RAPPORT DE PROJET

Refonte du site web du club d'échecs de Montpellier
le 23 février 2015, à Montpellier

Réalisé par

Maxime BERTRAND
Benjamin HOQUY
Sébastien LACOMBE
Jason THIBUR

IUT de Montpellier
pour l'obtention de la LP API-DAE

sous la direction de Xavier PALLEJA

Année universitaire 2014-2015

Sommaire

I Analyse **1**

1 Analyse du contexte **2**

2 Analyse des besoins fonctionnels **3**

3 Analyse des besoins non fonctionnels **4**
3.1 Contraintes techniques 4
3.2 Contraintes ergonomiques 4

4 Scénarios des cas d'utilisations **5**
4.1 Scénarios utilisateur non connecté 5
4.1.1 S'inscrire sur le site 5
4.1.2 Se connecter 5
4.2 Scénarios utilisateur 5
4.2.1 Publier un article 5
4.2.2 Modifier un article 7
4.3 Scénarios administrateur 7
4.3.1 Ajouter un menu/sous-menu 7
4.3.2 Accéder à l'interface d'administration 7
4.3.3 Ajouter un sponsor 7
4.3.4 Ajouter une publication 8
4.3.5 Refuser un article 8
4.3.6 Valider un article 8

II Rapport technique **9**

5 Conception **10**
5.1 Présentation des choix technologiques 10
5.1.1 Langages de programmation 10
5.1.2 Langages de structuration des données 10
5.1.3 Langages de configuration 10
5.2 Framework et outils intégrés 10
5.3 Description des fonctionnalités 11
5.3.1 Création d'une publication 11
5.3.2 Administration générale 13
5.3.3 Gestion des articles 13

5.3.4 Gestion des menus/sous-menus 17
5.3.5 Gestion des utilisateurs 20
5.3.6 Gestion des sponsors 21

6 Résultats et perspectives **22**
6.1 Résultats 22
6.1.1 Vitrine 22
6.1.2 Inscription d'un membre 22
6.1.3 Gestion des articles/sponsors 22
6.1.4 Gestion des catégories/produits 22
6.2 Perspectives 23
6.2.1 Boutique 23
6.2.2 Gestion d'équipes 23
6.2.3 Gestion des membres/inscription 23

III Rapport d'activité **24**

IV Manuel d'utilisation **26**

7 Accéder à l'interface d'administration **27**

8 Administration des publications **30**

9 Administration des sponsors **33**

A Script SQL phpmyadmin **36**

B Diagramme des classes **39**

Glossaire

Bundle Symfony La traduction la plus proche d'un Bundle Synfony est le "module", un bundle Symfony est un package regroupant une ou plusieurs fonctionnalités pouvant être ajouté à un site web Symfony pour faciliter sa création et/ou son utilisation. 14, 20

Entity Manager Classe permettant d'interfacer le monde de Doctrine et des classes implémentées par le développeur (Entité). En somme, il permet de gérer une entité depuis le code en appelant des méthodes raisonnant sur le modèle Objet. 13

ORM Object Relationnel Mapping. Technique de programmation faisant le lien entre le monde de la base de données et le monde de la programmation objet. Elle permet de transformer une table en un objet facilement manipulable via ses attributs. 9

Repository Classe PHP implémentée par Symfony2 dépendant d'une entité permettant de définir des requêtes DQL agissant sur son entité.. 14, 18

Service Classe PHP instanciée une seule fois consistant en une fonctionnalité bien définie et pouvant être utilisée de n'importe où dans l'architecture mise en place par symfony (Injection de Dépendances). 16–18

Service Container Ensemble de services, c'est-à-dire des classes PHP directement chargées et instanciées une seule fois au démarage du kernel (noyau) de Symfony. . 13, 14, 18

WYSIWYG What You See Is What You Get : Le Wisiwig est une interface utilisateur qui permet de composer visuellement le résultat voulu, typiquement pour un logiciel de mise en page, un traiement de texte ou d'image. C'est une interface « intuitive » : l'utilisateur voit directement à l'écran à quoi ressemblera le résultat final . 10

Remerciements

Nous aimerions remercier notre tuteur Xavier Palleja pour nous avoir guidé le long du projet tout en jouant le rôle du client exigeant et indécis.

Nous remercions aussi Mme Gelsomino pour son aide à la rédaction de ce rapport ainsi que M. Gautheret qui nous a initié à Symfony2.

Introduction

De nos jours, un site qui n'évolue pas n'attire pas les visiteurs. et lasse les coutumiers. Afin de garder un attrait, un site doit se renouveler de temps en temps. De plus, selon l'interface et les technologies utilisées la gestion de son site peut rapidement devenir complexe et fastidieuse.

C'est dans ce contexte que le club d'échecs montpelliérain Echecs Club Montpellier souhaite faire une refonte de son site internet afin de lui donner un aspect plus jeune, ajouter une web-boutique et surtout faciliter la gestion de ses publications.

En effet, le site actuel ne permet qu'une gestion minimale et fastidieuse des articles. Avoir un back-office clair afin de gérer ses publications ainsi que les commentaires peut rendre cette tâche bien plus agréable et intuitive.

C'est autour de cette problématique que le club d'échecs nous a contacté pour développer une refonte de leur site internet plus simple d'utilisation et intégrant une boutique, en utilisant la technologie Symfony2.

Après avoir définis le cadre du projet et les besoins généraux de celui-ci dans l'analyse, nous détaillerons l'ensemble des fonctionnalités, l'architecture développées pour arriver au résultat final dans le rapport technique. Enfin nous reviendrons dans le rapport d'activité sur les méthodes ainsi que les différents outils de travail utilisés pour réaliser ce projet.

Première partie

Analyse

1 Analyse du contexte

Le site du club d'échec de Montpellier a été développé avec Joomla pour une mise en place facile et rapide. Jusqu'à lors le site ne permettait qu'à des administrateurs de gérer le contenu du site.

Le club désire donc que les membres inscrits aient la possibilité de proposer leurs propres articles et qu'une administration puisse vérifier la pertinence de ces derniers.

L'administration pourra quant à elle, revoir l'architecture du site (pages, menus et sous-menus) à tout moment et ainsi s'assurer de la bonne lisibilité des informations partagées.

Le club possède une boutique physique interne et souhaiterait l'étendre sur son site web afin d'améliorer sa visibilité et son accessibilité. Pour cela, dans un premier temps il faudra mettre en place une vitrine informative des produits, et, le cas échéant donner la possibilité aux membres et aux visiteurs du site de passer leur commandes en ligne.

2 Analyse des besoins fonctionnels

Le nouveau site développé devra inclure :
— une vitrine présentant le club (informations essentielles, contact, sponsors)
— une boutique en ligne
— une partie administration ou back office
— une partie publication d'articles concernant le club
L'utilisateur enregistré pourra :
— consulter les nouveautés publiées par les autres utilisateurs enregistrés
— publier des articles dans les sections autorisées
— acheter différents produits vendus dans la boutique
L'administrateur pourra :
— modérer les articles publiés par les utilisateurs enregistrés
— modifier les informations essentielles statiques de la vitrine (contact, cours, organigramme, sponsors, etc.)
— gérer la boutique
— gérer l'architecture du site (navigation)

3 Analyse des besoins non fonctionnels

3.1 Contraintes techniques

Le site devra être développé avec le framework Symfony 2.

3.2 Contraintes ergonomiques

La charte graphique du site devra concorder avec les couleurs du club et de la commune(bleu).

4 Scénarios des cas d'utilisations

4.1 Scénarios utilisateur non connecté

4.1.1 S'inscrire sur le site

1. L'internaute clique sur le bouton « Inscription »
2. Le système affiche le formulaire d'inscription et demande à l'internaute de le remplir
3. L'internaute rempli le formulaire et appuie sur le bouton « Valider »
4. Le système envoie un mail de validation d'inscription à l'internaute contenant la lien pour terminer l'inscription.
5. L'internaute clique sur le lien de validation et termine son inscription.
6. Le système active le compte et affiche à l'internaute que l'inscription s'est bien finalisée

4.1.2 Se connecter

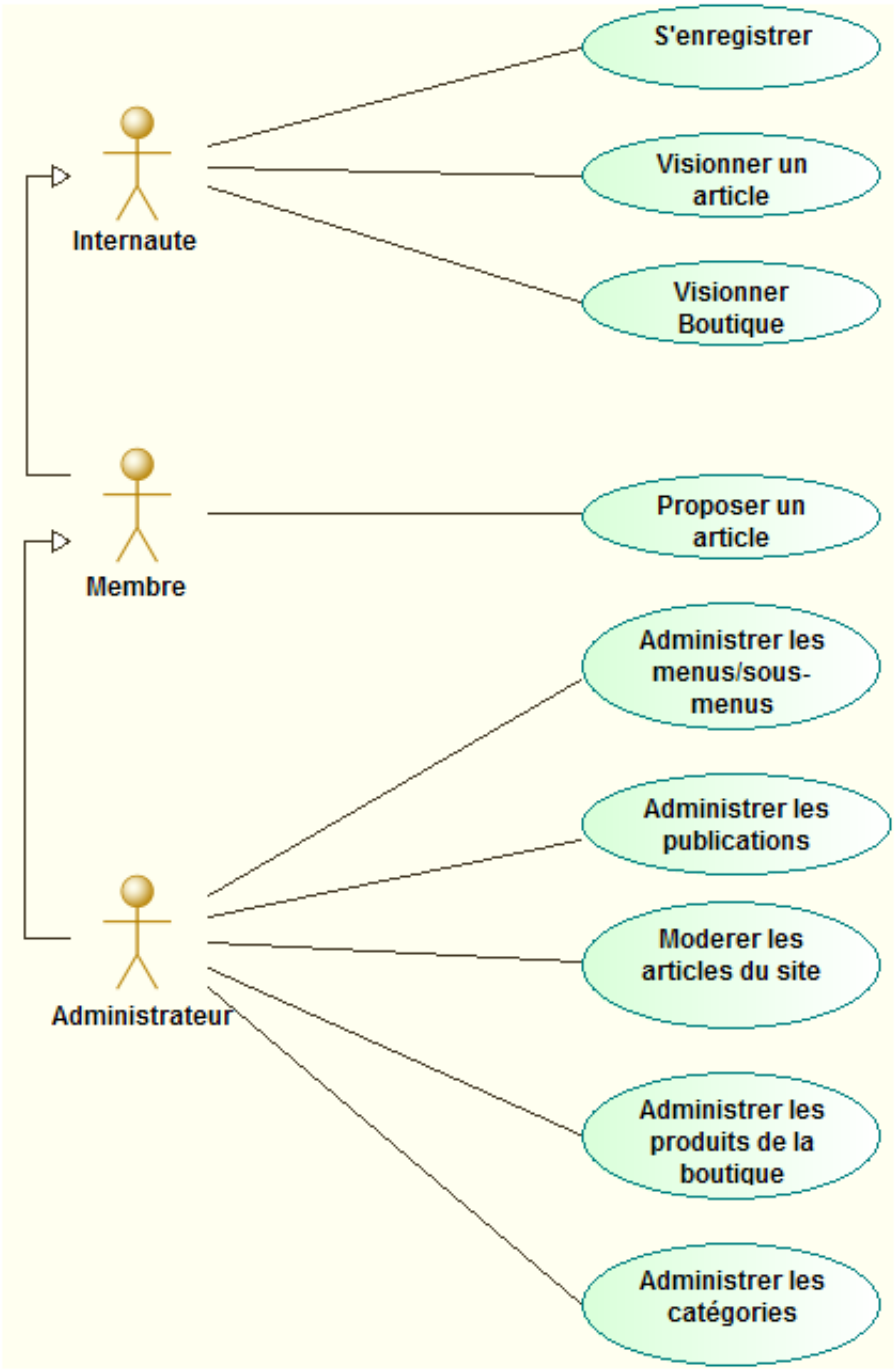
1. L'internaute clique sur le bouton « S'authentifier »
2. Le système affiche le formulaire de connexion
3. L'internaute rempli les champs du formulaire de connexion et appuie sur le bouton « Connexion »
4. Le système vérifie la validité des identifiants saisis précédemment
5. L'internaute est redirigé sur la « page d'accueil » avec un message de bienvenue et un onglet « mon compte »

4.2 Scénarios utilisateur

4.2.1 Publier un article

1. L'utilisateur clique sur "Proposer un article"
2. Le système affiche un formulaire contenant les informations essentielles d'un menu
3. L'utilisateur renseigne les champs du formulaire et le valide
4. Le sytème affiche un message de validation de l'article, l'article ne sera affiché que si un administrateur valide l'article

FIGURE 4.1 – Diagramme des cas d'utilisation



4.2.2 Modifier un article

1. L'utilisateur clique sur "Mes article"
2. L'utilisateur sélectionne l'article a modifier, le motif de refus est afficher
3. Le système affiche un formulaire contenant les informations essentielles
4. L'utilisateur modifier les champs du formulaire et le valide
5. Le sytème affiche un message de validation de l'article, l'article ne sera affiché que si un administrateur valide l'article

4.3 Scénarios administrateur

4.3.1 Ajouter un menu/sous-menu

Ajout d'un menu

1. L'administrateur va dans l'interface d'administration
2. L'administrateur clique sur le bouton "Ajouter un menu"
3. Le système affiche un formulaire contenant les informations essentielles d'un menu
4. L'administrateur renseigne les champs du formulaire et le valide
5. Le sytème affiche un message de validation de création du menu

Ajout d'un sous-menu

1. L'administrateur va dans l'interface d'administration
2. L'administrateur clique sur le bouton "Ajouter un sous-menu"
3. Le système affiche un formulaire contenant les informations essentielles d'un sous-menu ainsi qu'une liste déroulante avec les diffèrent menu.
4. L'administrateur renseigne les champs du formulaire et le valide
5. Le sytème affiche un message de validation de création du menu

4.3.2 Accéder à l'interface d'administration

1. L'administrateur clique sur l'onglet du menu « Administration » (disponible que pour l'administrateur)
2. Le système affiche la page d'administration contenant les liens pour administrer les différentes partie du site
3. Le système affiche les différentes parties que l'administrateur peut modifier sous forme de liens

4.3.3 Ajouter un sponsor

1. L'administrateur clique sur le lien modifier les sponsors.
2. Le système affiche la liste ordonnée des sponsors avec la possibilité de les modifier (ajout/suppression/modification)
3. L'administrateur clique sur « ajouter ».

4. Le système affiche un formulaire avec les informations du sponsor à renseigner (sous forme de formulaire : nom, lien hypertexte, ordre, image).
5. L'administrateur renseigne les champs et valide l'ajout.
6. Le système affiche un message de confirmation.

4.3.4 Ajouter une publication

1. L'administrateur va dans l'interface d'administration
2. L'administrateur clique le bouton "Publication"
3. Le système affiche les publications présents sur le site ainsi que l'option d'ajout d'une publication
4. L'administrateur clique sur le bouton "Ajouter une publication"
5. Le système affiche un formulaire contenant les informations essentielles d'une publication
6. L'administrateur renseigne les champs du formulaire et le valide
7. Le sytème affiche un message de validation de création de l'article dans le menu

4.3.5 Refuser un article

1. L'administrateur va dans l'interface d'administration
2. L'adminstrateur clique sur "Article"
3. L'administrateur clique sur l'article a refuser
4. L'administrateur selectionne et "refuser" et met un motif de refus puis valide le formulaire
5. Le sytème affiche un message de validation, l'article sera pas visible sur le site, l'utilisateur devra modifier sont article pour qu'il soit ressoumit a une validation par l'administrateur.

4.3.6 Valider un article

1. L'administrateur va dans l'interface d'administration
2. L'adminstrateur clique sur "Article"
3. L'administrateur clique sur l'article a valider
4. L'administrateur selectionne et "valider" et valide le formulaire
5. Le sytème affiche un message de validation, l'article sera visible sur le site

Deuxième partie

Rapport technique

5 Conception

5.1 Présentation des choix technologiques

5.1.1 Langages de programmation

Langage principal

PHP5 (utilisé par Symfony2)

Langages de requêtes

MySQL, DQL (Doctrine Query Language) : langage de requête utilisé par Doctrine dérivant du SQL. La différence majeure est que le DQL exécute des requêtes sur des entités Doctrine au lieu de les faire sur des tables SQL.

5.1.2 Langages de structuration des données

— *T*wig : moteur de template présent par défaut dans Symfony2
— *H*TML5
— *C*SS3

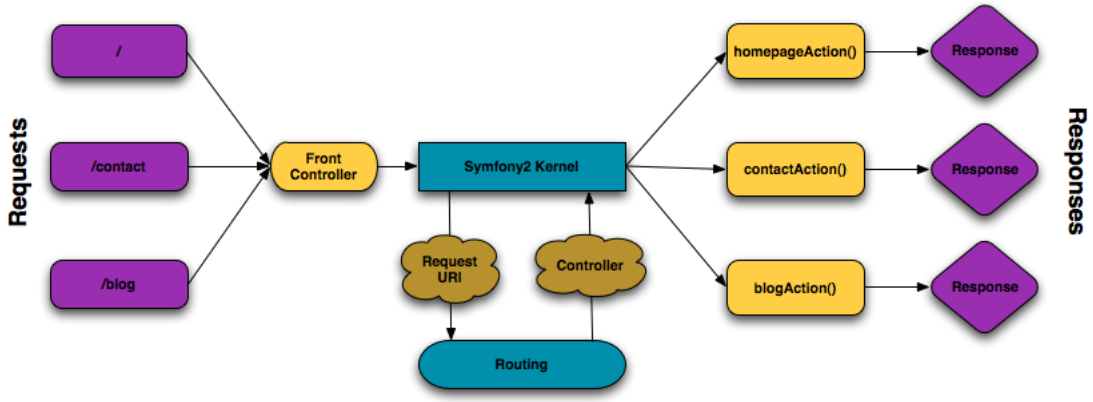
5.1.3 Langages de configuration

— *Y*ML : langage de configuration utilisé par Symfony2 décrivant plusieurs objets (attribut, valeur) servant à la configuration de Symfony et/ou de ses modules.
— *X*ML

5.2 Framework et outils intégrés

— *S*ymfony : Framework français écrit en PHP5, et conçu pour développer des applications en 3 couches, selon le modèle MVC (voir figure 5.1), permettant de produire du code propre et organisé.
— *D*octrine : ORM intégré par défaut dans le moteur de Symfony2. Il sert de couche de relation entre la base de données et les entités et permet d'interfacer les actions pouvant être appelées sur les entités en langage objet

FIGURE 5.1 – Fonctionnement de Symfony 2



5.3 Description des fonctionnalités

5.3.1 Création d'une publication

Une distinction a été faite entre les publications et les articles.Un article est rédigé par un utilisateur membre. Une publication, quant à elle, ne peut être créée que par l'administrateur du site. De ce fait, la création se fait sur l'interface d'administration et utilise, par conséquent, le module Sonata Admin Bundle.

Sonata Admin Bundle

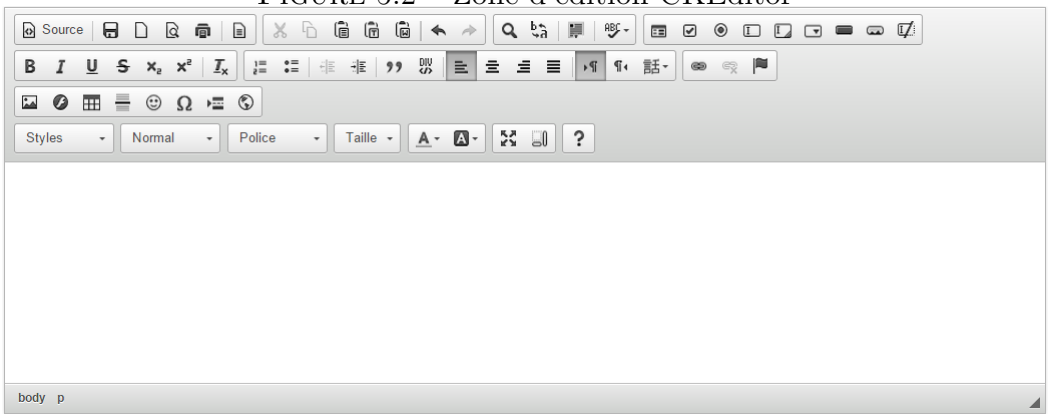
Sonata Admin Bundle est un module pour Symfony permettant d'administrer des entités Doctrine. Il est nécessaire d'implémenter une classe XxxAdmin héritant de Sonata/AdminBundle/Admin/Admin contenant des méthodes permettant de mapper les attributs de l'entité. Ce mapping est configuré dans le fichier services.yml.

```
    sonata.admin.articleUser:
        class: ECM\Bundle\ArticleBundle\Admin\ArticleAdmin
        tags:
            - name: sonata.admin
              manager_type: orm
              group: "Contenu"
              label: "Articles"
        arguments:
            - ~
            - ECM\Bundle\ArticleBundle\Entity\Article
            - ~
```

La classe précisée dans l'attribut `class` (ECM/Bundle/ArticleBundle/Admin/ArticleAdmin) sera utilisée par Sonata pour mapper les attributs de l'entité passée en argument dans l'attribut `arguments` (ECM/Bundle/ArticleBundle/Entity/Article).

```
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper
            ->add('titre', 'text')
            ->add('corps', 'ckeditor')
            ->add('menu', 'sonata_type_model', array('property' => '
   titre', 'btn_add' => false))
```

FIGURE 5.2 – Zone d'édition CKEditor




```
7             ->add('etat', 'choice', array('choices' => array('2' => '
  Accepter', '3' => 'Refuser'), 'expanded' => true))
8             ->add('motif');
9     }
```

Une fois la relation indiquée, il reste à signaler à Sonata les éléments à afficher sur son interface d'administration. La méthode configureFormFields indique à Sonata quels champs de l'entité Article seront affichés lors de la modification d'un article.

CKEditor

CKEditor est un éditeur de texte open source de type WYSIWYG pouvant servir à la création de contenu de pages web. Toute rédaction de texte liée à la publication d'articles est faite à l'aide de CKEditor.

```
1     /**
2      * @param FormBuilderInterface $builder
3      * @param array $options
4      */
5     public function buildForm(FormBuilderInterface $builder, array
    $options)
6     {
7         $builder
8             ->add('titre')
9             ->add('corps', 'ckeditor')
10            ->add('menu', 'entity', array(
11                'class' => 'ECM\Bundle\ModuleBundle\Entity\Menu',
12                'property' => 'titre',
13                'multiple' => false,
14                'required' => true,
15                'query_builder' => function (MenuRepository $rep) {
16                    return $rep->createQueryBuilder('m');
17                }));
18    }
```

Lors de la création du formulaire (buildForm()), la classe " ckeditor " est passée en paramètre au champ "corps" qui est de type texte. Durant la génération du formulaire, la zone de texte du champ "corps" sera replacé par l'interface d'édition de ckeditor (voir figure 5.2)

FIGURE 5.3 – Diagramme d'État-transition d'un article

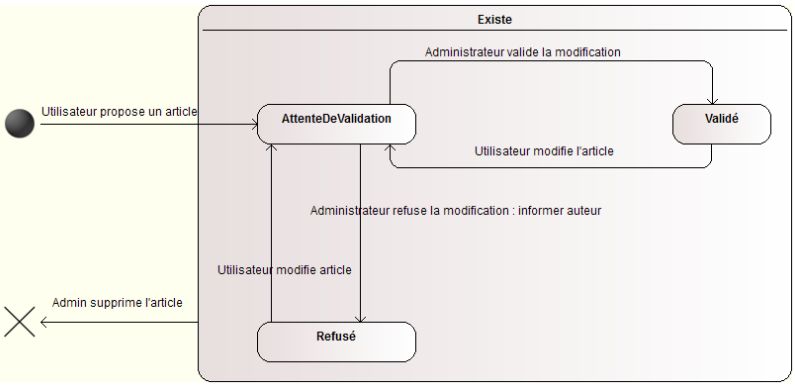
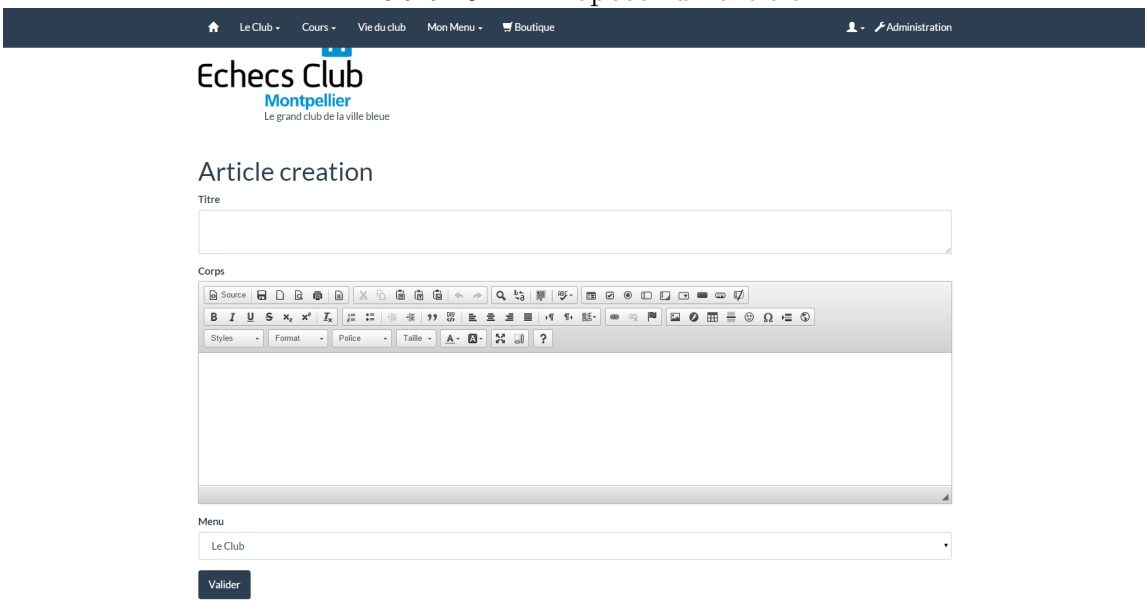


FIGURE 5.4 – Proposer un article



5.3.2 Administration générale

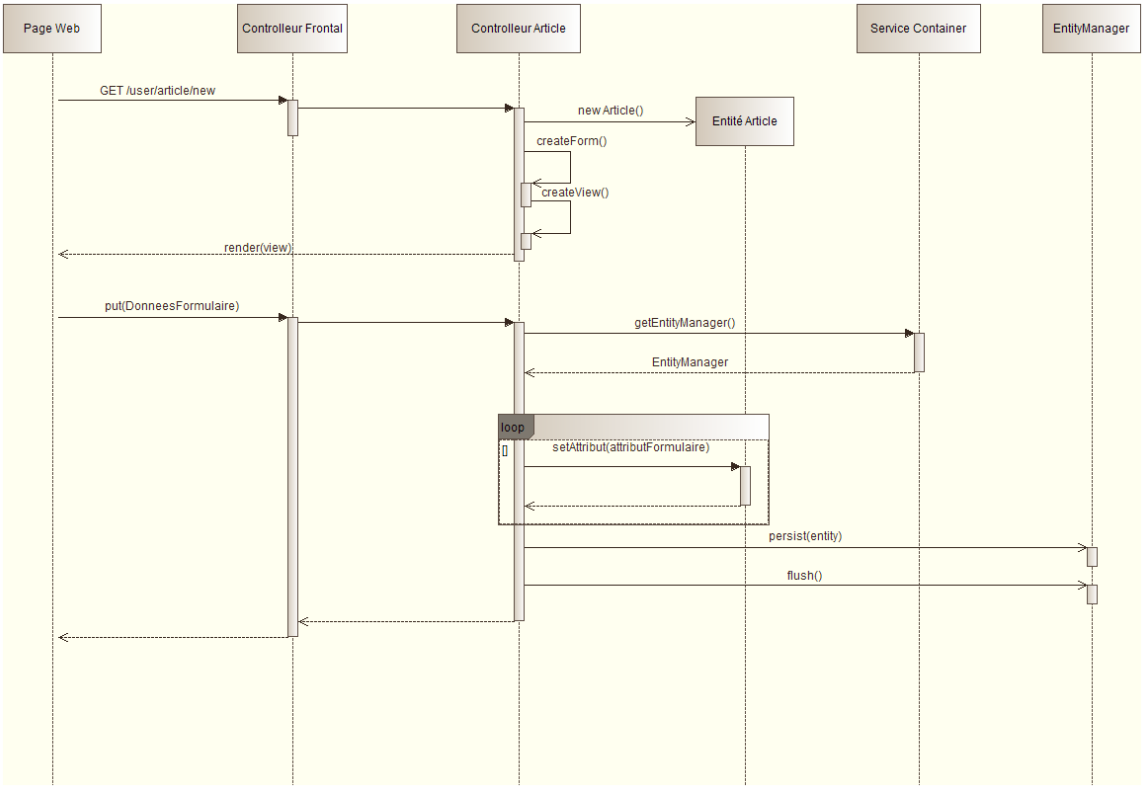
5.3.3 Gestion des articles

Proposition d'un article

Lorsque l'utilisateur demande à poster un article(voir figure 5.4), une entité Article est automatiquement créée par le contrôleur d'articles c'est à partir de cette entité que le contrôleur génère une vue avec les champs adaptés pour ensuite l'envoyer à l'utilisateur.

```
1     /**
2      * Creates a new Article entity.
3      *
4      */
5     public function createAction(Request $request)
6     {
7         $entity = new Article();
8         $form = $this->createCreateForm($entity);
9         $form->handleRequest($request);
10
11        if ($form->isValid()) {
```

FIGURE 5.5 – Diagramme de Séquence - Proposer un article



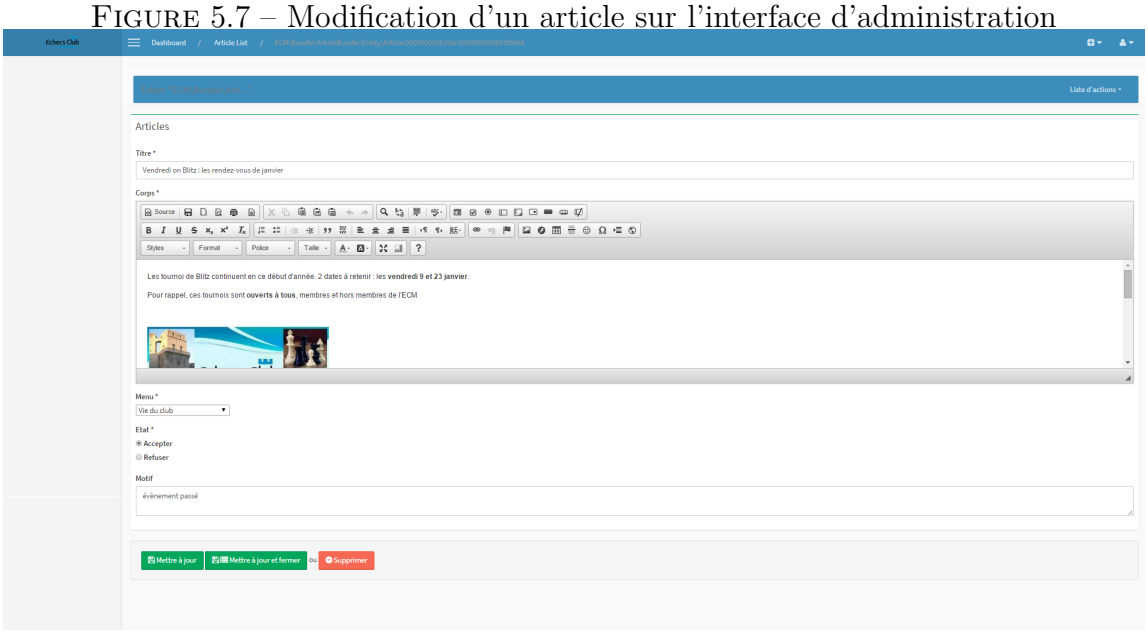

```
12              $em = $this->getDoctrine()->getManager();
13              $entity->setAuteur($this->get('security.context')->getToken
    ()->getUser());
14              $em->persist($entity);
15              $em->flush();
16
17              return $this->redirect($this->generateUrl('ecm_home_homepage
    '));
18          }
19
20          return $this->render('ECMArticleBundle:Article:new.html.twig',
    array(
21              'entity' => $entity,
22              'form' => $form->createView(),
23          ));
24      }
```

Une fois les champs remplis, l'utilisateur appuie sur le bouton de soumission. Le contrôleur d'article demande au Service Container de lui donner l' Entity Manager qui s'occupe alors de stocker l'entité dans la base de données (persist(entity)), puis qui valide les changements effectués dans la base (flush()).

L'article est alors créé, en attente de validation par l'administrateur. Il sera donc caché jusqu'à sa validation, l'auteur pourra tout de même vérifier l'état de son article ou le modifier à sa convence.

FIGURE 5.6 – Liste des articles sur l'interface d'administration

FIGURE 5.7 – Modification d'un article sur l'interface d'administration



Modération d'un article

Lorsque l'administrateur demande à afficher la liste des articles pour les administrer, le contrôleur article demande au Service Container le Repository de l'article en charge de récupérer les données de la base. Le contrôleur demande alors de récupérer tout les articles (findAll()), la liste des articles est alors renvoyée directement à la vue qui récupère les attributs nécessaires à la liste de chacun des articles. Ces étapes sont implicitement gérées par Sonata Admin Bundle Symfony

Une fois que l'administrateur a choisi quel article modérer, il le sélectionne. De la même manière, le contrôleur demande au Repository de l'article de lui trouver l'article en question à partir de son ID (findByID(idArticle)), de la même manière l'article est retrouné à la vue qui génère le formulaire en fonction des attributs à afficher (titre, auteur, corp, menu d'appartenance) (voir figure 5.7).

Après lecture l'administrateur choisi soit de le valider, soit de le refuser via les boutons radios de la page. Pour l'enregistrement des changements, le contrôleur fait appel à l'EntityManager à qui il demande de valider les changements fait en effectuant un flush()(voir figure 5.8).

FIGURE 5.8 – Diagramme de Séquence - Administration d'un article

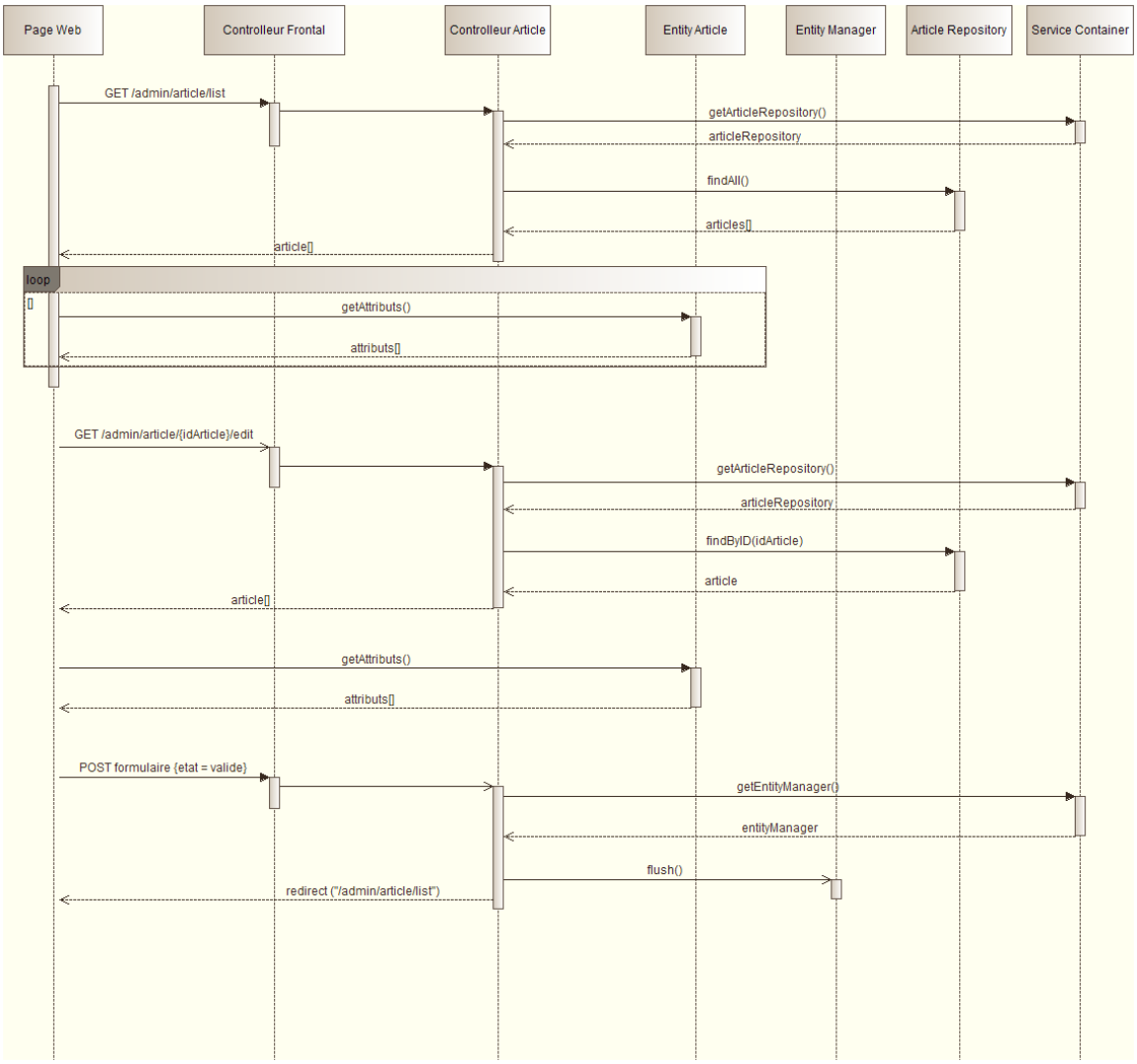


FIGURE 5.9 – Menu Administration

5.3.4 Gestion des menus/sous-menus

Menus statiques

KNP Menu Bundle est utilisé pour la création et l'affichage des menus. Le bouton "Administration" va servir d'exemple pour essayer de comprendre le fonctionnement des menus statiques(voir figure 5.9).

```
<div class="navbar navbar-default navbar-fixed-top" role="navigation">
    <div class="container">
{#{{ knp_menu_render('main', {'currentClass': 'active'}) }}#}
    {{ knp_menu_render('generated', {'currentClass': 'active'}) }}
    {% if is_granted('ROLE_USER') %}
        <ul class="nav navbar-nav navbar-right">
        {{ knp_menu_render('user', {'currentClass': 'active'}) }}
        {% if is_granted('ROLE_ADMIN') %}
            {{ knp_menu_render('admin', {'currentClass': 'active'}) }}
        {% endif %}
        </ul>
    {% else %}
        <ul class="nav navbar-nav navbar-right">
            {{ knp_menu_render('anonymous', {'currentClass': 'active'})
   }}
        </ul>
    {% endif %}
    </div>
</div>
```

Toujours pour suivre l'exemple avec le menu "Administration", la vue vérifie si l'utilisateur en cours est connecté en tant que "ROLE_ADMIN" grâce au test is_granted. S'il est connecté en tant qu'admin, la vue appelle le Service "admin" qui se charge d'afficher le menu administrateur.

```
    ecm_home.menu.admin:
        class: Knp\Menu\MenuItem # the service definition requires
   setting the class
        factory_service: ecm_home.menu_builder
```

```
        factory_method: createAdminMenu
        arguments: ["@request"]
        scope: request # needed as we have the request as a dependency
    here
        tags:
            - { name: knp_menu.menu, alias: admin }
```

Le Service ecm_home.menu.admin sert à appeler la méthode “createAdminMenu()” de la classe MenuBuilder. Il est utile de préciser que la requête (@request) est injectée dans le menuBuilder.

Cette méthode de la classe menuBuilder sert à la création du menu “Administration” Le menu retourné (menu) est contruit à partir de KNP/Menu/FactoryInterface. La fonction setChildrenAttribute sert a appliquer la classe css ‘nav navbar-nav’ au menu, la classe ‘nav navbar-nav’ est une classe Bootstrap, elle sert à la mise en forme du menu. Ensuite la fonction ‘addChild’ ajoute un enfant au menu en lui donnant en paramètre le libellé du menu (‘Administration’) et la route qui sera appelée lors du clique sur le menu (`route => sonata_admin_dashboard`) qui provient du fichier config.yml. La fonction “setAttribute” ajoute un attribut de type glyphicon de valeur “wrench” au menu.

```
    /*
     * Menu administrateur
     */
    public function createAdminMenu()
    {
        $menu = $this->factory->createItem('root');
        $menu->setChildrenAttribute('class', 'nav navbar-nav');

         $menu->addChild('Administration', array('route' => '
    sonata_admin_dashboard'))
             ->setAttribute('glyphicon', 'wrench');
        return $menu;
    }
```

Enfin, la méthode renvoie le menu à la vue qui se charge de l’afficher.

Menus Dynamiques

Il est possible de créer des menus personnalisés via l’interface d’administration. Ces menus peuvent également contenir des sous-menus. Pour gérer ce type d’architecture, un héritage entre menu et sous-menu est nécessaire.

Le processus d’affichage avec KNPMenuBundle ainsi que le Service utilisé pour générer le menu sont les mêmes que pour les menus statiques.

Héritage

L’héritage se configure dans les classes des entités Doctrine (Menu et SubMenu en l’occurrence).

```
/**
 * Menu
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="ECM\Bundle\ModuleBundle\Entity\
    MenuRepository")
 * @ORM\InheritanceType("JOINED")
 * @ORM\DiscriminatorColumn(name="type", type="string")
```

```
 * @ORM\DiscriminatorMap({"menu" = "Menu", "submenu" = "SubMenu"})
 */
class Menu extends ContainerAware
{
```

```
/**
 * SubMenu
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="ECM\Bundle\ModuleBundle\Entity\
   SubMenuRepository")
 */
class SubMenu extends Menu
{
```

`Inheritance Type` sert à déterminer comment Doctrine converti cet héritage en tables SQL (soit `JOINED`, ou `SINGLE_TABLE`). Ici, `JOINED` fait en sorte que deux tables soient crées (Menu et SubMenu). Menu contient un attribut discriminant (spécifié dans `Discriminator Column`) faisant la différence entre un Menu et un SubMenu. Lorsque ce discrimant vaut "menu", doctrine sait que c'est un menu. Au contraire lorsqu'il vaut "submenu", il sait que c'est un sous-menu et référence directement ce sous-menu dans l'autre table SQL SubMenu contenant tous les attributs spécifiés dans SubMenu et pas dans Menu.

Conteneur de Menus

Un Service jouant le rôle d'un Repository permet d'accéder aux menus contenus dans la base de données depuis le MenuBuilder de KNP.

Deux méthodes sont implémentées :

— getMenus permettant de récupérer tous les menus (et seulement les menus)

— getSubMenus permettant de récupérer tous les enfants d'un menu

```
class MenusContainer {

    private $doctrine;

    public function __construct(Container $container){
        $this->doctrine = $container->get('doctrine');
    }

    public function getMenus(){
        return $this->doctrine->getRepository('ECMModuleBundle:Menu')->
   getMenusOnly();
    }

    public function getSubMenus($menu){
        return $menu->getEnfants();
    }

}
```

La configuration de ce service se situe dans le fichier services.yml du bundle HomeBundle. Grâce à ce fichier, le Service est enregistré par le kernel de Symfony dans le Service Container.

```
    ecm_home.menu_builder:
        class: ECM\Bundle\HomeBundle\Menu\MenuBuilder
        arguments: ["@knp_menu.factory", "@service_container"]
```

Affichage des menus et sous-menus

Pour afficher les menus et sous-menus, la fonction `generateMenu()` parcourt le tableau de menus renvoyé par le conteneur de menus (`getMenus()`). Pour chaque menu, tous ses potentiels enfants sont récupérés et sont ajoutés en tant qu'enfant, via `addChild()`, au menu parent.

```
1    public function generateMenu(){
2
3        $menu = $this->factory->createItem('root');
4        $menu->setChildrenAttribute('class', 'nav navbar-nav');
5        $menus = $this->serviceContainer->get('ecm_home.menus.container'
    )->getMenus();
6        $menu->addChild('', array('route' => 'ecm_home_homepage'))
7            ->setAttribute('glyphicon', 'home');
8        foreach ($menus as $menuItem) {
9            $slug = $this->serviceContainer->get('slugify')->slugify(
    $menuItem->getTitre());
10           $menu->addChild($menuItem->getTitre(), array('route' => '
    ecm_articles_show_by_menu', 'routeParameters' => array('titreMenu' =>
     $slug)));
11           foreach ($menuItem->getEnfants() as $submenuItem) {
12               $menu[$menuItem->getTitre()]->setAttribute('dropdown',
    true);
13               $subSlug = $this->serviceContainer->get('slugify')->
    slugify($submenuItem->getTitre());
14               $menu[$menuItem->getTitre()]->addChild($submenuItem->
    getTitre(), array('route' => 'ecm_articles_show_by_menu', '
    routeParameters' => array('titreMenu' => $subSlug)));
15           }
16       }
17       $menu->addChild("Boutique", array('route' => '
    ecm_shop_allProduct'))
18           ->setAttribute('glyphicon', 'shopping-cart');
19
20       return $menu;
21   }
```

Slugify

Slugify est un petit module permettant de créer des slugs (jetons fabriqués à la volée à partir de chaînes de caractères). En l'occurence, ce module est utilisé pour générer des chaînes de caractères compatibles aux URLs (pour les menus).

5.3.5 Gestion des utilisateurs

Cette partie a été développé à l'aide du bundle FOSUserBundle.

FOSUserBundle

FOSUser sert à la gestion des membres (inscription / connexion / déconnexion / changement de mot de passe)

```
1 # FOSUser Configuration
2 fos_user:
```

```
    db_driver: orm # other valid values are 'mongodb', 'couchdb' and '
    propel'
    firewall_name: main
    user_class: ECM\Bundle\UserBundle\Entity\User
    registration:
        confirmation:
            enabled:    true
        form:
            type: ecm_user_registration
```

Un UserBundle a été développé pour gérer les utilisateurs. Ce Bundle Symfony étend de FOSUser, il hérite donc de toutes les méthodes et les fonctionnalités que propose FOSUserBundle. La mise en place de ce Bundle Symfony est simple : il suffit de spécifier la classe que l'on veut faire hériter de FOSUser UserBase.

5.3.6 Gestion des sponsors

La gestion de l'administration des sponsors est la même que pour les autres fonctionnalités (mapping de l'entité et configuration des champs affichés dans une classe `XxxAdmin`). Cependant, la gestion des sponsors nécessite un petit module permettant d'ordonner la position des sponsors.

Pix Sortable Bundle

Pix Sortable Bundle est un Bundle Symfony qui permet de trier et d'ordonner toute sorte entité. Pix Sortable Bundle sert à l'ordonnancement des sponsors, c'est à dire qu'il est possible de modifier l'ordre d'affichage des sponsors. Une petite fonction intégrée à `SponsorAdmin` permet d'interfacer ce module avec Sonata

6 Résultats et perspectives

6.1 Résultats

6.1.1 Vitrine

Le site contient une vitrine servant à présenter le club aux futurs adhérents ou aux personnes recherchant des informations sur le club. Mais aussi aux adhérents qui veulent se tenir informés des différentes news et évènements.

L'administrateur peut ajouter, modifier ou supprimer des publications ce qui rend la vitrine parfaitement dynamique.

6.1.2 Inscription d'un membre

Les internautes désirant s'inscrire au site, peuvent le faire via un formulaire dédié. Les inscriptions doivent être validées par mail ce qui a pour but d'empêcher les robots de s'inscrire.

Une fois inscrit, un utilisateur peut modifier sont mot de passe via le formulaire adéquat.

6.1.3 Gestion des articles/sponsors

Le site permet à l'administrateur de gérer les sponsors. Il permet d'en créer, de les modifier, ou de les supprimer, mais aussi de trier l'ordre d'affichage, en fonction de leur importance, les sponsors affichés sur le site (amenant sur le site web du sponsor).

Le site possède un système de publication et de modération des articles, un utilisateur inscrit peut proposer un article, via un formulaire correspondant. Cet article une fois validé est soumis à la modération de l'admin. Si l'article et accepté, il sera affiché sur le site. Dans le cas contraire l'utilisateur est invité à modifier son article.

6.1.4 Gestion des catégories/produits

Le site contient la liste des produits vendus par le club en fonction de leur catégorie. Ces produits et ces catégories sont donc administrables, l'administrateur peut créer, modifier ou supprimer des catégories. Le site permet la création, la modification et la suppression des produits, les produits appartiennent à une catégorie.

6.2 Perspectives

6.2.1 Boutique

Dans une évolution future, il sera utile d'ajouter un panier et de finaliser la partie boutique du site web, pour que les utilisateurs puissent acheter les produits directement sur le site web.

6.2.2 Gestion d'équipes

Il serait aussi possible d'ajouter une partie "équipes", pour faciliter la gestion de ces dernières par les chefs d'équipe lors de l'organisation de tournois.

6.2.3 Gestion des membres/inscription

Il faudrait rajouter une gestion des membres ainsi qu'un système permettant seulement aux adhérents du club de s'inscrire au site, ou de poster des articles. Une réunion avec le client devrait se faire pour décider de la solution la plus adaptée.

Troisième partie

Rapport d'activité

Gestion des itérations

Pour le démarrage du projet, nous avons fait une réunion avec notre tuteur, M. Palleja où nous avons discuté de ses attentes et du contenu global du projet. Nous avons fait un rapport préliminaire afin de bien vérifier la bonne compréhension du sujet que nous avons rendu la semaine suivante. Lors de la réunion qui suivi le rendu du rapport nous avons pu débattre des idée et proposition que nous avions à faire avant de nous lancer totalement dans le projet.

Par la suite, nous avons suivis une démarche agile en travaillant par itérations de deux semaines durant la période de cours, et en réduisant ce délais de moitié pour les dernières semaines consacrées au projet. Pour l'intégration nous avons utilisé le gestionnaire de version Git que nous avons intégré à nos Environnements de développement . Nous avons mis en place un Kanban via la site de Trello pour nous répartir les tâches lors des premières itérations.

Difficultés rencontrées

Mais, face aux difficultés techniques que nous avons rencontrées quant à l'utilisation de Sonata, nous nous sommes regroupés derrière un seul codeur pour comprendre le fonctionnement de Sonata et le mettre en place... Une fois ceci fait, nous avons réussi à nous diviser en deux équipes afin de terminer la mise en place de la gestion des articles et de la boutique.

Conclusion

En revenant sur notre progression, il aurait été préférable de ne pas utiliser Sonata Admin Bundle pour la gestion du site car, bien que ce module soit performant, l'équipe ne possédait pas les compétences nécessaires pour sa mise en place dans un délai acceptable. De plus, le module possédait énormément de fonctionnalités superflues pour notre projet. Une recherche plus approfondie d'un autre module, voire s'en passer aurait sûrement été préférable. D'un autre coté, d'un point de vue pédagogique l'installation de Sonata nous a permis de mieux comprendre le fonctionnement de Symfony2 et de l'installation de modules, l'installation de Sonata n'a pas été une totale perte de temps.

Quatrième partie

Manuel d'utilisation

7 Accéder à l'interface d'administration

Pour pouvoir accéder au back office il faut dans un premier temps, se connecter avec un compte administrateur (voir figures 7.1 et 7.2).

Une fois connecté il faut cliquer sur le bouton Administration du menu (voir figure 7.3). Ensuite le site redirige automatiquement vers le back office (voir figure 7.4).

Le back office est constitué de trois blocs, le premier bloc "contenu" va servir à ajouter/modifier/supprimer une publication et à la modération des articles postés par les utilisateurs.

Ensuite le second bloc, intitulé "modules" sert à ajouter/modifier/supprimer/ordonner les sponsors, les menus, les sous menu ,et a l'envoi d'images sur le serveur.

Pour finir, le dernier bloc sert à la gestion de la boutique, c'est dire à ajouter/modifier/supprimer des produits et des catégories.

Le fonctionnement du back office est sensiblement identique pour chaque item du site. Tous ces items à l'exception des Articles ont un bouton "ajouter" qui permet d'être redirigé vers le formulaire d'ajout correspondant à l'item. Il est aussi possible d'avoir la liste de toutes les entités par item. Par exemple : le bouton liste de la catégorie publication affichera la liste de toutes les publications qui sont enregistrées sur le site. La page liste permet de modifier et de supprimer une entité de l'item sélectionné La modification se déroule via le même formulaire que l'ajout, à la différence que, les champs sont pré-remplis avec les valeur de l'entité modifiée.

Pour illustrer le fonctionnement du back office, l'administration d'une publication et des sponsors vont être détaillés.

FIGURE 7.1 – Bouton connexion sur la page d'accueil

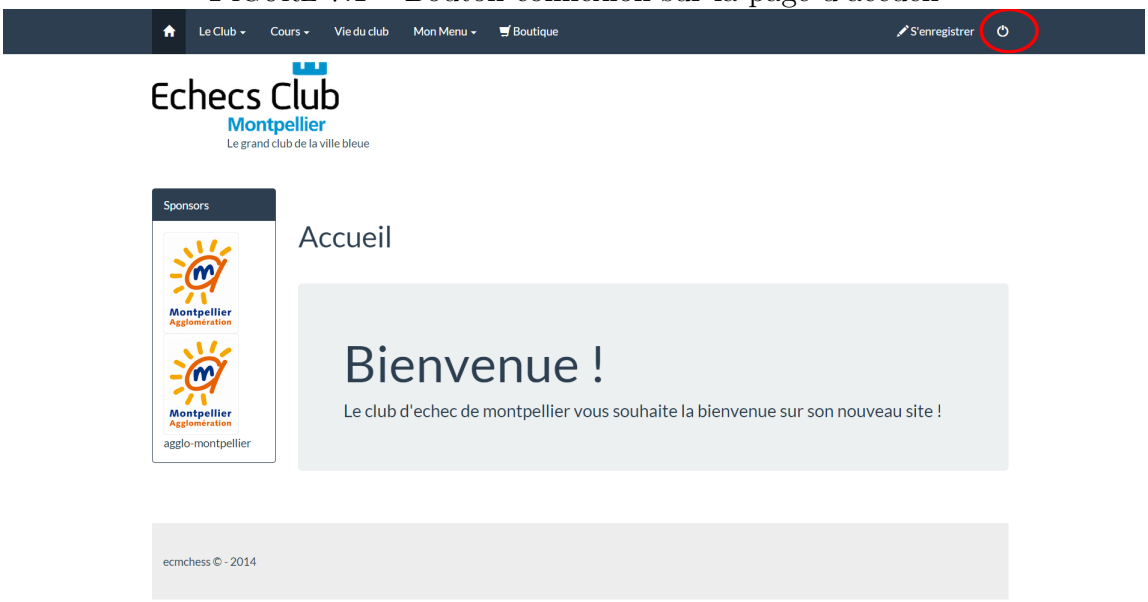


FIGURE 7.2 – Page de connexion

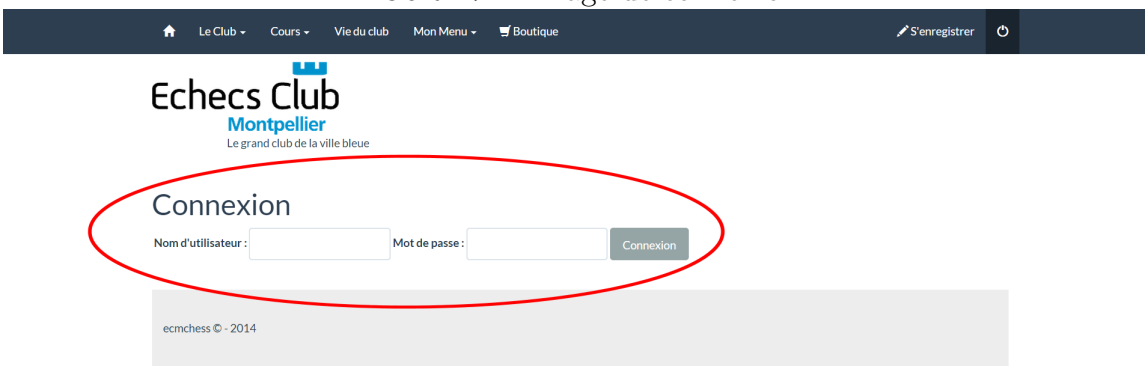


FIGURE 7.3 – Bouton administration sur la page d'accueil

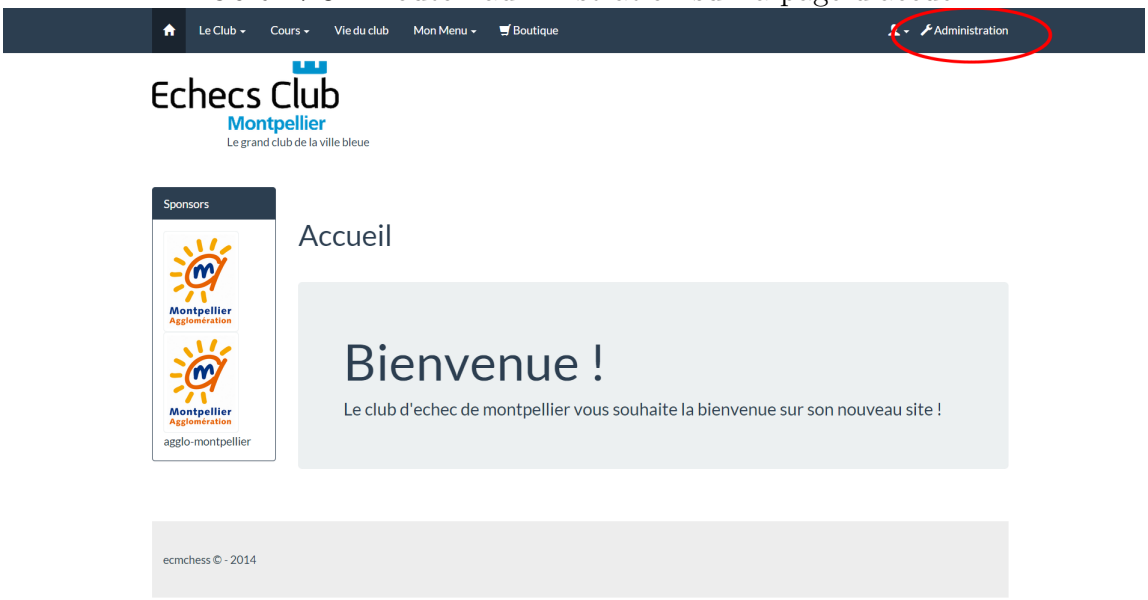
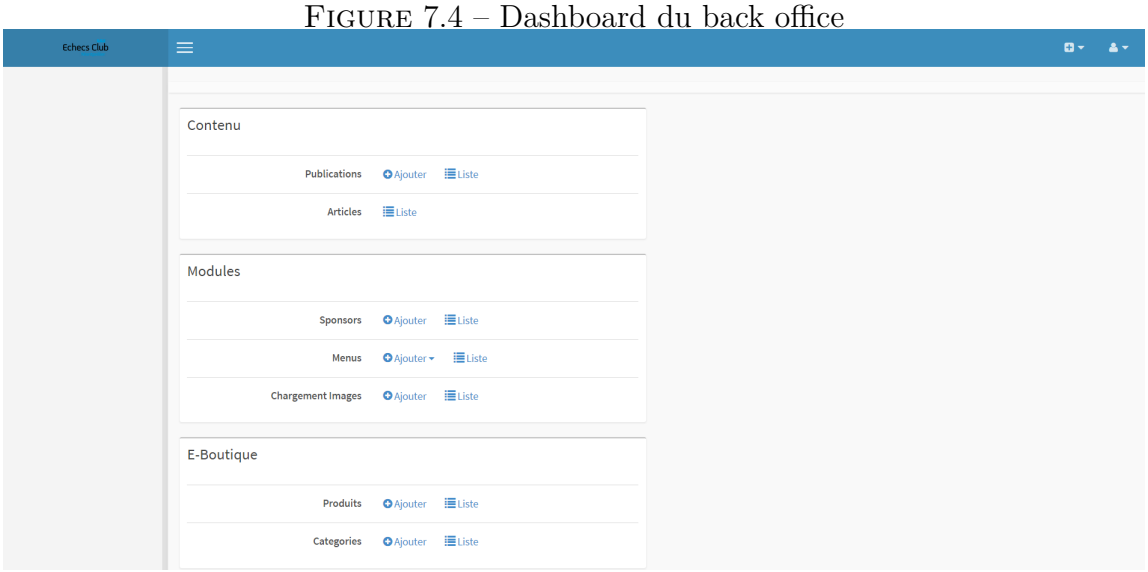


FIGURE 7.4 – Dashboard du back office



8 Administration des publications

Lors du clique sur le bouton liste le système affiche une page avec toutes les publications du site, chaque publication est modifiable et supprimable via les boutons prévu à cet effet. Il est possible aussi de modifier un article en cliquant sur son titre (voir figure 8.1). Lorsque l'on souhaite modifier un article , le système nous redirige vers la page de modification. Sur cette page il est possible de modifier le titre via la zone de texte correspondante, le corps du texte, quand a lui se gère via un champ CKEditor (dont nous avons détaillé le fonctionnement précédemment). Il est aussi possible de modifier le menu dans lequel la publication sera affichée, la liste déroulante correspondante contient les menus du site.

Pour valider on peut soit cliquer sur "Mettre à jour" pour valider et rester sur la même page, soit cliquer sur "Mettre à jour et fermer" ce qui enregistre les modification et renvoie sur la liste des publications.

Le bouton supprimer sert, quant à lui, à supprimer la publication (voir figure 8.2). Il est aussi possible d'ajouter une publication via le bouton "Ajouter", sur la page d'accueil du back office, l'ajout d'une publication se fait via le formulaire suivant, dont le fonctionnement ne diffère pas de la modification (voir figure 8.3).

FIGURE 8.1 – Bouton connexion sur la page d'accueil

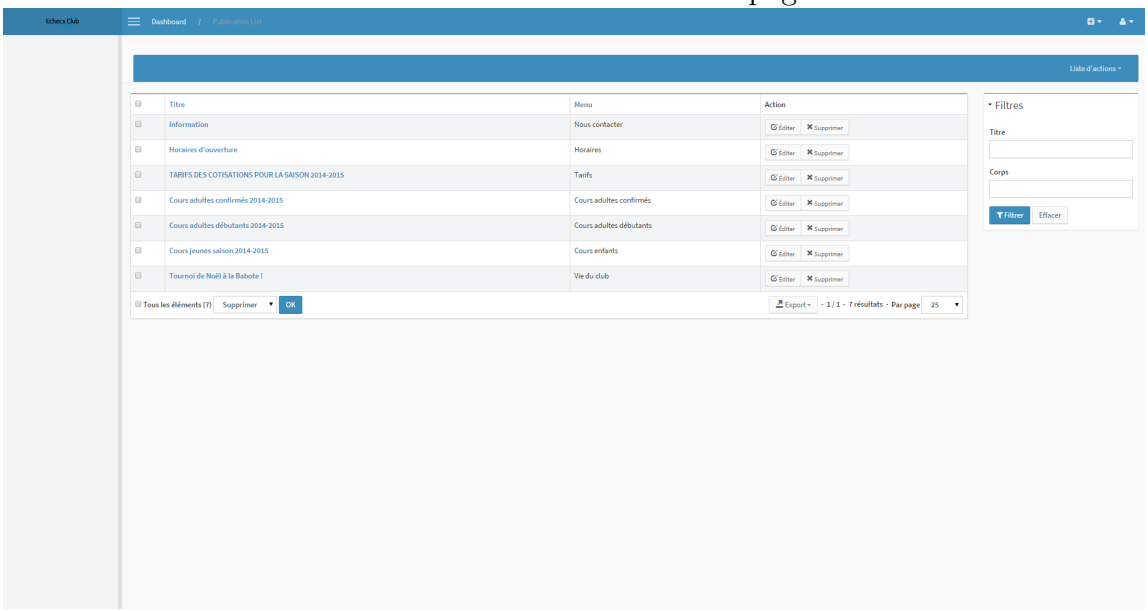


FIGURE 8.2 – Édition d'une publication

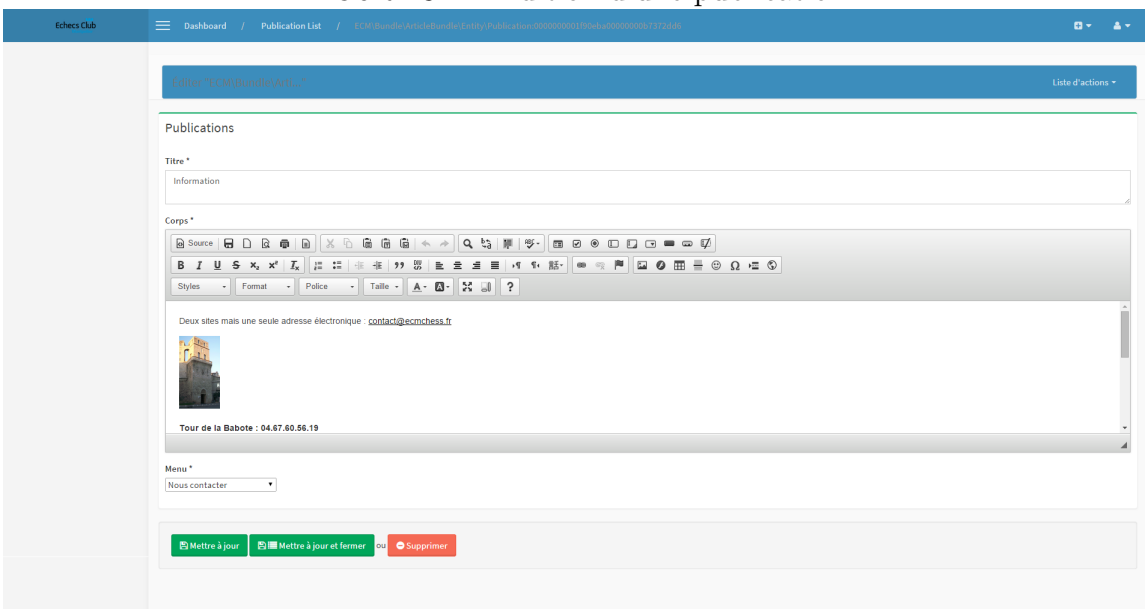
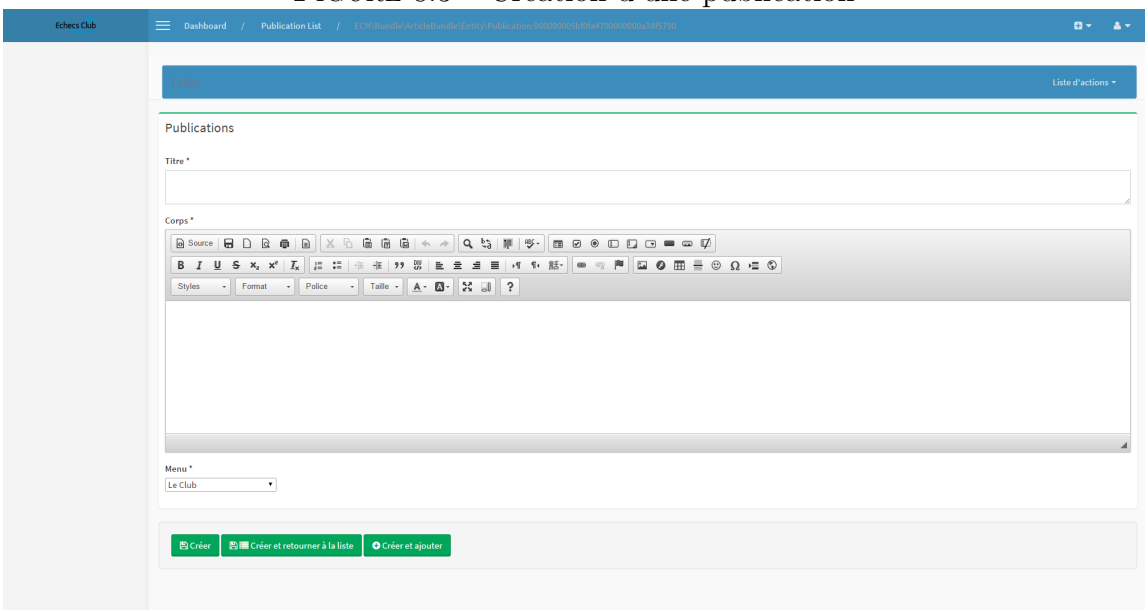


FIGURE 8.3 – Création d'une publication



9 Administration des sponsors

Pour ajouter un sponsor il faut cliquer sur le bouton “ajouter” de la page d’accueil du back office dans la partie sponsor du bloc “module”, le système nous affiche alors le formulaire adéquat.

Il suffit alors de renseigner le titre, le lien vers lequel pointera le sponsor, l’image qui s’affichera (l’image n’est pas obligatoire) ainsi que sa position (voir figure 9.1). La modification se fait via un formulaire sensiblement identique.

Il est aussi possible de modifier la position d’un sponsor de façon dynamique sur la page liste des sponsors, il suffit de cliquer sur les flèches pour faire monter ou descendre les sponsors (voir figure 9.2).

FIGURE 9.1 – Création d'un sponsor

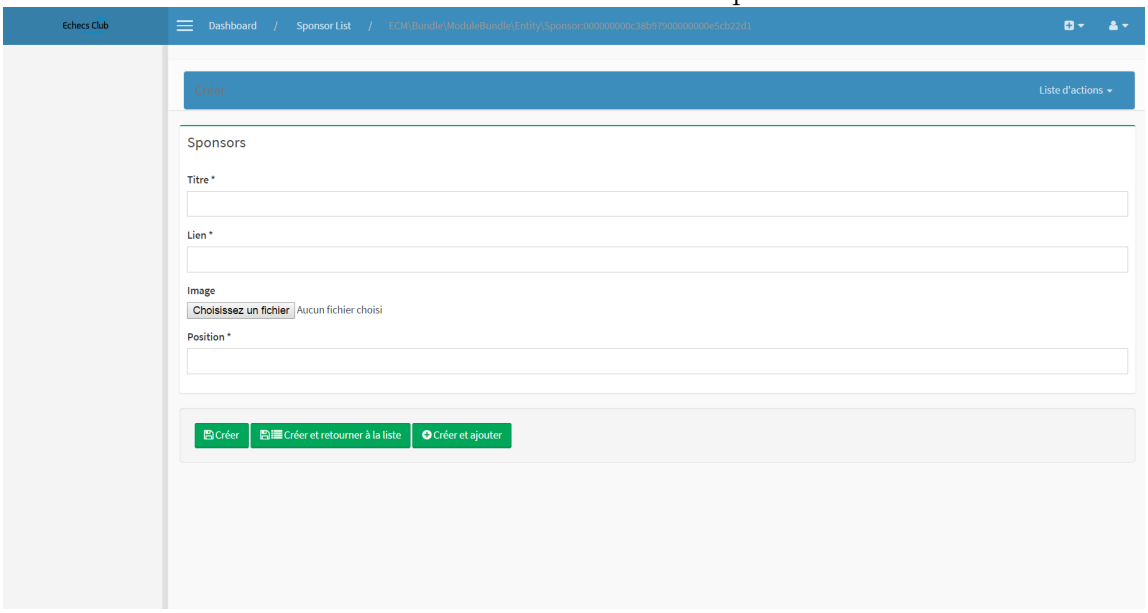
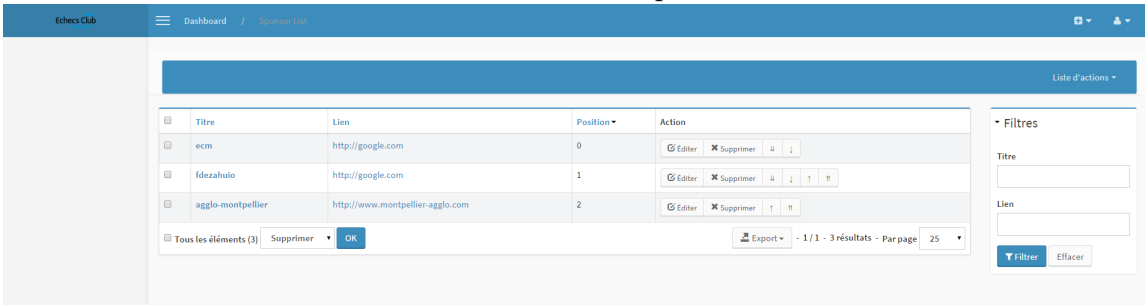


FIGURE 9.2 – Liste des publications



Conclusion

Nous avons réalisé le projet demandé en grande partie mais beaucoup d'améliorations et d'évolutions restent possibles, en effet, le site a fait peau neuve, possède une interface d'administration simple d'utilisation pour gérer l'architecture et le contenu du site, et permet à ses membres de poster (avec vérification préalable) des articles. Une vitrine permetant de présenter les produits a elle aussi été mise en place. En revanche, un panier pourrait être envisagé pour permettre aux utilisateur de faire leur commande en ligne. Cependant, certaines fonctionnalités n'ont pas pu être réalisées comme par exemple la gestion des équipes et une gestion plus approfondie des inscriptions.

Dans l'ensemble ce projet nous à permis de mieux appréhender les mécaniques de fonctionnement de Symphony, et nous a appris à nous méfier de l'installation hâtive de modules. Le travail en équipe nous a aussi poussé à utiliser des outils de travail communautaires (Git, Trello) qui sont largement utilisés en entreprise pour la gestion de projet.

A Script SQL phpmyadmin

```
1
2
3
4  CREATE DATABASE `lacombes` DEFAULT CHARACTER SET latin1 COLLATE
       latin1_swedish_ci;
5  USE `lacombes`;
6
7  -- --------------------------------------------------------
8
9
10
11 CREATE TABLE `Categorie` (
12   `id` int(11) NOT NULL AUTO_INCREMENT,
13   `nom` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
14   PRIMARY KEY (`id`)
15 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
       AUTO_INCREMENT=3 ;
16
17 -- --------------------------------------------------------
18
19
20 CREATE TABLE `Image` (
21   `id` int(11) NOT NULL AUTO_INCREMENT,
22   `nom` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
23   `updated` date DEFAULT NULL,
24   `urlImage` longtext COLLATE utf8_unicode_ci,
25   PRIMARY KEY (`id`)
26 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
       AUTO_INCREMENT=4 ;
27
28 -- --------------------------------------------------------
29
30
31
32 CREATE TABLE `Menu` (
33   `id` int(11) NOT NULL AUTO_INCREMENT,
34   `titre` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
35   `glyphicon` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
36   `type` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
37   PRIMARY KEY (`id`),
38   UNIQUE KEY `UNIQ_DD3795ADFF7747B4` (`titre`)
39 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
       AUTO_INCREMENT=22 ;
40
41 -- --------------------------------------------------------
42
```

```
43
44
45 CREATE TABLE `Produit` (
46   `id` int(11) NOT NULL AUTO_INCREMENT,
47   `categorie_id` int(11) DEFAULT NULL,
48   `nom` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
49   `prix` double NOT NULL,
50   `description` longtext COLLATE utf8_unicode_ci NOT NULL,
51   `image_id` int(11) DEFAULT NULL,
52   PRIMARY KEY (`id`),
53   KEY `IDX_E618D5BBBCF5E72D` (`categorie_id`),
54   KEY `IDX_E618D5BB3DA5256D` (`image_id`)
55 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
     AUTO_INCREMENT=3 ;
56
57 -- --------------------------------------------------------
58
59
60
61 CREATE TABLE `Publication` (
62   `id` int(11) NOT NULL AUTO_INCREMENT,
63   `menu_id` int(11) DEFAULT NULL,
64   `auteur_id` int(11) DEFAULT NULL,
65   `etat` int(11) NOT NULL,
66   `titre` longtext COLLATE utf8_unicode_ci NOT NULL,
67   `corps` longtext COLLATE utf8_unicode_ci NOT NULL,
68   `date` datetime NOT NULL,
69   `type` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
70   `motif` longtext COLLATE utf8_unicode_ci,
71   PRIMARY KEY (`id`),
72   KEY `IDX_29A0E8AECCD7E912` (`menu_id`),
73   KEY `IDX_29A0E8AE60BB6FE6` (`auteur_id`)
74 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
     AUTO_INCREMENT=23 ;
75
76 -- --------------------------------------------------------
77
78
79 CREATE TABLE `Sponsor` (
80   `id` int(11) NOT NULL AUTO_INCREMENT,
81   `titre` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
82   `urlImage` longtext COLLATE utf8_unicode_ci,
83   `lien` longtext COLLATE utf8_unicode_ci NOT NULL,
84   `position` int(11) NOT NULL,
85   `updated` date DEFAULT NULL,
86   PRIMARY KEY (`id`)
87 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
     AUTO_INCREMENT=4 ;
88
89 -- --------------------------------------------------------
90
91
92
93 CREATE TABLE `SubMenu` (
94   `id` int(11) NOT NULL,
95   `id_parent` int(11) DEFAULT NULL,
96   PRIMARY KEY (`id`),
97   KEY `IDX_CB3FDEBA1BB9D5A2` (`id_parent`)
```

```
98  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
99
100 -- ------------------------------------------------------
101
102
103
104 CREATE TABLE `fos_user` (
105   `id` int(11) NOT NULL AUTO_INCREMENT,
106   `username` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
107   `username_canonical` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
108   `email` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
109   `email_canonical` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
110   `enabled` tinyint(1) NOT NULL,
111   `salt` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
112   `password` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
113   `last_login` datetime DEFAULT NULL,
114   `locked` tinyint(1) NOT NULL,
115   `expired` tinyint(1) NOT NULL,
116   `expires_at` datetime DEFAULT NULL,
117   `confirmation_token` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL
      ,
118   `password_requested_at` datetime DEFAULT NULL,
119   `roles` longtext COLLATE utf8_unicode_ci NOT NULL COMMENT '(DC2Type:
      array)',
120   `credentials_expired` tinyint(1) NOT NULL,
121   `credentials_expire_at` datetime DEFAULT NULL,
122   PRIMARY KEY (`id`),
123   UNIQUE KEY `UNIQ_957A647992FC23A8` (`username_canonical`),
124   UNIQUE KEY `UNIQ_957A6479A0D96FBF` (`email_canonical`)
125 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
      AUTO_INCREMENT=14 ;
126
127 -- ------------------------------------------------------
128
129
130 -- ------------------------------------------------------
131
132
133
134 ALTER TABLE `Produit`
135   ADD CONSTRAINT `FK_E618D5BB3DA5256D` FOREIGN KEY (`image_id`)
      REFERENCES `Image` (`id`),
136   ADD CONSTRAINT `FK_E618D5BBBCF5E72D` FOREIGN KEY (`categorie_id`)
      REFERENCES `Categorie` (`id`);
137
138
139 ALTER TABLE `Publication`
140   ADD CONSTRAINT `FK_29A0E8AE60BB6FE6` FOREIGN KEY (`auteur_id`)
      REFERENCES `fos_user` (`id`),
141   ADD CONSTRAINT `FK_29A0E8AECCD7E912` FOREIGN KEY (`menu_id`)
      REFERENCES `Menu` (`id`) ON DELETE SET NULL;
142
143
144 ALTER TABLE `SubMenu`
145   ADD CONSTRAINT `FK_CB3FDEBABF396750` FOREIGN KEY (`id`) REFERENCES `
      Menu` (`id`) ON DELETE CASCADE,
146   ADD CONSTRAINT `FK_CB3FDEBA1BB9D5A2` FOREIGN KEY (`id_parent`)
      REFERENCES `Menu` (`id`);
```

B Diagramme des classes

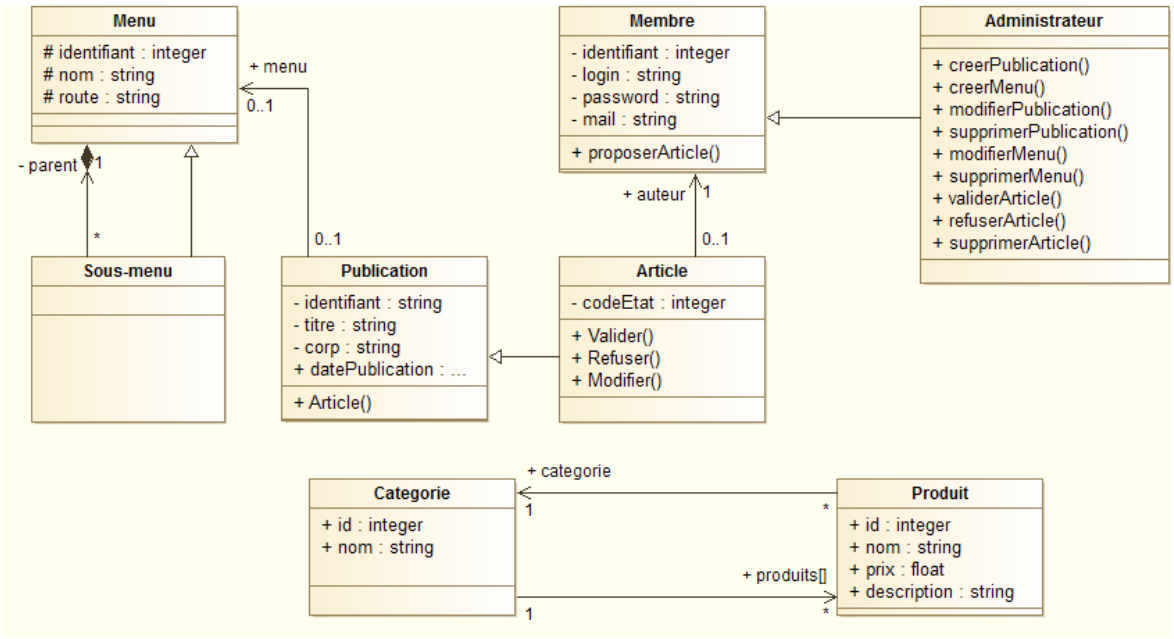


Table des figures

4.1 Diagramme des cas d'utilisation 6

5.1 Fonctionnement de Symfony 2 11
5.2 Zone d'édition CKEditor 12
5.3 Diagramme d'État-transition d'un article 13
5.4 Proposer un article 13
5.5 Diagramme de Séquence - Proposer un article 14
5.6 Liste des articles sur l'interface d'administration 15
5.7 Modification d'un article sur l'interface d'administration 15
5.8 Diagramme de Séquence - Administration d'un article 16
5.9 Menu Administration 17

7.1 Bouton connexion sur la page d'accueil 28
7.2 Page de connexion 28
7.3 Bouton administration sur la page d'accueil 29
7.4 Dashboard du back office 29

8.1 Bouton connexion sur la page d'accueil 31
8.2 Édition d'une publication 31
8.3 Création d'une publication 32

9.1 Création d'un sponsor 34
9.2 Liste des publications 34

Résumé

Le Projet ECMCHESS consiste à réaliser un site web vitrine pour un club d'échec montpellierain. Ce dernier devait pouvoir fournir divers informations sur le club, informer ses membres et permettre l'achat de matériel d'échec via une boutique en ligne Le site a été développé en utilisant le framework Symfony 2 ainsi que des modules notamment Bootstrap et Sonata. mots-clés : Échecs, Club d'échec, Symfony 2, boutique en ligne.

Abstract

ECMCHESS project consists in developping a website for a chessclub of Montpellier. It was aimed to show various inforation about the club, display news and events to the club's members and allow customer to buy chess's stuff on an online store. This web site has been developped using the Symfony 2 framework and bundles for instance Bootstrap and Sonata.

key words : Chess, chess club, symfony 2, e-shop.