direction notre controller dans sa partie « new » et ajoutons un peu de code pour traiter ces données selon nos besoins :

```
public function new(Request $request, ValidationRepository $validationRepository): Respo
{
    $destinataire = new Destinataire();
    $form = $this->createForm( type: DestinataireType::class, $destinataire);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();

        $date = new \DateTime( time: 'now');
        $date->setTimezone(new \DateTimeZone( timezone: 'Europe/Paris'));
        $destinataire -> setDateEnregistrementDestinataire($date);
        $validation = $validationRepository->find( id: 1);
        $destinataire->setIdValidation($validation);

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

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

    return $this->render( view: 'destinataire/new.html.twig', [
        'destinataire' => $destinataire,
        'form' => $form->createView(),
    ]);
}
```

En complément du code déjà présent grâce au CRUD, permettant l'enregistrement en BDD, nous ajoutons quelques lignes afin d'enregistrer la date d'enregistrement et de donner au statut de validation de notre souscripteur la valeur 1 qui correspond à « en attente de validation ».

## h -Attaquons la partie administration :

Notre destinataire peut s'inscrire et nous recueillons les données nécessaires pour lui envoyer son alerte.

Occupons-nous donc de la partie administration qui comprendra les outils de gestion des données (validations des souscripteurs, gestion des admins, gestion des différents items contenus dans les messages d'alertes....) et la création des alertes.

Cette partie ne doit être visible QUE pour les administrateurs enregistrés, il nous faut donc :

- un système de log pour les administrateurs
- cacher les liens en fonction des rôles
- protéger les urls

Retour dans la console de Symfony :

- Make:user (puis update de la bdd)
- Make:auth
- make:registration\_form

Symfony nous a créé tout ce dont nous avons besoin, reste quelques réglages à faire/vérifier :

- L'encodage du mot de passe  
dans config->packages->security.yaml

```
encoders:
    App\Entity\User:
        algorithm: bcrypt
```

dans le controller

```
$passwordEncoded = $encoder->encodePassword($user, $user->getPassword());
$user->setPassword($passwordEncoded);
```

- La définition des rôles : ici nous aurons besoin du `ROLE_SUPER_ADMIN` et du `ROLE_ADMIN`. Les deux auront accès aux pages de gestion des ressources et de création des alertes, le super admin sera seul autorisé à créer de nouveaux administrateurs. Nous créons donc en premier un super admin puis réglons les rôles à admin pour chaque nouvelle création :

Dans le UserController :

```
public function new(Request $request, UserPasswordEncoderInterface $encoder): Response
{
    $user = new User();
    $form = $this->createForm(type: UserType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $user->setRoles(['ROLE_ADMIN']);
        $entityManager->persist($user);
        $entityManager->flush();
    }

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

- Nous ajoutons un bouton admin dans notre barre de navigation qui amène sur la page de login :

Connectez-vous

Email

Mot de passe


Valider

nous avons ici choisi d'avoir l'Email unique comme login, et une fois connecté

### Security Token

torkhan@wanadoo.fr

Username



Authenticated

Property	Value
Roles	<div>[ "ROLE_SUPER_ADMIN" "ROLE_USER" ]</div>
Inherited Roles	<div>[ "ROLE_ADMIN" "ROLE_ALLOWED_TO_SWITCH" ]</div>

apparaît le lien vers la page de gestion de notre application et le bouton admin se change en déconnexion (logg\_out)

Recevoir les alertes !

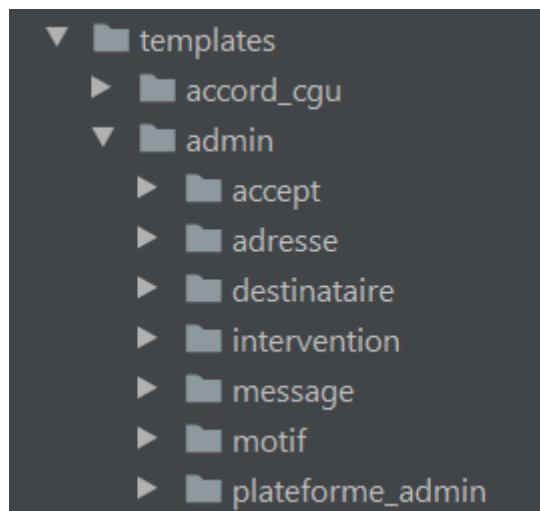
Plateforme de gestion

Deconnexion

grâce à notre twig

```
{% if is_granted('ROLE_ADMIN', 'ROLE_USER') %}  
    <a class=" ml-auto nav-item nav-link" href="{{ path('plateforme_admin') }}">Plateforme de gestion</a>  
    <a class=" mx-auto nav-item nav-link text-right" href="{{ path('app_logout') }}">Deconnexion</a>  
{% else %}
```

- N'oublions pas notre visiteurs mal intentionné et bien informé qui voudrait taper directement les urls sensibles, nous allons les protéger en créant un dossier admin dans



le « templates » et dans lequel nous glissons toutes les vues à protéger (bien sur il faudra modifier toutes les routes concernées dans les vues et controllers) et dans notre security,yaml :

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

Voilà nos urls sont protégées et si l'on tape admin dans l'adresse cela nous renvoi sur le log\_in.

La plateforme de gestion :

**Bonjour Leclercq Cédric!**

**Bienvenue sur votre palteforme de gestion**

Vous pourrez ici créer votre alerte et l'envoyer aux souscripteurs concernés, valider les demandes de soucription et gérer l'intégralité des ressources

### 1-Gérer les demandes de souscription

Page demande de souscription

### 2-Préparer votre alerte

Pour créer ou vérifier vos titres de messages

1-Gestion des titres de message

Pour créer ou vérifier les villes concernées

3-Gestion des villes

Pour créer ou vérifier vos types d'interventions

2-Gestion des types d'intervention

Pour créer ou vérifier le service concerné

4-Gestion des services

### 3-Créer votre intervention

Gestion des interventions

Nous avons réalisé cette page de manière à ce qu'elle soit le plus intuitive et facile d'utilisation possible, en segmentant les items . Je vais ici en expliquer certains :

-1 la gestion des demandes de souscription :

Comme vu plus haut notre admin doit valider (ou refuser) les demandes, nous allons donc lui faciliter la tâche en les regroupant dans un tableau où chacune pourra être validée séparément mais aussi toutes à la fois. Toujours dans un but de de lui rendre la vie plus simple, l'admin pourra cliquer sur les titres du tableau pour effectuer un tri par ordre alphabétique (réalisé en JQuery).



## Accepter/refuser les destinataires

Vous trouverez ici les destinataires ayant fait leur demande de souscription

Vous pouvez les accepter ou les refuser

Un Email indiquant votre choix leur sera automatiquement envoyé

Nom ↓	Prénom ↓	Numéro rue ↓	Adresse ↓	Complément adresse	Code Postal ↓	Ville ↓	Select	
toto	toto	952	rue de l'espérance		59119	Waziers	Valider	Refuser
Durand	roger	952	rue de l'espérance		59194	Râches	Valider	Refuser
Durand	roger	952	rue de l'espérance		59500	Douai	Valider	Refuser
Leclercq	df3	38	rue des alouettes		59151	Arleux	Valider	Refuser
Leclercq	testmail	92	rue du paradis		59151	Arleux	Valider	Refuser
sfsf	sfddfsdfsdfsdfsdfsdf	92	rue du paradis		59151	Arleux	Valider	Refuser
rzerzerzer	zrzerzerze	92	sdfsdf		59151	Arleux	Valider	Refuser

[Retour plateforme](#)[Valider tous](#)[Refuser tous](#)

Un peu de code pour expliquer ceci :

Dans un premier temps nous créons un AcceptController qui vas gérer tout cela et nous allons créer une fonction qui vas appeler dans notre Bdd les destinataires qui sont « en attente » de validation :

```
/**
 * @Route("/accept", name="accept")
 */
public function index(Request $request, DestinataireRepository $destinataireRepository, ValidationRepository $validationRepository)
{
    $formAccept = $this->createForm( type: DestinataireType::class);
    $formAccept->handleRequest($request);
    $results = $destinataireRepository->findOneBySomeField($validationRepository->find(id: 1));
    $destinataires = $destinataireRepository->findAll();

    return $this->render( view: 'admin/accept/index.html.twig', [
        'controller_name' => 'acceptController',
        'form' => $formAccept->createView(),
        'results' => $results,
        'destinataires' => $destinataires
    ]);
}
```

Ceci nous affiche notre tableau dans la vue twig.

Ensuite donnons vie aux boutons valider, refuser, valider tous, refuser tous (il s'agit simplement de modifier le statut de validation en récupérant l'id correspondante :

```
/**
 * @Route("utilisateur/{id}/refuser", name="refuser_destinataire")
 * @param Destinataire $destinataire
 * @return RedirectResponse
 * @throws Exception
 */
public function refuserDestinataire($id, Request $request, Destinataire $destinataire, ValidationRepository $validationRepository)
{
    $destinataire = $this->getDoctrine()->getRepository('persistentObject:Destinataire::class')->find($id);
    $date = new \DateTime('now');
    $date->setTimezone(new \DateTimeZone('Europe/Paris'));
    $destinataire->setDateModificationDestinataire($date);
    $destinataire->setDateValidationDestinataire($date);
    $validation = $validationRepository->find(3);
    $destinataire->setIdValidation($validation);

    $nom = $destinataire->getNomDestinataire();
    $prenom = $destinataire->getPrenomDestinataire();
    $email = $destinataire->getAdresseMailDestinataire();
    $messageEnvoye = 'Votre demande de souscription à été refusée';
}
```

Nous renvoyons sur la même page ce qui donne une impression de « disparition » des lignes concernées.

Un mail automatique est également envoyé à notre destinataire pour l'informer que sa demande de souscription a été traitée. Pour cela nous installons le bundle Swiftmailer :

```
> composer require symfony/swiftmailer-bundle
```

Un peu de paramétrage en suivant la documentation et c'est assez rapidement fonctionnel.

Je passe les détails des autres items de gestion qui sont là pour créer les paramètres de l'alerte (le service concerné, le type de travaux etc..) pour passer directement à ce qui est le cœur de notre application => l'envoi de l'alerte !

-2 l'envoi d'alerte :

## 5-Envoyer votre alerte

Envoyer l'alerte

Le bouton nous amène sur la page d'envoi où l'administrateur vas pouvoir choisir son alerte à envoyer et à qui. Nous devons donc lui donner la possibilité de ces différents choix :

D'abord à qui ?

Par définition une alerte de travaux ne vas concerner que certains habitants d'une rue dans une ville. Nous offrons donc à notre administrateur deux manières de rechercher, soit par un tri dans le tableaux des destinataires soit plus simplement en implémentant une recherche :

Nom

Nom de la rue

Ville

Valider

Flers-en-Escrebieux ▼

[Liste complète](#)

### Sélectionnez les destinataires

Nom ↓	Prénom ↓	Numéro rue ↓	Adresse ↓	Complément adresse	Code Postal ↓	Ville ↓
garou	gazrou	81	barbusse		59128	Flers-en-Escrebieux <input type="checkbox"/>
garou	garou	81	barbusse		59128	Flers-en-Escrebieux <input type="checkbox"/>
garou	garou	81	barbusse		59128	Flers-en-Escrebieux <input type="checkbox"/>
garou	garou	81	barbusse		59128	Flers-en-Escrebieux <input type="checkbox"/>
Leclercq	Coleen	92	des tests		59128	Flers-en-Escrebieux <input type="checkbox"/>
Leclercq	david	952	rue des alouettes		59128	Flers-en-Escrebieux <input type="checkbox"/>

Tous ☐

Ici avec une recherche par ville. L'administrateur vas pouvoir sélectionner un habitant ou tous via les cases à cocher. Un peu de jquery por détecter les coches et rendre opérationnelle la case tous :

```

//fonction de select all checkboxes
$('.checkAll').click(function(){
    if (this.checked) {
        $(".checkboxes").prop("checked", true);
    } else {
        $(".checkboxes").prop("checked", false);
    }
});
$(".checkboxes").click(function(){
    let nombreDeCheckboxes = $(".checkboxes").length;
    if(nombreDeCheckboxes == nombreDeCheckboxesChecked) {
        $(".checkAll").prop("checked", true);
    } else {
        $(".checkAll").prop("checked", false);
    }
});

```

Et du côté controller :

```
* @Route("/admin/recherche", name="recherche")
* @return Response
*/
public function index(Request $request, InterventionRepository $interventionRepository, MessageRepository $messageRepository)
{
    $recherche = new Destinataire();
    $messages = $messageRepository->getMessages();

    $formRecherche = $this->createForm( type: RechercheType::class, $recherche);
    $formRecherche->handleRequest($request);

    $nomDestinataire = $recherche->getNomDestinataire();
    $rueDestinataire = $recherche->getNomRueDestinataire1();
    $villeDestinataire = $recherche->getIdAdresse();

    $results = $destinataireRepository->RechercheDestinataire($nomDestinataire, $rueDestinataire, $villeDestinataire);

    return $this->render( view: 'admin/recherche/index.html.twig', [
        'controller_name' => 'RechercheController',
        'form' => $formRecherche->createView(),
        'results' => $results,
        'messages' => $messages ,
    ]);
}
```

Ceci fait, un menu déroulant pour sélectionner l'alerte et le bouton envoi :

**Selectionnez votre alerte**

Travaux de voiries - Dératisation ▼

**Envoyez l'alerte**

Envoyer

Retour plateforme

De notre côté beaucoup de chose à gérer pour que cela soit fonctionnel.

Il nous faut « capter » les destinataires cochés pour pouvoir obtenir leurs informations depuis la Bdd : un peu d'ajax s'impose !

```

function envoyer(e) {
    e.preventDefault();
    let idsDestinataires = [];
    let nombreDeCheckboxesChecked = document.querySelectorAll( selectors: '.checkboxes:checked');
    nombreDeCheckboxesChecked.forEach( callbackfn: function (data :Element) {
        let ids = data.value;
        idsDestinataires.push(ids);
    });
    let donneeErreur = document.querySelector( selectors: "#listeVide");

    if(idsDestinataires.length == 0){
        donneeErreur.classList.remove( tokens: 'd-none');
    }else{
        donneeErreur.classList.add('d-none');
        let idMessage = document.querySelector( selectors: "#selectMessage").value;

        $.ajax({
            url: "envoyerRecherche",
            method: 'POST',
            data: {
                "idUsers": idsDestinataires,
                "idMessage": idMessage
            },
            success: function (data) {

                if(data.envoye === 'ok'){
                    let donneeSuccess = document.querySelector( selectors: "#success");
                    donneeSuccess.classList.remove( tokens: 'd-none');
                }else{
                    let donneeErreur = document.querySelector( selectors: "#danger");
                    donneeErreur.classList.remove( tokens: 'd-none');
                }
            }
        });
    }
}

```

Qui renvoi sur notre controller :

```

function envoyerRecherche(Request $request,
    DestinataireRepository $destinataireRepository,
    Swift_Mailer $mailer,
    MessageRepository $messageRepository):Response{
    // récupération des données de la page recherche (les ids)
    $idUsers = $request->request->get( key: 'idUsers');
    // transformation du string vers tableau
    $listeDestinataires = array();

    foreach ($idUsers as $key=>$idUtilisateur) {
        $coordonneeDestinataires = $destinataireRepository->getDestinataires(intval($idUtilisateur));

        array_push( &array: $listeDestinataires, $coordonneeDestinataires);
    }
}

```

Ensuite, toujours dans le controller, il nous faut récupérer les informations de l'alerte sélectionnée :

```
// récupération des données de la page recherche (le message sélectionné)
$idMessage = $request->request->get( key: 'idMessage');

$contentMessage = $messageRepository->getContentMessage(intval($idMessage));

// recup message
$messageRecup = $contentMessage[0]->getContenuMessage();
$messageDateEnvoie = $contentMessage[0]->getDateEnvoi();//, "d-M-Y");
$dateformatEnvoie = date_format($messageDateEnvoie, format: "d-m-Y");

// recup type message
$messageTypeMessage = $contentMessage[0]->getIdTypeMessage()->getMessageType();

// recup info intervention
$nomIntervention = $contentMessage[0]->getIdIntervention()->getNomIntervention();
$villeIntervention = $contentMessage[0]->getIdIntervention()->getVilleIntervention();
$rueIntervention = $contentMessage[0]->getIdIntervention()->getRueIntervention();
$dateDebutIntervention = date_format($contentMessage[0]->getIdIntervention()->getDateDebutIntervention(), format: "d-m-Y");
$dateFinIntervention = date_format($contentMessage[0]->getIdIntervention()->getDateFinIntervention(), format: "d-m-Y");
```

Puis créer et envoyer les messages pour chaque destinataire sélectionné :

```
// corps du message
$corpsMessage= "";
$corpsMessage = "<h2>".$nomIntervention."</h2>";
$corpsMessage .= "<p>Date debut de l'intervention : ".$dateDebutIntervention."</p>";
$corpsMessage .= "<p>Date de fin de l'intervention : ".$dateFinIntervention."</p>";
$corpsMessage .= "<p>Le type d'intervention concerne ".$messageTypeMessage."</p>";
$corpsMessage .= "<p>Attention ce message vous concerne</p><p>".$messageRecup."</p><p>Date d'envoi du message : ".$dateformatEnvoie."</p>";

foreach ($listeDestinataires as $destinataires) {
    foreach ($destinataires as $key=>$destinataire) {
        //envoi des messages

        $message = (new \Swift_Message($nomIntervention))
            ->setFrom( addresses: 'xxxxx@gmail.com')
            ->setTo($destinataire['adresseMailDestinataire'])
            ->setBody($corpsMessage, contentType: 'text/html', charset: 'utf-8');
        $mailer->send($message);
    }
}

return $this->json([
    "code" => 200,
    "envoye" => "ok"
], status: 200);
```



## 6- Et si on testait tout ça ?:

a - Je rempli un formulaire d'inscription :

Nom

Test

Prénom

Complet

b - Vérification de son entrée en Bdd :

iter	Copier	Supprimer	123	1	1	18	NULL	Complet	Test
------	--------	-----------	-----	---	---	----	------	---------	------

c - Connexion administrateur :

### Connectez-vous

Email

xxxxx|@wanadoo.fr

Mot de passe

.....

d - Arrivée sur la plate forme d'administration :

### Bonjour Leclercq Cédric!

### Bienvenue sur votre palteforme de gestion

Vous pourrez ici créer votre alerte et l'envoyer aux souscripteurs concernés, valider les demandes de soucription et gérer l'intégralité des ressources

e - Validation de la nouvelle demande de souscription :

elle apparaît bien

Nom	Prénom	Numéro	Adresse	Complément	Code	Ville	Select
↓	↓	rue ↓	↓	adresse	Postal ↓	↓	
Test	Complet	92	rue du paradis		59247	Féchain	<input type="button" value="Valider"/> <input type="button" value="Refuser"/>

Je la valide

Nom	Prénom	Numéro	rue	Adresse	Complément	Code	Postal	Ville	Select
↓	↓	↓	↓	↓	adresse	↓	↓	↓	
Aucun destinaire ne correspond à votre recherche									

et vérifie dans la Bdd que son id validation est passée à « 2 »

Copier	Supprimer	123	1	2	18	NULL	Complet	Test
--------	-----------	-----	---	---	----	------	---------	------

Réception du mail de confirmation





f – sélection de l'alerte:

Nom	Nom de la rue	Ville	Valider
<input type="text"/>	<input type="text"/>	Féchain	

[Liste complète](#)

### Sélectionnez les destinataires

Nom ↓	Prénom ↓	Numéro rue ↓	Adresse ↓	Complément adresse	Code Postal ↓	Ville ↓
Test	Complet	92	rue du paradis		59247	Féchain <input checked="" type="checkbox"/>
						Tous <input type="checkbox"/>

### Selectionnez votre alerte

Coupure de courant - test

### Envoyez l'alerte

Envoyer

g - Envoi du mail

### Envoyez l'alerte

Envoyer

[Retour plateforme](#)

Le message a bien été envoyé

Request / Response

Performance

Validator

Forms

Exception

Logs

Events

Routing

Cache

Translation

Security

Twig

HTTP Client

Doctrine

E-mails

1

spoiled message

E-mail details

Subject

test

From

test@localhost.com

To

test@localhost.fr

Headers

Content-Transfer-Encoding: quoted-printable  
Content-Type: text/html; charset=utf-8  
MIME-Version: 1.0  
Date: Wed, 20 May 2020 09:21:37 +0000  
Message-ID: <a020af5a47773a8627191fd157726f1e@localhost>

Raw content

Rendered content

<h2>test</h2><p>Date debut de l'intervention : 06-May-2020</p><p>Date de fin de l'intervention : 07-May-2020</p><p>Le type d'intervention concerne Test</p><p>Attention ce message vous concerne</p><p></p><p>Date d'envoi du message : 05-05-2020</p>

Tout cela Fonctionne parfaitement !!

## 7- Exemple de recherche :

J'ai eu à effectuer de nombreuses recherches tant au niveau des documentations officielles que sur des pages et forum d'aide.

Pour en citer une pour laquelle j'ai rapidement trouvé ma solution :

J'avais un soucis avec l'envoi de mails qui partaient bien de Symfony mais n'arrivaient pas dans ma boîte mail. J'ai donc fait une recherche avec les termes suivants :

« swiftmailer symfony mail send but not received »

et j'ai rapidement trouvé la solution due à ma mauvaise configuration Swiftmailer via Gmail en transport, sur <https://stackoverflow.com/>

5 Answers

Active Oldest Votes

▲

14

▼

🕒


Problem solved!! Put this configuration instead of using the URL env variable

```
swiftmailer:
  transport:      gmail
  username:       ****@gmail.com
  password:       *****
  host:           localhost
  port:           465
  encryption:     ssl
  auth-mode:      login
  spool: { type: 'memory' }
  stream_options:
    ssl:
      allow_self_signed: true
      verify_peer: false
      verify_peer_name: false
```

share improve this answer follow

edited Mar 14 '18 at 20:20

answered Mar 6 '18 at 18:28

 Salve Safari

254 ● 1 ● 3 ● 10

1

Merci! That made the trick. Any idea why the .env file not working ? – Miles M. Jun 4 '18 at 1:49

Ici le questionneur se répondait à lui-même et diffusait la solution qu'il avait trouvé ailleurs.

« Problème résolu ! Mettez cette configuration au lieu d'utiliser la variable URL dans le .env »

## **8- Pour conclure et à suivre :**

Mon tuteur m'a proposé ici de participer à une belle application, qui de plus aura vocation à être publiquement utile. D'un point de vue technique, elle reprend et permet de mettre en pratique ce que nous avons vu durant ces neuf mois de formation.

Elle n'est pas terminée, loin s'en faut, et je vais continuer d'y travailler avec grand plaisir.

Par exemple, et après accord du client, il faudra implémenter la fonction Sms ; si les champs utilisés par le destinataire sont protégés ceux utilisés par les administrateurs ne le sont pas tous encore ; retravailler le corps du mail (ou du sms) pour le rendre plus « agréable »...

Cette « V1 » doit être soumise au client, nous attendons son retour.

D'un point de vue personnel :

J'ai décidé il y a deux ans de changer de voie, de me ré-orienter professionnellement et ce à 47 ans. Je suis ravi de l'avoir fait.

Et ce n'est que le début de l'aventure....

Cédric LECLERCQ

DWWM  
AFPA GRP 4

Nb : Vous ne trouverez pas ici de remerciements écrits : toutes les personnes que je souhaitais remercier pour leur engagement auprès de nous, leurs transmissions et leur suivi l'ont été de vive voix.

# **Annexes**

1 - Cahier des charges

2 – Étude sur l'utilisation de Twilio pour l'envoi de Sms (read-me rédigé par Noémie Hoffman).