1. Liste de compétence du référentiel couvertes par le projet

Liste de compétences

Ce projet illustre les différents points des compétences du référentiels :

Développer la partie Front-end d'une application web ou web mobile en enterrant les recommandations de sécurité :

1. Maquetter une application
   - Création d'une maquette fonctionnelle en amont du projet.

2. Réaliser une interface utilisateur web statique et adaptable
   - Mise en page avec Twig avec des rédaction HTML et CSS

3. Développer une interface utilisateur web dynamique
   - Mise en place des formulaires

Developer la parties back-end d'une application web, web mobile

1. Créer une base de données
   - Création de base de données avec MySql

2. Développer les composants d'accès au donné
   - Création du CRUD à partir des entités

3. Développer la partie back-end et administratif d'une application web web mobile

2.Résumé du projet

Contexte de réalisation

MOGGAR est une entreprise d'impression papier basée a bordeaux elle propose une impression de tous types de papier, elle offre des différentes détaille, coulure, résolution, ... Afin d'avoir plusieurs rendus possibles.

L'Imprimerie ne possède pas un site web c'est donc son premier site que je dois concevoir

Piste technique est fonctionnelles

Le but du projet consiste à concevoir un site web de nom de domaine (www.moggar.fr) permettant à l'entreprise MOGGAR deux choses :

1. D'avoir une vitrine proposant aux clients toutes les informations commerciales sur l'entreprise

2. Possibilité pour le client de faire un devis en ligne. Ce devis est enregistré dans une base de données et envoyé à l'imprimerie par mail.

Le site devrait être responsive et administrable.

Pour la gestion de données il faudrait donc que le site internet sois lié directement a une base de données.

Le projet est réalisé avec le Framework Symfony version 4.3 qui gère trais bien la gestion de donnée avec doctrine ORM et la possibilité d'intégrer des <<Bundle>> au Framework avec des version stable

Le site internet est structuré en 3 stacks Front/Back/API

La stack Symfony sera le stack principal de site pour l'administrateur et aussi pour le visiteur.

La stack Api a une grande importance sur le formulaire de devis qui fais son imbrication, les envois des formulaires, les requête des barres de recherches en AJAX.

Tout le projet a été développé sous Linux avec les logiciels (adobe XD, Balsamiq Wireframes, adobe Photoshop), éditeur de texte PHP Storm / Visual studio code, langage utiliser html ccs JavaScript/jQuery PHP, Framework Symfony, Bootstrap. Image libre de droit https://www.unsplash.com , phpMyAdmin, git/GitHub

Une fois le site internet crée il sera mis avec l'accord de l'entreprise en ligne sur un serveur 1&1 IONOS

3. cahier de charges technique du projet

Vous trouverez en annexe le cahier des charges technique et fonctionnelles le wireframe, les maquette du site internet

4. spécifications techniques du projet

Le site internet est lié à une base de données (BDD), elle permettra à recueillir des donné administres et aussi à conserver les messages et les devis envoyer par les utilisateurs

Le Framework Symfony gère parfaitement la BDD avec Doctrine (ORM) qui permet la création d'intenté et le mapping entre les objets et les éléments de la base de données avec une grande facilité et une gagne du temps

Aussi, Symfony prend les mesures de sécurité contre les injections SQL, attaque XSS, CSRF.

La plus grande fonctionnalité du site internet est la gestion d'élément administrable : la gestion des produits des types de papier à imprimer leur prix et tous les options supplémentaires avec l'uploade des images mise en avent, il pourra ajouter, modifier, lire et supprimer ses éléments.

Il est nécessaire que l'Admin soit connecté afin de pouvoir manipuler ses éléments

Le bundle FOS USER est mis en place pour l'interface de connexion, le mot de passe est enregistré en Sha1 sur la BDD pour la sécurité.

Lors de la création ou la modification d'un type de papier, un formulaire apparait avec diffèrent champ inputs :

L'administration de type de papier :

- Titre (Name): input type text
- Description: input type text
- Prix: input type number
- Format: input type checkbox
- Propriété : input type checkbox
- Media: input type checkbox
- Vernis: input type checkbox
- Quadrichromie: input type checkbox
- Orientation: input type checkbox

    - Optional:
- Page couverture: input type checkbox
- Papier couverture : input type checkbox
- Propriete contenu: input type checkbox
- Propriete couverture: input type checkbox
- Propriete papier contenu : input type checkbox
- Type de reliure : input type checkbox
- Vernis couverture : input type checkbox
- Image: input type file

L'administration de Format :

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L'administration de Propriété :

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L'administration de Media :

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L'administration de Vernis:

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L’administration de Quadrichromie:

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L’administration de Orientation :

- Titre (Name): input type text
- Description: input type text
- Prix: input type number

L’administration de Page couverture :

- Titre (Name): input type text

L’administration de Papier couverture :

- Titre (Name): input type text

L’administration de Propriete contenu:

- Titre (Name): input type text

L’administration Propriete couverture :

- Titre (Name): input type text

L’administration de Type de reliure :

- Titre (Name): input type text

L’administration de Vernis couverture :

- Titre (Name): input type text

L’administration des messages :

Les utilisateurs ont la possibilité d’envoyer un message a l’entreprise via in formulaire de contacte l’administrateur a le pouvoir de les lire et les supprimer. Les messages son envoyer par mail aussi

Vous trouverez en page du cahier de charge technique et fonctionnel d’information en ce qui concerne la gestion des messages

L’administration des devis :

Les devis envoyer son directement enregistrer sur la base de données et envoyer par mail avec tous les information, l’administrateur a le pouvoir de les retrouver sur le site internet et les supprimer

Le client souhaite qu’une barre de recherche sois à sa disposition pour effectuer des recherches de devis par nom ou prénom

Vous trouverez en page du cahier de charge technique et fonctionnel d’information en ce qui concerne la gestion des devis

5. *Réalisation comportant les extraits de code les plus significatif*

Contexte

Afin que les utilisateurs puissent envoyer un devis je devais mettre en place un formulaire avec des champ input imbriqué, avec un uploade de fichier et l'envois en ajax

Une fois l'envois réussie un mail est envoyé à l'admin avec les choix de l'utilisateur comme contenue

Ainsi, l'administrateur de son côté pourrait effectuer des recherches par nom ou prénom, lire et aussi la possibilité de supprimer les devis

Création de DEVIS

Avent tout J'ai créé une base de données <<moggar>> sur PhpMyAdmin le projet est lié à cette BDD

Sur le fichier << .env >> de Symfony (serveur version 5.2)

J'ai créé une entité DEVIS qui est composé d'attribut :

- Id
- Nom
- Prenom
- Email
- Phone
- Path
- Img-path
- Created_at

Je rajoute l'entité TYPE. Cette entité est réservée aux créations de type de papier est composé d'attribut :

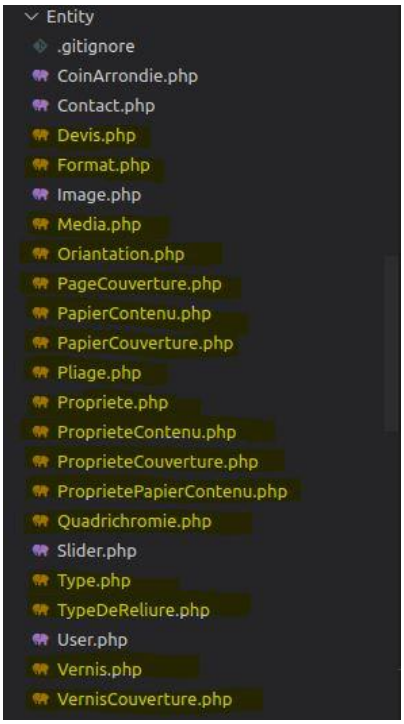
- Id
- Name
- Description
- Prix

Cette entité grâce aux annotations de l'ORM doctrine est en relation avec l'entité devis en MANY TO ONE.

```php
/**
 * @ORM\Entity(repositoryClass="App\Repository\DevisRepository")
 */
class Devis
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

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

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

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

```php
/**
 * @ORM\ManyToOne(targetEntity="App\Entity\Type")
 */
private $type;
```

Aussi j'ai créé des entités pour toutes les options possibles telle que format, orientation, quadrichromie...



La plupart de ces entités ont les mêmes attributs

- Id
- Name
- Description
- Prix

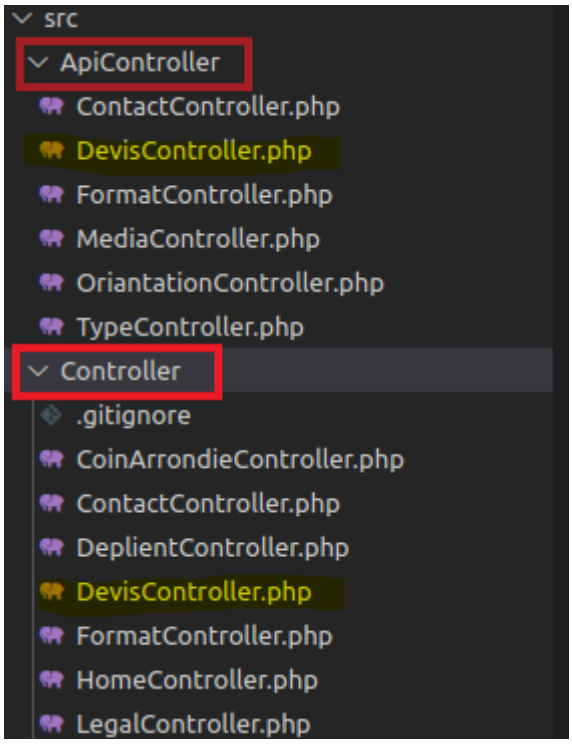
Ces entités sont en relation avec l'entité <<DEVIS>> en MANY TO ONE

Et en relation avec l'entité <<TYPE>> en MANY TO MANY

Vous trouverez schéma relationnel base de données sur la page

Pour faire les requête ajax j'ai mis en place le FOS rest bundle pour la Platform api

La stack api a sa propre HOST << api.moggar.fr/>>



Configuration de la route

Le host, resource et type pour les API Controller ont été paramétrés.

config > routes > ! annotations.yaml

```yaml
controllers:
    host: moggar.fr
    resource: ../../src/Controller/
    type: annotation
api_auth_login:
    host: api.moggar.fr
    path: /login
    methods: [POST]
api_controllers:
    host: api.moggar.fr
    resource: ../../src/ApiController/
    type: annotation

```

Api Davis Controller Namespace :

src > ApiController > ContactController.php

```php
<?php

namespace App\ApiController;


```

Host:

```php
/**
 * @Route("/devis", host="api.moggar.fr")
 */
```

Le formulaire est créé depuis le DevisType.php avec formBuiderInterface de symfony :

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('nom')
        ->add('prenom')
        ->add('email', EmailType::class)
        ->add('phone' ,TelType::class)
        ->add('type')
        ->add('media')
        ->add('format')
        ->add('vernis')
        ->add('oriantation')
        ->add('quadrichromie')
        ->add('propriete')
        ->add('pageCouverture')
        ->add('papierContenu')
        ->add('papierCouverture')
        ->add('proprieteCouverture')
        ->add('proprietePapier')
        ->add('typeDeReliure')
        ->add('vernisCouverture')
        ->add('proprieterContenue')

        ->add('file', FileType::class, array(
          'label' => 'image',
          'required' => false
        ))
    ;
}
```

Et insrer sur DevisController.php

```php
$devis = new Devis();
$form = $this->createForm(DevisType::class, $devis);
```

Ajouter des données

Dans une API REST**, l'**ajout de données doit passer par la **méthode HTTP POST** et retourner un code de réponse HTTP 201 "Created" lorsque l'ajout a fonctionné.

La méthode ajax() est utilisée pour exécuter une requête AJAX (HTTP asynchrone). data type JSON

Code JS :

```
$.ajax({
  url: "http://api.moggar.fr/devis/create",
  data: new FormData(form),
  type: "POST",
  contentType: false,
  processData: false,
  cache: false,
  dataType: "json",
  error: function (err) {
  $('#ERR_Form').text('formulaire non valide').css('color','red')
  },
  success: function (data) {
    $('form').hide();
    $('#success').val();
    $('#ERR_Form').text('formulaire envoyer ^^ ').css('color','green')
  },
  complete: function () {
    console.log("Request finished.");
  }
});
e.preventDefault();

});
```

Une fois la requête ajax réussi la fonction create et appeler sur DevisController.php :

```php
/**
 * @Rest\Post(
 *     path="/create",
 *     name="api_moggar_devis_created"
 * )
 */
public function create(
  Request $request,
  FormatRepository $formatDocumentRepository,
  MediaRepository $mediaRepository,
  OriantationRepository $oriantationRepository,
  ProprieteRepository $proprieteRepository,
  QuadrichromieRepository $quadrichromieRepository,
  TypeRepository $typeDocumentRepository,
  VernisRepository $vernisRepository,
  \Swift_Mailer $mailer
): View

{
  $doc = new Devis();
  $dataFile = $request->files->get('devis');
  $file = $dataFile['file'];

  $data = $request->request->get('devis');
  $nom = $data['nom'];


  $prenom = $data['prenom'];
```

Cette fonction contient tous les repository associer à l'entité devis et celle qui apparaissent sur les choix de formulaire

La variable **$data** récupère la requête de formulaire devis et récupère les valeurs des champ inputs

Example :

La valeur nom est récupérer et stocké sur la variable $nom

```php
$data = $request->request->get('devis');
$nom = $data['nom'];
```

Je vérifie si la valeur existe et si elle n'est pas vide et si la valeur et de type string

Et j'insert la valeur avec la fonction setters de l'entité DEVIS <<setNom>> et question de sécurité avec la fonction htmlspecialchars pour éviter les injection html, JS.

Sinon j'envoie une raiponce HTTP EXPECTATION FAILED avec le <<nom non valide>>

```php
if (isset($nom) && !empty($nom) && is_string($nom)) {
  $doc->setNom(htmlspecialchars($nom));
} else {
  return View::create('le nom envoye n\'est pas valide', Response::HTTP_EXPECTATION_FAILED);
}
```

Example 2 :

La valeur format

Il s'agit d'une valeur id récupérer depuis le formulaire care elle est en relation avec l'entité devis MANY TO ONE

Je récupère le repository de l'entité format

```
use App\Entity\Devis;
use App\Repository\DevisRepository;
use App\Repository\FormatRepository;
use App\Repository\ImageRepository;
use App\Repository\MediaRepository;
use App\Repository\OriantationRepository;
use App\Repository\ProprieteRepository;
use App\Repository\QuadrichromieRepository;
use App\Repository\TypeRepository;
use App\Repository\VernisRepository;
```

```
public function create(
  Request $request,
  FormatRepository $formatDocumentRepository,
  MediaRepository $mediaRepository,
  OriantationRepository $oriantationRepository,
  ProprieteRepository $proprieteRepository,
  QuadrichromieRepository $quadrichromieRepository,
  TypeRepository $typeDocumentRepository,
  VernisRepository $vernisRepository,
  \Swift_Mailer $mailer
): View

{
  $doc = new Devis();
  $dataFile = $request->files->get('devis');
```

La valeur format du formulaire est récupérer et stocké sur la variable $formatVal

La méthode find ($id) récupère l'entité correspondant à l'id et stocké sur la variable $format

```
$formatVal = $data['format'];
$format = $formatDocumentRepository->find($formatVal);
```

```php
if (is_numeric($formatVal) || empty($formatVal)) {
  $doc->setFormat($format);
}else{
  return View::create('le format envoye n\'est pas valide', Response::HTTP_EXPECTATION_FAILED);
}
```

Je vérifie si la valeur obtenue il s'agit bien d'une valeur numérique (id) ou bien une valeur vide

Et j'insert la variable $format avec la fonction setters de l'entité DEVIS <<setFormat>>

Sinon j'envoie une raiponce HTTP EXPECTATION FAILED avec le <<format non valide>>

Au final je récupère l'EntitéManager de Doctrine et j'utilise méthodes persist() et flush() pour la transactions.

```php
$entityManager = $this->getDoctrine()->getManager();
$entityManager->persist($doc);
$entityManager->flush();
```

Obtenir l'intégralité des informations

Pour obtenir l'intégralité des informations des devis, j'ai utilisé la méthode "**findAll**".

Cette méthode me renvois une collection de données, je vais devoir la convertir en json après l'avoir "**normalisée**", c'est à dire, convertie en tableau.

Pour ce là j'ai installé SensioFrameworkExtraBundle

J'ai dû donc avoir besoin de toutes ces classes

```php
use FOS\RestBundle\Controller\Annotations as Rest;
use Symfony\Component\Serializer\Normalizer\ObjectNormalizer;
use Symfony\Component\Serializer\Serializer;
```

j'ai récupéré la liste de devis et je les'ai push sur un array vide et le normalisée avec la fonction normalize que j'ai créer

```php
/**
 * @Rest\Get(
 *     path="/",
 *     name="api_moggar_index"
 * )
 */
public function index(DevisRepository $DevisRepository): View
{
  $data = $DevisRepository->findAll();
  $labels = [];
  foreach ( $data as $label ) {
    array_push($labels, $this->normalize($label));
  }
  return View::create($labels, Response::HTTP_OK);
}
/**
```

La fonction normalize :

J'ai spécifié les attributs que je souhaite sérialiser

```php
private function normalize($object)
{
  $serializer = new Serializer([new ObjectNormalizer()]);
  $object = $serializer->normalize($object, 'json',
    ['attributes' => [
      'id',
      'nom',
      'prenom',
      'email',
      'phone',
      'exemplaire',
      'propriete' => ['id','proprieteName','description','prix'],
      'quadrichromie' => ['id','quadrichromieName','description','
      'oriantation'=> ['id','oriantationName','description','prix'
      'meida'=> ['id','mediaName','description','prix'],
      'format'=> ['id','mediaName','description','prix'],
      'type'=> ['id','name'],
      'vernis'=> ['id','vernis'],
    ]]);
  return $object;
}
```

6. description de la veille sur la vulnérabilité de sécurité

Vulnérabilité en termes d'URL

Sur Symfony, si un mauvais URL est saisie une page erreur apparait. Ces pages peuvent laisser des données sensibles.

Avec le Framework Symfony, lorsque le site est mis en productions les page erreur serons remplacées par des page d'erreur simple comme par Example 404 not fond

Vulnérabilité en termes SQL

Comme pour tout Framework ou langage de programmation, la règle numéro #1 est la validation des inputs les conséquences des injections de code SQL peuvent être multiples

Sur Symfony pour la gestion de base de données nous utilisent l'entité manager de doctrine (ORM) qui lui Utilise PDO une extension définissant l'interface pour accéder à une base de données avec PHP celui-ci permet de se sécuriser de tout injection SQL

Vulnérabilité en termes d'injection XSS (cross-site Scripting)

XSS est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java, Flash…)

Les formulaires générer par le CRUD de Symfony permet de se protéger de ce genre d'attaque

Et Pour éviter que des informations malicieuses soient enregistrées dans la base de données, j'ai appliquée htmlspecialchars() sur les données avant qu'elles soient enregistrées.

Vulnérabilité en termes d'injection CRSF (cross-site request forgery)

Un utilisateur malveillant va parfois forcer une victime à faire une action spécifique sur un site internet

Symfony met en place un system de token CRSF sur chaque formulaire ainsi chaque formulaire dispose d'une identification, si l'identification ne correspond pas, les informations ne peuvent pas être modifiées.

Vulnérabilité en termes d'administration

Pour étende l'administration de site le client doit être connecter

J'ai mis un URL spécifique au niveau de la route qui dirige vers une page de connexion

Le mot de passe de l'administrateur et enregistrer sur la base de données en sha1 (Secure Hash Algorithme) très utile pour la sécurisation du stockage des mots de passe d'un site web

7. Description d'une situation de travail ayant nécessité une recherche durant le projet à partir de site anglophone

Pour son site internet, le client a demandé qui lui-même puisse insert des images et aussi d'ajouter la possibilité d'envoyer un fichier PDF sur le formulaire de devis

Après quelque recherche sur internet j'ai trouvé ce que je cherchais sur la documentation de Symfony

Grace a cette documentation, j'ai pu mettre en place le téléchargement des fichiers

Aussi le client m'a demandé qu'un mail sera envoyer lorsqu'un utilisateur envois un devis, pour cela j'ai dû configurer POSTFIX et swiftmailer bundle

Un site anglophone ma énormément aidé afin que postfixe sois installer :

Et la documentation Symfony pour maitre en place swiftmailer bundle est bien claire

Vous trouverez dans la partie ci-dessous les extrait du site internet

8. extrait des sites anglophone

Postfix / swiftmailer bundle :

Le URL de site :

https://devanswers.co/configure-postfix-to-use-gmail-smtp-on-ubuntu-16-04-digitalocean-droplet/

In this article we are going to configure Postfix to relay mail through Gmail's SMTP server on Ubuntu 20.04, 18.04 & 19.10. If you want to use a SMTP server other than Gmail, please see How to configure Postfix to use an External SMTP Server.

- Sur cette article on vas configurer postfix afin de relier les mail via gamil SMPT serveur sur ubuntu 20.04, 18.04 & 19.10 Si vous souhaitez utiliser un serveur SMTP autre que Gmail, veuillez consulter Comment configurer Postfix pour utiliser un serveur SMTP

If your Gmail account uses 2-Step Verification, you must create an application specific password.

If you're not using 2-Step Verification, please ensure that your Gmail account is configured to allow less secures apps.

- Si votre compte Gmail utilise la vérification deux étapes vous devez créer un mot de passe spécifique à l'application.

- Si vous utiliser pas la vérification a deux étapes veuillez-vous assurer que Les applications moins sécurisées est autoriser

1. **Installer postfix :**

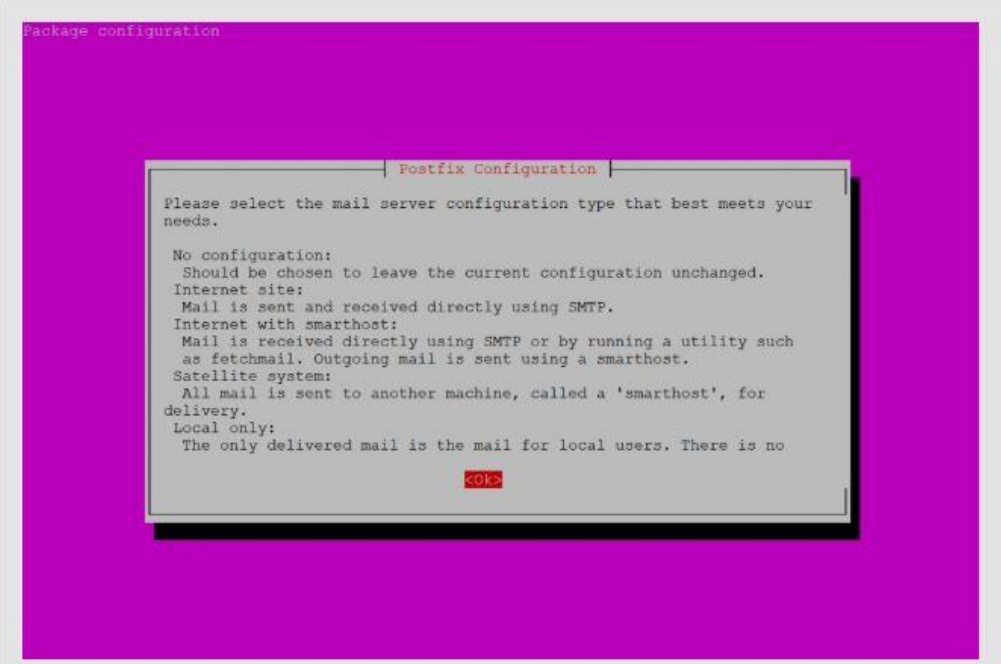
Let's update the package database first.

```
$ sudo apt-get update
```

Install `mailutils`, which will automatically install Postfix.

```
$ sudo apt install -y mailutils
```

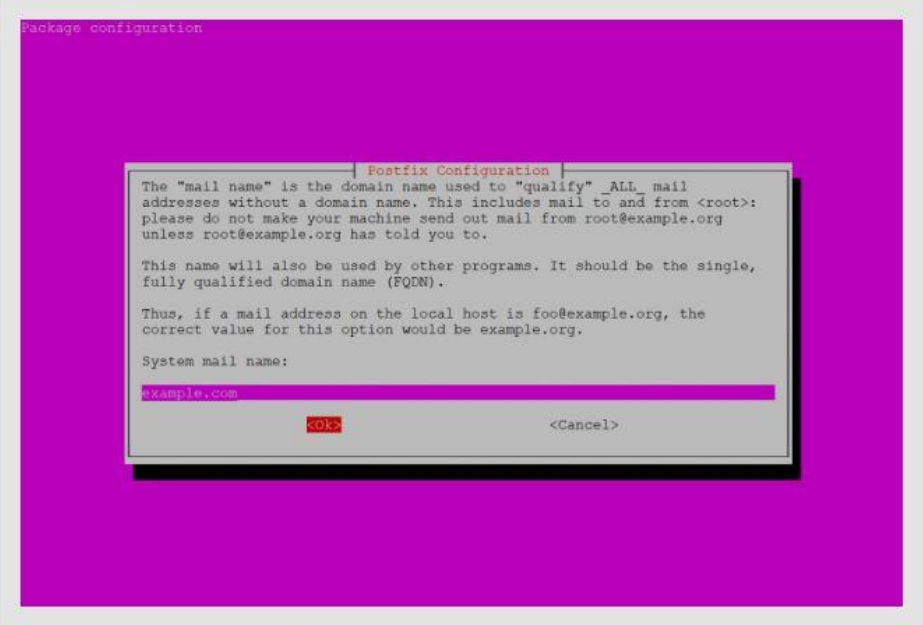
On the first Postfix configuration screen, select OK by pressing `TAB` and `ENTER`



Select Internet Site and press `ENTER`.

- En premier en vas matte a jour notre package de base de donnè
- On Installer mailutils qui vas installer postfix automatiquement
- Sur la première configuration de postfix sélectionner ok en appuyant sur les touches TAB et entrée
- Sélectionner internet site et appuyer sur la touche entrée

System mail name should be your domain name eg. `example.com`, press `ENTER`.



Package should now be installed.

- System mail Name devrait être votre nom de domaine ex. exemple.com, appuyer sur entrée
- Package devrais maintenait être installer

2. Configure Postfix

Edit the Postfix configuration file.

```
$ sudo nano /etc/postfix/main.cf
```

Find the following line `relayhost =` about 6 lines up from the bottom of the file and delete it.

Add the following to the end of the file.

/etc/postfix/main.cf

```
relayhost = [smtp.gmail.com]:587
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options = noanonymous
smtp_tls_CAfile = /etc/postfix/cacert.pem
smtp_use_tls = yes
```

Save file and exit. (Press `CTRL` + `X`, press `Y` and then press `ENTER`)

Configuration Posix

- Modifier le fichier de configuration postfix
- Trouver relayshot = environ la sixième ligne en commencent du bas et supprime la
- Enregistre le fichier et quitter

3. Create Password and DB Files

Create the `sasl_passwd` file which will store our credentials.

```
$ sudo nano /etc/postfix/sasl_passwd
```

Insert the following:

/etc/postfix/sasl_passwd

```
[smtp.gmail.com]:587 username@gmail.com:password
```

Replace `username` and `password` with your own.

Save file and exit. (Press `CTRL` + `X`, press `Y` and then press `ENTER`)

Create a hash database file for Postfix with the `postmap` command.

```
$ sudo postmap /etc/postfix/sasl_passwd
```

There should now be a file called `sasl_passwd.db` in the `/etc/postfix/` directory.

For added security, we will only allow **root** user to read and write to `sasl_passwd` and `sasl_passwd.db`

```
$ sudo chown root:root /etc/postfix/sasl_passwd /etc/postfix/sasl_passwd.db
```

```
$ sudo chmod 0600 /etc/postfix/sasl_passwd /etc/postfix/sasl_passwd.db
```

- Créé le SSL_PASSWD le fichier qui stockera nos informations d'identification
- Ajouter la ligne suivante :
- Remplacer USERNAME et PASSWORD avec le votre
- Enregistrer le fichier
- Créer une base de données de hachage avec la commande postmp
- Il devrait maintenant y avoir un fichier appelé sasl_passwd.db dans le répertoire /etc postfix/
- Pour plus de sécurité, nous autoriserons uniquement la root utilisateur à lire et écrire dans sasl_passwd et sasl_passwd.db

4. Sign Certificate

Now we are going to create the certificate.

```
$ cat /etc/ssl/certs/thawte_Primary_Root_CA.pem | sudo tee -a /etc/postfix/cacert.pem
```

There should now be a certificate file called `cacert.pem` in `/etc/postfix`

---

5. Send a Test Mail

We'll now send a test email message. Make sure to replace `test@example.com` with your own email address.

```
$ echo "Test Email message body" | mail -s "Email test subject" test@example.com
```

Don't forget to check your spam folder.

If you still haven't received any mail, check the mail error log.

```
$ sudo tail /var/log/mail.log
```

If the mail log is empty or doesn't exist, try parsing the syslog. This will return the last 50 entries for postfix.

```
$ sudo tail -f -n 50 /var/log/syslog | grep postfix
```

If the syslog is empty and you still haven't received any test email, it's possible that the test email was rejected by the recipient server. You should check to see if anything has bounced back to your mail folder.

```
$ sudo less /var/mail/$(whoami)
```

Press uppercase `G` to scroll to the bottom of the file and lowercase `q` to quit. The `$(whoami)` variable returns the currently logged in user.

---

6. Allow Less Secure Apps

If Gmail is not allowing postfix to connect via SMTP, you may need to enable "Allow Less Secure Apps" on your Gmail account.

Please see: Allow less secure apps to access your Gmail account

4. Signer le certificat

- Maintenait en vas créer un certificat :
- Il devrait maintient être crée

5. Envoyer un mail

- Maintenant en vas envoyer un mail, vérifier bien de remplacer test@exmaple.com avec votre propre adresse mail
- N'oublier pas de vérifier votre spam
- Si le mail n'a pas été envoyé vérifier l'erreur log.

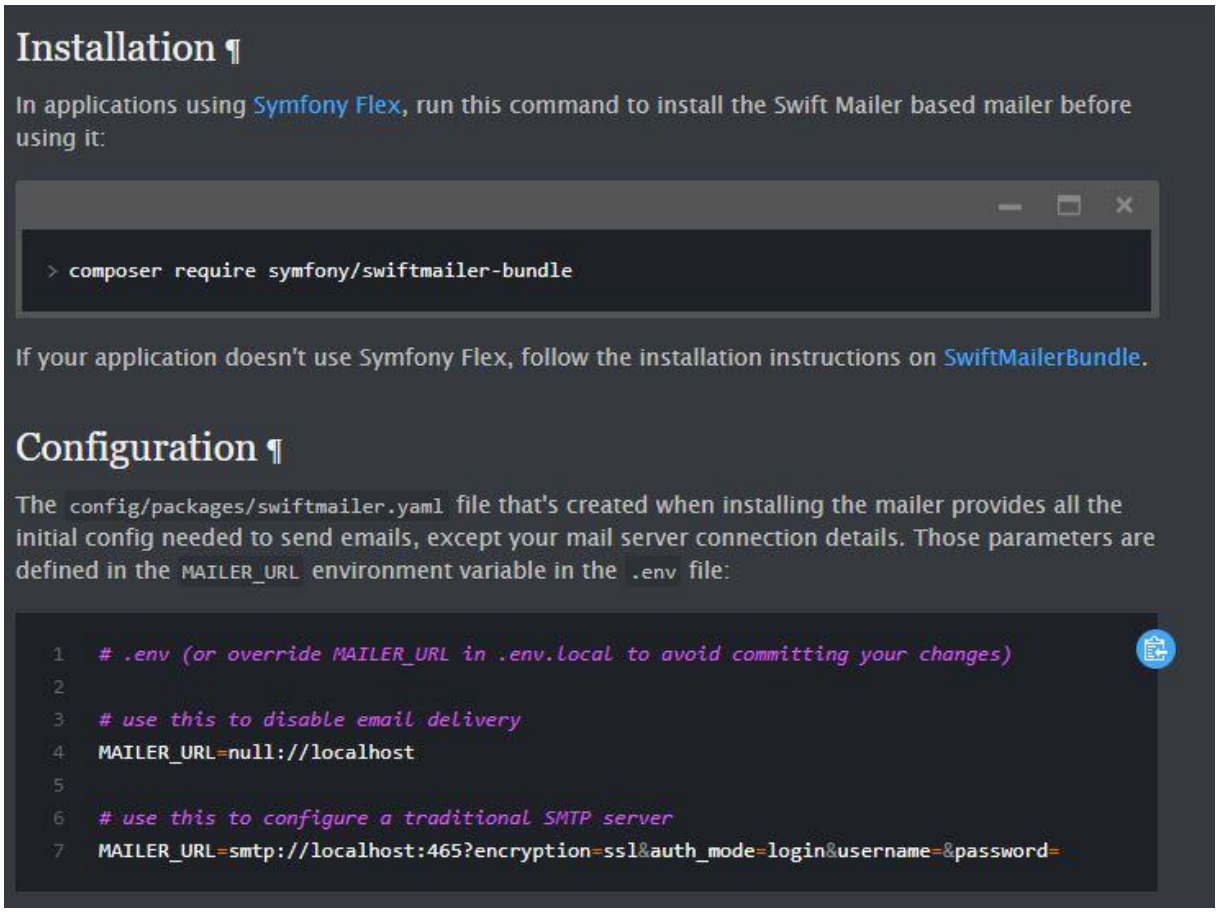
- Si le journal d'erreur est vide ou inexistant Essayez d'analyser le fichier syslog. Cela renverra les 50 dernières entrées pour postfix.
- Si le syslog. Est vide et vous n'avais aucun mail reçu il est possible que l'e-mail a été rejeté par le serveur destinataire.
- Appuyez sur G majuscule pour faire défiler vers le bas du fichier et sur q minuscule pour quitter. La variable $(whoami) renvoie l'utilisateur actuellement connecté.

6. Autoriser les applications moins sécurisées

- Si Gmail n'autorise pas postfix à se connecter via SMTP, vous devrez peut-être activer « Autoriser les applications moins sécurisées » sur votre compte Gmail.

swiftmailer bundle :

Le URL du site : https://symfony.com/doc/current/email.html

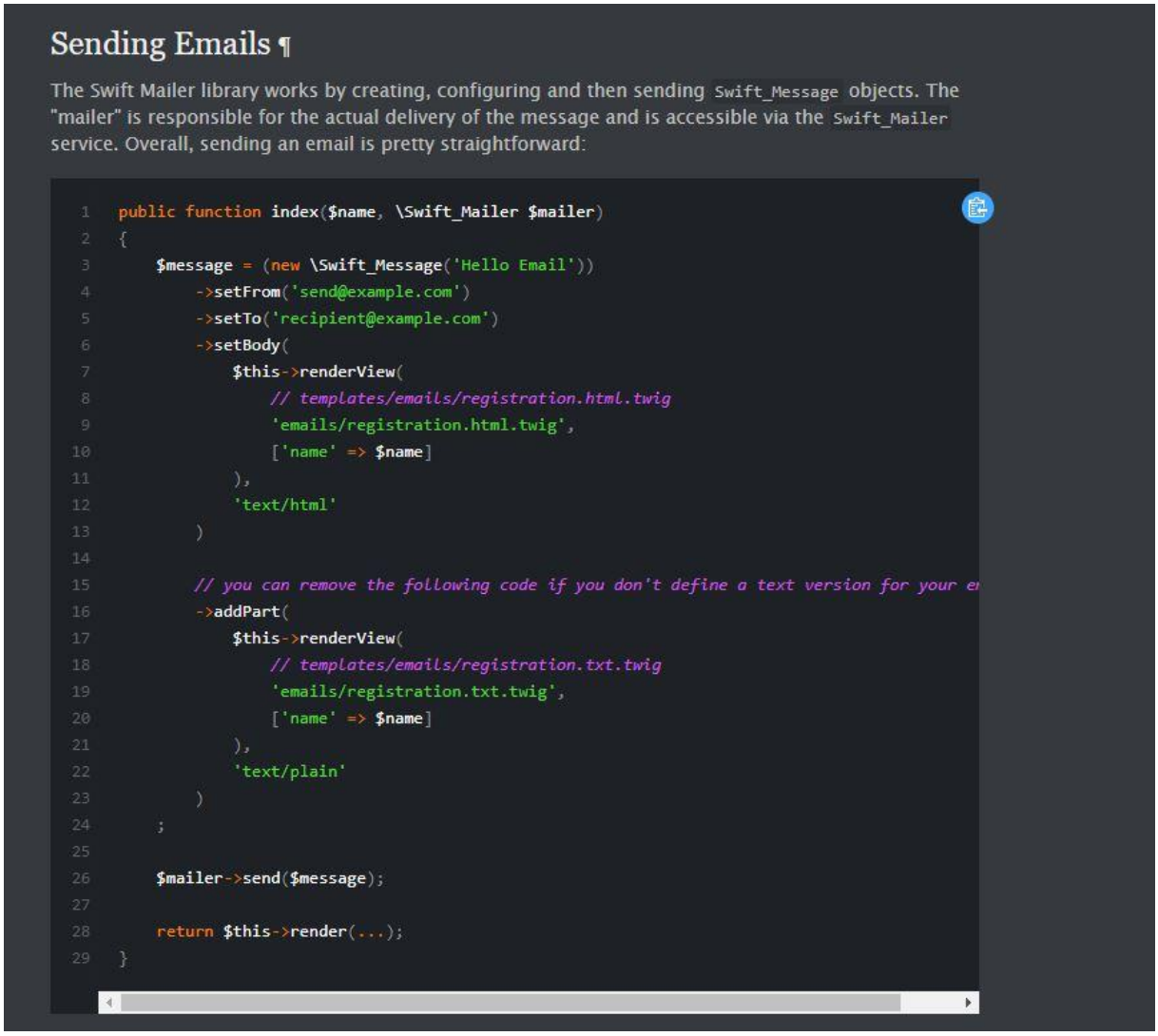


Installation :

- Dans les applications utilisant Symfony Flex, exécutez cette commande pour installer Swift Mailer avant de l'utiliser :
- Si votre application n'utilise pas Symfony Flex, suivez les instructions d'installation sur SwiftMailerBundle.

Configuration :

- Le fichier config /packages/swiftmailer.yaml a été créé. Lors de l'installation de l'expéditeur fournit toutes les configurations initiales nécessaires pour envoyer des
  E-mails, à l'exception des détails de connexion à votre serveur. Ces paramètres sont définis dans la variable MAILER_URL du fichier .env:



Sending Emails ¶

The Swift Mailer library works by creating, configuring and then sending `Swift_Message` objects. The "mailer" is responsible for the actual delivery of the message and is accessible via the `Swift_Mailer` service. Overall, sending an email is pretty straightforward:

```php
public function index($name, \Swift_Mailer $mailer)
{
    $message = (new \Swift_Message('Hello Email'))
        ->setFrom('send@example.com')
        ->setTo('recipient@example.com')
        ->setBody(
            $this->renderView(
                // templates/emails/registration.html.twig
                'emails/registration.html.twig',
                ['name' => $name]
            ),
            'text/html'
        )

        // you can remove the following code if you don't define a text version for your e
        ->addPart(
            $this->renderView(
                // templates/emails/registration.txt.twig
                'emails/registration.txt.twig',
                ['name' => $name]
            ),
            'text/plain'
        )
    ;

    $mailer->send($message);

    return $this->render(...);
}
```

Envois des Email :

- La bibliothèque Swift Mailer fonctionne en créant, en configurant puis en envoyant des objets Swift_Message. Le "mailer" est responsable de la livraison du message et est accessible via le service Swift_Mailer. Dans l'ensemble, l'envoi d'un e-mail est assez simple :

Les upload des fichiers :

Recherche de documentation de site web Symfony

URL: https://symfony.com/doc/current/controller/upload_file.html

Imagine that you have a `Product` entity in your application and you want to add a PDF brochure for each product. To do so, add a new property called `brochureFilename` in the `Product` entity:

```php
// src/Entity/Product.php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

class Product
{
    // ...

    /**
     * @ORM\Column(type="string")
     */
    private $brochureFilename;

    public function getBrochureFilename()
    {
        return $this->brochureFilename;
    }

    public function setBrochureFilename($brochureFilename)
    {
        $this->brochureFilename = $brochureFilename;

        return $this;
    }
}
```

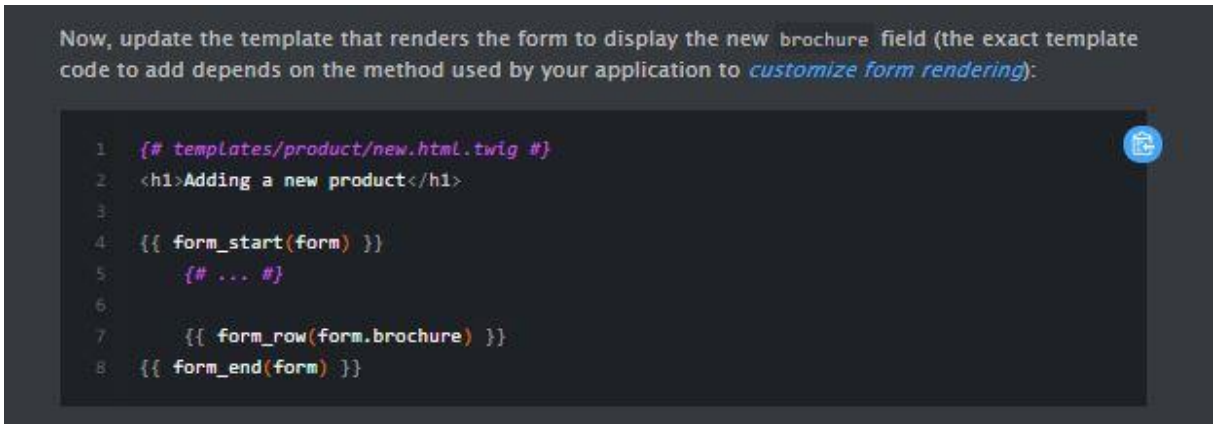
- Imaginez que vous ayez une entité Produit dans votre application et que vous souhaitiez ajouter une brochure PDF pour chaque produit. Pour ce faire, ajoutez une nouvelle propriété appelée brochureFilename dans l'entité Product :

Note that the type of the `brochureFilename` column is `string` instead of `binary` or `blob` because it only stores the PDF file name instead of the file contents.

The next step is to add a new field to the form that manages the `Product` entity. This must be a `FileType` field so the browsers can display the file upload widget. The trick to make it work is to add the form field as "unmapped", so Symfony doesn't try to get/set its value from the related entity:

```php
// src/Form/ProductType.php
namespace App\Form;

use App\Entity\Product;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Validator\Constraints\File;

class ProductType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            // ...
            ->add('brochure', FileType::class, [
                'label' => 'Brochure (PDF file)',

                // unmapped means that this field is not associated to any entity property
                'mapped' => false,

                // make it optional so you don't have to re-upload the PDF file
                // every time you edit the Product details
                'required' => false,

                // unmapped fields can't define their validation using annotations
                // in the associated entity, so you can use the PHP constraint classes
                'constraints' => [
                    new File([
                        'maxSize' => '1024k',
                        'mimeTypes' => [
                            'application/pdf',
                            'application/x-pdf',
                        ],
                        'mimeTypesMessage' => 'Please upload a valid PDF document',
                    ])
                ],
            ])
            // ...
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Product::class,
        ]);
    }
}
```

- A noter que l'attribut brochure est en chaine de caractère au lieu d'être chiffré car elle sert qu'à stocker le nom de fichier ou lieu des données

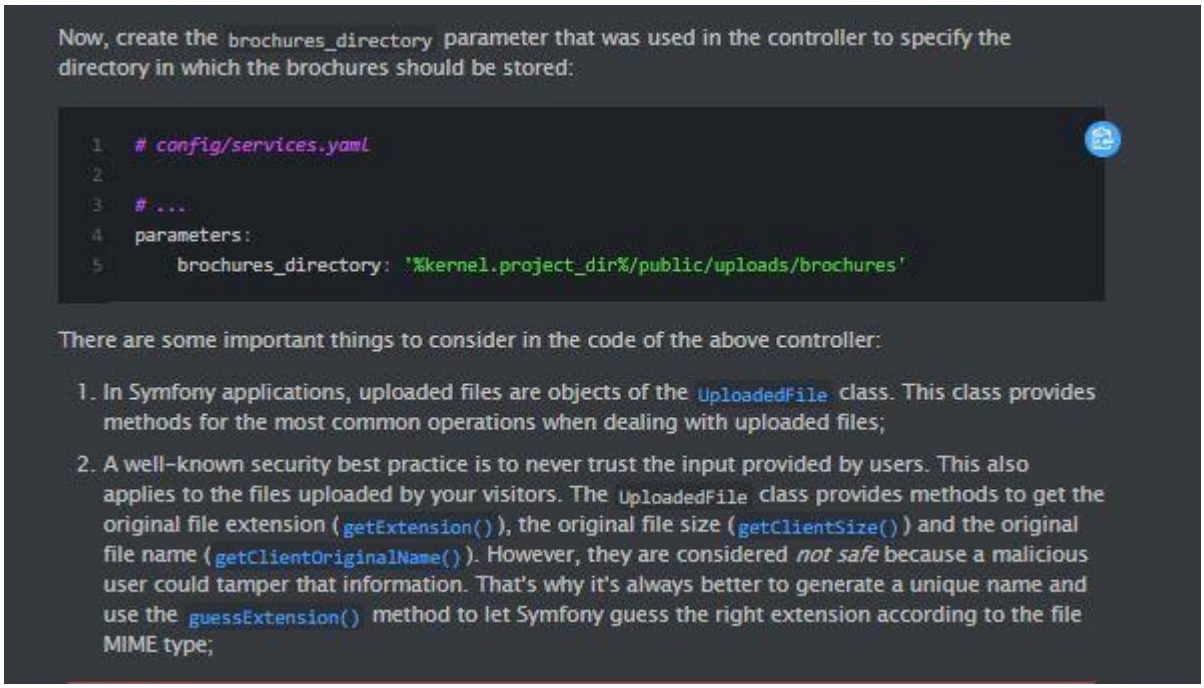


- Maintenant, mettez à jour le modèle qui rend le formulaire pour afficher le nouveau champ de brochure (le code de modèle exact à ajouter dépend de la méthode utilisée par votre application pour personnaliser le rendu du formulaire)

Finally, you need to update the code of the controller that handles the form:

```php
// src/Controller/ProductController.php
namespace App\Controller;

use App\Entity\Product;
use App\Form\ProductType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\File\Exception\FileException;
use Symfony\Component\HttpFoundation\File\UploadedFile;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

class ProductController extends AbstractController
{
    /**
     * @Route("/product/new", name="app_product_new")
     */
    public function new(Request $request)
    {
        $product = new Product();
        $form = $this->createForm(ProductType::class, $product);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            /** @var UploadedFile $brochureFile */
            $brochureFile = $form->get('brochure')->getData();

            // this condition is needed because the 'brochure' field is not required
            // so the PDF file must be processed only when a file is uploaded
            if ($brochureFile) {
                $originalFilename = pathinfo($brochureFile->getClientOriginalName(), PATHI
                // this is needed to safely include the file name as part of the URL
                $safeFilename = transliterator_transliterate('Any-Latin; Latin-ASCII; [^A-
                $newFilename = $safeFilename.'-'.uniqid().'.'.$brochureFile->guessExtensio

                // Move the file to the directory where brochures are stored
                try {
                    $brochureFile->move(
                        $this->getParameter('brochures_directory'),
                        $newFilename
                    );
                } catch (FileException $e) {
                    // ... handle exception if something happens during file upload
                }

                // updates the 'brochureFilename' property to store the PDF file name
                // instead of its contents
                $product->setBrochureFilename($newFilename);
            }

            // ... persist the $product variable or any other work

            return $this->redirect($this->generateUrl('app_product_list'));
        }

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

- Enfin, vous devez mettre à jour le code du contrôleur qui gère le formulaire



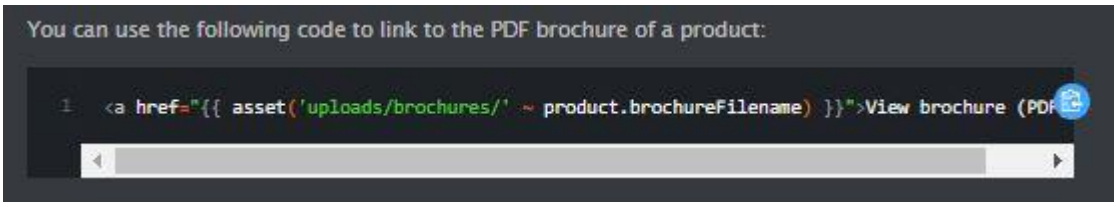
Now, create the `brochures_directory` parameter that was used in the controller to specify the directory in which the brochures should be stored:

```yaml
# config/services.yaml

# ...
parameters:
    brochures_directory: '%kernel.project_dir%/public/uploads/brochures'
```

There are some important things to consider in the code of the above controller:

1. In Symfony applications, uploaded files are objects of the `UploadedFile` class. This class provides methods for the most common operations when dealing with uploaded files;
2. A well-known security best practice is to never trust the input provided by users. This also applies to the files uploaded by your visitors. The `UploadedFile` class provides methods to get the original file extension (`getExtension()`), the original file size (`getClientSize()`) and the original file name (`getClientOriginalName()`). However, they are considered *not safe* because a malicious user could tamper that information. That's why it's always better to generate a unique name and use the `guessExtension()` method to let Symfony guess the right extension according to the file MIME type;

- Maintenant, créez le paramètre brochures_directory qui a été utilisé dans le contrôleur pour spécifier le répertoire dans lequel les brochures doivent être stockées
- Il y a des choses importantes à considérer dans le code du contrôleur ci-dessus :
- 1. Dans les applications Symfony, les fichiers téléchargés sont des objets de la classe UploadedFile. Cette classe fournit des méthodes pour les opérations les plus courantes lors du traitement des fichiers téléchargés ;
- 2. Ce qui est bon a savoir en terme de sécurité consiste à ne jamais faire confiance a des champ fournis par les visiteur. La classe UploadedFile fournit des méthodes pour obtenir l'extension de fichier d'origine (getExtension ()), la taille de fichier d'origine (getClientSize ()) et le nom de fichier d'origine (getClientOriginalName ()) cependant, ces methodes ne sont pas consideres comme sur car un utilisateur malveillant pourrait altérer ces informations C'est pourquoi il est toujours préférable de générer un nom unique et d'utiliser la méthode guessExtension ()



You can use the following code to link to the PDF brochure of a product:

```twig
<a href="{{ asset('uploads/brochures/' ~ product.brochureFilename) }}">View brochure (PDF
```

- Vous pouvez utiliser le code suivant pour créer un lien vers la brochure PDF d'un produit

CONCLUSION

Sur ce site internet le client peut en tant qu'administrateur, gérer les prix, les devis, les produit, les messages…, les visiteurs pourront visualisez ces éléments via la vue twig de symfony.

Le Framework symfony ma permit de créer un site web sur la façon la plus rapide et pratique. Notamment grâce à son ORM Doctrin concernant la gestion de base de donne MySql. Le Framework ma aussi permit à afficher le front de site avec ses information grâces au twig que j'ai pu personnaliser avec Bootstrap j'ai aussi inséré quelque script jQuery et utiliser JavaScript

Le client et moi sommes plutôt satisfait de site internet le site et toujours en construction. le client ma proposer plus d'option a ajouter et je dois revoir avec pour quelque détails au désigne afin de rendre le site plus attractif.