GUIRAUD Maxime

2ème année – DUT Informatique

Rapport de stage

Développement du site web de la Commune de Jallans

Maitre de Stage : M. Olivier Lecomte

Tuteur de Stage : Mme. Caroline Kalla

Année universitaire 2014/2015

Remerciement :

J'aimerais tout d'abord remercier l'ensemble du conseil municipal de la commune de Jallans de m'avoir accepté en stage et de m'avoir accueilli pendant 10 semaines.

Ensuite je remercie mon maitre de stage, Monsieur Lecomte Olivier, maire de Jallans, madame Dupont Angélique et Catherinot Yves, respectivement premier adjoint et second adjoint au maire ainsi que l'ensemble du personnel de la mairie pour leur disponibilité et leur accompagnement tout au long du stage.

Enfin, je tiens à remercier l'ensemble du corps enseignant du département informatique de l'IUT d'Orléans pour leurs enseignements et leur investissement tout au long de ces 2 années qui m'ont permis d'acquérir la plupart des connaissances dans le domaine de l'informatique dont je dispose aujourd'hui.

Sommaire :

Introduction :

Aujourd'hui, nous sommes dans l'ère du numérique. Internet est le reflet du monde dans lequel on vit, et notre réputation est construite par les informations à notre sujet présent sur le web. Il est donc tout naturel pour la commune de Jallans que de vouloir un site internet à son image. Une présence sur le web pouvant lui permettre un plus large rayonnement et une mise en valeur de son développement et son investissement journalier pour le bien-être de ses riverains. C'est donc dans cette optique que j'ai été pris en stage

Par ailleurs, ce stage est aussi l'aboutissement des 2 ans de ma formation à l'IUT (Institut Universitaire et Technologique) d'Orléans, département informatique. Ces dix semaines de stage vont permettre de mettre en application l'ensemble de mes connaissances acquises à l'IUT.

Tout au long de ce rapport, je reviendrais donc sur l'ensemble des processus de conception qui m'ont permis de créer le site web de la commune, en commençant par l'analyse du projet, puis en détaillant le développement du site. Enfin je reviendrais sur ce que m'a apporté ce stage.

II. Présentation de la commune de Jallans.

Jallans est une commune de plus de 800 habitants situé en Eure-et-Loir. La commune, regroupant le village de Jallans ainsi que le hameau de Jumeaux, a une superficie de 9km², principalement réservé à l'agriculture. De par sa proximité avec la ville de Châteaudun, la commune ne dispose pas de commerce et est avant tout un lieu d'habitation. Jallans possède cependant une école, ce qui lui permet d'attirer une population familiale.

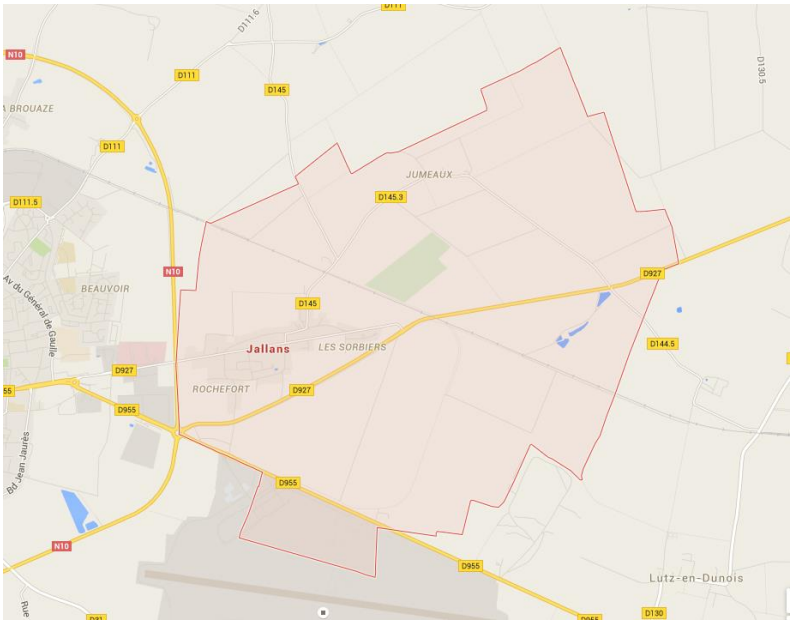


Figure 1- carte de la commune de Jallans, la zone rouge représente la superficie de celle-ci

La commune est connue en partie pour son Eglise et pour sa fontaine qui donne son nom au village. En effet, Jallans a depuis toujours une fontaine, aujourd'hui située dans une propriété privé. Jallans tire donc son nom du verbe jaillir.

Aujourd'hui, la commune a pour maire monsieur Olivier Lecomte, 35 ans, formateur et autoentrepreneur dans l'informatique.

Figure 2 - photo officiel de Monsieur Olivier Lecomte, Maire de la commune de Jallans

III. Sujet du stage

Conscient de l'atout qu'est un site internet aujourd'hui, mon stage a pour but de créer un site web pour la commune de Jallans. Il en existe déjà un, cependant celui-ci ne correspond plus au standard de navigation d'aujourd'hui et est extrêmement difficile à mettre à jour.

Il m'a donc été demandé de refaire un site web, plus en phase avec les technologies actuelles et facile d'utilisation, aussi bien pour l'utilisateur, qui doit pouvoir trouver facilement l'information qu'il recherche, que pour le personnel de mairie, qui doit pouvoir y ajouter du contenu facilement et l'administrer simplement.

Hormis les demandes présentées ci-dessus, j'ai comme initiative que de développer le site le plus agréable possible, que ce soit par son interface ou par ses fonctionnalités.

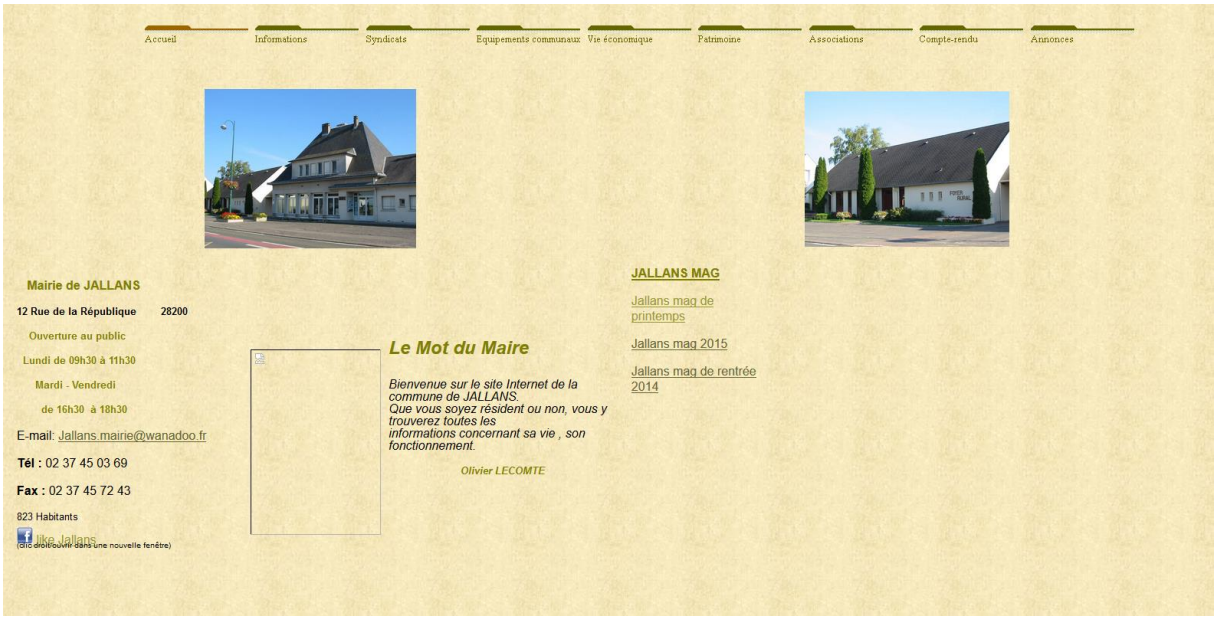


Figure 3 - Page d'accueil du site actuel de la commune, hormis le contenu, rien ne sera conservé

IV. Conception

a) Définition des besoins

Avant développer le site web, il me faut en définir l'ensemble de ses fonctionnalités et les outils nécessaires à sa réalisation. Cela s'est fait durant la phase de conception.

Dans l'optique de définir les fonctionnalités du site web, des entretiens avec mon maitre de stage, à savoir monsieur le Maire, ont eu lieu. Au cours de ces entretiens, une liste de fonctionnalités a émergé :

- On doit pouvoir publier des annonces de manière simple.
- Il faut pouvoir éditer le contenu du site facilement
- Il faut pouvoir accorder différents droits aux administrateurs.
- Les associations de la commune doivent pouvoir éditer le contenu lié à leur association.
- Le site web doit pouvoir témoigner de l'activité de la commune.

À la suite de ça, je réalise donc un **diagramme de Cas d'utilisation**, diagramme de conception permettant de d'obtenir une vision général du fonctionnement du système en développement, ici un site web.

Ce diagramme permet de résumer l'utilisation du site et d'affiner les fonctionnalités à implémenter. Il permet de voir simplement ce que doit pourvoir faire l'utilisateur.

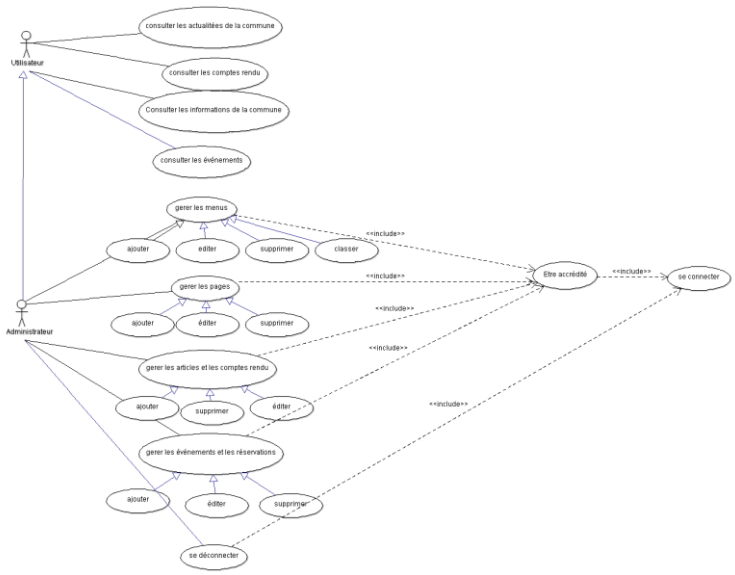


Figure 4 - Diagramme de cas d'utilisation, voir annexes 1

Dans le cas présent, l'administrateur doit pouvoir faire tout ce que peut faire l'utilisateur final, et toutes les fonctions d'administration. Ces fonctions nécessitent tout de même d'avoir les droits requis.

b) La maquette

J'ai réalisé plusieurs maquettes pour le site. Ces maquettes ont pour but de présenter plusieurs styles possibles applicables au site et de permettre au maire de choisir celui qu'il trouve le plus adéquate. Faire la maquette en amont permet aussi de guider de le développement et d'avoir rapidement une idée de ce à quoi doit ressembler le site web.

En premier lieux j'ai réalisé plusieurs maquettes sur papier pour trouver des idées applicables au site.

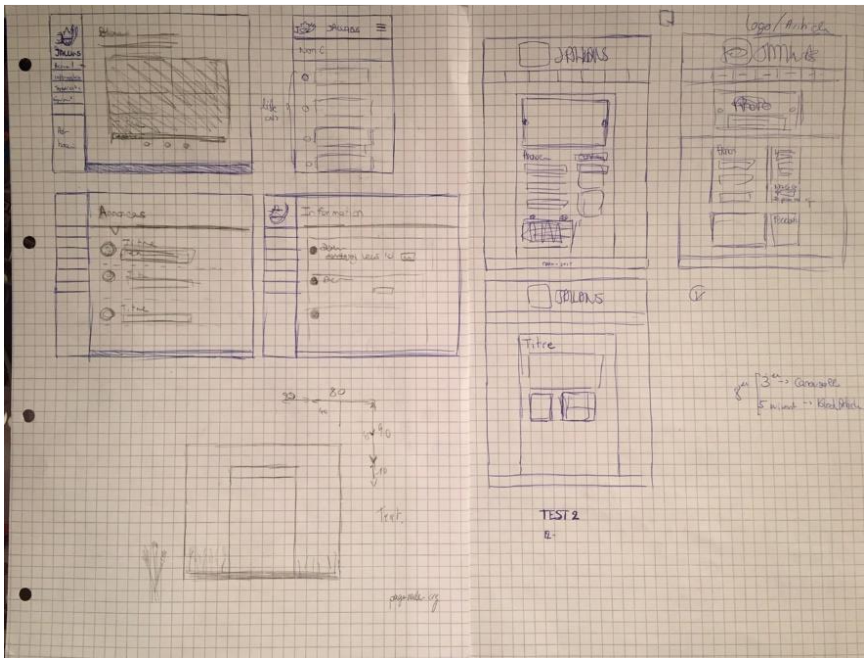


Figure 5 - Exemple de maquette préliminaire

Puis une fois les éléments trouvés, j'ai créé 2 maquettes, plus détaillées et complètes sur ordinateur pour présenter ce à quoi pourrait ressembler le site au final.

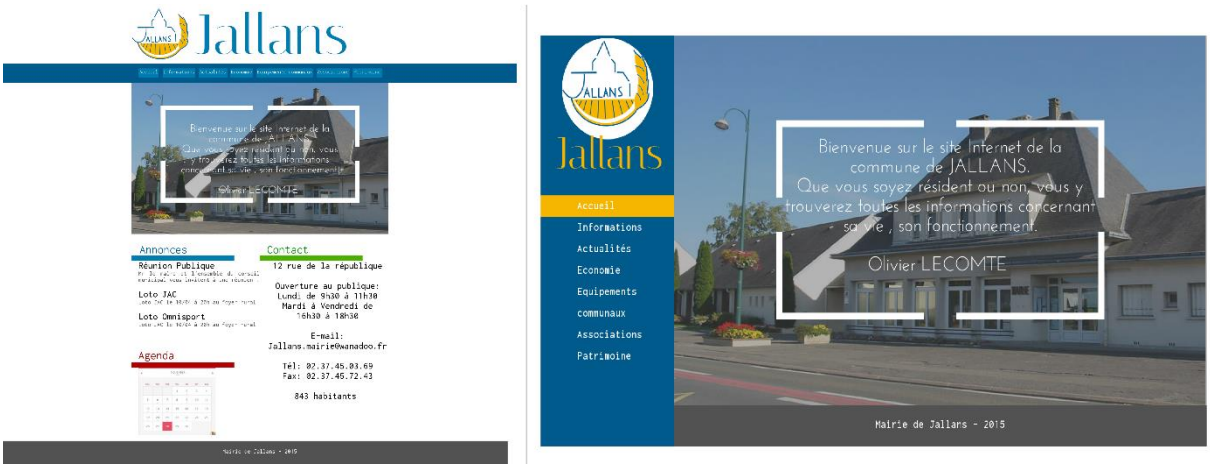


Figure 6 - Page d'accueil des 2 maquettes proposées

Après présentation, le premier choix a été retenu car jugé plus aéré et sobre.

c) Structure de la base de données

Etant donné les fonctionnalités nécessaires au site web, il faut conceptualiser une base de données pouvant répondre aux demandes du système. Pour ce faire j'ai fait un **Modèle Conceptuelle des Données** (MCD) ; ce diagramme permet définir les données utilisées par un système et les interactions entre elles.

Vu les besoins, la base de données est constituée de 5 tables :

- Une table **article** pour stocker les annonces et comptes rendus postés sur le site web.
- Une table **évènements** pour stocker les différents évènements liés à la commune.
- Une table **menu** pour stocker les menu du site, ce qui leurs permet d'être modulaires.
- Une table **pages** proche de la table article, pour stocker les pages d'informations du site, qui sont assez peu modifiés.
- Une table **utilisateurs** pour stocker les comptes administrateurs du site.

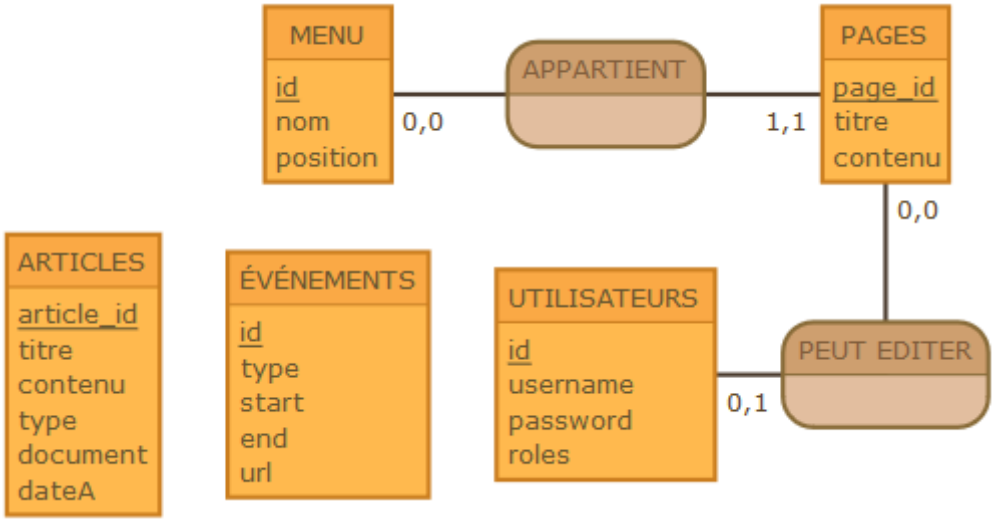


Figure 7 - Modèle Conceptuel des Données de la base de données

Il existe plusieurs relations entre les tables. Les tables pages et menus sont liées par une relation. Une page appartient à un menu ce qui permet par la suite d'obtenir un menu disposant de sous menu. De même, un utilisateur peut voir une page lui être associé pour qu'il puisse la modifier. Cela est utile dans le cas où l'utilisateur correspond au compte d'une association.

V. Développement

Une fois la conception préliminaire réalisée, viens la phase de développement. Cette phase, qui constitue la majeure partie du stage et s'articule autour de différentes thématiques dans le projet. On met en place, dans un premier temps, la structure. Puis vient ensuite l'implémentation des nombreuses fonctionnalités. Enfin viens l'interface utilisateur qui est à rendre agréable.

a) Les technologies utilisées

Avant toutes choses, il est nécessaire de présenter les technologies et logiciels utilisés pour la création de ce site web.

Les langages PHP et JavaScript sont utilisés pour la partie dynamique du site tandis que les langages HTML et CSS sont utilisés pour la partie statique du site, respectivement pour la structure des pages et l'aspect visuel.

Cependant, certaines technologies complémentaires sont aussi utilisées pour ce développement.

Silex :

Le site web repose en grande partie sur le *micro-framework* Silex. Un *framework*, en informatique, est un ensemble de composants chargés de gérer la structure d'un logiciel. Le *framework* défini l'architecture du logiciel et la manière dont il sera construit. En effet, les *frameworks* obligent les développeurs à respecter certaines règles de développements ; ces règles permettant d'avoir une uniformité entre les différents logiciels.

Créé par SensioLab, **Silex** est un *micro-framework* PHP basé sur le *framwork* **Symfony 2**. Il a pour avantage d'être très rapide à mettre en place, simple d'utilisation, et de quand même bénéficier des principales fonctionnalités de **Synfony 2**.

Twig :

Les différentes pages du site web sont générées par le *générateur de template* (*template engine)* **Twig**. Un *générateur de template* est un élément logiciel qui permet d'agréger les données avec les *templates*, qui sont des « patrons » dans lesquels sont ajoutées les données.

Twig est un *générateur de template* développé par SensioLab. Il est simple d'utilisation grâce à sa syntaxe proche du PHP et permet une génération rapide des pages.

MySql :

Pour stocker les données du site web, une base de données est utilisée. Le système retenu est **MySql** car c'est le système le plus utilisé pour le web. Cette base de données est administrée grâce au logiciel **PhpMyAdmin**.

Bootstrap :

L'interface est gérée par le *framework* CSS/Javascript Bootstrap. Ce *framework* a pour avantage d'intégrer un système de grille qui permet d'obtenir aisément une interface capable de s'adapter aux différentes tailles d'écrans (on parle d'interface *responsive*).

b) Mise en place de l'architecture Modèle-Vue-Contrôleur et de Silex

1. L'architecture MVC

Le cœur du développement commence par la mise en place de l'architecture du projet. Pour ce projet une architecture du type Modèle-Vue-Contrôleur (MVC) est adoptée. Cette architecture a pour principe de diviser l'application en trois parties distinctes pour faciliter sa gestion et son évolution. On a la Partie Modèle, c'est la partie qui gère le traitement des informations (tel que les interactions avec la base de donnée) ; la partie Vue qui va

permettre la mise en forme des informations et à l'utilisateur d'interagir avec l'application ; enfin, la partie Contrôleur fait le lien entre la partie Modèle et Vue. Concrètement, lorsque l'utilisateur change de page, il envoie un message au contrôleur qui indique au Modèle de charger les nouvelles données, données mise en forme par la Vue.

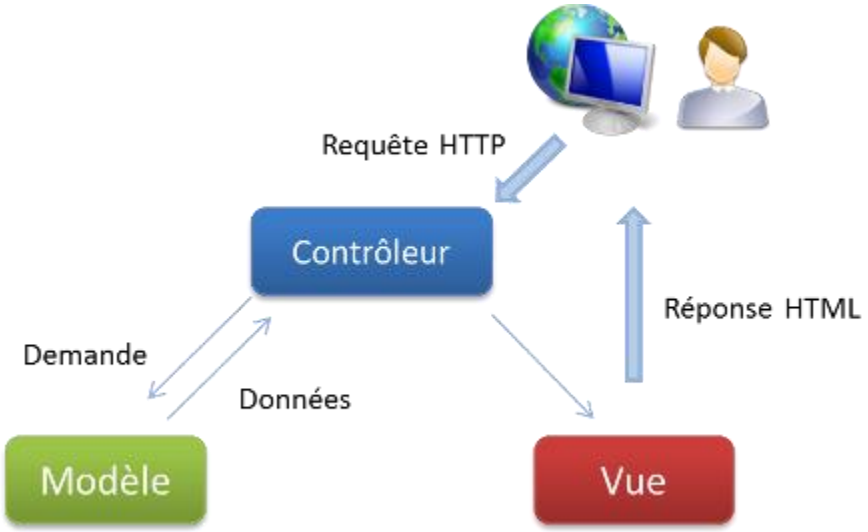


Figure 8 - Schéma résumant le principe Modèle-Vue-Contrôleur

Le *framework* Silex est conçu pour fonctionner selon ce principe. Pour installer Silex, j'ai utilisé **Composer**, un outil permettant de gérer les bibliothèques (collections de code implémentant différentes fonctions) PHP. Composer a pour avantage de gérer les bibliothèques grâce à un simple fichier .JSON

```
{
   "require": {
      "silex/silex": "~1.2",
      "symfony/twig-bridge": "~2.6",
      "twig/twig": "~1.16"
   }
}
```

Figure 9 - Fichier JSON, permet l'import de Silex et de Twig

Une fois le *framework* installé, un dossier **vendor** est créé. Ce dossier regroupe l'ensemble des bibliothèques utilisées par l'application. Je crée un dossier **site** qui contiendra l'ensemble de l'application. Ce dossier, pour respecter l'architecture MVC, a l'arborescence suivante :

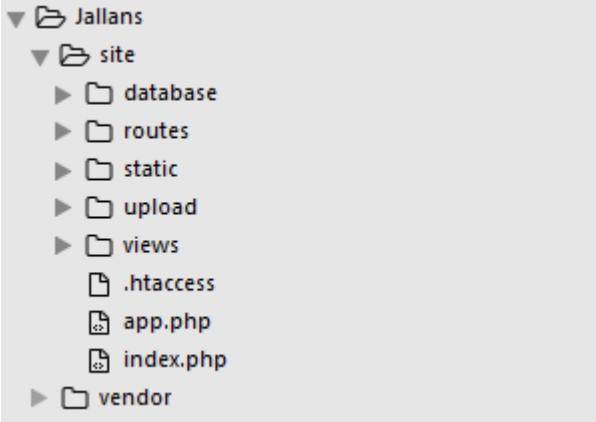


Figure 10 - Arborescence du projet

Le dossier **database** et le fichier **app.php** correspondent au Modèle ; le dossier **views** contenant les *templates* correspond à la Vue ; et le dossier **routes** et le fichier **index.php** correspondent au Contrôleur.

2. Le fonctionnement de Silex

Pour pouvoir utiliser Silex, il faut créer un objet *Application* qui est le contrôleur principal de l'application. Une fois créé, on peut lui ajouter les différentes bibliothèques précédemment ajouteées avec Composer.

```
//on importe les fichiers nécessaires au fonctionnement du framework
require_once __DIR__.'/../vendor/autoload.php';
//création du controlleur
$app = new Silex\Application();
```

Figure 11 - Code nécessaire pour utiliser Silex

Il faut ensuite créer les routes. Ces routes définissent l'ensemble des chemins accessibles depuis l'application et renvoient les informations associées.

```
//On créé une route qui accepte les requêtes GET
$app->get('/hello/{name}', function($name) use($app) {
    return 'Hello '.$name;
});
```

Figure 12 - Example de création d'une route

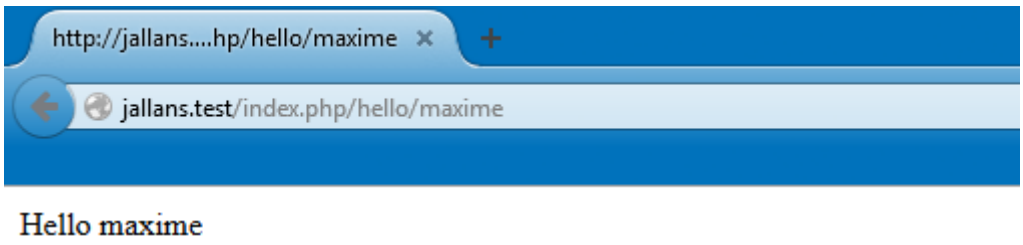


Figure 13 - Résultat obtenu en se rendant à l'url crée précédemment

Une fois les routes définies, la fonction run() permet de lancer le site.

Les bases étant posées, le développement peu maintenant commencer.

c) Création de la base de données et des objets d'interaction

Comme expliqué, une base de données est nécessaire. J'ai conçu cette base à l'aide de **phpMyAdmin** en suivant le MCD établit précédemment. Les attributs **Id** des différentes tables sont définis de manière à s'incrémenter automatiquement ce qui évite tout conflit de clé primaire (attribut unique des éléments d'une table).

Une fois toutes les tables créées, j'ai créé les objets PHP permettant d'accueillir les données extraites de la base de données (partie Model du principe MVC). Je crée respectivement une classe pour chaque table et obtient donc les classes:

- Utilisateur
- Article
- Menu
- Page
- Evènement

Vient ensuite la mise en place des Objets et méthodes permettant l'interaction avec la base de données. On appelle ces objets *DAO* pour *Data Access Object*, ils permettent d'uniformiser les communications. Les objets relatifs à chaque table étant relativement similaire, utilisons la classe articleDAO, gérant les interactions avec la table articles, comme exemple. L'objet articleDAO, prend en paramètre un objet de type *Connection* servant à se connecter à la base ; créé à l'aide de la bibliothèque *Doctrine*, une bibliothèque consacré à la gestion des connexions avec les bases de données. Ensuite, chaque méthode correspond à un appel à une requête bien précise.

```
public function getAll() {
    $sql = "SELECT * FROM articles order by article_id"; // requête sql
    $result = $this->db->fetchAll($sql); //on récupère les données bruts sélectionnés la requête
    $articles = array();
    foreach ($result as $row) {  //parcours du résultat pour pouvoir retourner une liste PHP
        $articleId = $row['article_id'];
        $articles[$articleId] = $this->Article($row);
    }
    return $articles;
}
```

Figure 14 – Exemple de fonction, la fonction getAll() de la classe ArticleDAO qui permet d'obtenir tous les articles présents dans la base de données

Ces appels, lorsqu'ils prennent une donnée rentrée par l'utilisateur utilisent des *Requêtes préparées* (ou *Prepared Statement*) pour effectuer les requêtes ce qui permet une exécution plus rapide et protège des injections SQL. C'est le cas de la fonction editArticle() qui sert à éditer un article.

```
public function editArticle($article){
    $req = $this->db->prepare('UPDATE articles SET titre=:t , contenu=:c , type=:ty, categorie=:ca, document=:d, dateA=:da WHERE article_id=:id'); // preparation de la requête.
        $titre = $article->getTitre();
        $contenu = $article->getContenu();
        $type = $article->getType();
        $categorie = $article->getCategorie();
        $document = $article->getDocument();
        $dateA = $article->getDate();
        $id = $article->getId();
        $req->bindParam(":t",$titre); // association des variable aux paramètres
        $req->bindParam(":c",$contenu);
        $req->bindParam(":ty",$type);
        $req->bindParam(":ca",$categorie);
        $req->bindParam(":d",$document);
        $req->bindParam(":da",$dateA);
        $req->bindParam(":id",$id);
        $req->execute();  // exécution de la requête
  }
```

Figure 15 - fonction editArticle(), permet l'édition des articles dans la base de données

En premier lieu la requête est préparée à recevoir les différents paramètres. Les paramètres sont ensuite liés à une variable. Enfin, la requête est exécutée. Cette méthode est sûre et explicite.

d) Implémentation des fonctionnalités

Comme expliqué précédemment, il a été établi que le site doit avoir certaines fonctionnalités. La plupart de celles-ci sont des fonctions de gestions du site, par conséquent, je crée une partie admin réservée à cet aspect.

1. La gestion du contenu

Le site étant le site d'une commune, il doit être possible d'y poster des articles d'actualité mais aussi de modifier les informations plus générales sur le site web (tel que l'histoire de la commune). C'est là que les tables articles et pages sont utiles. La table article regroupe l'ensemble des articles, billet d'actualité, tandis que la table pages regroupe les informations des pages « plus » statiques du site web. Ces deux types de contenu sont explicitement distingués.

Figure 16 - Menu de l'administration : les articles (appelés actualités) et les pages y sont séparés

Que ce soit pour l'ajout ou pour l'édition, des formulaires sont utilisés. Ces formulaires utilisent des requêtes POST, ce qui évite de transmettre les informations dans l'url.

```
<div class="Form">
    <form method="POST" action="{{ path('ajouterPage') }}"> //initialisation du formulaire, utilisation de la fonction path() de twig pour obtenir les liens pour valider le formulaire
        <div>
        <input class="textForm" type="text" name="titre" required placeholder="Titre">
        <textarea name="contenu" required class="ckeditor"></textarea>
        <h4>Menu </h4>
        <select class="textForm" name="type" size="1">
          {%for menu in menus%} // grâce à Twig, on peut boucler sur les éléments d'une variable.
                                // Ici, on parcourt les menus pour créer un menu déroulant
            <option>{{menu.nom}}</option>
          {%endfor%}
        </select>
        </div>
        <input type="submit" name="ajouter" value="ajouter" class="submit">
    </form>
  </div>
```

Figure 17 - Code du formulaire d'ajout de page utilisant Twig

J'ai ensuite créé les routes nécessaires au traitement des données. Pour cela j'ai fait des routes utilisant elles aussi des requêtes POST. Ces routes ont aussi pour particularité de prendre en paramètre un objet *Request* qui contient les informations transmises. Pour les récupérer, j'ai utilisé la méthode request->get() avec en paramètre une chaine de caractère correspondant au champ du formulaire dont je veux les informations.

```
$app->post('/admin/ajouterPage', function(Request $req) use ($app){
        $param = array('page_id' => 0,
                        'titre' => $req->request->get('titre'), //récupération des données du
                                                                champ titre du formulaire
                        'contenu' => $req->request->get('contenu'),
                        'type' => $req->request->get('type'));
        $page = $app['dao.pages']->Page($param);
        $app['dao.pages']->addPage($page);
        if($req->request->get('type') != 'Notes'){ $type='pages'; }
        else{ $type='Notes'; }
        return $app->redirect($app['url_generator']->generate('gestionPages', array('type' =>
$type))); //on redirige l'utilisateur vers l'url de gestion des pages.
        })->bind('ajouterPage');
```

Figure 18 - Route de validation du formulaire d'ajout de page.

Les routes sont des contrôleurs, elles font donc appel au Modèle pour traiter les données. Dans le cas présent, j'ai utilisé le DAO Pages déjà conçus pour ce traitement. Comme expliqué dans la partie précédente, la méthode addPage() utilise une requête préparés pour effectuer l'ajout à la base de données.

```
public function addPage($page){
    //on prépare la requête pour ensuite lui passer les valeurs en paramètre avant de l'exécuter
     $req = $this->db->prepare('INSERT INTO pages(titre, contenu, type) VALUES (?, ?, ?)');
       $rows = $req->execute(array($page->getTitre(), $page->getContenu(), $page->getType()));
    }
```

Figure 19 - Code de la fonction addPage() du Modèle.

Les articles, quant à eux, ont pour particularité de pouvoir leur associer un document, tel qu'un compte-rendu de réunion. Pour cela, j'ai précisé dans le formulaire la manière dont doivent être encodés les données grâce à l'attribut **enctype**. Ensuite, lors de la validation du formulaire, si un document a été ajouté, celui-ci est téléchargé dans le répertoire prévu à cet effet.

```
$file = $req->files->get('document');
        if($file != NULL){
                //on enlève les espaces du nom du fichier
                $document = str_replace(' ','_',$file->getClientOriginalName());
                //on télécharge le fichier avec la fonction move()
                $file->move(__DIR__.'/../upload/documents', $document);
        }else{
                $document = 'none';
        }
```

Figure 20 - Code d'enregistrement d'un fichier ajouté par l'utilisateur

L'application dispose aussi d'un troisième type de contenu, relativement différents, les événements. Les évènements servent, naturellement, à répertorier les différents évènements relatifs à la commune. Ils servent aussi à lister les réservations du foyer rural.

L'unicité des évènements est d'avoir une date de début et une date de fin. Ces dates devant être ajoutées à la base de données, il est nécessaire de les transformer au format correspondant. Les dates du formulaire sont au format Jours-Mois-Années, j'ai utilisé la méthode strtotime() pour les transformer en *timestamp* (nombre de secondes depuis le 1 janvier 1970 0h00). Une fois ce *timestamp* obtenu, je crée une nouvelle date au format Années-Mois-Jours, conforme à la base de données.

```
$debut => date("Y-m-d H:i:s ",strtotime($req->request->get(debut))),
```

Figure 21 - Transformation de la date de début d'un événement au bon format

Le contenu doit aussi être potentiellement modifié après coup. Pour cela, lors du chargement de la route menant à la page d'édition, le contrôleur fait charger les données déjà existantes. Ce qui permet de pré-remplir les champs du formulaire avec les informations existantes

```
$app->get('/admin/editionPage/{id}', function ($id) use ($app) {
        $menus = $app['menus'];
        $message = "Page à mettre à jour: ";
        //on récupère les données de la page sélectionné dans l'url via l'id.
        $page = $app['dao.pages']->getPageId($id);
        $titre = $page->titre;
        //génération du template avec les données
          return $app['twig']->render('admin/adminEditionPage.html.twig',
                                        array('titre' => $titre,
                                              'menus' => $menus,
                                              'message' => $message,
                                              'page' => $page));
})->bind('editionPage');
```

Figure 22 - Route menant vers la page d'édition de Page

Lors de la validation du formulaire, je fais cette fois-ci appel à la méthode d'édition de l'objet.

Enfin, le bouton « supprimer » des formulaires d'édition fait appel, au travers du contrôleur, à la méthode supprimer() du DAO respectif.

L'ensemble du contenu est donc gérable depuis l'application.

2. La gestion des utilisateurs

Il est possible d'administrer l'ensemble du site depuis l'application. Cependant, tout le monde ne doit pas y avoir accès, des comptes utilisateurs sont donc requis.

Deux types de comptes sont à différencier, le compte administrateur s'occupant de la gestion générale du site selon les droits qui lui sont attribués, et le compte association, réservé aux associations de la commune, pour leur permettre d'administrer les informations de leur page et d'ajouter les événements qu'ils organisent.

Lors de la création d'un compte administrateur, on peut définir séparément les droits dont disposera l'utilisateur grâce à des *checkbox*.

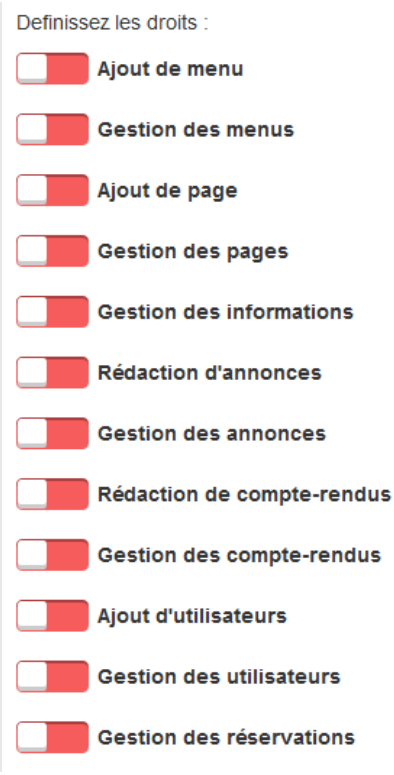


Figure 23 - Liste des checkbox du formulaire, dont l'aspect est modifié avec du CSS

À la validation du formulaire, une chaine de caractère est créée correspondant aux droits sélectionnés. Cependant les champs non cochés ne sont pas envoyés à la validation, il n'est pas possible de différencier les droits validés ou non. Pour parer à ce problème, je rajoute des champs cachés dans le formulaire.

```
//champ caché
<input id='switch1D'  type='hidden' value='unchecked' name='switch[]'>
//checkbox visible
<input type="checkbox" id="switch1" name="switch[]" class="switch"  />
```

Figure 24 - Champs input gérant le 1er droit

Une fois le formulaire validé, un script JQuery est exécuté et désactive les champs cachés inutiles (le champ visible étant coché).

```
function verifCheck(f){
//on parcourt toutes les checkbox cochées de la page et on désactive les champs cachés
   $("input.switch:checked").each (function () {
      $("#"+$(this).attr('id')+"D" ).prop('disabled', true);
   }); return true; }
```

Figure 25 - Script JQuery désactivant les champs cachés

Lors de la validation, le contrôleur parcourt les champs qui lui ont été transmis et crée la liste des droits.

Ces droits sont utilisés pour empêcher l'accès à certaines routes. La méthode acces() compare les droits de l'utilisateur aux droits requis et le redirige vers la page d'accueil de l'administration s'il ne les possède pas.

Pour le compte association, il n'est pas possible de définir les droits séparément, ils peuvent seulement gérer leur page et les évènements qu'ils pourraient organiser.

Ils ont tout de même comme particularité d'avoir une page qui leur est associé. Une liste des différentes associations de commune est donc présente à la création pour faire ce lien.

Figure 26 - Liste des association à la création d'un compte Association

Dans les 2 cas, les comptes nécessitent un mot de passe. Ce mot de passe sert, au moment de la connexion, à prouver l'identité de l'utilisateur. Il doit donc être protégé. J'utilise dans ce but le module **MessageDigestPasswordEncoder** du *framework* **Symfony 2.** Ce module permet de crypter les mots de passe de manière sûre.

Lors de l'enregistrement de mots de passe, je les crypte avec la méthode encodePassword(), qui utilise l'algorithme **SHA-512** ; à aucun moment les mots de passe sont stockés en clair dans la base de données.

```
$app['encoder']->encodePassword($req->request->get('password'), '')
```

Figure 27 - Code permettant de crypter le mot de passe rentré par l'administrateur

Au moment de l'identification, l'application crypte le mot de passe rentré par l'utilisateur et le compare à celui présent dans la base données grâce à la méthode isPasswordValid(), si les deux correspondent, l'utilisateur est connecté et ses informations stockées grâce aux variables **SESSION** de PHP.

```
//on récupère les informations de l'utilisateur qui souhaite se connecter avec son nom d'utilisateur
$userReq = $app['dao.utilisateurs']->getUtilisateurName($req->request->get('username'));
//Si, une fois chiffré, le mot de passe correspond à celui dans la BD, alors l'utilisateur est connecté
if( !is_null($userReq) and $app['encoder']->isPasswordValid($userReq->getPassword(),$req->request->get('password'), '')){
                $app['session']->set('user', $userReq->getUsername());
                $app['session']->set('roles', $userReq->getRoles());
                $app['session']->set('page', $userReq->getPage());
                $app['session']->set('log', true);
                return $app->redirect($app['url_generator']->generate('admin'));
                }
else{ return $app->redirect($app['url_generator']->generate('login')); }
```

Figure 28 - Code de la route de connexion à l'administration

Une fois son travail terminé, l'utilisateur peut se déconnecter. Cela a pour effet de détruire les variables de **SESSION**.

```
$app['session']->clear();
```

Figure 29 - Code exécuté à la déconnexion

3. Un site modulaire

Ma volonté était d'aboutir à un site qui puisse être au maximum gérable par l'utilisateur final, sans avoir à modifier le code. J'ai donc implémenté une gestion des menus. Les administrateurs peuvent donc ré-agencer les rubriques du site depuis l'interface.

Les menus se créent et s'éditent comme les pages ou les articles, a une différence près. Chaque page est lié à un menu (cf. MCD). Cela permet de regrouper les pages similaires dans les mêmes rubriques.

Par conséquent, lors de l'édition du nom d'un menu, il est nécessaire d'éditer aussi les pages. La fonction majPages() a été conçue à cet effet.

```
public function majPages($t1,$t2){
    $req = $this->db->prepare('UPDATE pages SET type=:t2 WHERE type=:t1');
    // met à jour les pages avec comme attribut type la valeur $t1
    $req->bindParam(":t1",$t1);
    $req->bindParam(":t2",$t2);
    $req->execute();
}
```

Figure 30 - Requête de mise à jour du type des pages

e) Réalisation de l'interface utilisateur

Le site web que j'ai développé est celui d'une commune, il se doit donc d'être simple d'utilisation et un minimum attractif visuellement. Cette volonté se traduit par différents aspects.

1. Une interface *responsive*

Aujourd'hui, la navigation sur internet se fait grâce à de nombreux appareils différents. Ordinateurs, Smartphones, Tablettes, tous ces appareils ont leurs spécificités. Il faut donc prévoir une interface accessible sur l'ensemble de ces supports.

J'ai donc utilisé **Bootstrap** car il permet de réaliser une interface *responsive*, c'est-à-dire qui s'adapte automatiquement en fonction du support. Sur un ordinateur classique, la page disposera de deux marges latérales tandis que sur appareil portable, les marges sont absentes et le contenu disposé en colonnes

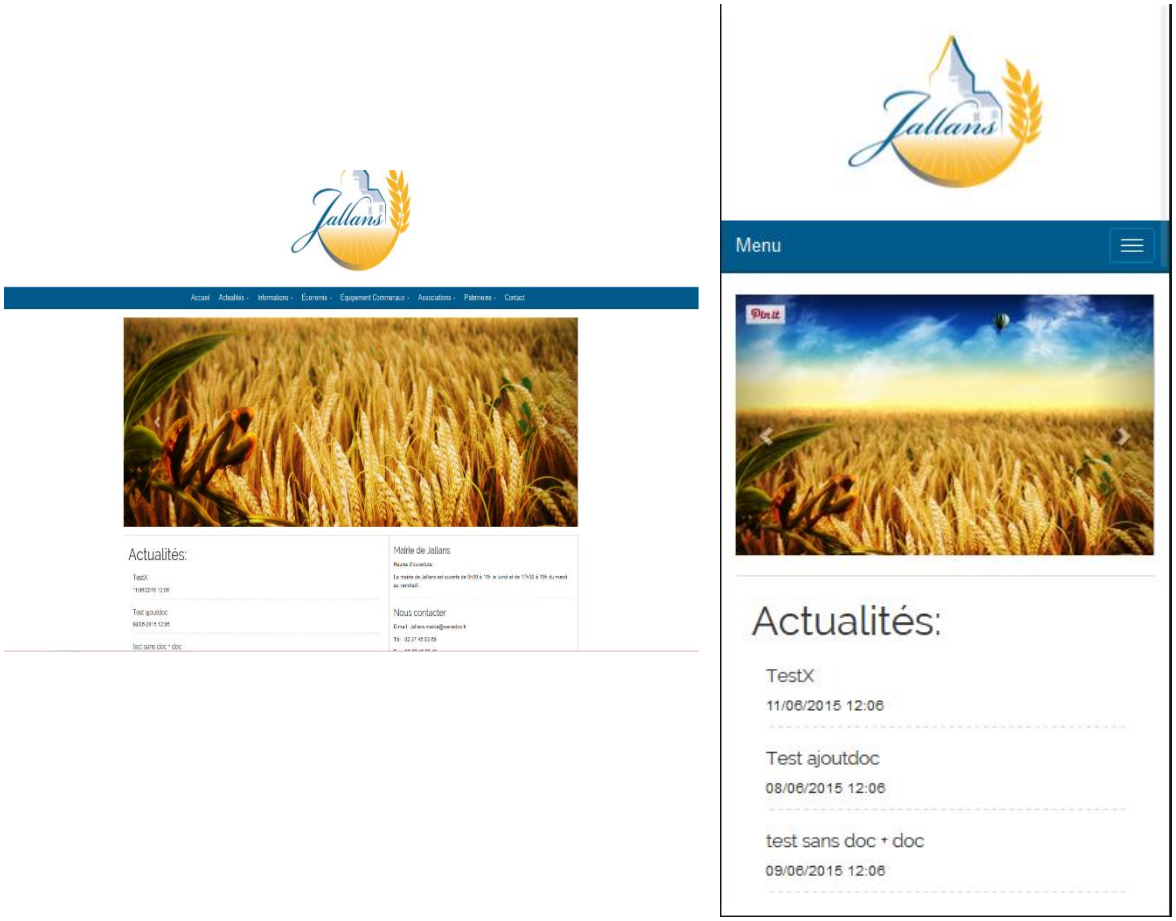


Figure 31 - à gauche l'affichage sur ordinateur, à droite l'affichage sur téléphone

De même le menu devient un menu déroulant sur mobile pour optimiser l'espace. De même certains éléments comme le logo voient leur taille réduite.

2. Une interface sobre

Pour faciliter la navigation et mettre en valeur l'information, le choix d'une interface sobre a été retenu. La charte graphique s'articule autour de trois couleurs. Le blanc, qui donne de la clarté et de la lisibilité au site. Le bleu qui se décline en deux versions, un bleu foncé et un bleu clair ; et enfin le jaune. Ces couleurs ont été choisies car ce sont celles de la commune.

Figure 32 - Exemple d'utilisation des 3 couleurs sur la page d'accueil

Toujours dans cette optique, les formulaires sont eux aussi réduis au plus simple. Le nom des champs est défini grâce au *placeholder* et j'ai laissé seulement la bordure du bas avec la règle de style *border*.

```
.textForm{
        outline: none;
        display: block;
        width: 100%; // force l'élément à remplir le conteneur en largeur
        padding: 7px; //applique une marge autour de l'élément
        border: none;
        border-bottom: 1px solid #ddd; //met une bordure d'1px d'épaisseur en bas de l'élément
        background: transparent;
        margin-bottom: 10px;
        font: 16px Arial, Helvetica, sans-serif; //modifie la police utilisée
        height: 45px;
}
```

Figure 33 - Code CSS appliqué aux champs des formulaires

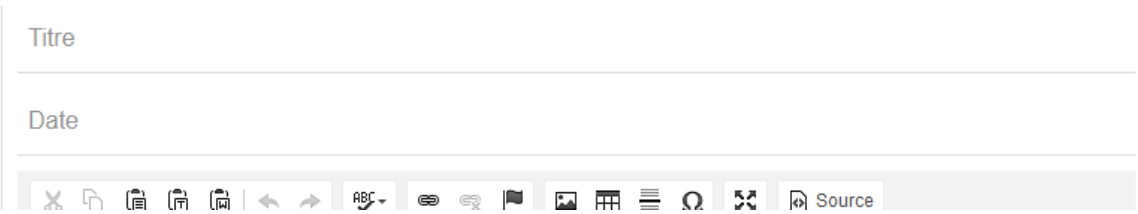


Figure 34 - Application du CSS sur les champs d'un formulaire

Les boutons sont eux aussi simplifiés dans leur aspect. Ils sont du même bleu clair que le menu pour un effort d'uniformité. La bordure basse, plus foncée, permet d'ajouter du relief à l'interface

Figure 35 - Boutons de validation des formulaires

3. Les modules complémentaires

Pour faciliter l'expérience utilisateur, j'ai utilisé plusieurs modules complémentaires. Le plugin CKeditor sert à la rédaction des articles et des pages. Ce plugin *open source* permet d'ajouter à un champ *textarea* des outils de traitement de texte. Il est donc plus aisé de mettre en forme le contenu rédigé.

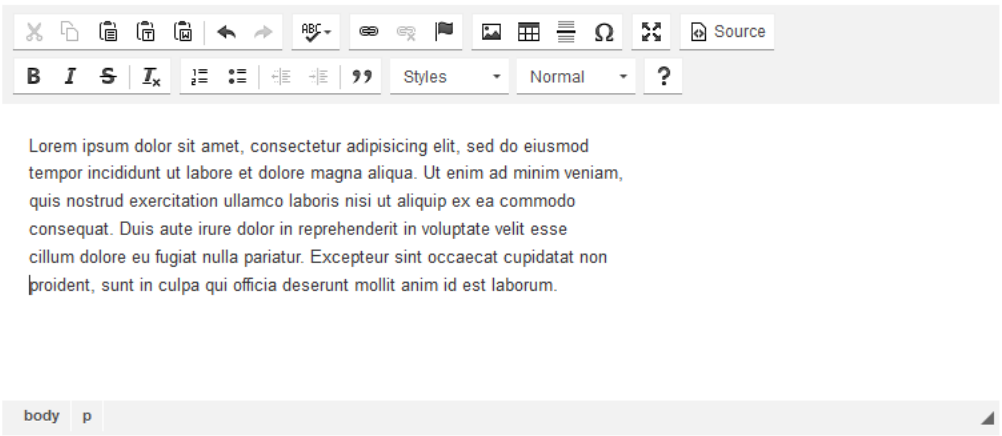


Figure 36 - Editeur de texte CKEditor

Le second plugin est utilisé pour simplifier la sélection de date dans les formulaires.

DateTime Picker , est un projet développé par le français Sébastien Malot, et qui permet d'obtenir un calendrier lorsque l'on clique sur un champs du formulaire. Entièrement paramétrable, ce sélectionneur de date a pour intérêt de proposer une sélection allant jusqu'aux secondes ce qui n'est pas le cas des autres.

```
<script type="text/javascript">
    $("#date").datetimepicker({ //On ajoute le datePicker au champs avec pour id « date »
        format: "dd-mm-yyyy hh:ii", //définition du format de la date
        language: "fr", //langue du calendrier
        todayHighlight: true,
        weekStart:1, //définition de différents paramètres de l'interface
        startView:2,
        minView:2,
        todayBtn:true, // ajout d'un bouton « aujourd'hui » au calendrier
        autoclose:true
    });
</script>
```

Figure 37 - Code JQuery pour ajouter une date picker à un formulaire

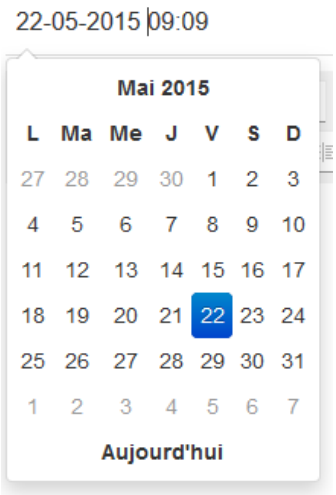


Figure 38 - DatePicker, permet de simplement sélectionner une date

Pour mettre en forme les événements de manière intuitive, j'utilise le plugin *FullCalendar*, qui est sous licence MIT (type de licence *open source*). Il propose un calendrier JQuery paramétrable et fonctionnant avec de simple objet javascript.

```
function renderCalendar(events) {
        $('#calendrier').fullCalendar({ //ajout du calendrier au <div> avec comme id "calendrier"
           header: {
              left: 'prev,next',
              center: 'title',
              right: 'month,agendaWeek'
           },
           lang: 'fr',
           buttonIcons: true, // afficher les boutons précédent/suivant
           weekNumbers: true,
           eventLimit: true, //limite le nombre d'évènements affichés par jour
           editable: false,
           events:events  //liste des événements à afficher dans le calendrier. }); }
```

Figure 39 - Code JQuery pour ajouter un calendrier à une page

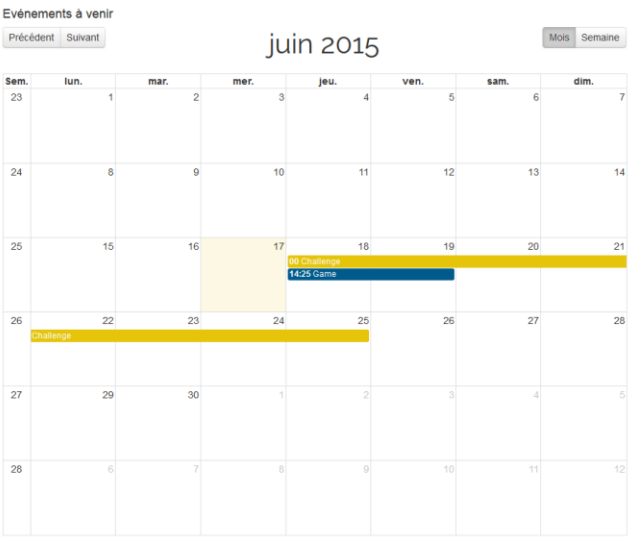


Figure 40 - Rendu du calendrier dans l'interface

Tous ces modules ont pour vocation des faciliter l'utilisation et la navigation dans l'application pour l'utilisateur.

VI. Organisation du travail

Ce stage, d'une durée de dix semaines, m'a demandé de mener dans sa globalité et à son terme un projet de développement. Pour réaliser au mieux ce projet, j'ai dû m'organiser au mieux et m'intégrer à mon espace de travail

a) L'organisation à la mairie de Jallans

À la mairie, je travaillais dans la salle de réunion, de 9H à 17h30 avec une pause de 2h le midi. Il m'est arrivé cependant de rester un peu plus tard le temps de finir ma tâche en cours.

De mon point de vu, je pense m'être bien intégré au sein du personnel. De plus, l'ensemble des gens étaient accessibles et toujours enclins à répondre à mes interrogations.

Un second écran a été mis à ma disposition pour un meilleur confort de travail.

b) Mon organisation personnelle

Pour planifier mon travail et connaitre mon avancement, j'ai utilisé l'outil de gestion **Trello**. Cette application, permet de définir et d'organiser les taches à réaliser. J'ai donc eu une méthode de travail proche la méthode *scrum*. C'est à dire que j'ai organisé mes taches par catégorie et par temps de réalisation. Une fois un ensemble de taches fini, je regardais que tout fonctionne puis passais à la suite.

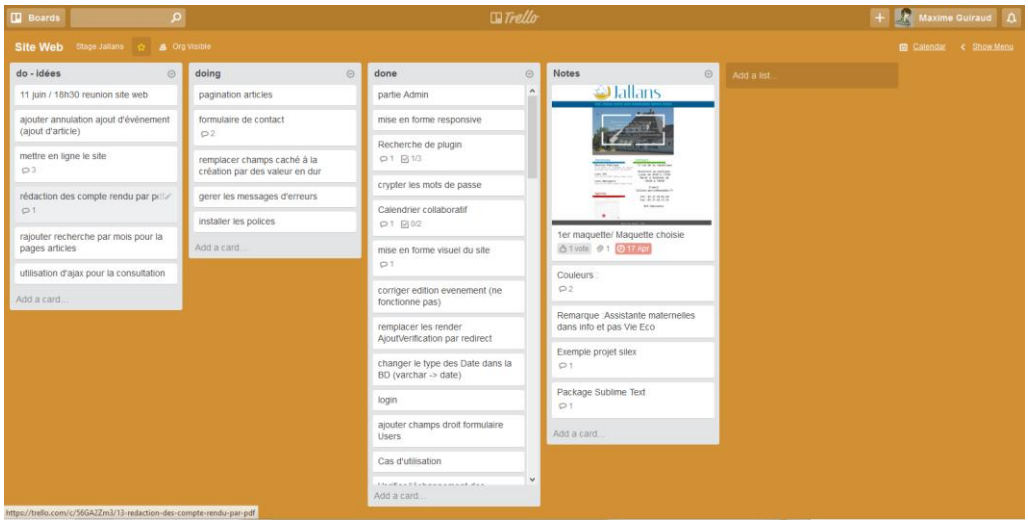
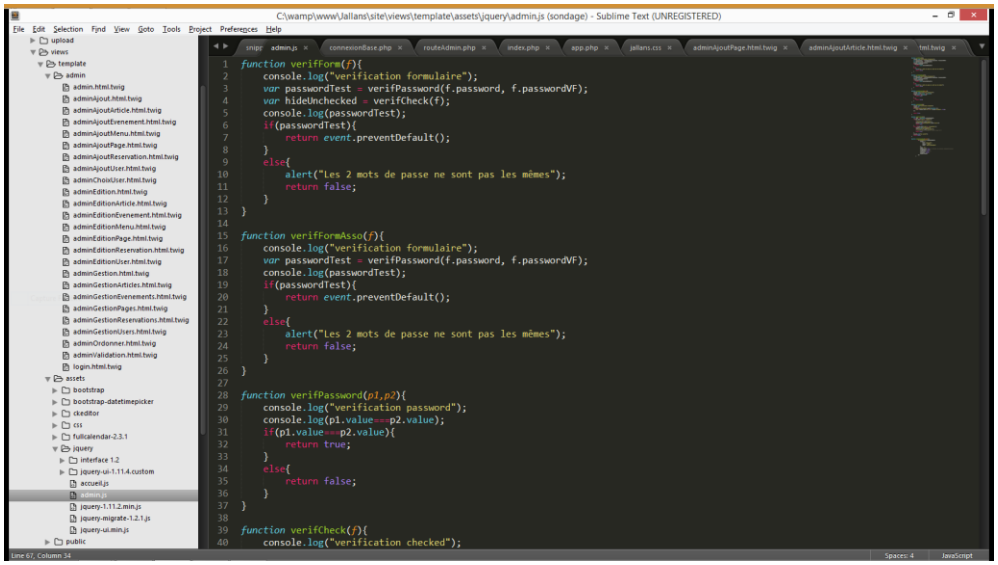


Figure 41 - Interface de l'application Trello, l'application utilise un système de cartes pour gérer les taches

Pour le développement, j'ai utilisé mon ordinateur personnel sous windows et l'éditeur de texte *Sublime Text*. Cet éditeur a comme intérêt d'offrir de nombreux raccourcis optimisant la rédaction et d'être compatible avec de nombreux langages différents, ce qui est utile pour le développement web où l'on utilise plusieurs langages en même temps.



```
function verifForm(f){
    console.log("verification formulaire");
    var passwordTest = verifPassword(f.password, f.passwordVF);
    var hideUnchecked = verifCheck(f);
    console.log(passwordTest);
    if(passwordTest){
        return event.preventDefault();
    }
    else{
        alert("Les 2 mots de passe ne sont pas les mêmes");
        return false;
    }
}

function verifFormAsso(f){
    console.log("verification formulaire");
    var passwordTest = verifPassword(f.password, f.passwordVF);
    console.log(passwordTest);
    if(passwordTest){
        return event.preventDefault();
    }
    else{
        alert("Les 2 mots de passe ne sont pas les mêmes");
        return false;
    }
}

function verifPassword(p1,p2){
    console.log("verification password");
    console.log(p1.value===p2.value);
    if(p1.value===p2.value){
        return true;
    }
    else{
        return false;
    }
}

function verifCheck(f){
    console.log("verification checked");
```

Figure 42 - Editeur de texte Sublime Text, utile notamment pour ses raccourcis

Pour pouvoir tester le code produit, il est nécessaire d'avoir un serveur. J'ai donc utilisé **Wamp**. C'est un logiciel permettant de générer un server local sur son ordinateur. Cela évite d'avoir à utiliser un vrai serveur et rend le développement plus rapide.

Enfin pour sauvegarder l'application et avoir accès aux versions du site à différent moment du développement, ce qui est utile en cas de bug, j'ai utilisé l'outil de versionnage Git.

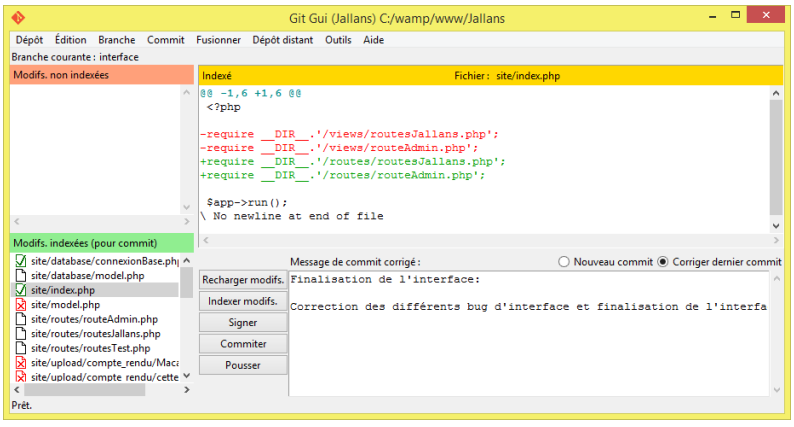


Figure 43 - Interface utilisateur pour le logiciel Git

VII. Conclusion

Ce stage a été une expérience bénéfique pour moi. Il m'a permis de mettre en application les connaissances acquises pendant mes deux années de formations à l'IUT ; mais aussi d'acquérir de nouvelles compétences.

D'un point de vue technique, je me suis principalement perfectionné en développement web. J'ai gagné de l'expérience en PHP, JQuery, HTML et en CSS.

Je me suis amélioré en base de données avec la conception d'une base de données et la réalisation de différentes requêtes.

Mais le plus important est l'utilisation de **Silex**. Grace à ce projet, j'ai pu découvrir un nouveau *framework*, enrichir mes connaissances et gagner en diversité dans les technologies que je maitrise.

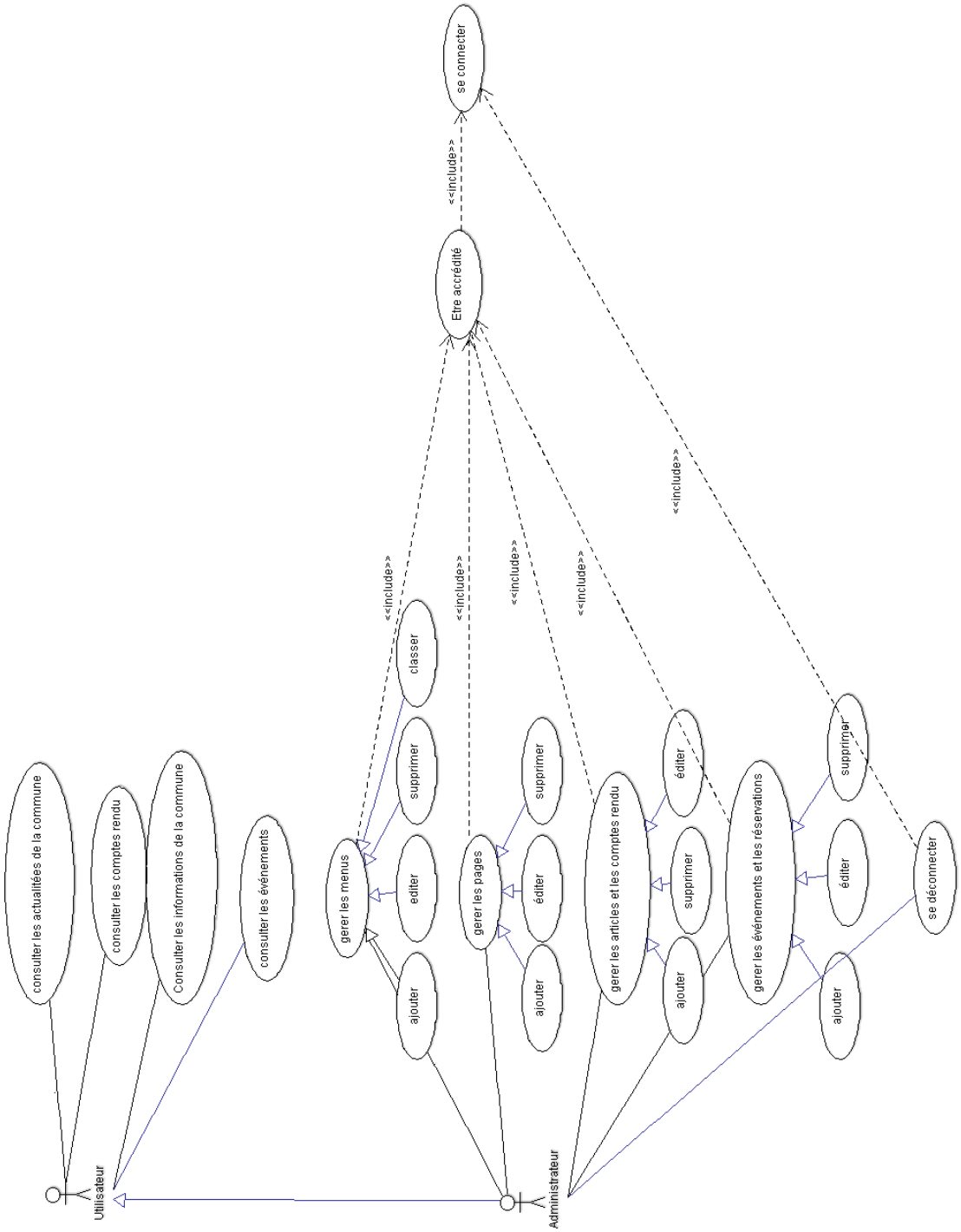
D'un point de vue humain ; ces dix semaines ont été pour moi une vraie mise en situation. C'est la première fois que je menais un projet d'une telle envergure, que ce soit par la taille que par la durée. Cela a été assez difficile de m'organiser et de diviser le travail en taches explicite. Malgré tous, ça m'a permis d'obtenir certaines méthodes de gestion et de l'expérience. Suite à ce stage, je sais maintenant faire face à certaines situations.

Dans l'ensemble, même si après coup certains détails du site web pourraient être améliorés, ce projet m'a permis de vivre une première expérience dans milieu professionnel et d'avoir une vision concrète du monde du travail ; notamment à travers les phases de recherche de stage.

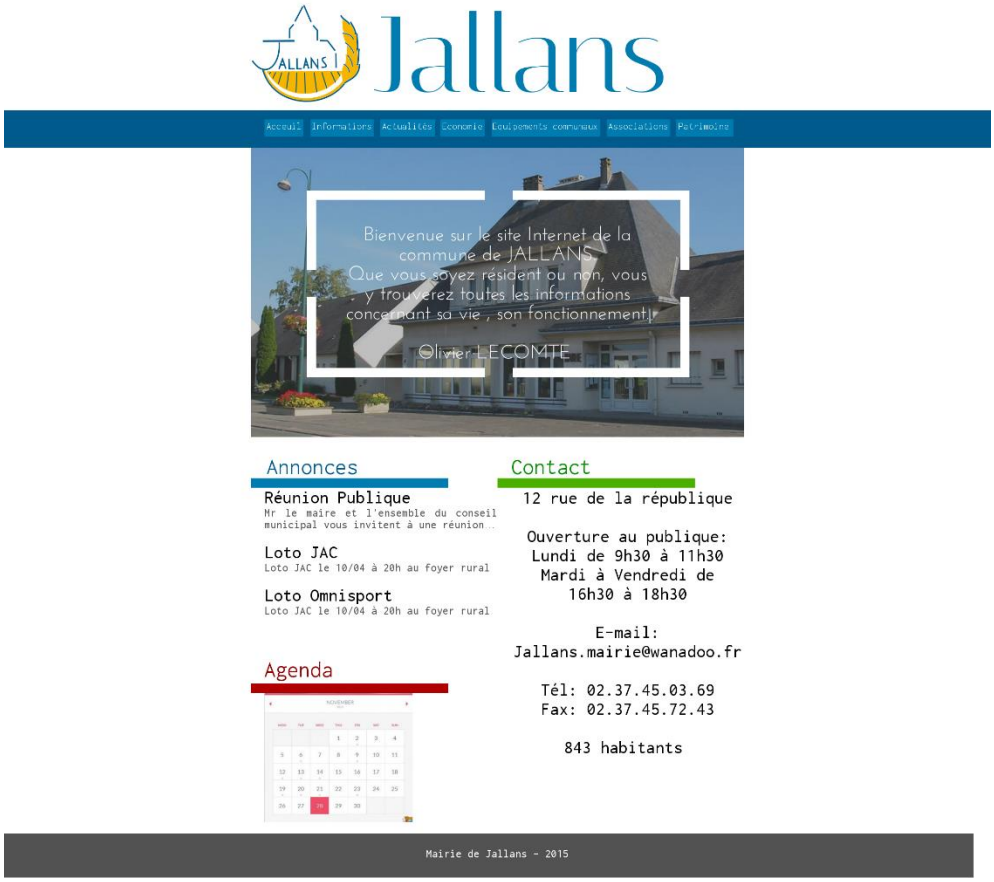
Enfin, après dix semaines de développement, la commune de Jallans dispose d'un site web en adéquation avec les nouvelles manières de naviguer sur internet. Celui-ci sera mis en ligne à la rentrée prochaine.

Annexes :

annexe 1 - Diagramme de Cas d'Utilisation



annexe 2 - Maquette 1 / accueil



annexe 3 - Maquette 2 / accueil