Documentation Technique – Projet Bannière Marketing (Refonte)

Contexte

Le projet vise à automatiser l'affichage des signatures de mails en fonction du service auquel appartient un utilisateur. L'application repose sur une hiérarchie bien définie :

• **Clients** : Ce sont les utilisateurs ou entreprises avec lesquels l'entreprise travaille.
• **Sociétés** : Chaque client appartient à une société spécifique.
• **Services** : Une société peut offrir plusieurs services.
• **Campagnes** : Chaque service peut être associé à une ou plusieurs campagnes marketing.
• **Bannières** : Chaque campagne utilise une ou plusieurs bannières publicitaires.
• **Affichage des bannières** : Les bannières seront affichées et visibles par les clients de la société.

Objectif du Projet

L'objectif principal est de moderniser une application web existante pour la rendre plus ergonomique et performante. Cette application est destinée à un usage interne et permettra une meilleure gestion des campagnes marketing et de leurs bannières associées.
L'application permet de gérer les bannières des signatures de mail pour les utilisateurs de la société Hefed.

Technologies Utilisées

- Outils : Git pour la gestion de version.
- Langages : PHP et JavaScript.
- Framework : Symfony.

  Bundles Utilisés : EasyAdmin, AssetMapper, Security, Reset-Password, VichUploader.

SOMMAIRE

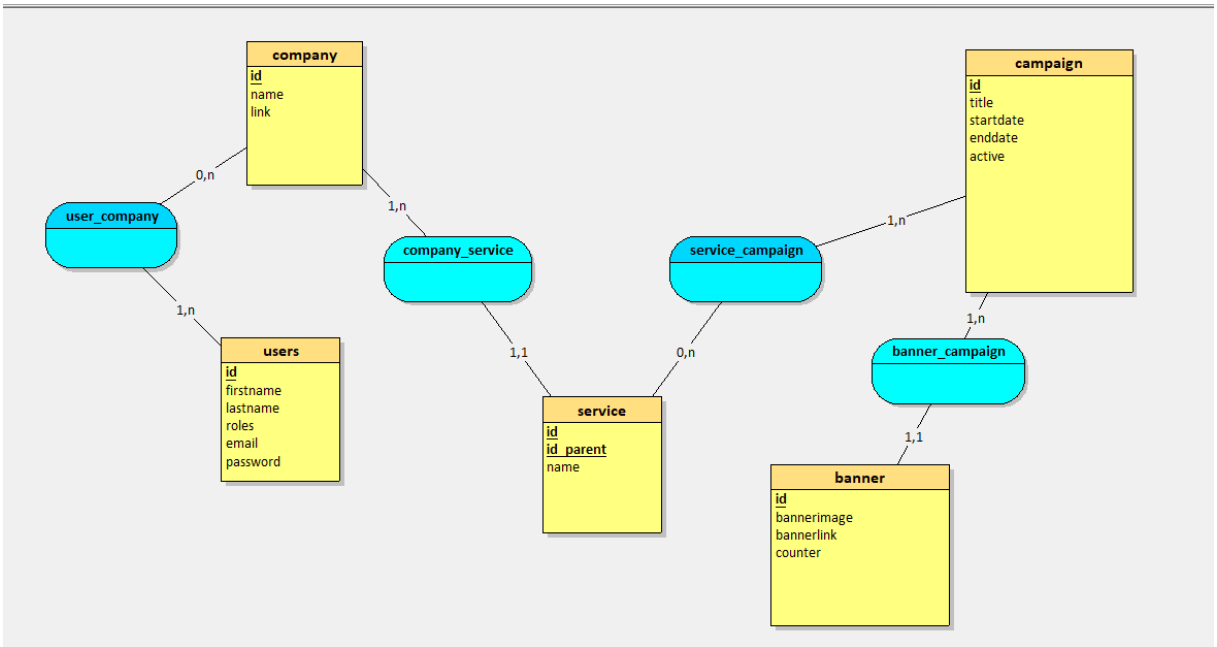
1. Base de données et Entité
    1.1 Structure de la Base de données
    1.2 Les Entités de Symfony
2. Sécurité
    2.1 Bundle Sécurity
    2.2 Permissions
    2.3 UserChecker
3. EasyAdmin
    3.1 CrudController
    3.2 Field
    3.4 Event
    3.5 Template
    3.6 Actions
4. Fonctionnalité
    4.1 Bundle VichUploader
    4.2 Bundle reset-password
5. les Routes Symfony
    5.1 les Routes
6. Workflow
    6.1 User
    6.2 Administrateur

1.Base de données et Entité

1.1 Structure de Base de données

« bannieremkt » est la base de Donnée MySQL utilisé pour ce projet

Il a été fait un MCD (Modèle Conceptuel des données) de la structure de Base de Données
Un Modèle Conceptuel de Données (MCD) a été élaboré pour structurer les relations entre les entités du projet. À partir de ce modèle, les entités Symfony ont été définies avec leurs relations respectives pour garantir une base de données bien organisée.



La base de données comprend 7 tables à partir des Entités

1.2 les Entités

1. User (users)

L'entité users représente les utilisateurs de l'application.

- Relation ManyToMany avec company
    - Un utilisateur peut être associé à plusieurs entreprises.
    - Une entreprise peut avoir plusieurs utilisateurs.
    - Cette relation crée une table intermédiaire user_company.

2. Company (company)

L'entité company représente une entreprise.

- Relation OneToMany avec service
    - Une entreprise peut avoir plusieurs services.
    - Une clé étrangère company_id est ajoutée dans la table service.

3. Service (service)

L'entité service représente un service lié à une entreprise.

- Relation ManyToMany avec campaign
    - Un service peut être associé à plusieurs campagnes.
    - Une campagne peut concerner plusieurs services.
    - Une table intermédiaire campaign_service est créée.
- Relation ManyToOne avec company
    - Chaque service appartient à une seule entreprise.
    - Une clé étrangère company_id est présente dans service.
- Relation récursive (auto-référence)
    - Un service peut être lié à un autre service via le champ id_parent.
    - Cela permet d'organiser les services sous forme hiérarchique (ex : sous-services).

4. Campaign (campaign)

L'entité campaign représente une campagne marketing ou publicitaire.

- Relation ManyToMany avec service
    - Une campagne peut être associée à plusieurs services.
    - Cette relation est gérée par la table campaign_service.

5. Banner (banner)

L'entité banner représente une bannière publicitaire.

- Relation ManyToOne avec campaign
    - Une campagne peut avoir plusieurs bannières.
    - Une clé étrangère campaign_id est ajoutée dans la table banner.

2.Sécurité

2.1 Bundle Sécurity

Le bundle Sécurity permet de rajouter un Controller « SecurityController.php » qui permet donc de crée une page de login sécurisé

Et qui rajoute aussi un fichier « security.yaml » dans lequel il y’a plusieurs paramètres

- password_hashers « permet le hash avec un algorithm « bcrypt » »

- user_checker « permet d'exécuter des vérifications personnalisées sur un utilisateur avant qu'il ne soit authentifié »
- form_login :
    - login_path « route ou ce situe le formulaire de login»
    - check_path « route qui verifie l'authentification »
    - enable_csrf « active ou non le token csrf »

  logout :

  path : « qui permet de choisir sa redirection lors de la deconnexion »

-switch_user : « qui permet l'impersonalisation »

-access_control :

- { path: ^/bannieres, roles: [ROLE_ADMIN, ROLE_USER] }

«Cela permet de gérer l'accès ce qui signifie pour cette exemple qu'une personne non authentifié ne peux pas accéder à cette route»

Ainsi qu'une fonction de déconnexion (logout)

2.2 Permissions

Roles :
Admin

Acces :

- UserCrudController (Gestion des utilisateurs)
- CampaignCrudController (Gestion des campagnes)
- CompanyCrudController (Gestion des entreprises)
- Page de connexion
- Route de déconnexion

Permissions :

- Admin possède tous les droits sur l'ensemble des entités.
- Il peut créer, modifier et supprimer les utilisateurs, campagnes et entreprises.

User

Acces
- CampaignCrudController (Gestion des campagnes)
- Page de connexion
- Route de déconnexion

Permissions :

- L'utilisateur peut créer, modifier et archiver des campagnes
- Il ne peut pas accéder à la gestion des utilisateurs ni des entreprises.

2.3 UserCheckerInteface

Le fichier UserChecker.php

Possède 2 function qui provienne de UserCheckerInterface de Symfony.
Cette interface permet de définir des règles de sécurité pour vérifier les informations d'un utilisateur avant et après l'authentification.

checkPostAuth(UserInterface $user): void
Cette méthode est exécutée après l'authentification réussie d'un utilisateur.
Elle est utilisée dans ce projet pour bloquer l'accès aux comptes inactifs après la soumission du formulaire de connexion, permettant ainsi une meilleure modération des utilisateurs.

checkPreAuth(UserInterface $user): void
Cette méthode est exécutée avant l'authentification d'un utilisateur.
Elle permet d'élaborer des règles de validation en amont, comme empêcher la connexion d'un utilisateur banni ou nécessitant une vérification supplémentaire.

3.EasyAdmin

3.1 CrudController

EasyAdmin est un bundle Symfony permettant de créer rapidement une interface d'administration. Les CrudController sont des contrôleurs dédiés à la gestion CRUD des entités dans l'administration. Chaque entité a son propre CrudController, qui définit son affichage et son comportement dans EasyAdmin.

Un CrudController hérite de AbstractCrudController.

Il est responsable de la configuration des actions, des champs et des permissions pour une entité donnée.

Il est enregistré dans DashboardController pour être accessible via l'interface d'administration.

Les CrudController du projet sont

UserCrudController	Est utilisé pour le rôle Admin
CompanyCrudController	Est utilisé pour le rôle Admin
CampaignCrudController	Est utilisé pour le rôle Admin et User
BannerCrudController	Est utilisé dans un collectionField* afin d'obtenir ces champs

*Voir Field

3.2 Field

Les champs permettent d'afficher le contenu de vos entités Doctrine sur chaque page CrudController . EasyAdmin fournit des champs intégrés pour afficher tous les types de données courants, mais vous pouvez également crée vos propres champs.

Voici les champs utilisés :

IdField

Field

UrlField

NumberField

DateTimeField
AssociationField

CollectionField
BooleanField
TextField
ChoiceField

Le TreeCampaignField est un champ personnalisé pour EasyAdmin permettant d'afficher une arborescence des services liés à une campagne sous forme de cases à cocher. Il repose sur :

- TreeCampaignType (FormType Symfony) : construit la structure des services.
- TreeCampaignField (Field EasyAdmin) : intègre le champ dans l'interface d'administration.

TreeCampaignType - Génération de l'arbre

Cette classe crée l'arborescence des services et détermine les éléments sélectionnés.

- buildView() : Récupère les entreprises et construit leur arborescence.

- buildCompanyTree() : Génère la structure des entreprises et leurs services.
- buildServiceTree() : Organise les services de manière récursive.
- hasChildChecked() : Vérifie si un service est déjà sélectionné.

```
final class TreeCampaignField implements FieldInterface
{
   use FieldTrait;


   public static function new(string $propertyName, $label = null): self
   {
      return (new self())
         ->setProperty($propertyName)
         ->setLabel($label)
         ->setFormType(TreeCampaignType::class);
   }
}
```

Il ne reste qu'à ajouter un Template personnalisé pour les formulaires

Dans le configureCrud de ton CrudController avec cette ligne
->addFormTheme('TreeCompany/custom_types.html.twig');

Le TreeCompanyField est un champ personnalisé pour EasyAdmin permettant d'afficher une arborescence des services. Il repose sur :

- TreeCompanyType (FormType Symfony) : construit la structure des services.
- TreeCompanyField (Field EasyAdmin) : intègre le champ dans l'interface d'administration.

TreeCompanyType Generation de l’arbres :
Cette classe crée l'arborescence des services.

- buildView() : Récupère les entreprises et construit leur arborescence.
- treeService() : affiche les services de manière recursive

```
final class TreeCompanyField implements FieldInterface
{
    use FieldTrait;

    /**
     * @param TranslatableInterface|string|false|null $label
     */
    public static function new(string $propertyName, $label = null): self
    {
        return (new self())
            ->setProperty($propertyName)
            ->setLabel($label)
            ->setFormType(TreeCompanyType::class);
    }
}
```

Il ne reste qu’à ajouter un Template personnalisé pour les formulaires

Dans le configureCrud de ton CrudController avec cette ligne
->*addFormTheme*('form/service_tree.html.twig');

3.4 Event

EasyAdmin déclenche plusieurs évènements Symfony lors de l'exécution de ses requêtes, vous pouvez donc écouter ces événements et exécuter votre propre logique.

Il y a tout d'abord sur ce projet des EventSubscriber qui ont pour rôles d'écouter des évènements Symfony

ils sont utilisé sur ce projet pour pouvoir soumettre les formulaires des champs personnalisés

Il a été ajouter des écouteurs sur ces évènements :

BeforeEntityPersistedEvent

- Déclenché avant la création d'une nouvelle instance d'une entité.
- Permet de modifier ou enrichir les données avant leur enregistrement en base.

BeforeEntityUpdatedEvent

- Déclenché avant la mise à jour d'une entité existante.
- Permet de vérifier ou modifier les données avant qu'elles ne soient enregistrées.

Dans le fichier CampaignSubscriber.php

Il y a 3 méthodes

getSubscribedEvents()
getSubscribedEvents est utilisé pour y ajouter des écouteurs exemple :

```
public static function getSubscribedEvents()
    {
        return [
            BeforeEntityPersistedEvent::class => ['handleEntityEvent'],
            BeforeEntityUpdatedEvent::class => ['handleEntityEvent'],
```

```
        ];
    }
```

handleEntityEvent(AbstractLifecycleEvent $event)
Cette fonction est utilisée pour traiter les données lors de la création ou modification d'une campagne.
Elle récupère la liste des services et sous-services sélectionnés via getSelectedServiceId($service) et met à jour l'entité en :

- Ajoutant les nouveaux services sélectionnés.
- Supprimant ceux qui ne sont plus cochés.

Cela garantit que la campagne reste toujours associée aux bons services.

getSelectedServiceId($service)
Cette fonction récupère l'ID d'un service et, de façon récursive, ajoute les IDs de ses sous-services s'il en a. Tous les IDs sont ensuite stockés dans un tableau et retournés.

Evenement Javascript dans EasyAdmin

EasyAdmin déclenche plusieurs événements JavaScript permettant d'interagir avec le formulaire et la gestion des collections d'éléments.

Type d'événement	Se produit lorsque	Détail de l'événement	Annulable
' ea.form.error'	L'utilisateur sou met un formulaire qui comporte des erreurs de validation	{page: pageName, form: form}	vrai
'ea.form.submit'	L'utilisateur soumet un formulaire	{page: pageName, form: form}	vrai

'ea.collection.item-added'	Article ajouté à la collection	{newElement: element}	FAUX
'ea.collection.item-removed'	Article retiré de la collection		FAUX

Exemple pour les bannières ou il y'a eu besoin de l'utiliser

App.js

```
document.addEventListener("ea.collection.item-added", (event) => {
    eventCount++;

    if (eventCount === 2) {
      btnAdd.classList.add("d-none");
    }
  });
```

Ce code compte le nombre de bannières ajoutées et cache le bouton d'ajout dès que la limite de 2 est atteinte.

3.5 Template

EasyAdmin importe des Templates par default que l'on peut surcharger

Pour cela il suffit d'aller dans vendor/easycorp/easyadmin-bundle/template

Et de choisir le template que tu veux modifier et ensuite tu crées un fichier .html.twig et tu dois étendre le template de ton choix comme ceci
Je l'ai utilisé ici afin de supprimer la barre de recherche

```
 {% extends "@!EasyAdmin/layout.html.twig" %}


{% block javascripts %}
    {% block importmap %}
        {{ importmap('app') }}
    {% endblock %}
{% endblock %}
{% block page_actions_wrapper %}
    <div class="page-actions">
        {% block page_actions %}
            {{ parent() }}
        {% endblock %}
    </div>
{% endblock %}
{% block search_wrapper %}{% endblock search_wrapper %}
```

3.6 Actions

configureActions(Action $actions) est une fonction qui permet de gérer ce que l'ont peut faire sur le CrudController , on peut modifier le fonctionnement des boutons, changer leur style les supprimer , crée de nouveaux Bouton.

Sur UserCrudController
Un bouton se connecter à été crée afin de permettre l'impersonnalisation d'un utilisateur.

Cette action permet aux administrateurs de se connecter en tant qu'un autre utilisateur

Sur CampaignCrudController

Un bouton de suppression personnalisé a été ajouté, remplaçant la suppression définitive par une archivage de la campagne.
Cela empêche les utilisateurs de supprimer définitivement des campagnes tout en permettant leur désactivation.

4. Fonctionnalité

4.1 Bundle VichUploader

Le Bundle VichUploader permet la gestion des fichiers médias, notamment l'upload d'images. Il crée automatiquement un dossier /uploads/images/ grave au fichier vich_uploader.yaml et enregistre uniquement le nom du fichier dans la base de données.

L'entité Banner a été modifiée pour inclure un champ permettant l'upload d'images :

```
#[Vich\UploadableField(mapping: 'images', fileNameProperty: 'picture_name')]
private ?File $picture = null;
```

mapping: 'images' définit la configuration d'upload.

fileNameProperty: 'picture_name' permet d'enregistrer uniquement le nom du fichier en base de données.

Dans le CrudController, ce champ est ajouté pour permettre l'upload des fichiers via EasyAdmin. Lors de l'enregistrement :

- L'image est stockée dans /uploads/images/.
- Seul le nom du fichier est sauvegardé en base de données.

4.2 Bundle resetpassword

Le Bundle ResetPassword permet la gestion de la réinitialisation des mots de passe. Il génère automatiquement plusieurs fichiers pour gérer le processus, notamment l'envoi d'e-mails et la mise à jour du mot de passe en base de données.

5.Les Routes Symfony

5.1Les Routes

Nom	Route	Detail
admin	/bannieres	Page d'Accueil
archived_campaign	/archived-campaign/{id_campaign}	Permet d'archivé une campagne avec un id
link_mkt	/lien/{link}/{id}	Création du lien qui sera clickable sur l'outils c# et qui va rediriger vers le site web associé à la bannière
image_mkt	/image/{im}/{id}	Création du lien d'image sur l'outils c# qui permet une visualisation de l'image de bannière
ids_compare	/ids-compare	Permet d'afficher seulement les services de la compagnie ou l'on se trouve

app_forgot_password_request	/reset-password	Page de formulaire sur la reinitialisation du mot de passe
app_check_email	/reset-password/check-email	Page qui informe Sur l'etat du mail
app_reset_password	/reset-password/reset/{token}	Page de demande de changement de mot de passe
app_login	/	Page de Login
app_logout	/logout	Deconnexion

6 Workflow

6.1 Worflow de l'utilisateur

Authentification

- L'utilisateur ne peut pas créer son propre compte.
- L'administrateur est responsable de la création et de la gestion des comptes utilisateurs.
- Lorsqu'un utilisateur est créé, il possède par défaut un rôle utilisateur standard (ROLE_USER).

Accès et Navigation

- Après l'authentification, l'utilisateur est redirigé vers la page d'accueil (/banniere).
- Dans la barre de navigation, il n'a accès qu'à la section Campagne.

Gestion des Campagnes et Bannières

- L'utilisateur peut créer, modifier et lire et supprimer les campagnes et des bannières.
- Ils possèdent aussi la gestion des bannières
- La suppression d'une campagne ne la supprime pas définitivement :

- La campagne est archivée.
- Elle disparaît de l’écran de l’utilisateur mais reste en base de données.

6.2 Worflow de l’administrateur

L’administrateur doit crée les comptes des utilisateurs

Il possède le rôle [ROLE_ADMIN]

Accès et Navigation

- Après l’authentification, l’admin est redirigé vers la page d’accueil (/banniere).
- Dans la barre de navigation, ils possèdent
    - Sociétés
    - Utilisateurs
    - Campagnes

Gestion des Campagnes, Bannières, Services, Sociétés, Utilisateurs

L’administrateur possèdent tous les droits
Il peut se connecter sur le compte de quelqu’un d’autre grâce à l’impersonnalisation.

Il peut modifier le nombre de click dans les bannières
Il peut Créer, modifier, supprimer, lire
Il peut aussi interagir avec les arbres dans CompanyCrudController comme créer de nouvelle branche , sous-branche.
Il dispose d’une lecture sur les campagnes qui sont archivé et possède un filtre pour les voir .