Rapport de Stage

Romain Bourel

Sommaire

Remerciements

Je souhaite remercier l’équipe de Challangel pour m’avoir accueilli et fait confiance durant cette année, ainsi que pour l’aide qu’ils m’ont apportée. Cela m’a permis d'agrandir mes compétences et connaissances techniques.

Je remercie tout particulièrement Cannelle, ma maître d’apprentissage, pour ses nombreux retours qui ont permis de me mettre sur la bonne voie et de pousser plus loin les possibilités, de faire un code plus clair, compréhensible et maintenable.

Je remercie également mes proches et surtout ma femme pour leur aide et leur soutien dans ce défi qu’a été ma reconversion. Grâce à eux j’ai pu me consacrer entièrement à mon projet et je suis fier du chemin accompli.

Présentation de L’entreprise

Créé en 2014 sur une idée originale de Marina Giacco, Challangel est une entreprise ayant pour but de créer des relations entre créateurs et marques via des appels à projet prenant la forme de challenges.

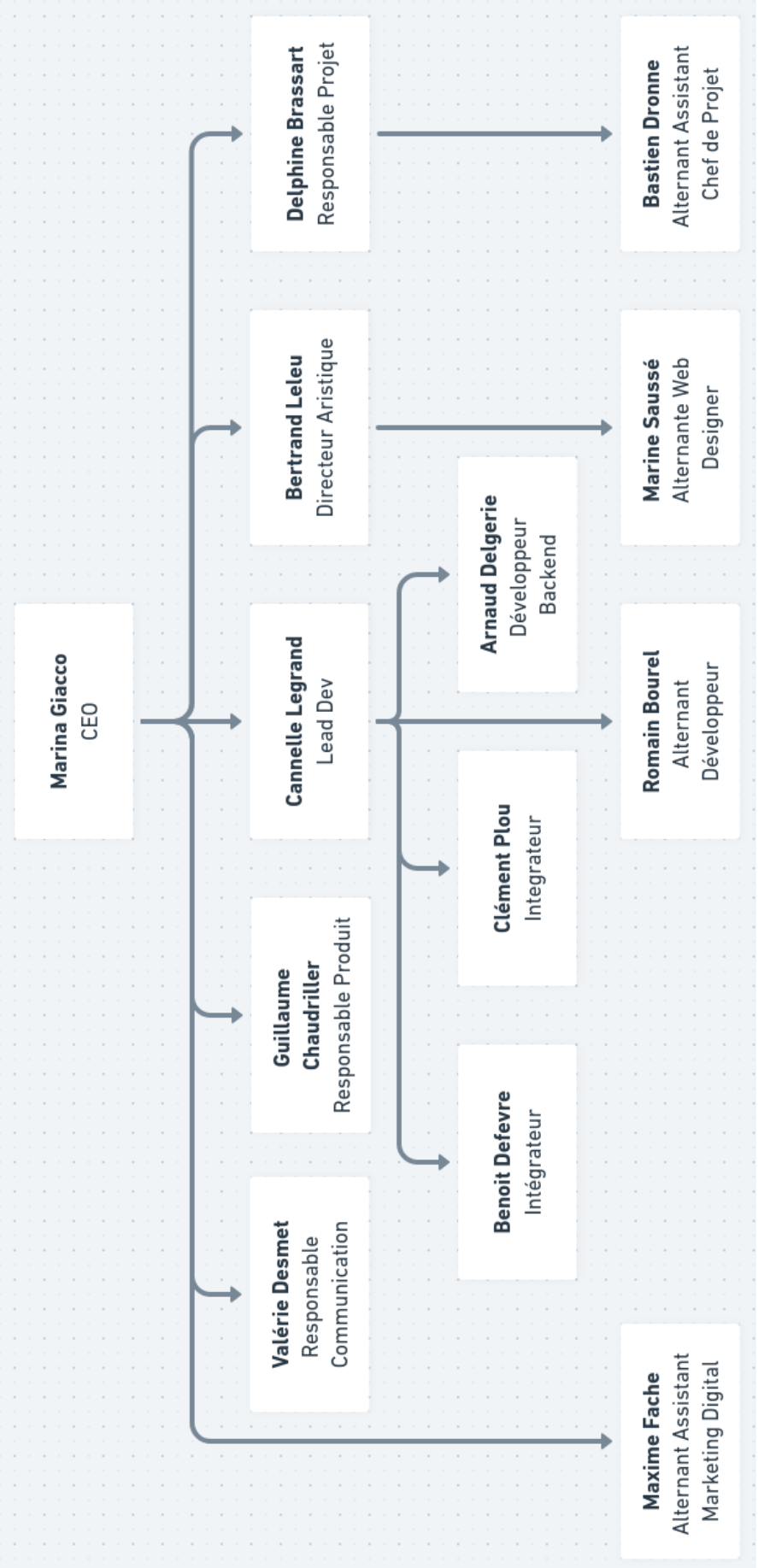
Forte de son expérience dans la gestion de sa communauté clients, l’entreprise grossit et se diversifie en proposant ByChall, une solution SaaS pour les grandes marques. Cette solution permet la création de plateformes communautaires, personnalisables, sécurisées, multilingues et responsives. Grâce à un grand nombre de modules administrables, ces plateformes permettent aux marques de co-créer avec leurs clients. Elles proposent une palette d’outils répondant aux différents besoins que peut avoir une entreprise en recherche de lien avec sa communauté : chroniques, configurateur de création, petites annonces, visio-conférence… sont autant de fonctionnalités qui sont venues accroître les possibilités.

Le but premier de ces plateformes est de permettre aux marques de se rapprocher de leurs clients et de pouvoir utiliser ce lien afin d’innover et proposer des offres et services adaptés au mieux à leur public.

Le site de Challangel continue de prospérer en faisant désormais partie de la liste des solutions SaaS.

Toutes les plateformes sont maintenues et améliorées par l’équipe de Challangel. De nouveaux modules viennent s’ajouter au fil du temps selon l’évolution des besoins et du secteur. Une partie de l’équipe s’occupe de la mise en avant de la solution auprès des clients, une autre de la réflexion et la création des nouveautés et les développeurs, dont je fais partie, de l’ajout et la mise en place de ces nouveautés.

Organigramme hiérarchique



Mes missions

Depuis mon arrivée, mes missions au sein de l'entreprise ont été variées.

La première d'entre elles fut l'installation de mon environnement de développement LAMP sous Apache 2, php7.4-fpm (nous sommes désormais passés à la version 8.0). Nous utilisons Symfony 5.4 avec webpack. Les vues sont gérées avec Twig et JS vanilla avec des restes de JQuery. Le Back Office du site, quant à lui, est fait avec le bundle Sonata.

J'ai principalement travaillé sur des tâches back-end de symfony. J'ai utilisé les Twig et JS mais je n'ai pas touché à la partie intégration du site.
L'organisation de mes missions se fait via des tickets sur Mantis ou gitLab.

Mantis :

Je me vois attribuer sur ce logiciel de suivi des bugs les différentes demandes de résolution. Elles sont reportées par l'équipe ou le client. Je dois ainsi comprendre le bug, trouver l'endroit où il se produit et analyser les logs. Cela est souvent évident mais nécessite parfois de longues heures pour tester et trouver pourquoi et comment celui-ci se produit. Une fois trouvé il faut réfléchir à la méthode permettant de le résoudre proprement et de manière compréhensible.

GitLab :

C'est sur GitLab que la lead dev répartit les nouvelles features à implémenter durant le sprint. Je dois prendre la prochaine tâche portant mon nom, en respectant l'ordre, et la déplacer de "Todo" à "Doing" le temps de la réaliser. Lorsqu'elle est réalisée, je l'envoie en validation où elle est relue. Après relecture, soit elle arrive dans "Done" soit elle revient dans "Todo". Ces features sont de nouvelles fonctionnalités avec le cahier des charges technique ainsi que les maquettes à mettre en place au besoin. Je dois comprendre ce qui est attendu et le réaliser. Si j'ai des questions sur les

attentes, je peux bien sûr les poser à l'équipe. Une fois ma partie finie, si celle-ci a le label “front-end” je l'assigne à l'intégrateur.

Workflow :

Nous utilisons Git en logiciel de versionning. Les fix (Mantis) sont faits sur des branches fix/xxxx et les features (GitLab) sur des branches feature/xxxx. Ces branches sont tirées depuis la branche dev. Il est possible parfois d'avoir des branches hotfix/xxxx pour les fix urgents posant un gros problème en production. Chaque fix et feature dispose d'un numéro d'identification (qui remplace ici les xxxx). Les différentes branches sont ensuite “incorporées”, on dit qu'on les merge, sur la branche dev puis testées et utilisées par l'ensemble de l'équipe, on appelle ça le recettage. Une fois ce recettage terminé on merge la branche dev sur la branche preprod, ce qui permet au client d'effectuer son propre recettage. Une fois cela terminé, on réitère l'action de la branche préprod vers la branche master. Pour les hotfix, la démarche est différente. On tire la branche hotfix depuis la master, on merge puis on fait redescendre le hotfix jusqu'à la branche dev.

Ce que j'ai fait et appris :

Grâce à cette formation en entreprise, j'ai pu améliorer mes connaissances et ma faculté de recherche au sein du repository mais également sur le web et dans la documentation. J'ai pu également travailler sur un projet en constante évolution faisant du multibranding. Cela veut dire que nous avons un code unique, commun à tous les sites que nous développons. Leur différenciation est rendue possible grâce à des fichiers particuliers : paramètres, trans, etc. Ces fichiers sont liés aux marques, ils vont surcharger le fonctionnement par défaut du site.

Par exemple :

- mettre le paramètre “challenge: enabled” de la marque X sur true permettra d'avoir le module de challenge activé sur leur site
- le paramètre “post: max-lenght” sur 400 limitera le nombre de caractères d'un post à 400.

Les différentes features et corrections que j'apporte doivent donc être réalisées en prenant en compte les nombreuses possibilités d'utilisation, m'obligeant à tester sur des configurations variées. Les fichiers trans.fr.yaml ou trans.en.yaml permettent,

quant à eux, de choisir le texte qui sera utilisé selon la langue, mais également selon la marque. Il y a une traduction par défaut qui peut être surchargée par rapport à la marque mais aussi la langue de l'utilisateur.

La structure du repository étant spécifique au projet de par son aspect multi-sites, il n'est pas possible d'utiliser les commandes du maker symfony permettant, par exemple, d'automatiser la création des entités, ce qui m'a obligé à comprendre plus précisément leur création et spécificités.

Voici maintenant une liste résumant différentes tâches réalisées au cours de l'année de manière générale puis plus précise :

- ajout/modification du wording dans le site.
- création/modification d'entités symfony.
- création de migrations afin de faire évoluer la base de données lors de certains changements dans les entités.
- utilisation de Sonata pour la gestion du Back Office.
- création de commandes pour les Cron Job.
- gestion du workflow d'un objet.
- création de KPI permettant de connaître différentes informations sur le site et son utilisation.
- création de DQL.
- création et modification du JS (écouteur d'événement, fonction asynchrone, class.
- implémentation de la résolution d'image HEIC
- gestion de la délivrance de mails (pas de mail, immédiat, date antérieure) au gagnant d'un challenge.
- création de widgets pour la page d'accueil du site.
- mise en place du téléchargement d'un rendez-vous en format ICS afin qu'il puisse se rentrer automatiquement dans l'agenda de l'utilisateur.
- mise en place d'une prévisualisation après avoir rempli le formulaire de création ou d'édition d'une annonce.
- mise en place de l'historisation du nombre de clics sur des boutons et liens désignés.
- …

Zoom sur une de mes tâches

Pour conclure, je vais maintenant vous présenter l'une des tâches que j'ai réalisée en autonomie avec la possibilité d'en discuter avec mes collègues. Celle-ci était de réaliser l'historisation d'un clic en base de données, afin de pouvoir afficher dans le BackOffice des statistiques sur les clics utilisateur.

J'ai choisi cette tâche car elle permet de montrer des détails techniques tout en étant facile d'accès pour un œil extérieur et rapidement explicable pour convenir au format demandé pour ce dossier.

Pour cette feature il m'était demandé d'historiser le nombre de clics sur différents éléments par jour, de manière à pouvoir plus tard rendre l'implémentation de cette historisation simple sur d'autres éléments.

Pour ce faire, j'ai d'abord réfléchi à la structure des tables. La question s'est posée de faire une seule table qui historisera tout les clics ou d'utiliser une table par entité lié, c'est à dire avoir une table pour les clics lié à l'entité utilisateur, une table pour les clics d'un post de blog, une pour les challenges… J'ai choisi de partir sur plusieurs tables afin d'éviter d'avoir une table avec d'innombrable champs de valeur nulle.

Pour que cela soit simple d'implémentation j'ai décidé de faire :

- un fichier JS qui détecte grâce à un dataset particulier les boutons à historiser,
- une entité abstraite que l'on aurait plus qu'a étendre pour n'avoir besoin que de surcharger la relation avec l'entité écoutée,
- un service qui automatise le travail des méthodes de route du controller simplifiant l'implémentation d'une nouvelle route.

Pour commencer, voici les trois éléments qu'il faudra créer lorsque l'on voudra faire une historisation de clics sur un nouvel élément :

...views/Discover/detail.html.twig

```
<button
  id="messaging_form_estimate"
  class="detail__product-actions__link detail__product-actions__link--estimate cta-link"
  data-type="estimate"
  data-object-id="{{ creation.id }}"
  data-action="{{ path('discussion_messages_show', {'recipient': creation.portfolio.user.id, 'id': creation.id, 'type': 'estimate'})
}}"
  data-click-history="{{ path('cha_click_history_creation', {type: constant('Cha\\UserBundle\\Entity
\\UserPortfolioCreationClickHistory::TYPE_ESTIMATE'), id: creation.id}) }}"
  data-click-history-listener="false"
>
    {{ source('@icons/icon-arrow-right.svg') }}{{ 'portfolio.ask_estimate'|trans }}
</button>
```

1) Ci-dessus l'ajout sur le bouton des datasets :

- "data-click-history" comprenant l'url de la route utilisée avec pour arguments l'id de l'objet et le type utilisé.
- "data-click-history-listener" égal à "false" qui passera à "true" une fois bindé pour éviter de lui lier plusieurs écouteurs de clics.

'UserPortfolioCreationClickHistory.php

```
namespace Cha\UserBundle\Entity;

use Cha\GeneralBundle\Entity\AbstractEntity\AbstractClickHistory;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Table(name="brand_user_portfolio_creation_click_history")
 * @ORM\Entity(repositoryClass="Cha\UserBundle\Repository\UserPortfolioCreationClickHistoryRepository")
 */
class UserPortfolioCreationClickHistory extends AbstractClickHistory
{
    public const TYPE_PRODUCT = 'product';
    public const TYPE_PRODUCT_HIGHLIGHTED = 'product_highlighted';
    public const TYPE_BRAND = 'brand';
    public const TYPE_BRAND_HIGHLIGHTED = 'brand_highlighted';
    public const TYPE_PROFIL = 'profil';
    public const TYPE_ESTIMATE = 'estimate';

    public const TYPES = [
        self::TYPE_PRODUCT,
        self::TYPE_PRODUCT_HIGHLIGHTED,
        self::TYPE_BRAND,
        self::TYPE_BRAND_HIGHLIGHTED,
        self::TYPE_PROFIL,
        self::TYPE_ESTIMATE,
    ];

    /**
     * @ORM\ManyToOne(targetEntity="Cha\UserBundle\Entity\UserPortfolioCreation", inversedBy="userPortfolioCreationClickHistories")
     */
    private ?UserPortfolioCreation $object =  null;

    public function getObject(): ?UserPortfolioCreation
    {
        return $this->object;
    }

    public function setObject(?UserPortfolioCreation $object): self
    {
        $this->object = $object;
        return $this;
    }
}
```

2) Ci-dessus, on retrouve la création de l'entité avec le champ objet, son getter et son setter. Le setter renvoie "$this" afin de pouvoir faire du "fluent setter". On y ajoute différentes constantes que nous pourrons alors utiliser. Cela permettra une meilleure maintenabilité. Si on fait évoluer une valeur, on aura

alors besoin de la changer uniquement à l'endroit où elle est définie. Ces constantes sont les différents types de clics historisés ainsi qu'une constante TYPES renvoyant un tableau avec tous les types de clics présents dans l'entité.

.../Controller/UserController.php

```php
/**
     * @Route(
     *      "user/click/history/{type}/{id}",
     *      name="cha_click_history_user",
     *      condition="request.isXmlHttpRequest()",
     *      requirements={"type"="\w+", "id"="\d+"},
     *      methods={"POST"}
     *      )
     */
    public function addCreationClick(string $type, User $user, ClickHistoryService $clickHistoryService): JsonResponse
    {
        $response = $clickHistoryService->addClick($user,  $type, UserClickHistory::class);
        return $this->json($response['data'], $response['status']);
    }
```

3) Pour finir, nous créons la route, présentée ci-dessus. Elle sera mise dans le controller qui utilisera les arguments de route et le ClickHistoryService que nous verrons plus bas.

Maintenant rentrons plus dans le détail de ce qui fait fonctionner tout cela.

assets/common/js/lib/click_history.js

```javascript
export default class ClickHistory {
    constructor() {
        document.querySelectorAll('[data-click-history-
listener="false"]').forEach(this.initClickListener);
    }


    initClickListener = (clickListener) => {
        clickListener.dataset.clickHistoryListener = 'true';
        clickListener.addEventListener('click', this.onClickCount);
    }


    onClickCount(e) {
        const redirect = 'true' === this.dataset.clickHistoryRedirect;
        if (redirect) {
            e.preventDefault();
        }
        fetch(this.dataset.clickHistory, {
            method: 'POST',
            headers: {'X-Requested-With': 'XMLHttpRequest',},
        })
            .then((response) => {
                if (redirect) {
                    window.location.assign(this.href);
                }
            })
    }
}
```

J'ai créé une class JS qui ajoute un listener / écouteur de clic sur toutes les balises comportant le "data-click-history-listener" égal à false. Cela permet d'envoyer en asynchrone la requête permettant d'historiser le clic.
Si la balise cliquée à un "data-click-history-redirect", un changement de page aura lieu une fois la requête effectuée.

.../ClickHistoryInterface.php

```php
namespace Cha\GeneralBundle;

interface ClickHistoryInterface
{
    public function getObject();

    public function getType(): ?string;

    public function setType(string $type): self;

    public function getNumberOfClick(): ?int;

    public function setNumberOfClick(int $numberOfClick): self;

    public function addClick(): self;

    public function getCreatedAt(): ?\DateTimeInterface;

    public function setCreatedAt(): self;
}
```

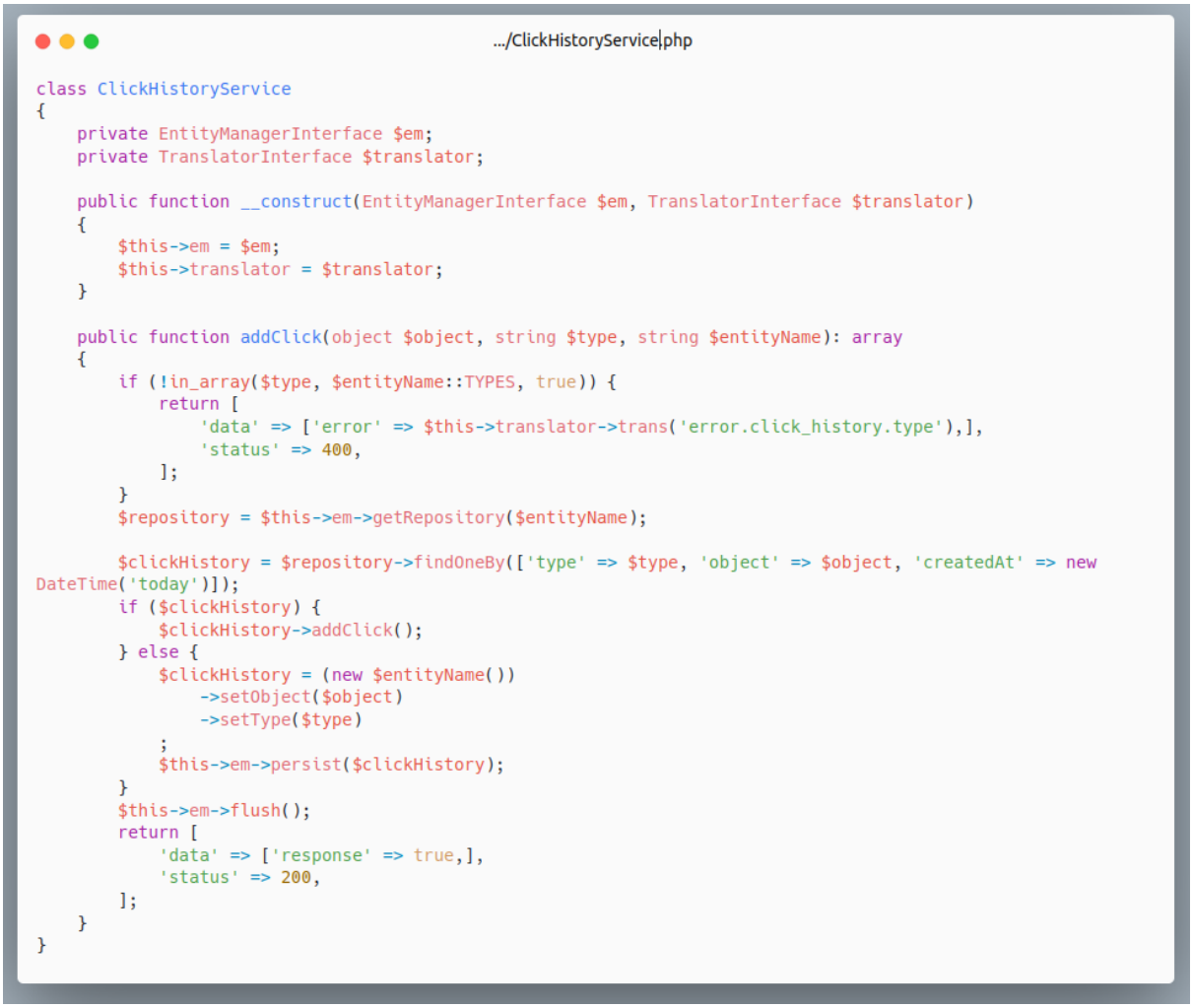
J'ai créé une interface regroupant toutes les fonctions dont l'entité a besoin.Cela évitera d'en oublier une en normant les développeurs lors de la création d'une nouvelle entité.

.../AbstractClickHistory.php

```php
/**
 * @ORM\MappedSuperclass()
 * @ORM\HasLifecycleCallbacks()
 */
abstract class AbstractClickHistory implements BrandEntityInterface, ClickHistoryInterface
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected ?int $id = null;

    /**
     * @ORM\Column(name="type", type="string", length=255)
     */
    protected ?string $type = null;

    /**
     * @ORM\Column(name="number_of_click", type="integer")
     */
    protected ?int $numberOfClick = null;

    /**
     * @ORM\Column(name="created_at", type="datetime")
     */
    protected \DateTimeInterface $createdAt;

    public function __construct()
    {
        $this->numberOfClick = 1;
    }

    public function getType(): ?string
    {
        return $this->type;
    }


    public function setType(?string $type): self
    {
        $this->type = $type;
        return $this;
    }

    public function getNumberOfClick(): ?int
    {
        return $this->numberOfClick;
    }

    public function setNumberOfClick(?int $numberOfClick): self
    {
        $this->numberOfClick = $numberOfClick;
        return $this;
    }

    public function addClick(): self
    {
        $this->numberOfClick++;
        return $this;
    }

    public function getCreatedAt(): ?\DateTimeInterface
    {
        return $this->createdAt;
    }

    /**
     * @ORM\PrePersist()
     */
    public function setCreatedAt(): self
    {
        $this->createdAt = new \DateTime('today');
        return $this;
    }
}
```

J'ai créé une class abstraite qui regroupe les champs communs à toutes les entités d'historisation de clics, ainsi que leurs getters et setters. Elle implémente mon interface décrite plus haut et BrandEntityInterface qui nous permet de lier l'entité à la table de la bonne marque (pour rappel le code et commun à tous les sites).

Il est à noter que le setCreatedAt dispose d'un événement PrePersist permettant de définir la date de création juste avant de le persister en base de données



J'ai également construit un service “ClickHistoryService” ci-dessus, avec une méthode addClick() à laquelle on passe les arguments de l’objet, son type et le nom de l’entityClickHistory.

Cela va permettre de définir s'il y a déjà eu un clic sur ce type de balise. On commence par vérifier si le type fait bien partie des types de l’entité. Ensuite on recherche avec un findOneBy s’il y a un objet correspondant à l’id, le type et la date du jour. Si un clic a déjà été fait ce jour, il sera déjà stocké en base de données et il nous restera à l'incrémenter. Sinon, on le crée et lui donne les informations nécessaires (son type et l’id de l’objet). On n’oublie pas de le persister en base de données (le createdAt se remplira automatiquement comme expliqué plus haut). Puis une fois la donnée sauvegardée, on renvoie le tableau qui sera utilisé par la route.

Il y a eu deux ou trois autres détails réalisés comme l’import du JS aux endroits où l’on en a besoin, la création d’une clef de traduction pour le message d’erreur, la

commande d’update de la base de données et l’ajout des datasets sur chaque balise en ayant besoin.

Cette feature s’est suivie de la création d’une seconde entité d’historisation de clics qui fut rapide à ajouter et de la création des KPI utilisant les données ainsi récoltées. J’ai donc dû créer la page en Back Office montrant différents tableaux et créer des DQL qui sont les requêtes que l’ORM doctrine traduit en requête SQL préparée

Conclusion

Pour conclure je suis heureux d’avoir fait cette reconversion. Elle me donne accès à un métier vaste et intéressant que j’aime pratiquer. Cette année fut pour moi très enrichissante. J’ai approfondi mes connaissances en PHP, JS et Symfony en entreprise et en cours. J’ai également découvert MongoDB et React durant les cours.

J’ai rejoint une équipe soudée avec une bonne ambiance dans laquelle je me sens bien. J’aime le fait de voir petit à petit mes ajouts devenir une fonctionnalité et de rechercher le moyen d’arriver au bout de la tâche demandée.

Mes envies pour la suite ne sont pas encore figées. J’aimerai trouver une entreprise pour continuer sur ma lancée en Symfony mais reste ouvert à d'autres langages. J'aime aussi les interactions en JS et suis intéressé par les avancées en IA. Pour demain, je souhaite rester sur le développement web mais le fait d’aller vers du machine learning dans un futur plus lointain est fort probable.

Merci de m’avoir lu.