Emile Cadorel
Nicolas Gaurrin
César Récoché
Thomas Buchet
Guillaume Dos Santos
Yoan Garnier
Cyprien Degeorge
Guillaume Gas

Rapport de Projet

Création d'un générateur de Site Sportif

IUT D'Orléans – Département Informatique **Années 2013/2014**

Sommaire

Introduction

Etudiants à l'IUT d'Orléans en département informatique, un certain nombre de projets, à réaliser en équipe pour la plupart, nous ont été confié durant nos deux années d'étude. C'est ainsi que dans le cadre de ces études, on nous a donné un projet tutoré à réaliser.

Ce projet a consisté en la réalisation d'un CMS orienté sport dont le but était de permettre à une équipe de sport d'avoir son propre site web facilement.

Cette application devait ainsi permettre à l'équipe d'organiser divers évènements, de planifier des matchs et d'afficher les résultats d'une façon claire.

Durant une période imposée de développement, à savoir 3 mois, nous avons été libre de décider de l'organisation du déroulement du développement, ainsi que des outils et des langages utilisés. Nous nous sommes ainsi mis d'accord ensemble sur ces points avant de nous répartir les tâches.

Je vous propose de découvrir plus en détails ce projet à travers ce document. Nous verrons dans un premier temps les principales fonctionnalités de l'application en commençant par l'installeur et pour finir avec le backend. Nous verrons ensuite la partie technique de l'application, à savoir les outils que nous avons utilisés, puis de quelle manière nous les avons utilisés pour obtenir ce rendu. Nous verrons aussi l'organisation globale de l'application et les règles que nous nous sommes imposé pour un développement en équipe.

Enfin, vous pourrez découvrir de quelle façon avons-nous décidé de nous répartir le travail et pourquoi avant de conclure.

Vous trouverez en annexe certains documents liés aux spécifications techniques.

A. Présentation du projet

Cette application ayant pour but de permettre à des personnes sans connaissances particulières en informatique, nous avons décidé de créer trois principaux outils :

1.Insalleur

Le but de l'installeur est de permettre à l'utilisateur d'installer facilement son site sur son serveur. Ainsi l'installeur possède une interface simple et intuitive sous forme d'étapes à suivre, et la dernière de ces étapes mène à la création de la base de données nécessaire au CMS.

L'installeur est composé de trois étapes sous forme de formulaires :

- Formulaire de connexion à la base de données. Celui-ci permet au CMS d'avoir les informations nécessaires pour se connecter à la base de données de l'utilisateur afin de créer les différentes tables. L'utilisateur doit donc au préalable installer une base de données MySql.
- Formulaire de création du compte administrateur du CMS. L'utilisateur peut ainsi se connecter à la page administration avec les identifiants saisis.
- Formulaire demandant le nom du site.

Enfin, cet installeur permet à l'utilisateur de visualiser un récapitulatif de toutes les données saisies et permettre de revenir sur une étape pour faire des corrections tout en gardant les données déjà saisies affichées.
Une fois la page de vérification passée, un message de réussite ou d'échec s'affiche après quelques secondes, le temps à mysql de créer chaque table et de réaliser les insertions de base (telle que le compte administrateur).
En cas de réussite, deux liens apparaissent : celui permettant d'atteindre le backend, et l'autre le frontend.
En cas d'échec, des détails de l'erreur sont affichés et une nouvelle tentative est proposée.

2. Backend

La partie backend concerne toute l'interface permettant au créateur du CMS d'y apporter des modifications. Il est le seul ayant un droit d'accès et cette partie est donc protégée par une page de login.

L'utilisateur peut d'une part modifier l'apparence du CMS, la page de configuration lui laisse le choix entre plusieurs templates différents (c'est à dire des styles différents). Ceux-ci sont listés à partir du répertoire *templates,* ce qui veut dire que l'utilisateur peut très bien décider de créer son propre template en se basant simplement sur ceux déjà créés.

Outre l'aspect visuel du site, le backend permet à l'administrateur de gérer l'ensemble du contenu du site. Il a ainsi accès aux utilisateurs, aux news, aux évènements, aux équipes, aux matchs, aux tournois et enfin aux photos.
Pour chacune de ces fonctionnalités, l'administrateur peut effectuer des ajouts, des modifications et des suppressions. Ces nombreux formulaires lui évitent de modifier directement la base de données et lui permettent d'avoir une vision claire et globale du contenu de son site.

3. Frontend

Cette troisième partie concerne tout l'aspect visible par le public du site.
Les divers templates déjà présents à l'installation donnent accès aux fonctionnalités suivantes :

- Une page de news. Le visiteur peut a accès aux dernières news publiées, ainsi qu'au plus anciennes. Un petit formulaire en tête de page lui permet d'effectuer une rapide recherche. De plus un utilisateur connecté a la possibilité d'écrire un commentaire.
- Un calendrier qui regroupe tous les évènements de l'équipe. Tout comme pour les news, le visiteur connecté peut écrire un commentaire.
- Une page regroupant les équipes. Elle permet de retrouver les informations relatives aux équipes enregistrées ainsi qu'aux joueurs.
- Une page "Matchs" afin de voir les derniers matchs joués, ainsi que ceux prévus. Pour chaque matchs on retrouve comme informations les deux équipes, les points, et le lieu de la rencontre. Un export sous forme d'un fichier pdf est possible afin de pouvoir récupérer les résultats sous forme papier.
- Une page tournois afin de voir les différents tournois passés ou en cours. Une présentation sous la forme d'un arbre coloré permet une vision claire de l'enchainement des matchs et de l'évolution des équipes dans le tournoi.
- Une page de profil pour l'utilisateur connecté. C'est à partir de cette page que l'utilisateur a la possibilité de modifier ses informations personnelles.
- Et pour finir, des pages de connexion et d'inscription sont bien sûr présentes.

Nous avons réfléchi à la façon de mettre en évidences tous ces éléments afin que l'utilisateur ait un accès facile à toutes ces fonctionnalités. Il est en effet désagréable, pour un visiteur, de chercher pendant plusieurs minutes une fonctionnalité en particulier.
Bien sûr, tous ces éléments ne sont pas accessibles par tout le monde. Un système de droit doit permettre de donner des privilèges différents à certains utilisateurs. Enfin certaines fonctionnalités ne sont accessibles que pour les utilisateurs connectés (par exemple l'ajout de commentaire de news).

B. Outils utilisés

1. Gestionnaire de version collaboratif

Notre choix de gestionnaire de version collaboratif c'est tourné vers le logiciel git. En effet son utilisation est simple, et le site github propose des fonctionnalités intéressantes.

Tout d'abord son interface nous permet d'avoir une vision globale du projet, avec les derniers commits réalisés, la possibilité de visualiser rapidement le code source ou encore la participation de l'équipe et l'évolution du développement sous forme de graphiques. De plus github permet la création "d'issues", c'est à dire la liste des tâches à accomplir ou des bugs à corriger par exemple. Ces issues peuvent être regroupées dans des "Milestones", ce qui permet de les assigner à certaines parties de l'application. Enfin, ces milstones nous informent de l'avancement du développement au travers d'une barre accompagnée d'un pourcentage de progression.

Git étant disponible à la fois sous linux et sous windows, il nous a semblé l'outil idéal pour développer notre projet en équipe.

2. Base de données

Nous avons décidé d'utiliser MySql comme base de données. Tout d'abord parce qu'elle est gratuite, et puis pour sa simplicité. De nombreux tutoriels sont présents sur internet à son sujet et sa documentation est bien construite.

Enfin, l'utilisation de phpMyAdmin nous permet d'utiliser cette base de données encore plus simplement. C'est avec cette interface que nous pouvons effectuer des tests rapidement, par exemple des tests de contraintes entre les différentes tables. Elle permet de plus d'avoir une vue d'ensemble de la base de données et des enregistrements effectuées.

3. Langages utilisés

Les langages utilisés sont HTML5 et CSS3 pour l'affichage. Ceux-ci sont en effet très répandus, et nous permettent de répondre à nos attentes assez facilement. Ils s'utilisent assez bien avec les autres langages que nous utilisons.

Nous utilisons ensuite le javascript pour certaines parties du CMS, principalement pour une vérification plus poussée des formulaires du côté client. L'Ajax est utilisé de paire avec le javascript afin d'envoyer certaines requêtes au serveur sans avoir besoin de recharger la page. Par exemple pour le cas du champ de recherche de news avec autocompletion.

Enfin, nous utilisons le PHP comme langage de script côté serveur afin de dialoguer avec la base de données et de faire appel à nos templates à envoyer au client. Nous avons choisi ce langage car il est aussi simple à utiliser que puissant. La possibilité d'utiliser la programmation orienté objet avec ce langage nous a plu car c'est un bon moyen de garder un code lisible et bien structuré.

4. Moteur de template

Nous avons découvert le moteur de template Smarty durant nos recherches. Cet outil codé en PHP est d'une simplicité et d'une utilité remarquable. Il nous permet, une fois les informations extraites de la base de données, de les envoyer facilement dans le template voulu avant de l'envoyer au client. Il contribu ainsi à garder une bonne organisation dans notre code, et nous permet de garder une certaine séparation entre la partie développement PHP et le développement HTML/CSS.

Maintenant que vous avez pu prendre connaissance des outils que nous avons utilisé, nous allons vous montrer de quelle manière nous les avons utilisé dans les trois principales parties de l'application. Ainsi nous allons commencer par voir le fonctionnement de l'installeur, puis nous verrons ce que cache le backend, pour enfin voir comment le frontend permet d'avoir accès à toutes les fonctionnalités vues précédemment.

C. Description technique de l'application

1. Structure de la base de données

Nous avons essayé d'enregistrer un maximum d'informations tout en gardant la structure la plus simple possible. Ainsi, par exemple, chaque table possède un Id ce qui facilite l'accès aux différentes informations (ex : des fonctions php permettent l'accès aux données en fonction de l'Id passé en paramètre, idem pour l'insertion ou la modification de données).
Vous trouverez un diagramme UML de la base de données en annexe.

2. Structure de l'installeur

Afin d'avoir un installeur facile à utiliser, nous avons décidé de couper l'installation en plusieurs étapes. De plus, nous voulions que l'utilisateur puisse revenir aux étapes précédentes en cas d'erreur de saisie par exemple. Nous nous sommes donc penché sur le mécanisme des sessions afin de conserver les informations, d'une part, et d'autre part l'avancement de l'utilisateur afin de lui donner la possibilité, ou non d'atteindre une autre étape.

Vous pouvez ainsi voir dans le code suivant l'initialisation des variables de session permettant de gérer l'avancement de l'utilisateur dans l'installation :

install/index.php

```
<?php
session_start();
if((!isset($_SESSION['started']) || !$_SESSION['started'])  && !$DEBUG)
{
    unset($_SESSION['steps']);
    $_SESSION['steps']['step1'] = false;
    $_SESSION['steps']['step2'] = false;
    $_SESSION['steps']['step3'] = false;
```

```
    $_SESSION['steps']['step4'] = false;
    $_SESSION['bdd'] = NULL;
    $_SESSION['started'] = true;
} else if(!isset($_SESSION['steps']) && $DEBUG) {
    require_once("debug.php");
}
?>
```

Vient ensuite le déroulement des différentes étapes, le fichier index.php comporte un *switch* permettant d'exécuter telle ou telle étape :

install/index.php

```
<?php
if(isset($_POST['step']) || isset($_GET['step'])) {

    $step = $_REQUEST['step'];

    switch($step) {
        case "step_0":
            start_step_one();
            break;
        case "step_1":
            if(!$_SESSION['steps']['step1']) {
                recup_info("step_1");
                $_SESSION['steps']['step1'] = true;
            }
            start_step_two();
            break;
        /* suite... */
?>
```

Enfin, la principale fonctionnalité de l'installeur est la création de la base de données et les insertions de base dans celle-ci. Pour effectuer ces deux tâches, nous avons dans un premier temps créé deux fichiers sql comportant respectivement les requêtes de créations et les requêtes d'insertions.
Nous avons ensuite codé une fonction php permettant d'exécuter chaque requête en lisant le fichier (le délimiteur étant le point-virgule afin de sélectionner chaque requête correctement).

install/php/fonctions.php

```
<?php
function exec_sql_file($bdd, $file) {

    $f = file($file);
    $content = "";
    $res = false;

    if($f) {
        foreach($f as $l) {
            $content .= $l;
        }
        $test = explode(";", $content);
```

```
        $bdd->beginTransaction();
        //$bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $res = true;
        $i = 0;
        foreach($test as $req) {
            if(!empty($req) && $req != ";") {
                $r = $bdd->query($req);
                $i++;
                if(!$r) {
                            echo '<font color="red">[!] Erreur lors de
l\'éxecution de la requête n°'.$i.': '.$req.'</font><br>';
                    $res = false;
                }
            }
        }

        $bdd->commit();
    }
    return $res;
}
?>
```

3. Structure du backend

Toute l'organisation du backend tourne autour d'un fichier nommé *index.php*. C'est dans celui-ci qu'une première vérification est faite pour savoir si l'administrateur est connecté ou non. Dans tous les cas, c'est cette page qui inclu les éléments demandés par l'utilisateur.

Voici une partie du code en question, simplifié :

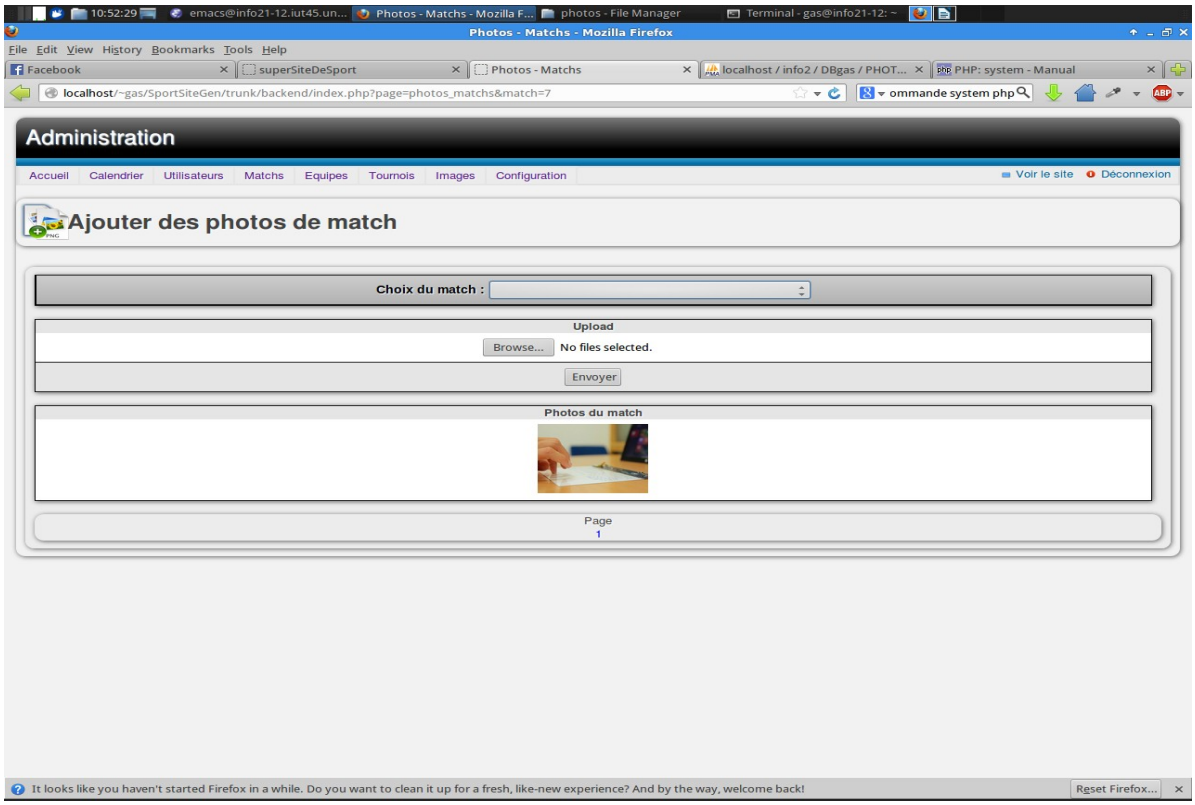
```
<?php
session_start();
require("../mysql_connect.php");
require("php/fonctions.php");
/* ... */

/* Si l'admin n'est pas connecté, on l'envoie sur la page de login */
if(!isset($_SESSION['admin_connected'])||!$_SESSION['admin_connected'])
{
    /*...*/
    $_SESSION['admin_connected'] = false;
    include("html/authen.html");
} else { /* Sinon on vérifie si il a demandé une page en particulier */
    if(isset($_GET['page']) || isset($_POST['page'])) {
        $page = $_REQUEST['page'];

        /* On gère chaque cas demandé par l'utilisateur */
        switch($page) {
            /*ARTICLE*/
            case 'article':
                include("php/article.php");
```

```
                break;
            case 'new_article':
                include("html/newarticle.html");
                break;
            case 'edit_article':
                include("html/editarticle.html");
                break;
           /* autres cas... */
           /* Cas où la page demandée n'est pas connue */
            default:
                include("html/accueil.html");
        }
    } else {
        include("html/accueil.html");
    }
}
?>
```

Comme vous pourrez le voir ici, ainsi que dans d'autres parties de l'application, nous nous sommes penchés sur les erreurs à ne pas commettre au niveau de la sécurité. Ainsi nous n'incluons pas directement la page passée dans l'url, mais nous analysons quelle page est demandée avant d'effectuer une action.

Le reste du backend a une structure assez répétitive. C'est une suite de formulaires qui, une fois que l'utilisateur a cliqué sur validé, envoient leurs données sur des scripts php. Ceux-ci vérifient alors la validité des données pour ensuite faire les modifications nécessaires dans la base de données.
Il n'y a que la partie permettant la gestion des images qui diffère un peu car nous avons dû nous renseigner sur l'upload de fichiers, la copie de fichiers et la création de dossiers. De plus, nous voulions que ces photos soient correctement rangées dans les dossiers afin qu'ils soient facilement accessible en aillant connaissance de l'Id d'un match, par exemple.
Nous avons donc décidé de créer trois dossiers, pour les photos de matchs, de tournois, et enfin toutes les autres. Lors de l'enregistrement des photos, si l'upload a fonctionné correctement, le script php enregistre le nom de la photo et le stocke avec le chemin vers le fichier dans la base de données.

Exemple avec la page d'upload de photos de match



4. Structure du frontend

La structure du code a été une étape délicate dans un premier temps car il fallait mettre en lien tous les éléments de l'application, tout en faisant transiter les données et paramètres correctement. C'est ainsi qu'après un premier essai, nous nous sommes dirigé vers l'orienté objet et le moteur de template smarty.

Comme pour le backend, la structure du frontend se base sur le fichier index.php, du moins en partie. Nous avons créé de nombreux objets permettant d'obtenir toutes les données nécessaires depuis la base de données. Par exemple la classe *News* nous permet d'accéder aux données, d'en modifier et d'en supprimer facilement. Celà nous permet d'avoir un code lisible et propre. De plus c'est une façon simple de travailler en équipe, une personne qui avait pour tâche de créer la classe News n'avait plus qu'à la "*push*" sur le dépôt. Les autres développeurs n'avaient plus qu'à lire les commentaires du code pour savoir quelle fonction utiliser.

Le fichier d'index créé ainsi les objets de base tel que celui nécessaire à l'initialisation du CMS (afin de récupérer le template à utiliser par exemple), ou encore celui permettant de récupérer le header du site.

Afin de récupérer ensuite les données de corps du site (par exemple les news, les évènements), c'est à dire la partie que l'on a appelé "*content*", nous avons créé une classe *Content*, qui, suivant la page demandée, va faire appel à la classe nécessaire pour renvoyer les données demandées.

Voici le code correspondant, tout d'abord vu depuis l'index :

frontend/index.php

```
<?php
// On récupère le corps du texte suivant ce qui a été demandé
$content = new Content($bdd, $template, $smarty);
if(isset($_GET['page']) || isset($_POST['page'])) {
    $page = $_REQUEST['page'];
     $param = get_params($_GET, $_POST); /* On récupère tous les params
sauf la page */
    $content_html = $content->get_html($page, $param);

    if ($page == "deconnexion"){
      unset($_SESSION);
      session_destroy();
      header("Location:  index.php");
    }
} else {
    $content_html = $content->get_html("accueil");
}
?>
```

La fonction *get_params* sert ici à récupérer dans un tableau toutes les données passées dans $_GET ou $_POST. Ce tableau est ensuite passé à l'objet *content*. Cela permet au développeur d'avoir accès aux variables passées dans l'url ou par les envoies de formulaire n'importe ou dans la suite du code.

Voici maintenant une partie simplifiée de la classe Content :

frontend/php/Content.class.php

```
class Content {

    private $_bdd; /* base de donnees rattache */
    private $_template; /* le nom du dossier de template courant */
    private $_smarty; /* systeme de parsing */

    /*
      \brief construit l'objet
      \param bdd la base de donne
      \param template le nom du template a parse
      \param smarty le systeme de parsing
    */
    public function __construct($bdd, $template, $smarty) {
        $this->_bdd      = $bdd;
        $this->_template = $template;
        $this->_smarty   = $smarty;
    }

    /*
      \brief genere le code html de la page news
      \param param les parametre optionnel de chargement
      \return page parser avec smarty
```

```
    */
    public function get_html_news($param) {
        $news_obj = new News($this->_bdd, $param);
        $news     = $news_obj->get_content();
        $com      = $news_obj->get_content_com();
        $simple = array();
             $simple['one'] = isset($param['v1']) && $param['v1'] ==
"lire_news";
        if( $simple['one'] ) {
            $simple['connected'] = isset($_SESSION['connected']);
            $simple['Id'] = $news[0]['Id'];
        }
        $this->_smarty->assign("News", $news);
        $this->_smarty->assign("Com", $com);
        $this->_smarty->assign("NSimple", $simple);
               return               $this->_smarty->fetch("templates/".
$this->_template."/html/news.html");
    }
```

Comme vous pouvez le voir, l'objet News nous permet de récupérer les données nécessaires pour afficher les news, puis il nous reste à donner ces données au moteur de template smarty afin d'obtenir le code html voulu en fonction du template enregistré au préalable.

Voici maintenant un exemple d'utilisation de smarty, avec ici encore, les news :

frontend/templates/debug/news.html

```
<table align="center">
    <tr>
        <th>NEWS</th>
    </tr>
    {if $NSimple != 1}
    <tr>
        <th>
         <form method="post" action="index.php?
page=news&v1=research_news">
           <input type="text" autocomplete="off" name="v2"
id="champ_recherche" />
           <input type="text" style="display:none"/>
           <input type="submit" value="Rechercher" />
        </form>
       <div id="output" class="hide result"></div>
       <script type="text/javascript" src="js/oXHR.js"></script>
       <script type="text/javascript"
src="js/news_research.js"></script>
       </th>
    </tr>
    {/if}
    {foreach from=$News item=news}
    <tr>
       <td><a href="index.php?
page=news&v1=lire_news&v2={$news.Id}"><u>{$news.titre}</u></a> -
<i>{$news.date}</i></td>
```

```
    </tr>
    <tr>
        <td>{$news.contenu}</td>
    </tr>
    {/foreach}
</table>
```

Comme vous pouvez le constater, smarty nous permet de faire des conditions. Ici le but est d'afficher la barre de recherche uniquement quand on affiche toutes les news et non une en particulier.
Smarty nous permet aussi de faire des boucle à l'aide du mot-clé *foreach* ce qui est très pratique. Il nous suffi de passer un tableau au moteur de template afin d'afficher simplement toutes les news.

La barre de recherche, quant à elle, utilise la technologie *Ajax*. Nous voulions un système rapide, et aillant déjà vu ce genre de système de recherche nous voulions apprendre son fonctionnement. Nous avons donc découvert l'Ajax, qui nous permet de dialoguer avec la base de données sans avoir à rafraichir la page toute entière. C'était une bonne occasion d'approfondir nos connaissances en javascript et découvrir l'ajax.
Comme vous pouvez le voir dans le code ci-dessus, le fichier *news_research.js* est inclu. C'est lui qui surveille les changements effectués sur la barre de recherche, et envoie les requêtes à la base de données. C'est alors un fichier php qui effectue la recherche dans la base de données, pour ensuite renvoyer les résultats de la manière suivante :
titre1|titre2|titre3...

Le script javascript prend alors le relais une fois la réponse reçue, et créé un tableau en utilisant le caractère '|' comme délimiteur.
Les éléments trouvés sont alors affichés sous le champ de recherche.

Voici une partie du code javascript/ajax permettant la recherche :

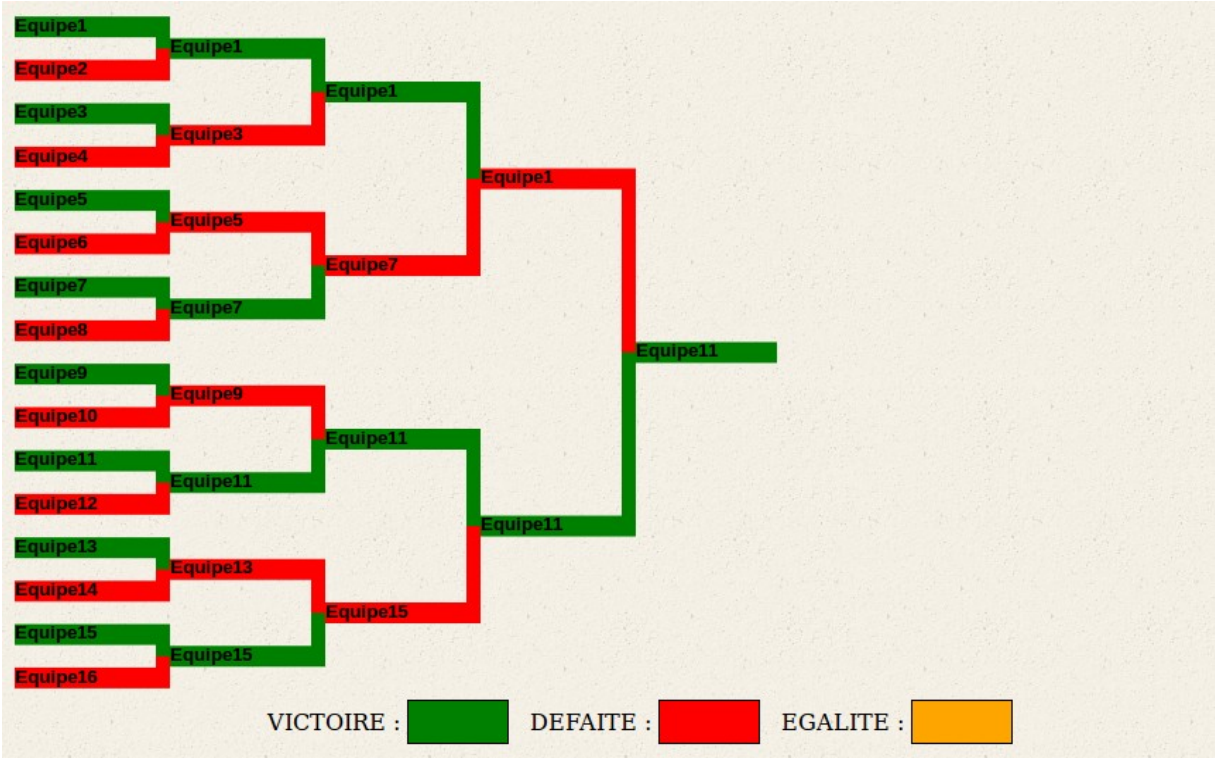
Frontend/js/news_research.js

```
function request(val) {
   xhr.open("GET", "php/research_ajax.php?val="+encodeURIComponent(val), true);
   /* Status de l'objet, 4 le serveur a fini son travail, 200 et 0 car pas d'erreur (ex 404...) */
   xhr.onreadystatechange = function() {
      if(xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
          traitement(xhr.responseText); //données textuelles récupérées (responseXML -> arbre XML)
       }
   };

   xhr.send(null);
   return xhr;
}
```

```
    function traitement(sData) {
        if(sData) {
            results.innerHTML = "";
            var array = sData.split('|');
            var div, cn;
            var output = document.getElementById("output");

            for(var i = 0; i < array.length; i++) {
                div = document.createElement("div");
                div.className = "unselected_result result";
                cn = document.createTextNode(array[i]);

                div.appendChild(cn);
                div.onClick = function() {
                    chooseResult(this);
                };
                output.appendChild(div);
            }

            output.className="af";
        }
    }
```

Comme vous pouvez le constater, la fonction *request* s'occupe d'envoyer une requête au serveur. Les résultats sont alors envoyés dans la fonction traitement qui s'occupe de découper la chaîne récupérée et de créer les tags html nécessaires pour l'affichage des résultats.

La réalisation de la page des tournois fût une autre partie intéressante de ce projet avec un affichage sous la forme d'un arbre.
Voici un exemple d'un tel arbre :



Pour ce cas nous avons une fois de plus utilisé le javascript pour sa réalisation. La fonction d'affichage de l'arbre en javascript utilise le systeme de caneva. Elle prend un tableau de match identifié comme appartenant à un tournoi. Cette identification se fait en php dans la classe Tournoi. Les matchs d'un tournoi disposent d'un numero de tour ainsi on peut connaitre l'emplacement de chaque match. Le javascript affiche ainsi les matchs les un après les autres sous forme de barre coloré (la couleur indiquant la victoire d'une équipe).Un arbre mal forme (log2(nbMatch) = un nombre flottant) ne s'affichera pas. Si une arbre est non finis parceque tout les matchs n'ont pas été entrée par l'utilisateur mais qu'il respecte le log2, celui ci s'affichera partiellement pour n'afficher que les matchs entrés dans la partie administration.

Enfin, le dernier élément qui nous a particulièrement intéressé fût l'export des fiches en matchs vers des fichiers pdf. Pour celà, nous avons trouvé une bibliothèque php : html2pdf. Il nous suffi de récupérer les données grâce aux classes créées à cet effet, puis nous transmettons ces données à smarty avec la page template déstinée à affiche les résultats. Enfin il nous reste plus qu'à récupérer le code html renvoyé par smarty pour le donner à un objet *html2pdf* qui nous créé le fichier pdf correspondant.

Voici un exemple simplifié de l'utilisation de smarty et de html2pdf :

Frontend/php/export_pdf.php

```
$req = $bdd->prepare("select * from MATCHS where Id = :id");
$req->execute(array(":id" => $_GET['id']));
     if($req->rowCount() !=0) {
         $data = $req->fetch();

          $smarty = new Smarty();
          $smarty->assign("match", $data);
          $content = $smarty->fetch("../templates/fiche_match.html");

         $html2pdf = new HTML2PDF('P', 'A4', 'fr');
         $html2pdf->WriteHTML($content);
         $html2pdf->Output('test_'.$data['DateMATCHS'].'.pdf');
```

D. Conventions de codage

Nous nous sommes mis d'accord sur un certain nombre de convention decodage afin d'avoir un code propre. Voici ces différentes conventions :

1. Conventions concernant le PHP :

L'indentation doit permettre une bonne lisibilité du code : une tabulation soit 4 espaces.

Pas de retour à la ligne avant un ' {' et '}' doivent rester seuls sur une

ligne.

Exemple :

```
if(test) {
    echo 'test';
}
```

Les conditions qui n'engendrent qu'une seule ligne de code doivent tout de même avoir des '{}'.

Concernant les classes, que des caractères alphanumériques (pas de '_'). Elles doivent de plus commencer par une majuscule, tout comme le début de chaque mot composant le nom de la classe (à la manière de Java).

Les fonctions peuvent avoir un '_' entre les mots qui composent leur nom.

Les variables possèdent les même normes que le nom des classes, exception faite des variables de classe qui commencent par un 'm_' (abréviation de my).

Les constantes commencent par une '_' et un '_' sépare chacun des mots de cette constante. Elles doivent être entièrement en majuscule.

Ex : _URL_SITE

2. Conventions concernant le HTML et le CSS :

Le html soit doit avec des balises en minuscules et une indentation correcte : une tabulation (4 espaces).

Le code css à l'intérieur d'un fichier html est à proscrire.

Le code css devra respecter les mêmes normes d'appellation que le php. Quant aux accolades, les deux doivent se trouver seules sur une ligne.

3. Conventions concernant le SQL :

Le nom des tables est à noté en majuscule.

Le nom de chaque champs est à noté avec une majuscule en début de mot, et à chaque début des autres mots.

E. Répartition du travail

Lors de notre première réunion, nous avons identifié les différentes parties qui composeraient le projet et nous nous sommes réparti le travail de la façon suivante.

1. Installeur

Cyprien s'est occupé de la partie base de données. Il a ainsi utilisé la méthode Merise afin de déterminer quelles seraient les tables nécessaires.

Nicolas s'est occupé de la partie HTML/CSS des différentes étapes.

Guillaume G. s'est occupé de la partie PHP de l'installeur.

2. Frontend

Développement de la partie template (HTML/CSS) : Guillaume D., César, Nicolas.
Développement de la partie PHP/Javascript/Ajax : Emile, Guillaume G.

3. Backend

Développement de la partie HTML/CSS : Thomas, Nicolas.
Développement de la partie PHP : Guillaume G, Emile, Thomas.

F. Améliorations possibles

Nous avons pensé à quelques améliorations à apporter au projet afin de le rendre plus complet. En voici les principales :

1. Supporter d'autres bases de données

Nous pourrions proposer à l'utilisateur lors de l'installation d'utiliser Oracle par exemple (ou autre). Cela supposerait de créer une système de connexion à la base de donnée particulier qui permettrait la connexion à différents types de base de données.

2. Afficher des statistiques

Il serait intéressant d'afficher des statistiques concernant les matchs d'une ou plusieurs équipes durant une saison, avec par exemple un graphique montrant leur évolution.

2. Prise en chaque plus complète et précise des sports

Pour le moment l'application permet de gérer une équipe de sport mais aucune information précise n'est enregistrée concernant un sport particulier. Pour pourrions proposer différents sports pris en charge, et faire une base de données plus étoffée à ce sujet (avec des détails concernant les points, les pénalités par exemple..).

Conclusion

En conclusion, nous pouvons dire que ce projet nous a permis d'apprendre de nombreuses choses dans différents domaines.

Nous avons dû d'abord apprendre à effectuer un travail de préparation efficace étant donné la taille du projet à réaliser. Nous devions donc anticiper les diverses tâches que nous aurions à accomplir afin de nous les répartir correctement.

De plus, durant le développement de ces diverses parties nous avons été confronté à de nombreux bugs que nous n'avions pas forcément déjà rencontré auparavent, ce qui nous a fait apprendre beaucoup de choses sur les différents langages que nous avons eu à utiliser. C'est d'ailleurs dans un tel contexte qu'on se rend compte de l'utilité et du fonctionnement de ces différents outils, que ce soit pour le développement ou le travail d'équipe.

Enfin, ce projet nous a donné une meilleure idée du travail que nous aurons à fournir en entreprise, sur de gros projet, de la complexité à laquelle nous serons sans doute confronté, et au futur travail d'équipe auquel nous aurons affaire.

Annexes

MLD de la base de données (voir page 7):