

RAPPORT DE STAGE

du lundi 20 juillet au mercredi 26 août 2020



Les Bulles de Liberté

10 Cité Fieux, 71100 SAINT-RÉMY

Maître de Stage : Mme Paola VELON-COMTE,
Présidente de l'association

Marceau GÉRARD

Session WebForce3 Le Creusot – Mars 2020

Remerciements

Je voudrais remercier l'équipe de WebForce3 : M. BADAIRE, M. BARTHE, nos formateurs M. DEMON, Mme. DONJON et M. BOURILLOT, pour leur disponibilité, leur soutien et leur écoute, malgré ce contexte compliqué de confinement.

Je remercie aussi tous mes comparses de formation, avec qui j'ai appris à travailler en équipe, à distance, à utiliser nos forces respectives dans la complémentarité pour créer des projets où nous avons pu trouver un sens à notre participation.

Je tiens enfin à remercier la région de Bourgogne-Franche-Comté et Pôle Emploi, qui m'ont permis de suivre cette formation de Développeur Web & Web Mobile, qui me donne bon espoir pour la suite de ma vie professionnelle.

Table des matières

| | |
|--|------|
| 1. Introduction | p.4 |
| 2. Liste des compétences du référentiel couvertes par le projet | p.5 |
| 3. Résumé du projet | p.6 |
| 4. Cahier des charges, expression des besoins, spécifications fonctionnelles | p.7 |
| 5. Spécifications techniques du projet | p.8 |
| 5.1 Authentification | p.8 |
| 5.2 Gestion des Données de l'Utilisateur | p.8 |
| 5.3 Hébergement | p.9 |
| 5.4 Événements | p.10 |
| 6. Système de participation en détail | p.12 |
| 6.1 Côté back-end | p.12 |
| 6.2 Côté front-end | p.14 |
| 7. Sécurité du projet | p.17 |
| 8. Galerie d'images et carrousel : une utilisation de jQuery et de Twig | p.18 |
| 9. Possibilités d'amélioration | p.21 |
| 10. Conclusion | p.22 |
| 11. Annexe | p.23 |

1. Introduction

Avec un Baccalauréat Général Littéraire en poche, j'ai décidé en 2010 de quitter la Bourgogne pour commencer des études de Cinéma. Déjà, les aspects technique et créatif du montage vidéo m'attiraient énormément.

Également, étant multi-instrumentiste autodidacte, j'ai pu me confronter à différents logiciels de MAO (Musique Assistée par Ordinateur), et ainsi, me rendre compte de mon affinité particulière avec l'informatique.

Pour des raisons financières et familiales, je suis revenu en Saône-et-Loire, où j'ai travaillé dans la grande distribution de 2016 à 2019 en tant que préparateur de commandes. Mais le caractère épuisant et aliénant de cet emploi m'a convaincu qu'il fallait que je me construisse un projet professionnel plus en adéquation avec mes capacités et mes valeurs.

2. Liste des compétences du référentiel couvertes par le projet

Je vais ici dresser la liste des compétences professionnelles demandées dans le référentiel du titre professionnel Développeur Web et Web Mobile.

La suite de ce rapport contiendra des explications sur comment chacune de ces compétences a été utilisée sur le site créé durant ce stage.

Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Développer la partie back-end d'une application web ou web mobile

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

Compétences transversales

- Utiliser l'anglais dans son activité professionnelle
- Actualiser et partager ses compétences

3. Résumé du projet

Après ces quatre mois à WebForce3, à l'IUT du Creusot, puis à domicile, j'ai choisi Les Bulles de Liberté pour faire mon stage professionnel de 6 semaines, du 20 juillet au 26 août 2020.

Les Bulles de Liberté est une association (loi 1901) ayant pour but d'aider les personnes en situation de handicap mental, en leur proposant des activités, en compagnie des familles, ainsi que de personnes issues du milieu médico-social. La présidente de l'association m'a accompagné durant ce stage en télétravail, pour créer entièrement un site. Le vice-président, M. DE JESUS m'a quant à lui accompagné pour définir l'identité visuelle du site.

Leur envie était d'avoir un support numérique pour promouvoir leur association, faciliter l'accès aux dons, et échanger des documents avec les différents acteurs, autrement que par l'échange de mails, très chronophage et énergivore, mais de manière plus formelle que par de simples conversations sur les réseaux sociaux.

4. Cahier des charges, expression des besoins et spécifications

Le but principal du site, exprimé par Mme VELON-COMTE, était de faciliter les échanges avec et entre les bénévoles et autres participants aux événements :

- leur proposer un espace personnel, où ils peuvent renseigner et modifier leurs coordonnées ;
- récolter leurs participations afin de mieux répartir les tâches.

Le site devait aussi proposer d'autres fonctionnalités :

- envoi de documents pour les foyers d'accueil (affiches à imprimer), ou pour les familles (autorisations, droit à l'image) ;
- galerie de photos des événements passés ;
- redirection vers la cagnotte HelloAsso déjà créée.

J'ai eu accès à plusieurs ressources de la part du vice-président :

- logo de l'association en différents formats ;
- maquette de la page d'accueil, en version mobile et PC, avec ma participation (**voir annexe 1**) ;
- polices d'écriture utilisées ;
- affiches d'événements.

Durant notre première réunion, nous avons défini un plan du site, avec les différentes pages et ce qu'elles contiennent. Enfin, je leur ai introduit le concept de diagramme de base de données afin de pouvoir rapidement créer un schéma UML (**voir annexe 2**)

J'ai décidé de coder le site en utilisant *Symfony*, un framework MVC que j'ai découvert durant ma formation. Son adaptabilité et sa facilité d'utilisation me semblaient convenir aux besoins définis ci-dessus.

5. Spécifications techniques du projet

5.1 Authentification

Le site comporte un « espace bénévole », qui permet aux concernés par le biais de différents formulaires, de s'inscrire, de se connecter et de modifier des informations personnelles (numéro de téléphone, mot de passe, etc). Une grande partie de ce système d'authentification a été installé automatiquement via la commande *php bin/console make:auth* via la console.

L'inscription demande plusieurs champs (dont certains obligatoires), dont le champ d'adresse e-mail qui est utilisé comme identifiant pour la connexion. L'administrateur du site devra aussi se créer un compte pour profiter pleinement des ressources qui lui sont proposées et dont il sera question un peu plus bas dans ce document.

La connexion demande à l'utilisateur l'adresse e-mail, enregistrée comme identifiant unique dans la base de données, ainsi que le mot de passe. Un système de gestion d'erreurs peut informer à l'utilisateur si son compte n'existe pas, ou si le mot de passe n'est pas le bon (**voir Annexe 3**).

5.2 Gestion des Données de l'Utilisateur

Le formulaire d'inscription demande à l'utilisateur de renseigner plusieurs données personnelles qui seront stockées dans la Base de Données, il faut donc veiller à respecter le Règlement Général de Protection des Données, ou **RGPD**. Il s'agit d'un texte réglementaire européen qui encadre le traitement des données sur tout le territoire de l'Union Européenne. Ce texte est entré en application le 25 mai 2018.

Ce règlement définit une donnée personnelle comme étant « toute information se rapportant à une personne physique identifiée ou identifiable. Il existe 2 types d'identifications :

- identification directe (nom, prénom etc.)
- identification indirecte (identifiant, numéro etc.). »

Afin de respecter au maximum ce règlement, j'ai veillé à ce que seules des données nécessaires à l'association soient récupérées, à ce que certaines données (le numéro de téléphone) soient facultatives lors de l'inscription, et à ce que toutes les

informations recueillies par l'utilisateur lui soient accessibles (via la page de profil), à la consultation, à la rectification et à la suppression.

Pour des raisons de sécurité, j'ai rappelé lors de notre dernier entretien avec M. DE JESUS que les données qu'ils collecteront devront être sécurisées. L'accès à la Base de Données devra être protégé par un mot de passe suffisamment fort : au moins 8 caractères, 1 majuscule et 1 caractère spécial.

Enfin, je leur ai donné toutes les informations relatives à la rédaction des mentions légales, car un site d'association de loi 1901 doit y renseigner l'identité de l'association et les moyens ouverts à l'utilisateur de la contacter. Il faut aussi avertir l'utilisateur de la politique de collecte et de traitement des données personnelles mise en œuvre par l'association.

L'identité de l'association demeure dans tous les cas dans le *footer* du site (bas de page), ainsi qu'un lien vers la page de mentions légales dans le plan du site. Ce lien est aussi accessible depuis la page d'inscription, avec une case à cocher pour accepter les Conditions Générales d'Utilisation, car les mentions légales de l'association peuvent figurer sur les CGU du site. L'inscription est impossible si ces conditions ne sont pas acceptées.

5.3 Hébergement

Le site n'étant à l'heure actuelle, pas entièrement terminé, il n'est pas encore hébergé. Mais j'ai l'intention de le terminer après ma période de stage, puis de l'héberger en passant par OVHcloud, en compagnie de M. DE JESUS. En lui proposant les services de cette entreprise française, il m'a donné son accord pour que nous passions par elle. Leurs offres garantissent une compatibilité avec la version 7.4 de PHP (que j'ai utilisé), une base de données SQL, et une protection anti-DDoS (une attaque DDoS vise à rendre indisponible un ou plusieurs services vulnérables, en saturant la bande passante par exemple).

5.4 Événements

La gestion des événements fait partie des éléments les plus importants du site. Visibles directement sur la page d'accueil, les quatre événements les plus récents sont affichés dans des « cards » (encart rectangulaire) importés depuis le framework Bootstrap, permettant d'adapter le contenu en fonction du format du support depuis lequel on accède au site (responsive design) :

- Les quatre événements s'affichent les uns en dessous des autres en format mobile, avec affichage de l'image principal, et du titre par dessous avec les dates, sur un overlay permettant de mieux détacher le texte de l'image.
- À partir du format tablette (soit 768 pixels de large), les quatre *cards* s'organisent en colonne de deux, avec l'image principal en haut, le titre et les dates en dessous.
- Enfin, le format PC (à partir de 1200 pixels) organise les quatre *cards* sur une seule ligne, séparées d'une marge, pour aérer le contenu.

Cette gestion de l'affichage (ou non) des éléments constitutifs d'une *card* n'est pas uniquement gérée par les classes *Bootstrap*, j'ai ajouté du CSS pour correspondre au maximum à la maquette, par le biais des Media Queries, un module CSS3. Je m'en suis servi pour conditionner le contenu selon la largeur de l'écran.

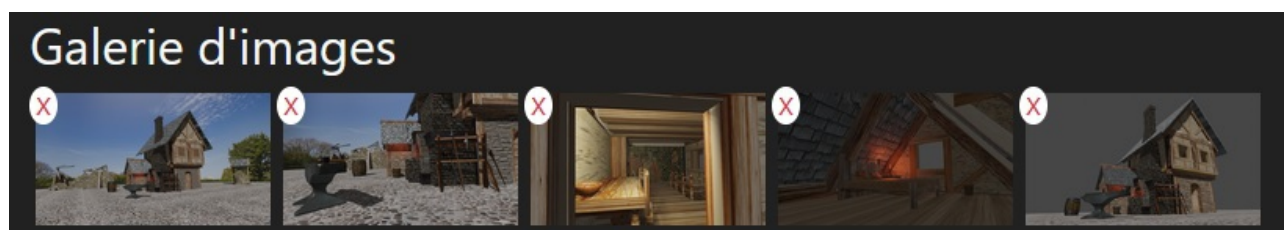
L'entité *événement*, appelée « article » dans la base de données, est une des plus fournies (**voir Annexe 4**). L'administrateur (car il n'y a que lui qui ait la permission d'en créer) doit renseigner un titre, une description, une photo principale et deux dates (une de début et une de fin).

Les champs *description* et *détails* apparaissent visuellement comme un éditeur de texte, grâce au bundle *CKEditor*, dans sa configuration la plus basique. Cette offre permet à l'administrateur de mieux présenter son texte, pour un rendu plus agréable et lisible.

Il peut aussi ajouter plusieurs photos, qui seront ensuite visibles dans la page d'un seul Événement, et cliquables pour les afficher dans un carrousel, pour en avoir une version agrandie. Ces photos se retrouveront aussi dans la page *Galerie* du site avec les mêmes fonctionnalités. Ce carrousel est, comme les *cards* importé grâce à Bootstrap.

Le champ optionnel *missions* est également un champ de texte, mais celui-ci ne sera visible que par les utilisateurs connectés, car il est voué à contenir des informations qui concernent l'aspect organisationnel. Cette restriction est possible grâce à Twig, un moteur utilisé dans les vues de Symfony.

L'administrateur a aussi la possibilité de modifier les données d'un événement, dans un formulaire dédié et interactif, notamment dans le système de suppression de photos de la galerie : quand on clique sur la croix, un message nous demande de confirmer ce choix (pour éviter les erreurs). S'il est validé, la photo se supprime sans que la page ne recharge, grâce à une requête AJAX (AJAX est une technologie utilisant Javascript, dite asynchrone : le code continue de se lire quand l'appel vers le serveur est effectué).



J'ai enfin utilisé du JSON pour les participations des bénévoles, dont je parle plus en détail dans la suite de ce dossier. JSON est un format de données textuelles utilisable en Javascript. Ci-dessous, le tableau JSON créé lors de l'ajout de participation de la part d'un utilisateur :

| JSON | Données brutes | En-têtes |
|-----------------|-------------------------|------------------------------|
| Enregistrer | Copier | Tout réduire Tout développer |
| code: | 200 | |
| message: | "Participation ajoutée" | |
| participations: | 1 | |

6. Système de participation en détail

La gestion des participations aux événements était très certainement la fonctionnalité dont Mme VELON-COMTE et M. DE JESUS avaient le plus besoin, car, l'association grandissant, il devient de plus en plus difficile d'organiser quelque chose sans s'ajouter un travail de préparation considérable. J'ai donc apporté un soin particulier à l'élaboration du système de participation, tant du côté des bénévoles, que du côté des administrateurs.

6.1 Côté Back-end...

J'ai tout d'abord créé une entité *Participation*, sa table correspondante dans la base de données et le Repository (un Repository ou *Repo* est une classe qui permet de faire le lien entre une entité et sa table). L'objet *Participation* reçoit deux choses (en dehors de son propre id) : l'id de son *Article* (qui réfère à son événement), et l'id de son *User* (qui réfère à l'utilisateur). (**voir Annexe 5**)

Dans le Contrôleur, j'ai créé une fonction *participation()*, dont un des paramètres de route définit l'URL contenant l'id de l'*Article*, qui sera récupéré grâce à la méthode GET. Cette fonction sera appelée lorsque l'utilisateur connecté cliquera sur le lien du bouton pour ajouter ou enlever sa participation, car il enverra l'id en données GET dans l'URL.

```
/**
 * Permet à un user d'ajouter ou d'enlever sa participation
 *
 * @Route("/evenement/{id}/participation", name="event_participation")
 *
 * @param \App\Entity\Article $article
 * @param \Doctrine\ORM\EntityManagerInterface $manager
 * @param \App\Repository\ParticipationRepository $participationRepo
 * @return \Symfony\Component\HttpFoundation\Response
 */
public function participation(Article $article, EntityManagerInterface $manager, ParticipationRepository $participationRepo) : Re:
{
```

J'ai utilisé l'instruction *if* pour gérer les conditions suivantes : si l'utilisateur n'est pas connecté, on crée un tableau JSON avec le code d'erreur et un message. Par contre, si l'utilisateur est effectivement connecté, deux possibilités s'offrent à nous : soit il ne participe pas à l'événement (c'est-à-dire qu'il n'existe aucune entrée dans la table *Participation* reliant cet utilisateur à cet *Article*), et donc il faut ajouter sa participation, soit au contraire, l'utilisateur participe déjà et dans ce cas il faut retirer sa participation.

```
551         // On récupère l'user
552         $user = $this->getUser();
553
554         // S'il n'est pas connecté
555         if(!$user){
556             // Tableau JSON avec le code d'erreur et le message
557             return $this->json([
558                 'code' => 403,
559                 'message' => 'Il faut être connecté'
560             ], 403);
561         } else {
562             // ...
563         }
564     }
```

Je me suis ensuite posé la question : « *comment peut-on savoir si un utilisateur a ajouté sa participation à un événement ?* » Pour y répondre, j'ai créé une nouvelle fonction *willCome()* dans l'entité *Article*, qui prend en paramètre une instance de la classe *User*, et qui retourne un booléen (*oui* ou *non*, *vrai* ou *faux*, *1* ou *0*).

Pour chaque participation d'un article (on parcourt chaque participation grâce à la commande *foreach()*), on compare l'id de l'*User* qui participe avec celui qui est connecté. S'il y a une concordance, la fonction retournera un *true*, sinon un *false*.

```
211     /**
212      * Permet de savoir si un User participera à cet événement
213      *
214      * @param User $user
215      * @return boolean
216      */
217     public function willCome(User $user) : bool
218     {
219         foreach ($this->participations as $participation){
220             if($participation->getUser() === $user){
221                 return true;
222             }
223         }
224         return false;
225     }
```

De retour dans le contrôleur, dans la partie où l'utilisateur est connecté, j'ai donc utilisé cette fonction *willCome()*. Ainsi, dans l'éventualité où l'utilisateur participe déjà, j'ai utilisé le Repository de *Participation* pour récupérer sa participation, et l'enlever de la base de données.

Il faut aussi mettre à jour l'Article pour décrémenter (réduire de 1) le nombre de participations. On peut enfin envoyer un tableau JSON avec le code de succès et le message. (**voir Annexe 6**)

À l'inverse, si l'utilisateur ne participe pas déjà, on ajoute une *Participation*, en l'« hydratant » (complétant) avec l'Article et l>User qui correspondent, et on incrémente (augmente de 1) le nombre de participations dans l'Article, sans oublier le tableau JSON. (**voir Annexe 7**)

6.2 Côté Front-end...

Pour traiter la requête, je me suis servi d'axios, dont j'avais découvert l'existence durant ma formation, sur mon projet de soutenance. Axios est une bibliothèque JavaScript fonctionnant comme un client HTTP. J'ai commencé par télécharger la version minifiée du fichier, pour l'appeler dans une balise script uniquement dans les vues nécessitant son utilisation : la page listant les événements à venir, et la page d'un article en particulier. J'ai d'ailleurs choisi de ne pas afficher le bouton de participation sur la page des événements futurs en version mobile, car l'ajout d'un bouton sur le peu de place que prend une *card* aurait rendu l'expérience utilisateur plus difficile selon moi.

Dans le fichier script.js, j'ai d'abord créé un écouteur d'événement sur chaque lien de bouton de participation, au clic principal de la souris. Cet écouteur appelle la fonction *onClickBtn()*, que nous expliquerons juste après, et qui se situe juste avant l'écouteur dans le document.

```
30 // On sélectionne les boutons de participation et on leur ajoute un écouteur
31 // d'évènement au clic, en appelant la fonction qu'on a créé ci-dessus
32 document.querySelectorAll(".participation-link").forEach(function(link){
33     link.addEventListener('click', onClickBtn);
34 });
```


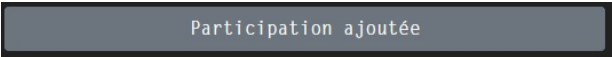


JE PARTICIPE !

Dans cette fonction *onClickBtn()*, il faut d'abord empêcher le chargement de la page de participation, qui s'active normalement lors du clic sur le lien, avec la méthode *preventDefault()*. Puis, j'ai créé une méthode qui récupère l'URL du lien cliqué, pour la donner à *axios*. Si la requête a bien fonctionné, alors on autorise le rafraîchissement de la page avec *location.reload()*. Sinon, en cas d'erreur, on affiche un message d'alerte générique avec la méthode *alert()*. Si le code statut récupéré depuis le tableau JSON est 403, cela signifie qu'il n'y a aucun utilisateur connecté, alors on affiche un message particulier. (**voir Annexe 8**)

Enfin, dans ce fichier *script.js*, j'ai ajouté une autre fonction qui permet aux utilisateurs PC de changer l'aspect visuel de chaque bouton « Participation ajoutée » lors du passage de la souris, grâce aux classes Bootstrap. Le contenu textuel du bouton change alors en même temps grâce à la fonction jQuery *textContent*.

```
30 // On sélectionne les boutons de participation et on leur ajoute un écouteur
31 // d'évènement au clic, en appelant la fonction qu'on a créé ci-dessus
32 document.querySelectorAll(".participation-link").forEach(function(link){
33     link.addEventListener('click', onClickBtn);
34 });
35
36 // Au passage de la souris sur le bouton "Participation ajoutée", on change le bouton
37 document.querySelectorAll(".participated").forEach(function(e){
38     e.addEventListener('mouseenter', function(mouse){
39         mouse.target.classList.replace("btn-secondary", "btn-danger");
40         mouse.target.textContent = "Annuler ma participation";
41     });
42     e.addEventListener('mouseleave', function(mouse){
43         mouse.target.classList.replace("btn-danger", "btn-secondary");
44         mouse.target.textContent = "Participation ajoutée";
45     });
46 })
```



L'utilité de cette fonction pourrait être simplement esthétique, mais je pense qu'il permet de mieux comprendre la double utilité de ce bouton, s'inscrire et se désinscrire d'un événement. Le simple bouton gris « Participation ajoutée » me semblait moins explicite.

Je vais finir ce tour d'horizon du système de participation par le fichier html géré par Twig où se situe le bouton de participation.

Participer à un événement c'est bien, mais participer à un événement qui n'est pas déjà passé, c'est encore mieux ! Sur le site des Bulles de Liberté, j'ai choisi de différencier les événements futurs et passés en fonction de la date de fin renseignée par l'admin : si cette date est ultérieure à la date actuelle, le bouton de participation s'affiche.

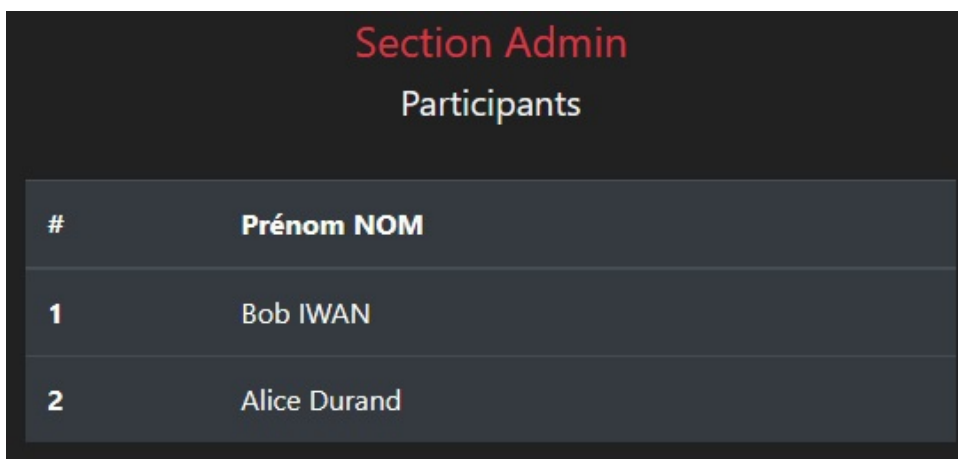
Le cas échéant, j'ai conditionné le contenu du lien vers la participation en fonction de deux choses :

- si l'utilisateur est connecté et qu'il participe déjà (Twig peut utiliser la fonction *willCome()* détaillée plus haut), le contenu textuel du bouton sera « Participation ajoutée »
- sinon (si l'utilisateur n'est pas connecté OU qu'il ne participe pas) le contenu textuel du bouton sera « JE PARTICIPE ! » (**voir Annexe 9**)

La capture d'écran en annexe montre que l'administrateur aura accès à des informations supplémentaires : d'abord le nombre total de participants. J'avais prévu de laisser ce nombre visible à toutes les bénévoles connectés, mais lors de notre réunion de fin de stage, M. DE JESUS m'a demandé de restreindre cette information pour éviter des biais éventuels. L'accord au pluriel du mot *participant* est effectif lorsque le nombre de participants dépasse 1, grâce à une autre condition *if* en Twig.

Enfin, s'il y a au moins un participant, l'administrateur verra un tableau avec le prénom et le nom de chaque participant. Une boucle *for* ajoute une ligne au tableau pour chaque participant.

(**voir Annexe 10**)



| Section Admin | |
|---------------|--------------|
| Participants | |
| # | Prénom NOM |
| 1 | Bob IWAN |
| 2 | Alice Durand |

7. Sécurité du projet

Un des gros avantages de Symfony est la gestion de la sécurité grâce au bundle *Security*. Ce bundle est installé d'office sur un projet Symfony neuf, mais il se met à jour à partir de la création d'une entité *User*. Si on ouvre le fichier *security.yaml*, on peut y trouver plusieurs choses :

- La présence d'un encodeur de mot de passe : cela permet de ne pas stocker directement en clair les mots de passe des utilisateurs dans la base de données, mais un hashage du mot de passe. Dans *security.yaml*, l'algorithme utilisé est automatiquement Sodium, qui utilise Argon2 comme fonction de dérivation de clé.
- Une hiérarchie des rôles utilisateurs : c'est ce qui permet de différencier un utilisateur lambda d'un utilisateur connecté, ou d'un administrateur. Le bundle *security* va récupérer cette information dans la table *roles* dans la base de données. Cette table est constituée d'un tableau JSON : si l'utilisateur connecté a un tableau JSON comprenant "*ROLE_ADMIN*", *security* comprend que l'utilisateur connecté est un admin. Si ce tableau comprend "*ROLE_USER*" ou un tableau vide, c'est un utilisateur classique, dans notre cas, un bénévole.
- La redirection vers une page lors de la déconnexion de l'utilisateur.

8. Galerie d'images et carrousel : une utilisation de jQuery et de Twig

La page *Galerie* est sûrement celle qui m'a donné le plus de fil à retordre. Elle devait recevoir toutes les images (secondaires, pas la principale qui est affichée sur les cards) de tous les articles. Chaque image devait être cliquable et ouvrir un carrousel d'images.

Or, ce carrousel ne devait pas contenir les images de l'intégralité de la page *Galerie*, mais uniquement celles de l'article relié à l'image cliquée. De surcroît, il fallait que cette image soit celle affichée en premier dans le carrousel, grâce à la classe Bootstrap « *active* » sur sa *div* parente.

Pour récupérer dans le script l'image cliquée seule, j'ai choisi d'ajouter l'attribut *id* à la balise *img*, mais étant donné que l'affichage des images est géré par une boucle *for* (Twig), et qu'un attribut *id* doit être unique dans un document, il fallait que je trouve une solution pour que chaque image ait un *id* différent en utilisant Twig.

J'ai fait des recherches sur la documentation officielle, et j'ai trouvé ceci sur la page concernant *for* (<https://twig.symfony.com/doc/3.x/tags/for.html#for>):

The loop variable ¶

Inside of a *for* loop block you can access some special variables:

| Variable | Description |
|-----------------------------|---|
| <code>loop.index</code> | The current iteration of the loop. (1 indexed) |
| <code>loop.index0</code> | The current iteration of the loop. (0 indexed) |
| <code>loop.revindex</code> | The number of iterations from the end of the loop (1 indexed) |
| <code>loop.revindex0</code> | The number of iterations from the end of the loop (0 indexed) |
| <code>loop.first</code> | True if first iteration |
| <code>loop.last</code> | True if last iteration |
| <code>loop.length</code> | The number of items in the sequence |
| <code>loop.parent</code> | The parent context |

```
1 {% for user in users %}
2   {{ loop.index }} - {{ user.username }}
3 {% endfor %}
```

Voici ma traduction personnelle :

À l'intérieur d'un bloc de boucle for, vous pouvez accéder à quelques variables spécifiques :

| | |
|-----------------------|--|
| <i>loop.index</i> | <i>L'itération actuelle de la boucle. (commence à 1)</i> |
| <i>loop.index0</i> | <i>L'itération actuelle de la boucle (commence à 0)</i> |
| <i>loop.revindex</i> | <i>Le nombre d'itérations restantes dans la boucle (en commençant à 1)</i> |
| <i>loop.revindex0</i> | <i>Le nombre d'itérations restantes dans la boucle (en commençant à 0)</i> |
| <i>loop.first</i> | <i>Retourne vrai s'il s'agit de la première itération</i> |
| <i>loop.last</i> | <i>Retourne faux s'il s'agit de la dernière itération</i> |
| <i>loop.length</i> | <i>Le nombre d'éléments de la boucle/séquence</i> |
| ... | |

J'ai donc ajouté *loop.index* dans l'attribut *id* de la balise *img*. J'ai également renseigné l'id de son Article correspondant pour que la première image de chaque article n'ait pas le même attribut, la deuxième idem, etc :

```
id="art-{{ article.id }}-img-{{ loop.index }}"
```

Ce problème réglé, un autre est apparu : avec jQuery, comment récupérer les informations contenus dans cet *id*, sachant que jQuery ne comprend pas Twig ? J'ai choisi d'ajouter deux attributs *data-* à la balise *img*, pour permettre au script de traiter ces données. Voici ce que ça donnait dans mon fichier Twig... :

```
data-number="{{ loop.index }}" data-artnumber="{{ article.id }}"
```

... et voici ce que ça donnait dans mon fichier js :

```
displayImage($(this).data('image'), $(this).data('artnumber'), $(this).data('number') );
```

À ce moment, jQuery pouvait sélectionner un élément par son *id*, mais il me fallait aussi sélectionner avec elle toutes les autres images de la boucle. Avec jQuery, je peux les sélectionner grâce à la méthode `.siblings()`, que j'ai couplé avec la méthode `.add()` pour ne pas oublier l'élément appelé. J'ai donc créé un *array* (un tableau contenant plusieurs éléments), que j'ai hydraté avec chaque image grâce aux méthodes `.push()` et `.each()` .

En lisant attentivement la documentation liée à cette dernière méthode, j'ai remarqué ceci :

.each()

Categories: [Miscellaneous](#) > [Collection Manipulation](#) | [Traversing](#)

.each(function)

Description: *Iterate over a jQuery object, executing a function for each matched element.*

🔗 .each(function)

function

Type: [Function](#)([Integer](#) index, [Element](#) element)

A function to execute for each matched element.

Voici ma traduction personnelle :

Description : exécute une fonction pour chaque élément constituant d'un objet jQuery.

Type : Fonction(itération de type Entier, élément de type Élément)

Une fonction qui s'exécute pour chaque élément correspondant.

La méthode `.each()` me permet donc de passer des informations en paramètres de *function()*. J'ai ainsi utilisé le premier paramètre de type *Integer* (nombre entier) dans l'attribut *class* de la *div* parente de l'image, car c'est elle qui doit recevoir la classe *active*, pour que l'image s'affiche dans le carrousel. J'ai utilisé le deuxième paramètre pour indiquer à l'attribut *src* de la balise *img* du carrousel le nom du fichier.

9. Possibilités d'amélioration

À la fin de ce stage, plusieurs choses restent à créer ou à configurer sur mon projet :

La page *Qui sommes-nous* a été conçue pour recevoir un texte, via un formulaire contenant un champ géré par le bundle *CKEditor*. Mais idéalement, cette page devrait recevoir une liste des membres principaux de l'association, avec une photo, le prénom, le nom, et une petite présentation personnelle. J'ai donc prévu de rajouter un système de photo de profil pour que la photo qui apparaît soit celle de l'utilisateur. La présentation personnelle sera le contenu de la table *motivation*.

La page d'inscription n'est actuellement munie que d'un formulaire. M. DE JESUS voudrait rajouter un texte sur cette page, afin de mieux guider les futurs bénévoles vers les attentes de l'association.

Il n'y a pour le moment aucun moyen d'accéder à une page de profil autre que la sienne. Il serait intéressant de permettre aux utilisateurs de visiter le profil des autres. De cette manière, je pourrais permettre aux administrateurs de consulter le champ *motivation* des nouveaux inscrits afin de les accepter, ou non.

Le système de pagination, installé grâce au bundle *knp_paginator* est opérationnel mais aucun CSS n'y a été ajouté en plus du thème Bootstrap. Il faudrait changer certaines couleurs pour coller au maximum au thème du site.

Des pages d'erreurs personnalisées amélioreraient l'expérience utilisateur.

Un système de réunion (avec création d'une nouvelle Entité) serait utile. Les administrateurs pourront prévoir des réunions en renseignant un nom, une date, un lieu, etc. Ces futures réunions seraient visibles par les bénévoles sur une page dédiée, où ils pourraient ajouter leur participation, comme pour les événements. Quelques jours après la date effective de la réunion, elle se supprimerait de la base de donnée.

10. Conclusion

J'ai pu durant ce stage atteindre les objectifs du cahier des charges . Mme VELON-COMTE et M. DE JESUS m'ont félicité à la fin de mon stage pour le travail que je leur ai présenté. Je les ai à mon tour remerciés de m'avoir accordé ce stage, et leur ai garanti que je m'occuperai du développement du site jusqu'à sa mise en ligne.

Je suis satisfait du rendu visuel de mon projet, et ai hâte de pouvoir procéder aux améliorations dont j'ai fait la liste plus haut. Ce stage de 6 semaines m'aura prouvé que je suis capable de mettre en place un projet Symfony seul, même si j'ai préféré l'esprit d'équipe qui régnait lors de notre projet de soutenance devant le jury de WebForce3. Cette préférence m'a conforté dans l'idée que travailler dans une entreprise avec plusieurs développeurs serait un choix qui me conviendrait davantage que de travailler seul, d'un point de vue humain et social.

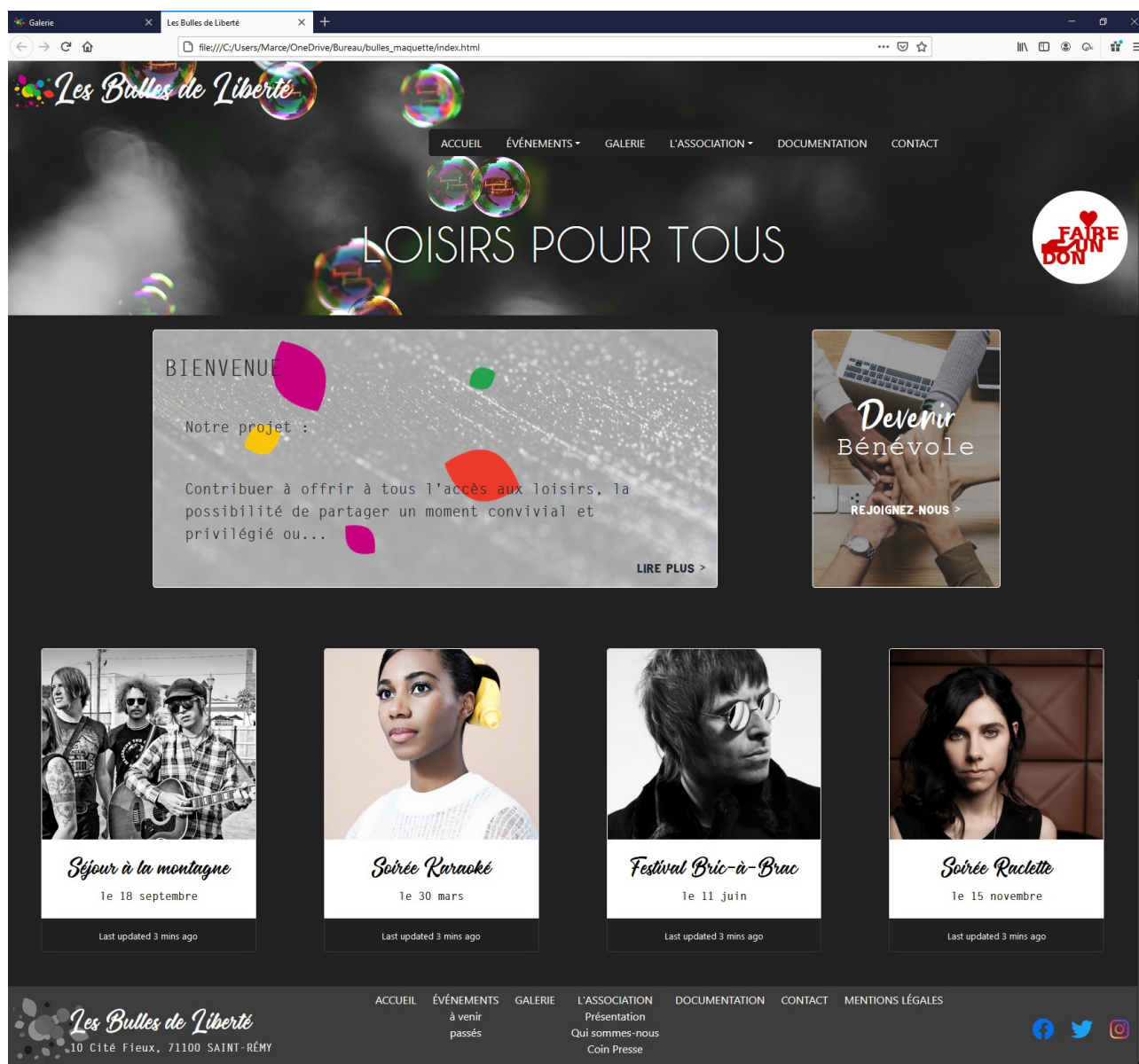
Je me suis senti progresser durant ce stage. Ce sentiment est réconfortant et encourageant. Si j'ai pu être effrayé au début par le caractère adaptatif du métier de développeur web, je me rends compte aujourd'hui que le fait de constamment réapprendre était quelque chose qui me convenait. J'ai pu vérifier ceci lors des mises à jour des bundles Symfony et de PHP durant la formation.

Je voudrais encore remercier l'équipe de WebForce3 pour m'avoir révélé ce qui est manifestement devenu une nouvelle passion.

11. Annexe

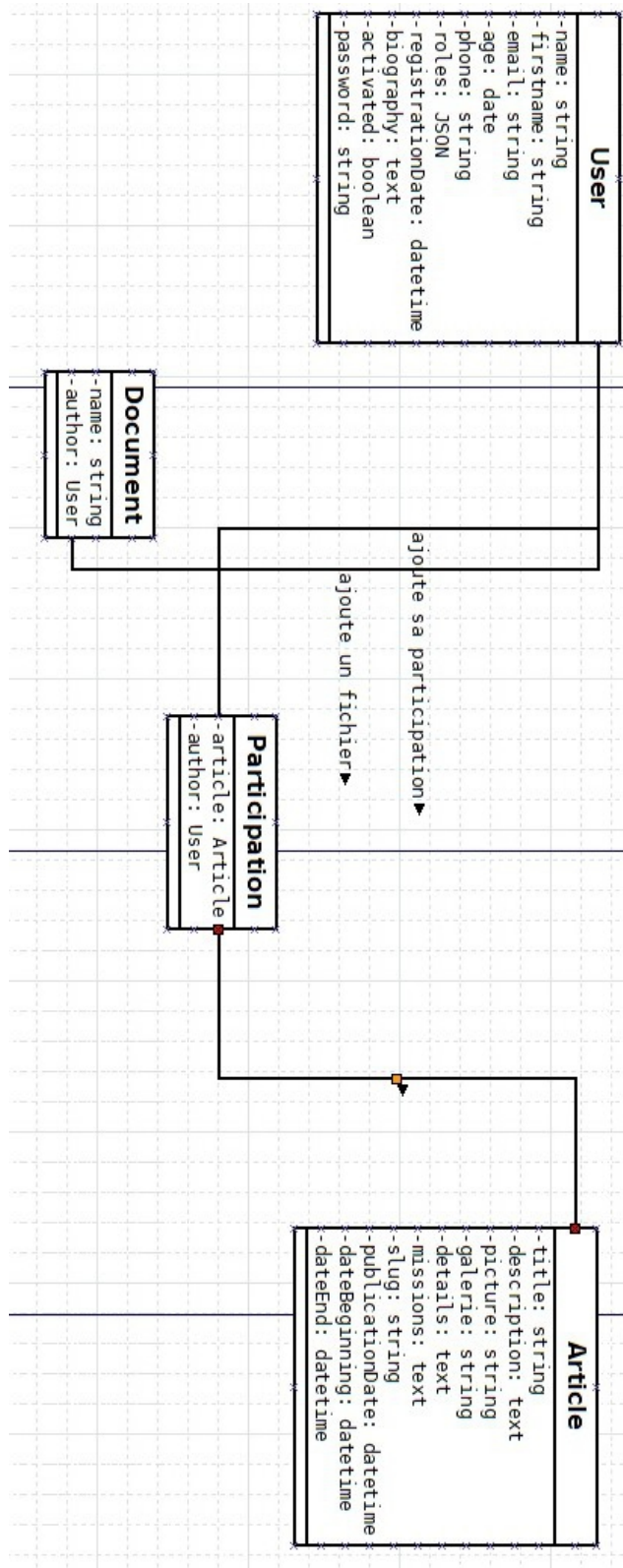
Annexe 1 :

Maquette de la page d'accueil version PC



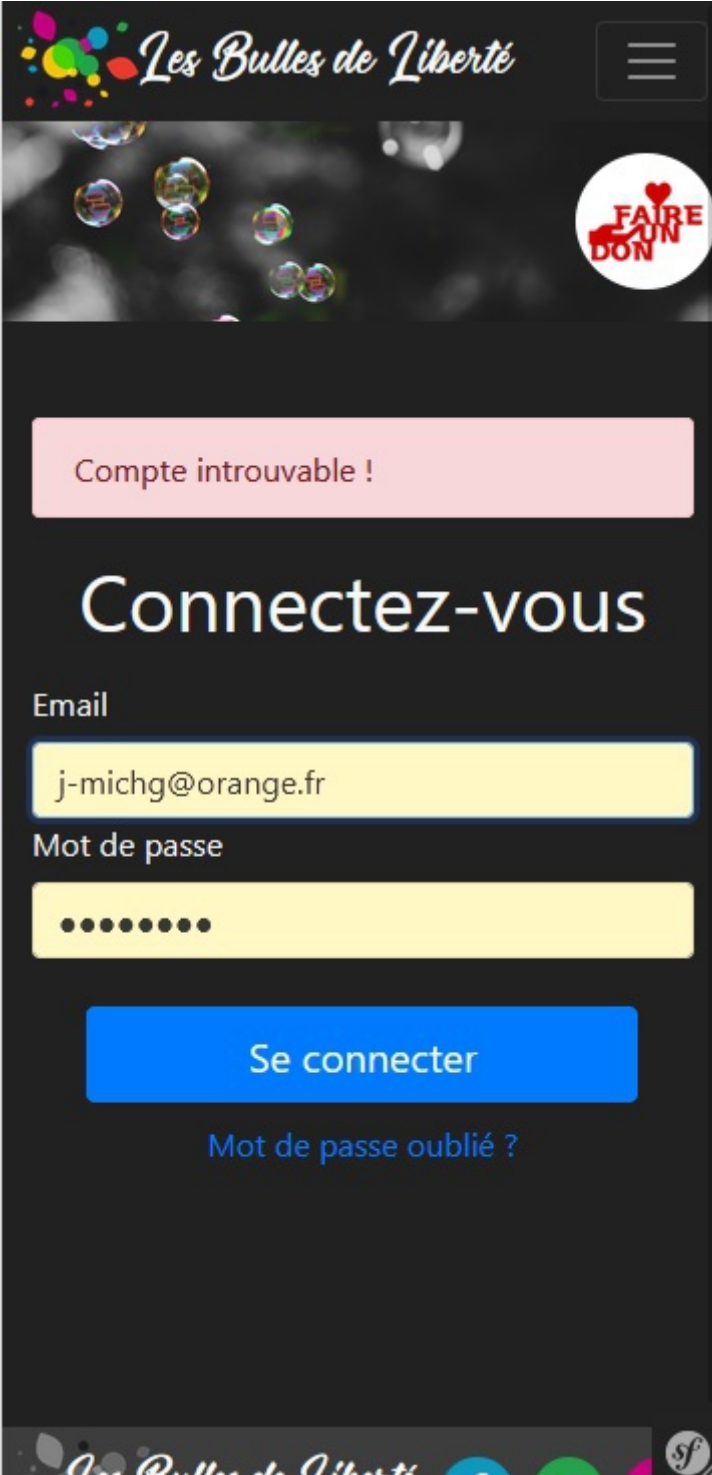
Annexe 2 :

Schéma UML constitué à la 1ère réunion



Annexe 3 :

Connexion impossible, version mobile



The image shows a mobile application interface for 'Les Bulles de Liberté'. At the top, there is a header with the app's logo on the left and a hamburger menu icon on the right. Below the header is a banner image featuring colorful bubbles and a circular logo with the text 'FAIRE UN DON' and a heart icon. The main content area has a dark background. A pink error message box at the top of the main area reads 'Compte introuvable !'. Below this, the text 'Connectez-vous' is displayed in large white letters. Underneath, there are two input fields: 'Email' with the value 'j-michg@orange.fr' and 'Mot de passe' with masked characters. A blue 'Se connecter' button is positioned below the password field. A link for 'Mot de passe oublié ?' is located below the button. At the bottom of the screen, there is a footer with the app's name and social media icons.

Les Bulles de Liberté

Compte introuvable !

Connectez-vous

Email

j-michg@orange.fr

Mot de passe

●●●●●●●●

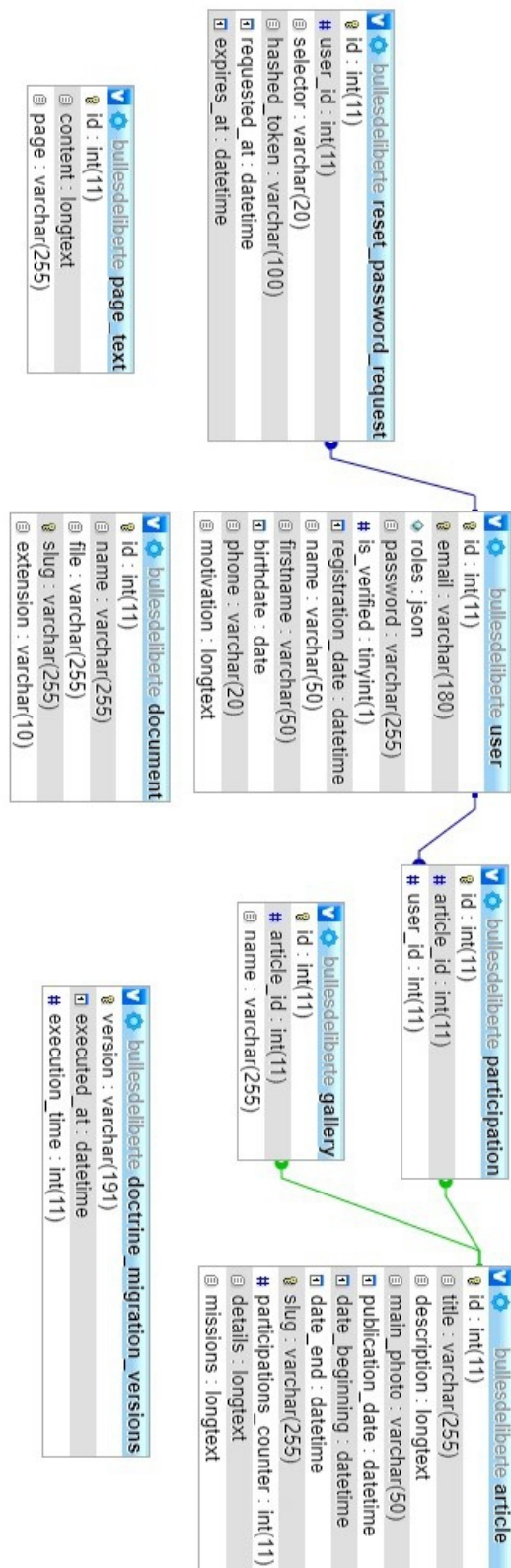
Se connecter

[Mot de passe oublié ?](#)

Les Bulles de Liberté

Annexe 4 :

Schéma de la Base de Données à la fin du stage
(via le designer de PHPmyadmin)



Annexe 5 :

l'entité Participation au détail
(sans les Getters & Setters)

58 lines (46 sloc) | 1 KB

```
1  <?php
2
3  namespace App\Entity;
4
5  use App\Repository\ParticipationRepository;
6  use Doctrine\ORM\Mapping as ORM;
7
8  /**
9   * @ORM\Entity(repositoryClass=ParticipationRepository::class)
10  */
11  class Participation
12  {
13      /**
14       * @ORM\Id()
15       * @ORM\GeneratedValue()
16       * @ORM\Column(type="integer")
17       */
18      private $id;
19
20      /**
21       * @ORM\ManyToOne(targetEntity=Article::class, inversedBy="participations")
22       */
23      private $article;
24
25      /**
26       * @ORM\ManyToOne(targetEntity=User::class, inversedBy="participations")
27       */
28      private $user;
```

Annexe 6 :

Extrait de code

Si l'utilisateur participe, on enlève sa participation

```
561     } else {
562
563         // Si l'user participe déjà, on retire sa participation
564         if($article->willCome($user)){
565
566             $participation = $participationRepo->findOneBy([
567                 'article' => $article,
568                 'user' => $user
569             ]);
570
571             // On baisse le nb de participations
572             $participationsCounter = $article->getParticipationsCounter();
573             $article->setParticipationsCounter(--$participationsCounter);
574
575             // On enlève cette participation dans la BDD
576             $manager->remove($participation);
577             $manager->persist($article);
578             $manager->flush();
579
580             // Tableau JSON avec le code de succès et le message
581             return $this->json([
582                 'code' => 200,
583                 'message' => "Participation supprimée",
584                 'participations' => $article->getParticipationsCounter()
585             ], 200);
586
587             // Sinon, il ajoute sa participation
588         } else {
```

Annexe 7 :

Extrait de code

Si l'utilisateur ne participe pas, on ajoute sa participation

```
587         // Sinon, il ajoute sa participation
588     } else {
589
590         // On spécifie quel user participe
591         $participation = new Participation();
592         $participation
593             ->setArticle($article)
594             ->setUser($user)
595         ;
596
597         // On augmente le nb de participations
598         $participationsCounter = $article->getParticipationsCounter();
599         $article->setParticipationsCounter(++$participationsCounter);
600
601         // On ajoute sa participation dans la BDD
602         $manager->persist($participation);
603         $manager->persist($article);
604         $manager->flush();
605
606         // Tableau JSON avec le code de succès et le message
607         return $this->json([
608             'code' => 200,
609             'message' => 'Participation ajoutée',
610             'participations' => $article->getParticipationsCounter()
611         ], 200);
612     }
613 }
614 }
```

Annexe 8 :

Extrait de code

Fonction de participation dans le script

```
3 // Fonction permettant de gérer les participations
4 function onClickBtn(event){
5
6     // On empêche le chargement de la page de participation, créée dans le MainController
7     event.preventDefault();
8
9     // On récupère l'url
10    let url = this.href;
11
12    // Grâce au bundle axios...
13    axios.get(url).then(function(){
14
15        location.reload();
16
17    }).catch(function(error){
18
19        // Si l'utilisateur n'est pas connecté...
20        if(error.response.status === 403){
21            // ... on affiche une erreur dans une fenêtre d'alerte
22            window.alert("Il faut être connecté pour liker un article !");
23            // ...sinon, pour n'importe quelle autre erreur, on affiche un autre alert
24        } else {
25            window.alert("Une erreur s'est produite, veuillez réessayer plus tard.");
26        }
27    });
28 }
```

Annexe 9 :

Extrait de code

Vue Twig avec affichage du bouton de participation

```
{% if event.dateEnd > date() %}
<div class="row w-100 ml-0 my-4">
  <div class="col-10 col-md-4 offset-md-4 offset-1 d-flex flex-column mb-4">
    <a href="{{ path('event_participation', {id: event.id}) }}" class="d-inline-block participation-link h-75 text-center">
      {% if app.user and event.willCome(app.user) %}
        <p class="btn btn-secondary participation m-0 participated w-100">Participation ajoutée</p>
      {% else %}
        <p class="btn btn-primary participation m-0 w-100">JE PARTICIPE !</p>
      {% endif %}
    </a>
  {% if is_granted('ROLE_ADMIN') %}
    <span class="p-count text-light text-center mt-2">
      {{ event.participationsCounter }} participant{% if event.participationsCounter > 1 %}s{% endif %}
    </span>
  {% endif %}
</div>
{% endif %}
```


Annexe 10 :

Extrait de code

Vue twig – section Admin avec tableau des participants

```
119     {# Partie admin avec liste de bénévoles #}
120     {% if is_granted('ROLE_ADMIN') and event.participationsCounter > 0 %}
121         <hr class="bg-light col-lg-4 offset-lg-4 ml-auto">
122
123         <div class="row w-100 ml-0">
124             <div class="col-10 offset-1 col-lg-4 offset-lg-4 text-center">
125                 <h4 class="text-danger">Section Admin</h4>
126             </div>
127         </div>
128
129         <div class="row w-100 ml-0 mb-4">
130             <div class="col-12 col-md-8 offset-md-2 text-center">
131                 <h5 class="text-light">Participants</h5>
132             </div>
133         </div>
134
135         <div class="row w-100 ml-0">
136             <div class="col-12 col-md-4 offset-md-4">
137                 <table class="table table-dark">
138                     <thead>
139                         <tr>
140                             <th scope="col">#</th>
141                             <th scope="col">Prénom NOM</th>
142                         </tr>
143                     </thead>
144                     <tbody>
145                         {% for participation in event.participations %}
146                             <tr>
147                                 <th scope="row">{{ loop.index }}</th>
148                                 <td>{{ participation.user.firstname }} {{ participation.user.name }}</td>
149                             </tr>
150                         {% endfor %}
151                     </tbody>
152                 </table>
153             </div>
154         </div>
155
156         <hr class="bg-light col-lg-4 offset-lg-4 ml-auto">
157
158     {% endif %}
```