



Dossier de projet professionnel

Titre professionnel développeur web
et web mobile (DWWM)

réalisé par
Sofiane MSATFA

2021-2022

Table des matières

1	Introduction	3
1.1	Cahier des charges	4
1.1.1	Objectifs	4
1.1.2	Contraintes techniques	5
2	Mise en place du projet	6
3	Récupération des données	8
3.1	Configuration	9
3.2	Récupération via scraping	12
3.2.1	Connexion	12
3.2.2	Analyse	14
3.2.3	Insertion	17
3.3	Récupération via fichiers Excel	20
3.3.1	Insertion	21
3.4	Distances	22
3.4.1	Coordonnées géographiques	22
3.4.2	Matrice des distances	25
3.5	Habilitations	26
3.6	Mise à jour	30
4	Algorithme	31
4.1	Principe général	31
4.2	Etape 1 : regroupement des matchs par date	33
4.3	Etape 2 : récupération des arbitres potentiels	34
4.4	Etape 3 : processus de sélection	35
4.4.1	Matchs uniques	35
4.4.2	Matchs tournois	36
4.5	Validation	37
4.6	Rapport	38
5	Interface	39
5.1	Liste des matchs	41
5.2	Désignations	43
5.2.1	Récupération des arbitres	44
5.2.2	Désignations automatiques	46
5.3	Sécurité	47
6	Documentation	48
7	Conclusion	49

1 Introduction

Dans le cadre de ma formation développeur web et web mobile, j'ai eu la chance d'assister Mr. Vincent VAURETTE qui est en charge de la cellule de transition numérique de la Fédération Française de Volley-Ball (FFVB).

Le but de son travail est de développer des outils qui facilitent la gestion de tout ce qui est en lien avec le monde du volley-ball, et de redonner un coup de jeune à l'intranet de la fédération.

Durant trois mois, j'ai eu la responsabilité de la transition numérique de l'interface de désignation des arbitres.



La Fédération Française de Volley-Ball est une association créée en 1936 qui constitue l'instance dirigeante du volley-ball en France. Plusieurs centaines de matchs se déroulent par week-end, et la désignation des arbitres les concernant est entrêmement chronophage.

Le site actuel de la fédération date de 2009 et manque d'ergonomie. Celui-ci demande plusieurs click pour effectuer une seule désignation et ne permet pas d'afficher toutes les informations nécessaires pour celles-ci. Les gestionnaires sont alors contraints de passer par un tableur externe pour procéder aux désignations, puis ensuite de rentrer manuellement celles-ci pour chaque match.

L'objectif principal qui m'a été confié a été la création d'un outil permettant d'automatiser ces désignations. Celui-ci devait pouvoir associer un ou plusieurs arbitres habilités et disponibles à chaque match d'une période donnée, tout en laissant le choix de leur modification.

1.1 Cahier des charges

1.1.1 Objectifs

Quels sont les objectifs demandés ?

- L’objectif principal est l’automatisation des désignations des arbitres pour une période donnée. Il faut que cet algorithme puisse proposer un arbitre habilité et disponible pour chaque match de la période voulue.
- Un autre objectif essentiel est la mise en place d’une matrice des distances entre chaque arbitre et chaque club, qui pourra servir à faciliter la désignation des arbitres pour les gestionnaires.
- Enfin, il faut également améliorer l’interface graphique et ajouter quelques fonctionnalités.

Quelles sont les fonctionnalités à ajouter à l’interface ?

- Il faut que l’interface affiche tous les matchs à venir de la saison sous forme de tableau, qu’on pourra également trier par poule si voulu. Pour le confort des utilisateurs, Le tri par poule doit se faire sous forme de SELECT avec la possibilité de rechercher une valeur dans celui-ci.
- Le tri des matchs par colonne doit être possible et une pagination doit être mise en place.
- (Optionnel) Une barre de recherche doit également être disponible pour trouver une donnée précise dans ce tableau.
- Il faut pouvoir désigner les arbitres directement depuis cette interface, en affichant un SELECT avec une liste de tous les arbitres disponibles. Cette liste d’arbitres doit afficher plusieurs informations essentielles les concernant : le numéro de licence, le niveau, le grade, et la distance les séparant du gymnase du match. Les arbitres doivent également être triés par état de disponibilité pour le match sélectionné, avec un code couleur différent pour chacun.
- La désignation doit se faire de façon automatique au changement du SELECT pour éviter de perdre les sélections en cas d’oubli.
- L’interface doit également permettre de lancer l’algorithme d’automatisation des désignations en permettant de choisir une période voulue et de lancer les désignations via un bouton.

Précisions concernant l'automatisation des désignations ?

- Il faut pouvoir distinguer les matchs pour lesquelles deux arbitres sont nécessaires, et proposer le bon nombre d'arbitres à chaque fois.
- Il ne faut proposer que des arbitres qui sont habilités à arbitrer, et disponibles le jour du match : c'est à dire qui n'ont pas précisé d'indisponibilité, et qui ne sont pas déjà désignés sur un autre match.
- Prendre en compte les exceptions : les arbitres ne peuvent pas arbitrer un match joué par leur club, et les matchs du club à qui ils ont donné leurs points d'arbitrage.
- La distance séparant les arbitres du gymnase doit aussi être prise en compte, mais être flexible pour éviter les désignations répétitives des mêmes arbitres pour les mêmes gymnases.
- L'algorithme doit proposer des arbitres pour chaque match de la période, mais également permettre le changement manuel de ceux-ci directement sur l'interface. De plus, la validation des désignations automatiques, contrairement aux désignations manuelles, devra se faire grâce à un bouton de confirmation.
- Certains matchs sont de type 'tournois' et se déroulent donc au même endroit : l'algorithme doit proposer le même arbitre pour chaque paquet de trois matchs de la même poule du même tournoi.

1.1.2 Contraintes techniques

Les technologies utilisées sont :

- Bootstrap 4.6 (une autre version peut cependant être utilisée)
- Javascript / JQuery 3.6
- PHP 8 (aucun framework)
- MySQL 8.0

2 Mise en place du projet

Afin de répondre à ces objectifs, il m’a fallu mettre en place un plan d’action en prenant compte les technologies utilisées pour ce projet.

Mon travail devait pouvoir être maintenu facilement par mon tuteur, et pour ce faire la documentation et l’organisation des fichiers étaient primordiales.

Une documentation complète a été réalisée en fin de projet pour apporter une précision supplémentaire aux commentaires écrits avec le standard d’écriture PHP Docs, permettant ainsi de faciliter la compréhension et la reprise du projet par mon tuteur.

J’ai décidé de travailler en Programmation Orientée Objet (POO) pour faciliter la réutilisation des composants.

J’ai donc pensé à organiser mes fichiers par entité qui compose ce projet : chaque classe concernant une entité se trouve dans un dossier différent des autres et permet ainsi de facilement s’y retrouver dans les fichiers.

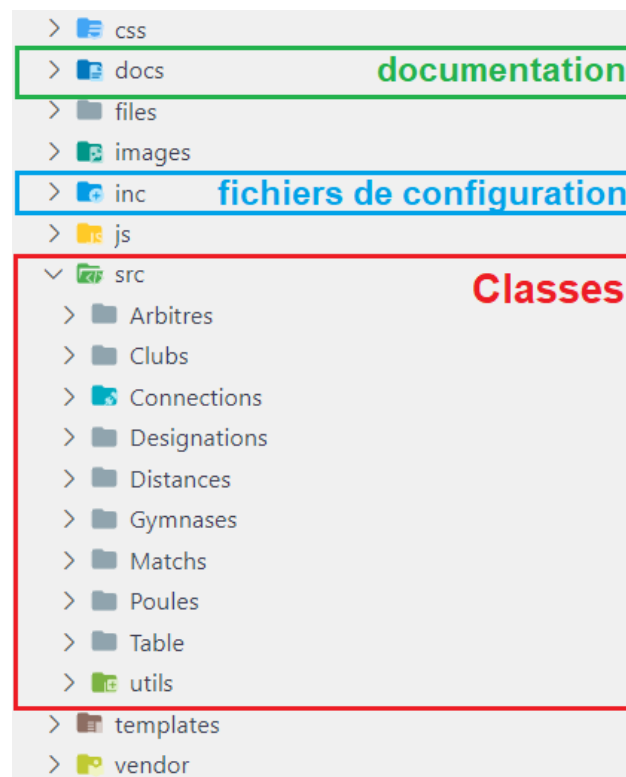


FIGURE 1 – Arborescence des dossiers du projet

Pour effectuer cette séparation des fichiers sans compliquer l'utilisation des classes, j'ai adopté le concept de namespace et utilisé le gestionnaire de dépendances Composer pour ce projet.

Composer est un gestionnaire de dépendances pour PHP. Il permet de réduire l'importation de fichiers en automatisant cette tâche.

➔ Je commencerai par expliquer les méthodes utilisées pour récupérer les données nécessaires à ce projet, puis j'exposerai les modifications apportées à l'interface graphique et enfin je parlerai en profondeur de l'algorithme d'automatisation des désignations.

3 Récupération des données

Par manque d'accès à la base de données fédérale, la première étape consistait à récupérer les données nécessaires à la mise en place des outils.

Après une analyse complète des informations nécessaires au projet, j'en suis arrivé au modèle de données UML suivant :

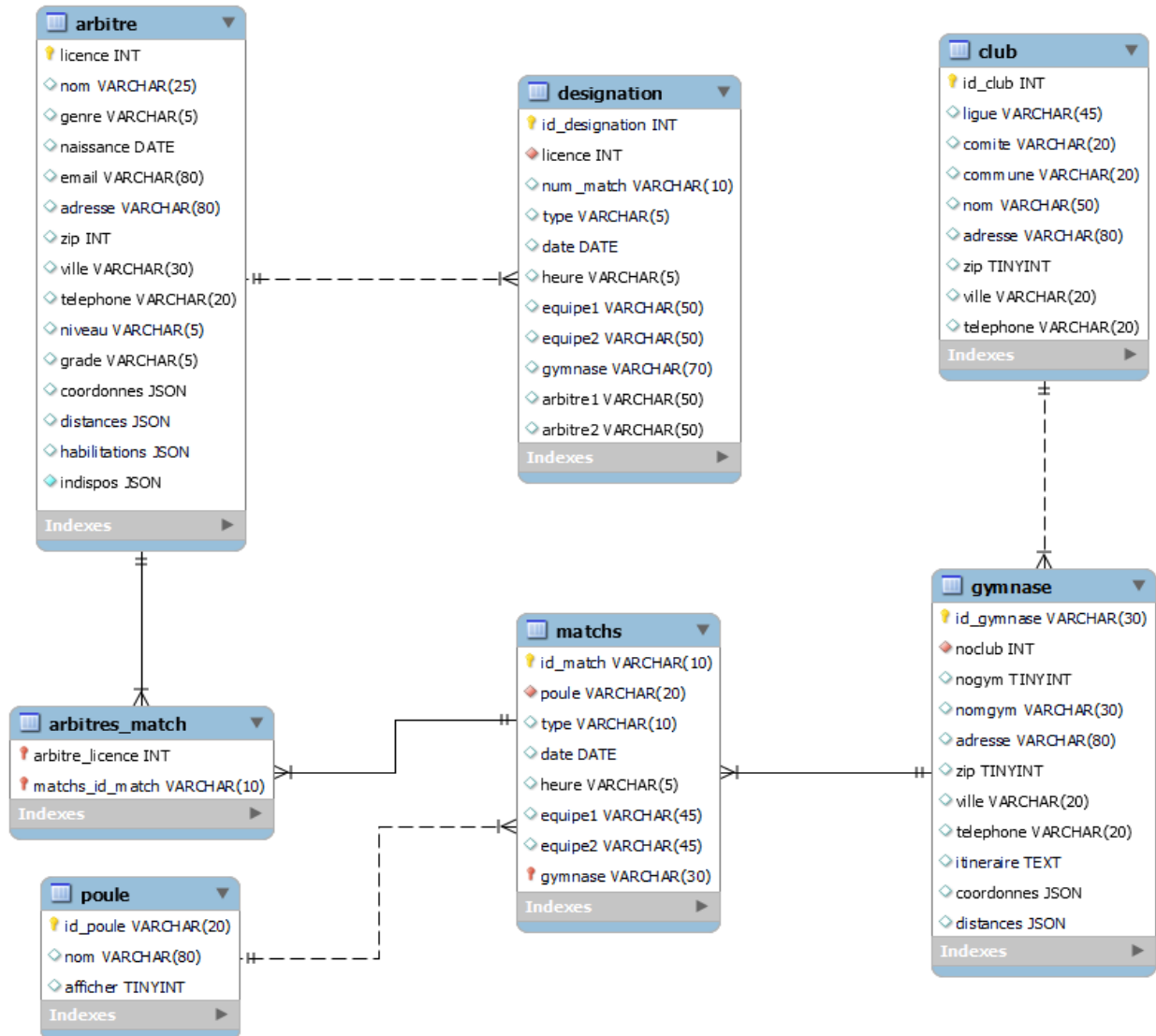


FIGURE 2 – Modèle de données UML effectué avec MySQL Workbench

Certaines données étaient disponibles au format csv (Excel), et d'autres demandaient à être récupérées sur l'intranet actuel de la fédération. Il m'a donc fallu extraire ces dernières via Web Scrapping pour ensuite les insérer dans ma base de données.

3.1 Configuration

Dans un premier temps, afin de faciliter la configuration de l'accès à la base de données lors de la mise en production, j'ai mis en place un fichier de configuration général `globals.php`. Ce fichier contenait un panel de variables de configuration différentes en fonction de l'environnement (développement ou production).

```
switch ($_SERVER['HTTP_HOST']) {
    case 'environnement_production':

        define('DBNAME', "bdd_prod");
        define('DBUSER', "user_prod");
        define('DBPASS', "pass_pro");

        // Gestion des options PDO
        define('PDO_OPTIONS', array(
            PDO::ATTR_ERRMODE => PDO::ERRMODE_SILENT,
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
        ));

        break;

    case 'localhost':

        define('DBNAME', "bdd_dev");
        define('DBUSER', "user_dev");
        define('DBPASS', "pass_dev");

        // Gestion des erreurs
        ini_set('display_errors', 1);
        ini_set('display_startup_errors', 1);
        error_reporting(E_ALL);

        // Gestion des options PDO
        define('PDO_OPTIONS', array(
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
        ));

        break;
}
```

FIGURE 3 – Fichier de configuration d'environnement

Les paramètres de connexion à la base de données, et l’affichage des erreurs sont différents en fonction de l’environnement : les erreurs sont remontées seulement en environnement de développement.

Mon tuteur m’a également demandé de la flexibilité pour le nom des tables constituant la base de données, car il risquait d’être amené à modifier certains d’entre eux dans le futur.

J’ai décidé pour cela d’inclure un fichier `config.json` permettant la modification totale du nom des tables et des colonnes, sans impacter la structure interne de celles-ci.



FIGURE 4 – Contenu du fichier de configuration du nom des tables

Pour récupérer les informations de ce fichier, j’ai également créé un Trait et une méthode associée que je pourrai réutiliser dans mes classes pour récupérer ces informations.

```
trait Config
{
    public static function config()
    {
        $filePath = dirname(__FILE__, 3) . "/config.json";
        return json_decode(file_get_contents($filePath), true);
    }
}
```

FIGURE 5 – Trait d’accès au nom des tables dans le fichier de configuration associé

Ce trait récupère le fichier `config.json` relativement à l'arborescence des dossiers du projet, et renvoi le contenu de celui-ci sous forme de tableau associatif grâce à la fonction `json_decode()`.

Enfin, dans un but de travailler régulièrement avec la base de données, j'ai mis en place une classe `Database` permettant d'effectuer de façon statique une connexion à celle-ci via une instance de l'objet **PHP Data Object (PDO)**.

```
class Database
{
    private static $connection = null;
    private static $hostname = 'localhost';

    public static function connect(): ?PDO
    {
        $hostname = self::$hostname;

        if(is_null(self::$connection)){
            try{
                self::$connection = new PDO("mysql:host=$hostname;dbname=" . DBNAME, DBUSER, DBPASS, PDO_OPTIONS);
            }catch(PDOException $e){
                die($e->getMessage());
            }
        }
        return self::$connection;
    }

    public function disconnect()
    {
        self::$connection = null;
    }
}
```

FIGURE 6 – Composant d'accès à la base de données

3.2 Récupération via scraping

Pour avoir accès aux données indisponibles sous format Excel, il m’a fallu utiliser **libcurl** pour récupérer le contenu du site actuel de la fédération.

Il s’agit d’une bibliothèque permettant de se connecter et de communiquer avec différents types de serveurs, et ce, avec différents types de protocoles.

Celle-ci m’a donc permis via le protocole HTTP de me connecter au site de la fédération avec des identifiants fournis par mon tuteur, de stocker les paramètres de connexion, et d’accéder au contenu HTML complet du site sous forme de chaîne de caractères.

Il m’a alors fallu analyser cette chaîne de caractères pour en tirer les informations qui m’intéressent, puis insérer celles-ci dans la base de données.

3.2.1 Connexion

Le langage PHP permet le support de **libcurl** via l’extension ‘CURL’ (Client URL Request Library), qui est disponible sous forme d’objet.

Celui-ci possède trois modules en fonction de l’utilisation voulue : un gestionnaire simple, un gestionnaire multiple et un gestionnaire de partage. Les ressources liées à ces gestionnaires sont respectivement **CurlHandle**, **CurlMultiHandle** et **CurlShareHandle** depuis la version PHP8

Dans le cadre de ma problématique, l’utilisation du gestionnaire simple était suffisante. Cependant, une connexion au site de la fédération était nécessaire avant toute récupération de contenu sur celui-ci.

Pour faire cela sans multiplier inutilement les connexion pour mes différentes requêtes, j’ai choisi de créer une classe permettant de configurer les paramètres de connexion et de les stocker dans l’objet **CurlHandle** qui me permettait ensuite d’effectuer mes requêtes.

Une fois la connexion effectuée, un cookie était également généré par la librairie pour permettre l’authentification des prochaines requêtes.

```

class Ffvb
{
    private $username;
    private $password;
    private $connection = null;

    public function __construct(string $username, string $password)
    {
        $this->username = htmlspecialchars($username);
        $this->password = htmlspecialchars($password);
    }

    public function connect(): ?CurlHandle
    {
        $ffvbPortal = "https://sitedelafederation.com/connexion";
        $postFields = "username=" . $this->username . "&password=" . $this->password;

        $cookieJar = $GLOBALS['designationRoot'].'/cookies/cookie.txt';
        $cookieJar = str_replace('/', DIRECTORY_SEPARATOR, $cookieJar);

        $CURL_OPTIONS = array(
            CURLOPT_URL => $ffvbPortal,
            CURLOPT_SSL_VERIFYPEER => false,
            CURLOPT_VERBOSE => true,
            CURLOPT_USERAGENT => 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0',
            CURLOPT_POST => 1,
            CURLOPT_POSTFIELDS => $postFields,
            CURLOPT_HTTPHEADER => array('Content-Type: application/x-www-form-urlencoded'),
            CURLOPT_FOLLOWLOCATION => true,
            CURLOPT_COOKIEJAR => $cookieJar,
            CURLOPT_COOKIESESSION => true,
            CURLOPT_RETURNTRANSFER => true
        );

        if(!is_null($this->connection)) throw new Exception('Une connexion est déjà active');

        $this->connection = curl_init();
        curl_setopt_array($this->connection, $CURL_OPTIONS);

        return $this->connection;
    }
}

```

FIGURE 7 – Composant de paramétrage d'une connexion au site de la fédération

3.2.2 Analyse

Les différents outils mis en place pour la récupération des données sur le site de la fédération prennent en paramètre le **CurlHandle** de la connexion configuré précédemment pour effectuer leurs requêtes.

Après avoir défini le chemin vers les cookies de ma connexion, il m'a suffi d'envoyer une requête HTTP à la bonne URL pour y récupérer le contenu brut sous forme de chaîne de caractères.

```
private function htmlArbitres(): string|bool
{
    $urlArbitres = "https://sitedelafederation.com/arbitres";
    $postFields = "arbitres=all";

    $cookieJar = $GLOBALS['designationRoot'] . '/cookies/cookie.txt';
    $cookieJar = str_replace('/', DIRECTORY_SEPARATOR, $cookieJar);

    $CURL_OPTIONS = [
        CURLOPT_URL => $urlArbitres,
        CURLOPT_COOKIEJAR => $cookieJar,
        CURLOPT_USERAGENT => 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0',
        CURLOPT_POSTFIELDS => $postFields,
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => false
    ];

    curl_exec($this->connection);
    // connected
    curl_setopt_array($this->connection, $CURL_OPTIONS);

    return curl_exec($this->connection);
}
```

FIGURE 8 – Méthode de récupération du contenu d'une page de l'intranet

Le contenu étant sous forme de chaîne de caractères, le principal problème était de récupérer seulement les informations utiles depuis cette String. Les données sont sous forme de tableaux, et le ciblage de ceux-ci est impossible dans cet état.

Heureusement, PHP possède nativement l'objet **DOMDocument** qui permet d'analyser une chaîne de caractères afin de récupérer les éléments voulus comme nous l'aurions fait en JavaScript. En effet, cet objet possède plusieurs méthodes similaires et permettait donc de répondre à mon problème.

Cette étape d'analyse étant nécessaire pour toutes mes classes de récupération de données via **CURL**, il m'a semblé logique de créer un objet spécialement conçu pour ça.

De plus, les données à récupérer étant pour la plupart sous forme de tableaux, j'ai également pensé à créer une méthode permettant de récupérer le détails des tableaux contenus dans une chaîne de caractères à partir de l'instance **DOMDocument** liée à celle-ci.

```

public static function getTablesDetails(string $source): array
{
    $dom = self::getDomDocument($source);
    $dom->preserveWhiteSpace = false;

    $tables = $dom->getElementsByTagName('table');
    if (empty($tables))
        throw new Exception(__CLASS__ . ' : aucune table trouvée sur la page demandée.');
```

```

    foreach ($tables as $idt => $table) {
        $headers = $table->getElementsByTagName('th');

        $data[$idt]['headers'] = NULL;
        foreach ($headers as $node) {
            $data[$idt]['headers'][] = $node->nodeValue;
        }

        $rows = $table->getElementsByTagName('tr');
```

```

        foreach ($rows as $idr => $row) {
            $cols = $row->getElementsByTagName('td');

            if ($cols) {
                foreach ($cols as $idc => $col) {
                    $data[$idt][$idr][$idc] = self::nodeDetails($col);
                }
            }
        }
    }
}

```

FIGURE 9 – Méthode d’analyse du contenu d’une page HTML

Les détails des colonnes sont récupérés grâce à la méthode `nodeDetails()` qui permet de scanner chaque colonne en récupérant le texte et les attributs si disponibles. Cette méthode effectue ce travail de façon dynamique pour chaque sous-nœud d’une colonne en se rappelant elle-même.

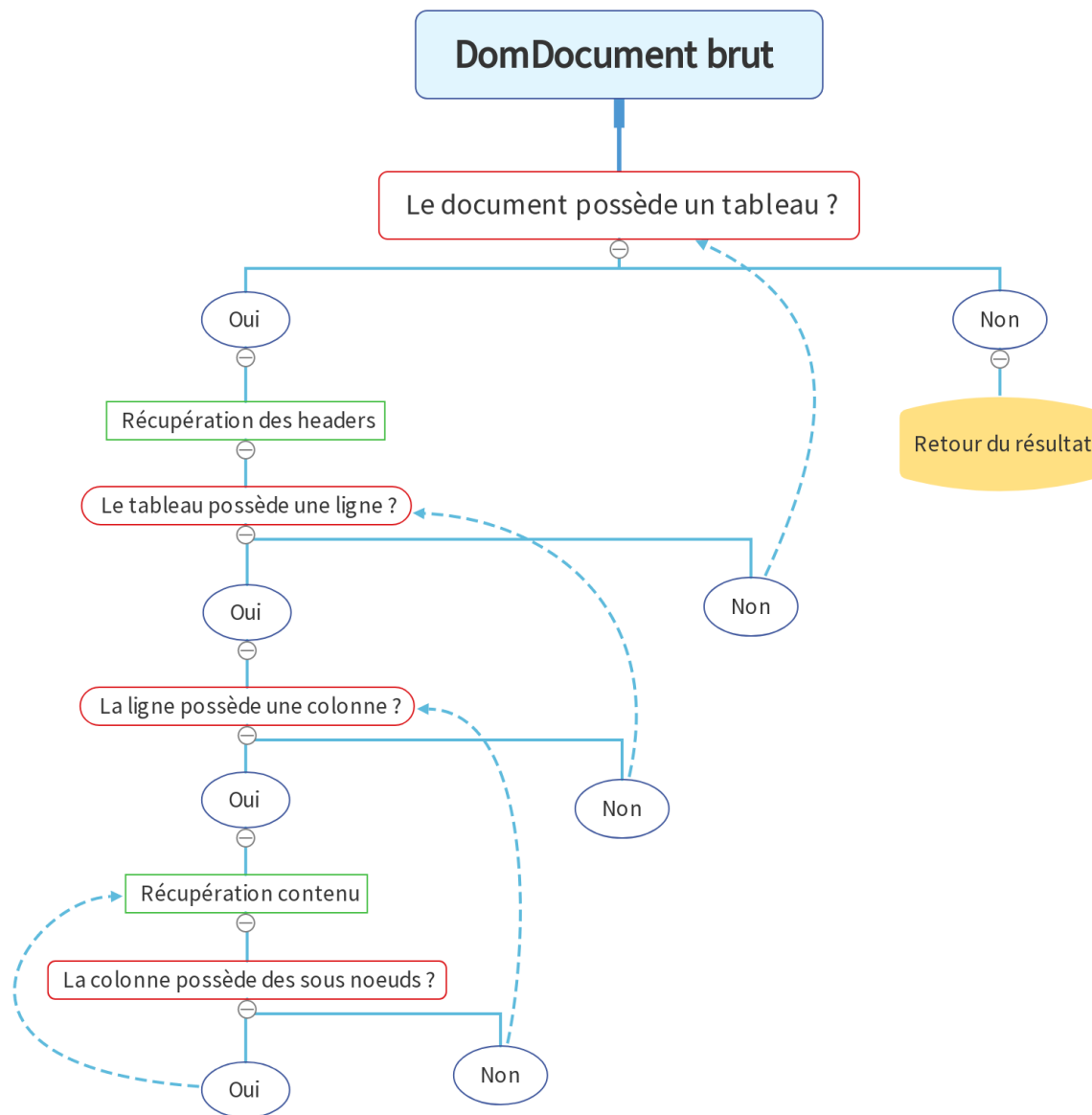


FIGURE 10 – Logique suivie par la méthode d’analyse du contenu d’une page

Une fois ces objets et méthodes mis en place, la récupération et l’analyse des pages intéressantes du site de la fédération étaient beaucoup plus simples.

J’ai donc ensuite créé une méthode propre à chacune de mes classes permettant de cibler et nettoyer les données renvoyées par les méthodes précédentes. Celles-ci ciblent les attributs et textes renvoyés par la méthode `getTablesDetails()` pour afficher un tableau contenant l’essentiel des données voulues.

3.2.3 Insertion

Pour stocker ces données dans la base de données, j’ai d’abord récupéré le nom des tables et colonnes concernées dans le fichier config.json (voir configuration), puis j’ai réutilisé ma classe Database pour effectuer une connexion avec celle-ci.

```
use Config;

public function insert(): void
{
    /** récupération des noms de tables/colonnes nécessaires

    list(
        'table' => $table,
        'primary_key' => $primary_key,
    ) = self::config()['table_name'];

    /** Connexion BDD et début de la transaction

    $dbh = Database::connect();
    $dbh->setAttribute(PDO::ATTR_AUTOCOMMIT, FALSE);

    $dbh->beginTransaction();
}
```

FIGURE 11 – Initialisation de l’insertion des données

Pour effectuer cette tâche de façon rapide et sécurisée, j’ai préféré utiliser des transactions et des requêtes préparées lors de la création des données. Les requêtes préparées permettent d’augmenter les performances en minimisant les allers-retours avec la base de données. Quant à elles, les transactions permettent d’éviter la perte de données dans le cas où certaines des requêtes ne passeraient pas.

De plus, l’objet PDO donne la possibilité de nettoyer les données lors de l’exécution en utilisant des variables nommées ou anonymes dans la requête, permettant ainsi d’éviter les **injections SQL**.

Les injections SQL sont des méthodes permettant de détourner les requêtes SQL en injectant des bouts de codes malicieux.

```

try {
    /* création de la table si besoin */
    $create = "CREATE TABLE IF NOT EXISTS $table (
        $primary_key VARCHAR(50) PRIMARY KEY UNIQUE NOT NULL,
        colonne VARCHAR(50) NULL
    )ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;";

    $dbh->query($create);

    /* vérification des données de la table */
    $query = "SELECT * FROM $table WHERE $primary_key = :id";
    $select = $dbh->prepare($query);

    /* requête pour insérer les données */
    $query = "INSERT INTO $table($primary_key, colonne)
        VALUES (:pk, :col)";
    $insert = $dbh->prepare($query);

    /* requête pour mettre à jour les données */
    $query = "UPDATE $table SET colonne = :col
        WHERE $primary_key = :pk";
    $update = $dbh->prepare($query);

    /* récupération des données formatées */
    $data = $this->get();

    foreach ($data as $key => $value) {

        $select->execute([':pk' => htmlspecialchars($key)]);

        $params = [
            ':pk' => htmlspecialchars($key),
            ':col' => htmlspecialchars($value)
        ];

        if ($select->rowCount() == 0) {
            $insert->execute($params);
        } else {
            $update->execute($params);
        }
    }

    $dbh->commit();
} catch (PDOException $e) {
    $dbh->rollBack();
    throw new PDOException($e->getMessage());
}

```

FIGURE 12 – Insertion de toutes les données de façon sécurisée

La logique de la méthode `insert()` est de vérifier si les informations à insérer existent déjà dans la base de données, de les mettre à jour dans ce cas, et de les insérer dans le cas contraire. Ceci évite alors les erreurs SQL dues à la multiplicité des clés primaires.

Cette étape d'insertion des données m'a également permis d'uniformiser les données récupérées sur différents points d'entrées du site de la fédération, comme par exemples les matchs régionaux et nationaux.

En effet, les matchs régionaux et nationaux se trouvent sur deux parties différentes du site officiel et la récupération des informations liées à ceux-ci via scraping ne renvoient pas exactement le même type de données. Cette étape d'insertion était donc l'occasion d'uniformiser les informations qui seraient disponibles pour les uns mais pas les autres, en définissant des valeurs par défaut quand nécessaire.

3.3 Récupération via fichiers Excel

Heureusement, certaines données étaient disponibles directement sous format Excel. Le traitement de celles-ci était donc beaucoup plus facile que précédemment.

Il m'a simplement fallu placer les fichiers Excel dans un dossier, et définir le bon chemin d'accès à ceux-ci dans mon script.

Ensuite en ouvrant un stream du fichier grâce à la fonction `file`, qui renvoie un tableau du contenu, j'ai pu analyser celui-ci grâce à une combinaison des fonctions `array_map` et `str_getcsv` pour en tirer un tableau multidimensionnel des lignes du fichier Excel.

Chaque ligne du fichier Excel était sous forme de tableau de chaîne de caractères qu'il fallait ensuite séparer par colonnes, en utilisant le séparateur utilisé dans le fichier.

```
$rows = array_map(
    function ($element) {
        return str_getcsv($element, ';');
    },
    file($filePath)
);
```

FIGURE 13 – Parsing du fichier Excel

Puis en itérant dans chaque ligne de mon fichier, j'ai pu insérer les données voulues dans la base de données en tenant compte de l'encodage des caractères qu'il fallait convertir pour être en accord avec l'encodage de la base de données.

Les fichiers Excel sont encodés en **Windows-1252** (ou **CP1252**) et doivent donc être convertis en **UTF-8** pour prendre en compte les caractères accentués par exemple.

```
$headers = array_shift($rows);
foreach ($rows as $row) {
    // encodage des caractères en utf-8
    $row = array_map(
        function ($element) {
            return mb_convert_encoding($element, 'UTF-8', 'CP1252');
        },
        $row
    );

    // insertion des données
}
```

FIGURE 14 – Encodage des caractères du fichier

3.3.1 Insertion

Après avoir récupéré les noms des tables et des colonnes dans `config.json`, j'ai encore une fois utilisé des requêtes préparées et des transactions par soucis de performance et de sécurité.

Les valeurs étant déjà encodées et donc prêtes à être ajoutées à la base de données, j'ai suivi le même principe que pour l'insertion des données via `CURL`, c'est à dire :

1. Vérifier si la ligne existe déjà
2. Si elle existe, alors mettre à jour les données ('UPDATE table')
3. Si elle n'existe pas, insérer les données ('INSERT INTO table')

3.4 Distances

Une tâche nécessaire à la mise en place de l'algorithme d'automatisation des désignations d'arbitres était de pouvoir quantifier la distance séparant les arbitres des gymnases. Cette donnée devait servir de critère lors de la sélection automatique, mais devait également être disponible directement sur l'interface.

Les distances entre les arbitres et les gymnases devaient donc être stockées en base de données sous forme de matrice pour permettre de récupérer facilement une distance entre le domicile d'un arbitre et l'adresse d'un gymnase. Face à ce problème, j'ai pensé à deux façon différentes de procéder :

- En créant une table unique associant une ligne par arbitre et une colonne par gymnase, l'intersection de chaque définissant une distance
- En stockant la matrice au format JSON dans une colonne de la table des arbitres, qui associe pour chaque arbitre l'id de chaque gymnase et sa distance avec celui-ci, puis effectue de façon symétrique cette étape dans la table des gymnases.

Le nombre d'arbitres et de gymnases étant évolutif et de l'ordre de plusieurs centaines, la deuxième option me paraissait plus facile à maintenir et à mettre en place.

3.4.1 Coordonnées géographiques

La première étape était de réfléchir à un moyen de récupérer les distances entre deux adresses. Pour ça deux méthodes s'offraient à moi :

- La première était d'utiliser l'API Google Geocoding pour tous les arbitres et les gymnases. Cette API permet de récupérer les coordonnées GPS d'une adresse (longitude et latitude), ce qui pouvait me permettre de calculer la distance à vol d'oiseau séparant chaque arbitre et chaque gymnase grâce à une formule mathématique (Loi des cosinus).
- La deuxième méthode était d'utiliser l'API Google Matrix. Cette API permet de récupérer directement la matrice des distances entre une adresse de départ et plusieurs adresses d'arrivée.

Pour commencer, j'ai opté pour l'utilisation de la première méthode car celle-ci me permettait de réduire le nombre total de requêtes envoyées à Google.

En effet, cette méthode ne demandait qu'une requête par arbitre et par gymnase, ce qui équivalait pour n arbitres et m gymnases à un total de $n + m$ requêtes. La deuxième méthode demandait quant à elle un total de $n * m$ requêtes. Google n'autorise cependant qu'un nombre limité de requêtes gratuites par jour que je voulais éviter de dépasser.

La recherche de la formule mathématique permettant de calculer une distance entre deux points à partir de leurs coordonnées GPS a été faite sur des sites anglophones à cause du manque d'informations sur les sites francophones.

In the picture above, this would be $\cos(\delta)$.

So you can compute the angle t as a function of a_1, b_1, a_2, b_2 , which are the latitudes and longitudes of our cities p and q . Then visualize what you've got: draw a great circle through the points p and q . This is just a plain old Joe-Schmoe circle of radius r , and the **angle t is the angle of the arc that subtends p and q** . The problem from here on out is just figuring out what the arc length between p and q is. The relevant formula is $\text{Arc length} = t/360 * 2\pi * r$. So that's your formula. By substitution, we have

```
Arccos[Cos[a1] Cos[b1] Cos[a2] Cos[b2] + Cos[a1] Sin[b1] Cos[a2] Sin[b2]  
+ Sin[a1] Sin[a2]]/360 * 2Pi * r
```

Oh, by the way, **West longitude means negative values of b** , and **South latitude means negative values of a** .

Here is that great circle through P and Q :

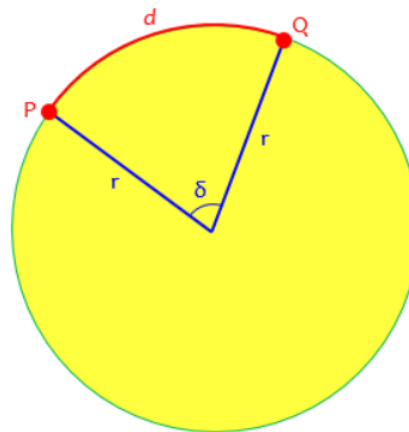


FIGURE 15 – Article anglophone donnant une formule pour calculer une distance à partir des coordonnées GPS

Traduction : On peut calculer l'angle delta comme une fonction de paramètres a_1, b_1, a_2 et b_2 représentant respectivement les latitudes et longitudes des points P et Q . Pour visualiser ce qu'on a : on dessine un cercle passant par nos points P et Q . Il s'agit simplement d'un cercle de rayon r , l'angle delta étant l'angle de l'arc qui relie les points P et Q . Le problème ici est de savoir quel est la distance de l'arc entre les points P et Q . La formule pertinente est : $d = \delta/360 * 2\pi * r$

Pour effectuer le calcul de distance à partir de cette formule, j'ai mise en place un Trait possédant les méthodes nécessaires à la conversion de ces coordonnées en distances.

J'ai également mis en place les outils nécessaires à l'utilisation de la deuxième méthode pour l'utiliser à terme, car celle-ci permet la récupération d'informations plus riches et précises

```

Trait DistancesConverter
{
    public static function directDistance(float $longA, float $latA, float $longB, float $latB)
    {
        // radiant de la terre en km
        $radius = 6378137;

        $longA = self::radiant($longA);
        $latA = self::radiant($latA);

        $longB = self::radiant($longB);
        $latB = self::radiant($latB);

        $delta = abs($longB - $longA);

        $angle = acos(sin($latA) * sin($latB) + cos($latA) * cos($latB) * cos($delta));

        return $radius * $angle;
    }

    // converti une coordonnée décimale en coordonnée radiale
    public static function radiant(float $coordinate)
    {
        return $coordinate * pi() / 180;
    }

    // converti une coordonnée radiale en coordonnée décimale
    public static function decimal(float $radiant)
    {
        return $radiant * 180 / pi();
    }
}

```

FIGURE 16 – Trait pour convertir des coordonnées GPS en distance

comme le temps de trajet en fonction du moyen de transport par exemple.

J'ai commencé par créer une classe avec deux méthodes distinctes pour effectuer les différentes requêtes.

Les méthodes geocode et matrix récupèrent respectivement les réponses renvoyées par les API Google Geocoding et Google Matrix, sous forme de tableau associatif grâce à la fonction `json_decode` pour une manipulation ultérieure.

J'ai ensuite réalisé une classe pour la récupération et la mise en place des données géographiques dans la base de données. La récupération se base sur les méthodes de la classe `DistanceRequest`, pour chaque arbitre et chaque gymnase disponibles dans la base de données (voir section sur la récupération des données).

La logique suivie par les méthodes de cette classe est d'abord de créer la colonne ac-


```

class DistancesRequest
{
    // API Google geocode
    public static function geocode(string $address)
    {
        $address = htmlspecialchars($address);
        $address = urlencode($address);
        $key      = API_KEY;
        $url      = "https://maps.googleapis.com/maps/api/geocode/json?address={$address}&key={$key}";

        $res = json_decode(file_get_contents($url), true);

        return $res;
    }

    // API Google matrix
    public static function matrix(string $origin, string ...$destinations)
    {
        $origin = htmlspecialchars($origin);
        $destination = implode('|', $destinations);
        $key      = API_KEY;
        $url      = "https://maps.googleapis.com/maps/api/distancematrix/json?destinations={$destination}&origins={$origin}&key={$key}";

        $res = json_decode(file_get_contents($url), true);

        return $res;
    }
}

```

FIGURE 17 – Composant pour envoyer des requêtes aux API Google

cueillant les coordonnées dans les tables si inexistante, puis d’insérer les coordonnées en se basant sur la réponse de l’API Google Geocoding.

À l’issu de ce processus, les tables des arbitres et des gymnases avaient toutes les deux une colonne supplémentaire avec les coordonnées géographiques de leur adresse. Ces coordonnées étaient sous format **JSON** et stockaient **null** si l’adresse était indisponible ou mal formatée.

3.4.2 Matrice des distances

Comme je l’ai expliqué dans l’introduction de ce chapitre, l’objectif avec ces coordonnées était de calculer la matrice des distances entre chaque adresse d’arbitre et de gymnase.

Je me suis basé sur les outils du composant **DistancesConverter** pour effectuer le calcul des distances, et j’ai ensuite stocké ces matrices sous forme de tableau associatif au format **JSON** pour plus de flexibilité.

Ces matrices sont construites de façon à associer l’ID de chaque gymnase à la distance en mètres le séparant d’un arbitre, pour chaque arbitre puis de façon symétrique pour chaque gymnase.

Pour bien finir, et dans un objectif d’automatisation de ce processus dans le futur, j’ai finalement créé un composant dédié à la mise en place de ces matrices qui se base sur les classes de récupération de coordonnées et de calcul de distance à partir de celles-ci. Ce dernier appelle de façon procédurale les méthodes permettant de récupérer les coordonnées GPS et de calculer la matrice des distances pour chaque arbitre et chaque gymnase.

3.5 Habilitations

Un autre critère essentiel à la mise en place de l'automatisation de la désignation des arbitres était la prise en compte des habilitations des arbitres. En effet, cet algorithme devait pouvoir proposer la désignation d'arbitres habilités pour chaque match.

Le problème était que ces habilitations n'étaient pas disponibles directement dans les informations liées aux arbitres, et ne pouvaient pas être récupérées via CURL comme pour les informations précédentes. Il fallait donc pouvoir récupérer celles-ci d'une façon différente.

Ici encore j'ai pensé à deux méthodes qui présentaient chacune leurs avantages et inconvénients :

- la première méthode était de déduire ces habilitations à partir de la liste passée des désignations, en prenant en compte le type de match le plus élevé qu'avait arbitré chaque arbitre, pour en déduire un plafond d'habilitation.
- La deuxième méthode était de déduire les habilitations de chaque arbitre à partir du grade et du niveau de leur licence.

La première méthode présentait quelques problèmes : le plus important était le manque d'informations concernant une bonne partie des arbitres qui n'avaient pas pu arbitrer les matchs de la saison en cours. De plus, les arbitres pour lesquels les informations étaient disponibles ne reflétaient pas forcément le niveau maximum d'habilitation qu'ils possédaient. J'ai estimé un taux d'erreur avec cette méthode de l'ordre de 30% environ.

Concernant la deuxième méthode, il faut savoir que les habilitations d'arbitres ne sont pas toujours définies par leur niveau et grade de licence, mais que ces données sont quand même fortement liées aux habilitations. Le taux d'erreur avec cette méthode était de l'ordre de 15%.

Le taux d'erreur étant plus faible, j'ai donc décidé de me reposer sur la deuxième méthode pour effectuer la déduction des habilitations, et de mettre en place une interface graphique permettant la mise à jour de celles-ci. Cette interface permettrait aux gestionnaires d'effectuer les modifications nécessaires pour les arbitres pour lesquels les habilitations ne correspondraient pas.

Pour effectuer la mise en place des habilitations initiales des arbitres, j'ai créé une unique méthode dans une classe dédiée à cette tâche

```

$query = "UPDATE $arbitres_table SET habilitations = :json
WHERE licence = :licence";
$update = $dbh->prepare($query);

$query = "SELECT * FROM $arbitres_table WHERE habilitations IS NULL";
$select = $dbh->query($query);

// On assigne les habilitations en fonction du niveau et du grade
$habilitationsKeys = ['ARM', 'ARF', 'ARG', 'ARL', 'ARN', 'ARO', 'ARQ', 'AR
foreach ($select->fetchAll() as $arbitre) {

    /* Aucunes habilitation de base
    $habilitations = array_fill_keys($habilitationsKeys, 0);

    // Habilitations départementales
    $habiliteDep = function () use ($habilitations) {
        return array_replace($habilitations, [
            'ARM' => 1,
            'ARF' => 1
        ]);
    };

    /* Habilitations régionales ( + départementales )
    $habiliteReg = function () use ($habiliteDep) {
        return array_replace($habiliteDep(), [
            'REM' => 1,
            'REF' => 1
        ]);
    };
};

```

FIGURE 18 – Mise en place des requêtes et fonction nécessaire à la mise en place des habilitations

```

// Assignment des habilitations en fonction du niveau et du grade
switch ($arbitre['niveau']) {
    case 'Dép.':
        $habilitations = $arbitre['grade'] == 'RE' ?
            $habiliteReg() :
            $habiliteDep();
        break;
    case 'Lig.':
        $habilitations = $arbitre['grade'] == 'NA' ?
            $habiliteNat() :
            $habiliteLig();
        break;
    case 'Nat.':
        $habilitations = $habiliteNat();
        break;
    case 'Féd.':
        $habilitations = $habiliteFed();
        break;
}

$json = json_encode($habilitations);

$update->execute([
    ':json' => $json,
    ':licence' => $arbitre['licence']
]);

```

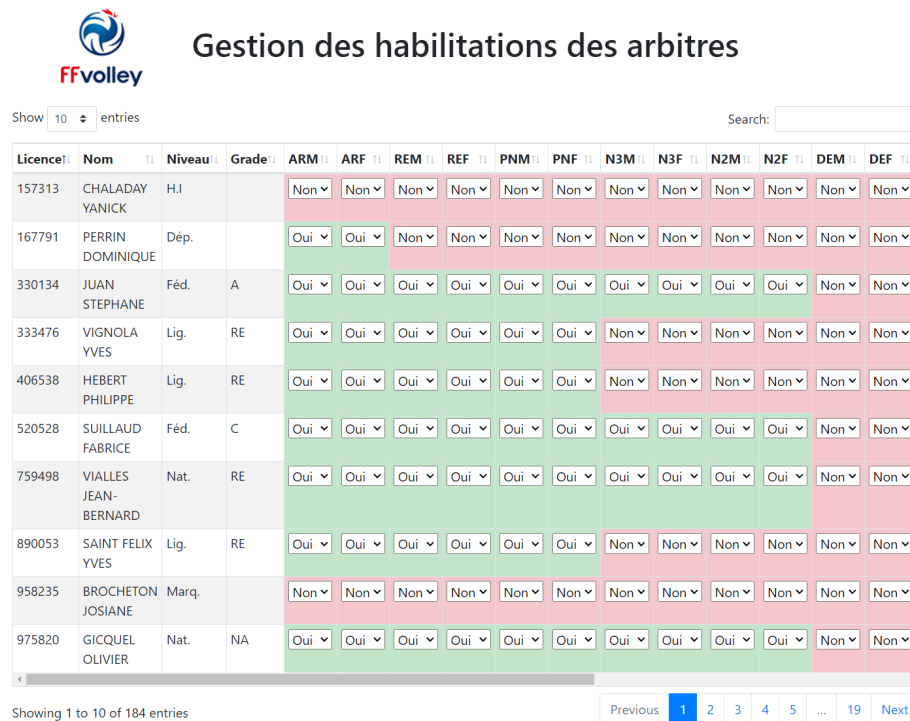
FIGURE 19 – Détermination des habilitations initiales de l'arbitre

Les différents types d’habilitations nous ont été indiqués par notre tuteur et sont définis par des critères subjectifs basés sur les différents types de matchs possibles. Les clés correspondantes à ces habilitations sont stockées dans un tableau indexé, et initialisées à zéro pour chaque arbitre. Cette valeur signifie que les arbitres n’ont aucunes habilitations au départ.

Puis en fonction du niveau et du grade de leur licence, les habilitations des arbitres sont modifiées à un pour chaque clé correspondante au niveau d’habilitation associé. Ces habilitations étant cumulées de façon croissante à chaque niveau supplémentaire, j’ai utilisé la fonction `array_replace()` pour retourner un tableau des habilitations complètes de chaque arbitre.

Une fois ce tableau défini, il fallait finalement l’insérer sous format **JSON** dans la base de données. Pour faire cela, j’ai utilisé la méthode `json_encode()` sur le tableau des habilitations et je l’ai inséré à l’aide d’une requête SQL de type INSERT.

Une fois les habilitations initiales insérées dans la base de données, j’ai ensuite pu travailler sur l’interface de mise à jour de celles-ci. J’ai donc mis en place une page annexe qui affiche la liste de tous les arbitres, auxquels on associe la liste de toutes les habilitations préremplies sous forme de SELECT. Chaque select permet de choisir entre “Oui/Non”, permettant ainsi de choisir ou non d’accorder l’habilitation à l’arbitre voulu.



Gestion des habilitations des arbitres

Search:

Show 10 entries

Licence	Nom	Niveau	Grade	ARM	ARF	REM	REF	PNM	PNF	N3M	N3F	N2M	N2F	DEM	DEF
157313	CHALADAY YANICK	H.I		Non	Non	Non	Non	Non	Non	Non	Non	Non	Non	Non	Non
167791	PERRIN DOMINIQUE	Dép.		Oui	Oui	Non	Non	Non	Non	Non	Non	Non	Non	Non	Non
330134	JUAN STEPHANE	Féd.	A	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non
333476	VIGNOLA YVES	Lig.	RE	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non	Non	Non	Non	Non
406538	HEBERT PHILIPPE	Lig.	RE	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non	Non	Non	Non	Non
520528	SUILLAUD FABRICE	Féd.	C	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non
759498	VIALLES JEAN-BERNARD	Nat.	RE	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non
890053	SAINT FELIX YVES	Lig.	RE	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non	Non	Non	Non	Non
958235	BROCHETON JOSIANE	Marq.		Non	Non	Non	Non	Non	Non	Non	Non	Non	Non	Non	Non
975820	GICQUEL OLIVIER	Nat.	NA	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Non	Non

Showing 1 to 10 of 184 entries

Previous 1 2 3 4 5 ... 19 Next

FIGURE 20 – Interface de mise à jour des habilitations

Un script javascript permet ensuite au changement d'un select, d'envoyer via ajax le numéro de licence et les nouvelles informations d'habilitations à une page dédiée de notre back-end. Cette page récupère ensuite les habilitations actuelles de l'arbitre, les transforme en tableau associatif grâce à la fonction `json_decode()`, puis met à jour l'habilitation voulue. Elle encode ensuite ce nouveau tableau sous format JSON et l'insère dans la base de données.

Pour plus de détails concernant la partie graphique de cette page, merci de vous reporter au chapitre sur les interfaces.

3.6 Mise à jour

Toutes les informations récupérées et calculées jusqu'à maintenant, devaient pouvoir être mises à jour régulièrement via un script simple. J'ai donc mis en place un composant spécialement dédiée à cette action, qui lance de façon procédurale la mise en place de la base de données et la récupération complète de tous les éléments précédemment introduits.

Cette classe prend en paramètre des identifiants fédéraux valides, nécessaires au lancement des méthodes CURL, ainsi qu'une clé API Google active.

```
public function __construct(  
    ?string $username = null,  
    ?string $password = null,  
    ?string $googleKey = null  
)  
{  
    $this->username = htmlspecialchars($username);  
    $this->password = htmlspecialchars($password);  
    $this->googleKey = htmlspecialchars($googleKey);  
}  
  
public function insert(): string  
{  
    $this->insertArbitres();  
    $this->insertIndispos();  
    $this->insertDesignations();  
    $this->insertClubs();  
    $this->insertGymnases();  
    $this->insertMatches();  
    $this->insertPoules();  
    $this->insertMatrix();  
  
    return 'Done';  
}
```

FIGURE 21 – Méthode principale du composant de mise à jour de la BDD

Un appel à la méthode `insert()` de cette classe crée la base de données définie dans le fichier de configuration `globals.php` (voir configuration), en prenant en compte le nom des tables et colonnes définies dans le fichier `config.json`, puis insère toutes les données nécessaires au projet depuis les différentes classes de celui-ci.

4 Algorithme

Une fois tous les outils en main, j'ai pu travailler sur l'algorithme d'automatisation des désignations. Pour rappel, cet algorithme devait prendre en compte trois critères importants : les disponibilités des arbitres, leurs habilitations et la distance qui les sépare du lieu de la rencontre.

L'étape la plus importante a été de réfléchir à une façon de procéder aux désignations selon ces critères et de proposer celles-ci sans les valider directement.

Dans un souci de maintenabilité et d'évolutivité, j'ai fractionné cet algorithme en plusieurs composants qui font chacun référence à une étape ou sous-étape de celui-ci. De cette façon, la compréhension de l'algorithme devenait plus facile et son amélioration plus accessible.

4.1 Principe général

Une première étape a été de regrouper tous les matchs de la période voulue par date, puis de vérifier les disponibilités des arbitres pour chacun d'entre elles et d'obtenir un tableau des arbitres disponibles du jour.

Il fallait ensuite prendre en compte individuellement pour chaque match les habilitations de ces arbitres, pour en tirer une sous liste d'arbitres potentiels.

À partir de cette sous-liste d'arbitres potentiels pour chaque match, il fallait ensuite prendre en compte le critère des distances de façon assez subtile pour ne pas influencer complètement sur les désignations.

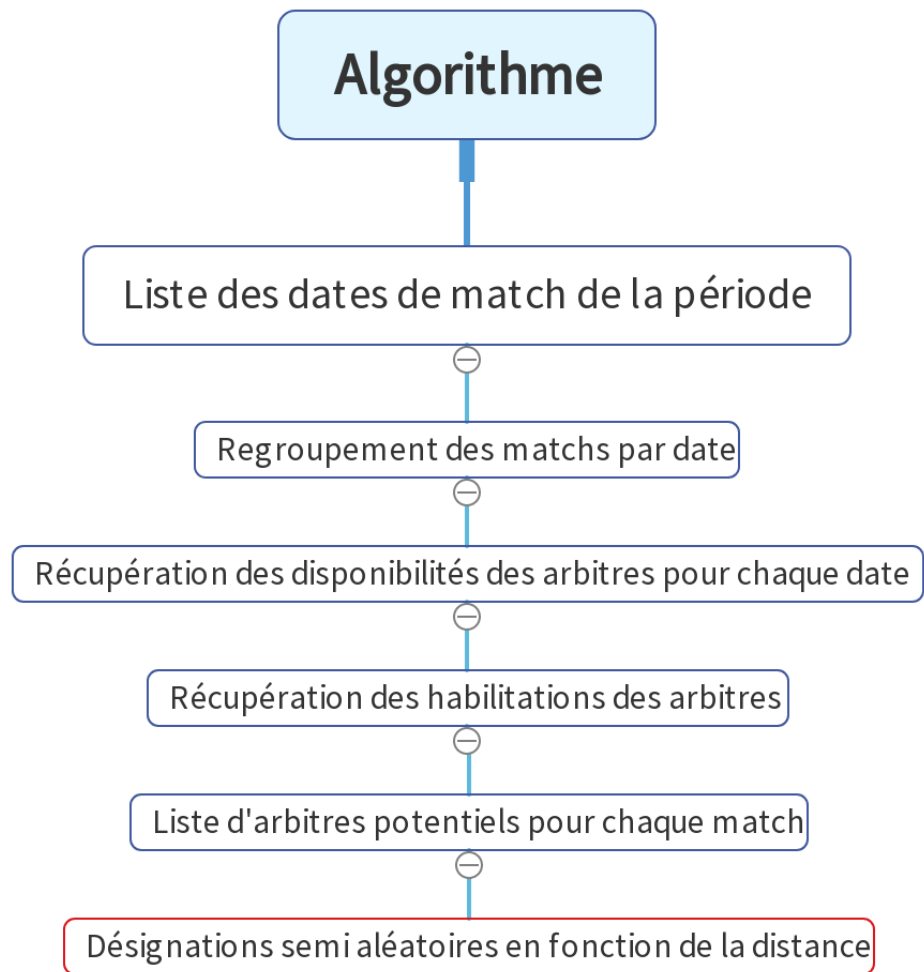


FIGURE 22 – Logique suivie par l’algorithme

4.2 Etape 1 : regroupement des matchs par date

Le regroupement des matchs par date était une première étape importante car elle devait également permettre de trier les matchs par type pour la suite de l'algorithme.

En effet, les matchs peuvent être de type tournois dans le cas où plusieurs d'entre eux se déroulent au même endroit, ou de type uniques dans le cas contraire. Ainsi, en faisant ce tri à ce moment, il était possible de prendre en compte différemment les désignations dans la suite du programme.

De plus, l'algorithme devait pouvoir effectuer les désignations d'une période donnée, ou de l'ensemble des matchs de la saison si aucune date n'était fournie. Il fallait donc également prendre en compte ces deux scénarios différents.

Le composant qui s'occupe de cette première étape prend donc en paramètres les dates de la période souhaitée, qui ont comme valeur par défaut les dates de la période allant d'aujourd'hui à la fin de saison.

```
class DesignationsBase
{
    private string $date_debut;
    private string $date_fin;

    public function __construct(?string $date_debut, ?string $date_fin)
    {
        $this->date_debut = $date_debut ?? date('Y-m-d', strtotime('now'));
        $this->date_fin = $date_fin ?? currentSaison('end');
    }

    // methodes
}
```

FIGURE 23 – Constructeur du composant principal

Ce composant sous forme de classe possède la méthode `group()` qui permet de regrouper les matchs de la période par date, tout en faisant le tri entre les matchs tournois et uniques. Cette méthode récupère la liste des dates distinctes de cette période puis associe, pour chaque date, la liste des matchs tournois et uniques à leur tableau des matchs respectifs.

```

// on récupère Les différentes date de la période
$query = "SELECT DISTINCT $col_date FROM $match_table
WHERE $col_date between ? AND ?
GROUP BY $col_date";
$select = $dbh->prepare($query);
$select->execute([$this->date_debut, $this->date_fin]);

$dates = $select->fetchAll(PDO::FETCH_COLUMN);

/* récupération des matchs uniques
$query = "SELECT * FROM $match_table
        WHERE ($col_date, $col_gymnase, $col_arb1, $col_arb2) IN
        (SELECT $col_date, $col_gymnase, '', '' FROM $match_table
        WHERE $col_date = ?
        GROUP BY $col_date, $col_gymnase HAVING count(*) = 1)";
$uniques = $dbh->prepare($query);

/* récupération des matchs tournois
$query = "SELECT * FROM $match_table
        WHERE ($col_date, $col_gymnase, $col_arb1, $col_arb2) IN
        (SELECT $col_date, $col_gymnase, '', '' FROM $match_table
        WHERE $col_date = ?
        GROUP BY $col_date, $col_gymnase HAVING count(*) > 1)";
$tournois = $dbh->prepare($query);

```

FIGURE 24 – Préparation des requêtes qui permettent de grouper les matchs

4.3 Etape 2 : récupération des arbitres potentiels

Cette liste de dates associées à leurs matchs m'a ensuite permis d'appliquer le premier critère de disponibilité. Pour vérifier les disponibilités des arbitres de chaque match, j'ai réutilisé des méthodes implémentées lors de l'insertion des indisponibilités et des désignations dans la base de données. Alliées ensemble, ces méthodes permettent de définir si un arbitre est disponible pour une date donnée. Elles m'ont donc permis d'attacher à chaque match un tableau des arbitres disponibles.

Ensuite, il a fallu définir pour chaque match une sous liste d'arbitres potentiels habilités à partir de ces tableaux d'arbitres disponibles. Pour faire cela j'ai utilisé la classe dédiée au filtre des habilitations en appliquant pour chaque arbitre potentiels une méthode qui renvoi un booléen si l'arbitre est habilité ou non pour le match voulu.

4.4 Etape 3 : processus de sélection

J'avais donc pour chaque match une liste d'arbitres habilités potentiels. La prochaine étape était donc de procéder aux désignations.

Contrairement aux matchs uniques, les matchs tournois doivent avoir un arbitre pour chaque groupe de trois matchs de la même poule. Il fallait donc prendre en compte différemment la sélection en fonction du type de match.

Autre contrainte mineure, il fallait que la désignation prenne en compte le nombre exacte d'arbitres pour chaque match.

4.4.1 Matchs uniques

La seule chose à prendre en compte pour la sélection des arbitres pour les matchs uniques était le nombre d'arbitres nécessaires pour chaque match. Cette donnée n'est pas explicite et dépend de la poule et du niveau du match. J'ai donc créé une méthode dans la classe `DesignationsBase` qui permet de déterminer si un match demande deux arbitres ou non.

```
// Tableau des poules régionales de type 'double arbitre'
$poules_reg_double_arb = ['PM'];

try {
    $query = "SELECT * FROM $match_table WHERE $match_pk = ?";
    $select = $dbh->prepare($query);
    $select->execute([htmlspecialchars($matchID)]);

    $match = $select->fetch();

    foreach ($poules_reg_double_arb as $pl) {
        if (
            $match['type'] == 'REG' &&
            strpos($match['matchID'], $pl) !== false
        )
            return true;
    }

    if ($match['type'] == 'NAT' || $match['type'] == 'LNV') return true;

    return false;
} catch (PDOException $e) {
    throw new PDOException($e->getMessage());
}
```

FIGURE 25 – Détermination du nombre d'arbitres pour le match

Le nombre d'arbitres est déterminé en fonction du niveau du match (régional ou national), et prend en compte les exceptions.

Une fois le nombre d'arbitres défini, la distance entre chaque arbitre potentiel est récupérée dans la base de données et associée avec les arbitres dans un tableau trié par distance croissante. Ce tableau final permet de procéder aux désignations de façon semi aléatoire en augmentant progressivement la probabilité de sélection des arbitres en fonction de leur distance avec le gymnase.

Un score aléatoire est attribué à chaque arbitre. Ce nombre aléatoire est compris entre 0 et 100, puis multiplié par une valeur allant de 2 à 0,5. Les arbitres voient leur score ajusté à la baisse entre ces deux valeurs en fonction de leur distance avec le gymnase. Ceci induit alors que le score de l'arbitre le plus proche est multiplié par deux et celui du plus éloigné est divisé par deux.

De cette façon, les arbitres dont la distance avec le gymnase est faible ont plus de chance d'être sélectionnés par l'algorithme, sans pour autant enlever complètement la probabilité de sélection des arbitres dont la distance est plus élevée.

Une fois un arbitre sélectionné, celui-ci est ensuite retiré du tableau global des arbitres disponibles du jour pour ne plus être resélectionné pour d'autres matchs le même jour.

4.4.2 Matchs tournois

Le processus de sélection des matchs tournois se rapproche de celui des matchs uniques. La sélection reste semi aléatoire de façon à mettre l'accent sur les arbitres dont la distance avec le gymnase est plus faible.

Cependant, par souci d'optimisation, il fallait pouvoir désigner un seul arbitre par paquet de trois matchs de la même poule d'un tournoi. De cette façon, un arbitre se déplacerait pour arbitrer trois match consécutifs d'un tournoi.

J'ai pris en compte cette différence en ajoutant quelques ligne de code permettant de désigner le même arbitre pour trois match consécutif de la même poule.

4.5 Validation

Une particularité de ces désignations automatiques était de les rendre temporaires jusqu'à validation par un gestionnaire fédéral. Cette étape de validation devait alors se faire en deux temps : une étape pour procéder aux désignations temporaires, puis une étape de validation de ces dernières.

Pour faire face à ce problème, j'ai pensé à mettre en place une table temporaire identique à celles des matchs, qui permettrait d'afficher les désignations d'arbitres temporaires tout en permettant ensuite leur validation dans la table officielle.

En effet, notre page principale des désignations est basée sur la table des matchs, et la création d'une table identique permet alors de faciliter l'affichage des nouvelles désignations sans devoir reconfigurer les désignations manuelles.

La validation des désignations automatiques se fait donc en basculant alors les informations de cette table temporaire vers la table officielle des matchs.

4.6 Rapport

Une demande postérieure concernant cet algorithme a été de pouvoir suivre le procédé de désignation des arbitres avec un rapport complet des étapes. Grâce à la découpe des différentes fonctions de celui-ci, l'ajout de ce rapport a été rapide en ajoutant simplement l'écriture de descriptions durant chaque étape du processus.

```
public static function process()
{
    // ouverture du fichier de rapport
    $filePath = $rapportDir.date("dmYHi", strtotime("now")).'.txt';
    $filePath = str_replace('/', DIRECTORY_SEPARATOR, $filePath);

    $file = fopen($filePath, 'a+');

    // récupération des arbitres potentiels, désignation et insertion de ceux-ci

    fwrite($file, "Début de la récupération des arbitres potentiels...\n");
    $arbitres_potentiels = self::getArbitresPotentiels();

    fwrite($file, "Début du calcul des désignations...\n");
    $designations = self::processDesignations($arbitres_potentiels);

    fclose($file);

    foreach($designations as $matchID => $arbitres){
        $arbitre1 = $arbitres[0] ?? null;
        $arbitre2 = $arbitres[1] ?? null;

        self::insertDesignations($matchID, $arbitre1, 1);
        self::insertDesignations($matchID, $arbitre2, 2);
    }
}
```

FIGURE 26 – Methode appelée pour lancer l'algorithme - incorporation du rapport d'erreurs

5 Interface

Le dernier axe important de la transition numérique qui m’a été confiée, a été la refonte de l’interface de gestion des arbitres et y intégrant les nouveaux outils créés. Il m’a fallu moderniser cette interface en gardant la nouvelle identité du site mise en place par mon tuteur.

Au total, l’interface devait posséder trois pages distinctes pour chacune des fonctionnalités principales de cette partie du site. Celles-ci sont :

- L’interface principale pour les désignations des arbitres
- Une interface secondaire pour la gestion des habilitations.
- Une dernière interface pour la gestion des poules à afficher ou non

The figure displays four screenshots of the FFvolley web interface, illustrating the different pages for managing referees and matches.

Menu désignations: This page features the FFvolley logo and three main navigation buttons: "Affichage des poules" (blue), "Habilitations des arbitres" (yellow), and "Désignation des Arbitres" (red).

Gestion des habilitations des arbitres: This page shows a table of arbitrators. The table has columns for "Licence", "Nom", "Niveau", "Grade", and various skill categories (ARM, ARF, REM, REF, PMM, PNF, NBM, NBP, NZM, NZF, DEM, DEF). Each cell contains a status (e.g., "Non", "Oui") and a dropdown arrow.

Désignations des arbitres: This page shows a table of matches. The table has columns for "numero", "type", "date", "heure", "equipe 1", "equipe 2", "arbitre 1", "arbitre 2", and "gymnase". Each cell contains a status (e.g., "Non", "Oui") and a dropdown arrow.

Gestion des tables à afficher: This page shows a table of match schedules. The table has columns for "D", "nom", and "afficher". Each cell contains a status (e.g., "Non", "Oui") and a dropdown arrow.

FIGURE 27 – Les différentes pages de l’interface

Ces interfaces exclusivement composées de tableaux. Ceux-ci devaient faciliter l’expérience utilisateur en intégrant une pagination, un tri par colonne et une barre de recherche.

Heureusement, l’outil **Datatables** permettait de répondre à l’ensemble de ces contraintes. Cet outil basé sur JQuery permet d’intégrer plusieurs fonctionnalités à un tableau html dont la pagination, la barre de recherche et le tri en fonction des colonnes.

Pour la réalisation de ces pages, j'ai d'abord procédé au maquetage de l'application avec le logiciel dédié Adobe XD. Le but était de mettre en place un schéma visuel à suivre qui me permettrait de me concentrer uniquement sur l'implémentation du côté technique.

Une fois cette maquette réalisée et validée par mon tuteur, je me suis donc basé sur cette celle-ci pour réaliser mes pages.

Pour faciliter le côté responsive, et pour être en accord avec les outils déjà utilisés pour ce projet, j'ai utilisé la librairie Bootstrap pour le côté statique et JQuery pour le côté dynamique.

Faisant partie d'un intranet fermé au public, le référencement de ces pages n'était pas une priorité. J'ai cependant tout de même structuré ces pages en exploitant au mieux les balises html de chaque élément composant celles-ci, afin d'exploiter au moins le référencement naturel.

5.1 Liste des matches

L'interface principale devait intégrer la liste des matches complets à venir pour le reste de la saison. Elle devait également permettre le tri de ces matches en fonction des différentes poules possibles sans recharger la page.

J'ai mis en place un composant sous forme de classe permettant de gérer ces deux différents scénarios, et de renvoyer au choix le tableau de tous les matches à venir ou le tableau des matches à venir d'une poule. Pour fonctionner, cette classe prend en paramètre l'ID d'une poule et renvoi alors le tableau html composé de tous les matches à venir associés, récupérés depuis la base de données. Ce paramètre peut être nul et renvoi dans ce cas la liste complète des matches à venir sans distinction de poules.

```
public static function printMatches(?string $poule)
{
    // récupération du noms des tables et colonnes nécessaires
    $html = "<table>";

    // colonnes d'en tête
    $headers = ['numero', 'date', '...'];
    $html .= "<thead><tr>";
    foreach($headers as $header){
        $html .= "<th>$header</th>";
    }
    $html .= "</tr></thead>";

    // corps du tableau
    $html .= "<tbody>";
    foreach(self::getMatches($poule) as $match){
        // constitution d'une ligne pour chaque match
    }

    $html .= "</tbody>";

    $html .= "</table>";

    return $html;
}
```

FIGURE 28 – Méthode de construction de tableau pour une poule

De cette façon, je n'avais qu'à attacher un évènement au changement du SELECT affichant les poules, qui enverrait via AJAX la valeur de celle-ci à une page annexe. Cette page n'aurait alors qu'à renvoyer le bon tableau grâce à cette méthode.

AJAX (Asynchronous Javascript and XML) est une méthode permettant de faire des requêtes à un serveur web de façon asynchrone, et de récupérer la réponse de celui-ci sans recharger la page.

Pour faciliter l'envoi et la gestion de ces requêtes, j'ai utilisé la bibliothèque JQuery déjà très largement utilisée dans le cadre de cette transition numérique.

```
$("#poule").change(function () {  
    let form = $("#form");  
    let url = form.attr("action");  
  
    $.ajax({  
        type: "post",  
        url: url,  
        data: form.serialize(),  
        success: function (res) {  
            $("#response").html(res);  
        },  
    });  
});
```

FIGURE 29 – Fonction AJAX au changement d'une poule

5.2 Désignations

L'interface actuelle du site de la fédération rend la désignation des arbitres chronophage car celle-ci demande d'effectuer plusieurs clics afin de désigner les arbitres d'un match.

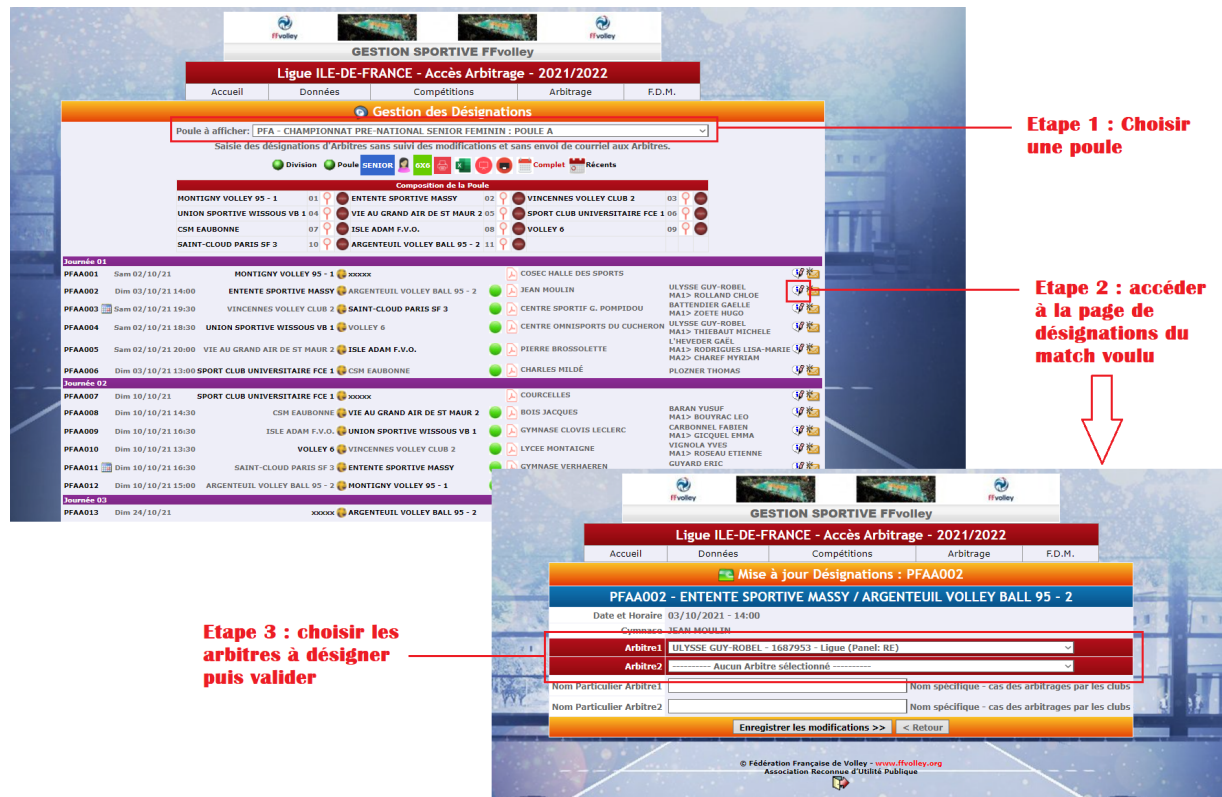


FIGURE 30 – Processus de désignation des arbitres actuel

La nouvelle interface devait également servir à lancer l'algorithme d'automatisation des désignations, mais également permettre de procéder aux désignations manuellement tout en offrant le maximum d'informations possibles concernant les arbitres à désigner, le tout sur la même page.

Les désignations manuelles devaient pouvoir se valider de façon dynamique via AJAX, tandis que celles proposées par l'algorithme devaient pouvoir être validées via un bouton.



Désignations des arbitres

Poule à afficher : Toutes les poules Afficher seulement les matchs d'une poule précise

Procéder aux désignations pour la période du au Calculer Envoyer Lancer les désignations automatiques

Show entries Search:

numero	type	date	heure	equipe 1	equipe 2	arbitre 1	arbitre 2	gymnase
AAF007	REG	18/06/2022	10:00	SAND SYSTEM ASSOCIATION 1	PUC VOLLEY-BALL	2040515 - AGESILAS VAEI - I	Désigner un arbitre manuellement	JULES LADOUMEGUE
AAF008	REG	18/06/2022	10:00	ASV - ARTS ET SPORTS A VILLEBON 1	VOLLEY-BALL LA ROCLETTE	2043473 - DUMONT ELSA - I		JULES LADOUMEGUE
AAF009	REG	18/06/2022		SAND SYSTEM ASSOCIATION 1	PUC VOLLEY-BALL			JULES LADOUMEGUE

FIGURE 31 – Centralisation des informations sur la nouvelle interface

5.2.1 Récupération des arbitres

Le calcul des disponibilités des arbitres pour chaque match étant trop long à effectuer au chargement de la page, celles-ci sont récupérées via AJAX quand l'utilisateur clique sur un SELECT.

FFvolley

Poule à afficher : Toutes les poules

Procéder aux désignations pour la période du au Calculer Envoyer

Show entries Search:

numero	type	date	heure	equipe 1	equipe 2	arbitre 1	arbitre 2	gymnase
ACF028	REG	18/06/2022		AS. SP DE SARTROUVILLE	ASV - ARTS ET SPORTS A VILLEBON 1	1577681 - MOURIER MARTIN		CARNOT
ACF029	REG	18/06/2022		VOLLEY-BALL LA ROCLETTE	VOLLEY-CLUB NOGENTAIS 2			
BAF007	REG	18/06/2022	10:00	CONFLANS-ANDRESY JOUY VB 1	VAIRES BEACH VOLLEY			
BAF008	REG	18/06/2022	10:00	ASV - ARTS ET SPORTS A VILLEBON 2	CONFLANS-ANDRESY JOUY VB 2			
BAF009	REG	18/06/2022		CONFLANS-ANDRESY JOUY VB 1	CONFLANS-ANDRESY JOUY VB 2			

Liste des informations des arbitres récupérées via AJAX

- 1592683 - TOUSSAINT THOMAS - Nat. - (Panel: NA) - 8.23 km
- 1521612 - TROOST MAIKEL - Nat. - (Panel: NA) - 33.33 km
- 1218989 - TRUONG KIM - Nat. - (Panel: NA) - 327.32 km
- 1687953 - ULYSSE GUY-ROBEL - Lig. - (Panel: RE) - 17.63 km
- 759498 - VIALLES JEAN-BERNARD - Nat. - (Panel: RE) - 36.43 km
- 333476 - VIGNOLA YVES - Lig. - (Panel: RE) - 14.54 km
- 1154224 - VILLESALMON ERIC - Lig. - (Panel: RE) - 14.91 km
- 1882642 - WAGUET ROMANE - Lig. - (Panel: NA) - 19.55 km
- 2047212 - WERY DÉBORA - Lig. - (Panel: RE) - 63.13 km
- 1151990 - YACK JOHN - Féd. - (Panel: C) - 12.91 km
- 1066407 - ZIANE NOUR-EDDINE - Féd. - (Panel: C) - 39.08 km
- 1818787 - ZURBACH KILIAN - Lig. - (Panel: RE) - 14.24 km
- 1971022 - AGOUTIN CORALIE - Dép. - 57.83 km
- 2116899 - AIGLON CLEMENT - Marq. - 21.38 km
- 2070249 - ANAT LIAM - Marq. - 26.47 km
- 2141964 - BALLOT BRIAN - Dép. - 7.94 km
- 2146215 - BELARBI LENAIS - Jeu. - 36.24 km
- 2415062 - BENMALEK SOFIANE - Jeu. - 28.35 km

FIGURE 32 – Nouveau processus de désignation des arbitres

Pour faire ça, j'ai encore une fois utilisé JQuery pour envoyer une requête AJAX à une page qui récupère et me renvoi les bonnes informations d'arbitres pour le match voulu.

```

function getDisposWithAjax(querySelector) {
    querySelector.one("click", function () {
        let select = this;
        let match = $(this).data("match");
        let selected = $(this).val();

        $.ajax({
            type: "post",
            url: "display_design.php",
            data: {
                match: match,
                selected: selected,
            },
            beforeSend: function () {
                $("#loader").addClass("is-active");
            },
            complete: function () {
                $("#loader").removeClass("is-active");
            },
            success: function (res) {
                $(select).append(res);
            },
        });
    });
}

```

FIGURE 33 – Fonction AJAX pour récupérer les arbitres

Cette façon de procéder a permis au chargement de la page de passer d'environ 180 secondes à moins de 2 secondes. Pour obtenir ces informations de performance, j'ai simplement affiché la différence en microsecondes entre la fin de mon script et le début.

```

$debug_start=microtime(true);
// le code à performer
var_dump(microtime(true) - $debut_start);

```

Au changement de sélection d'un arbitre pour le match, celui-ci est ensuite automatiquement intégré à la base de données grâce à l'envoi de ces informations via AJAX à une page annexe. Ceci permet alors d'éviter la perte de données à un oubli.

5.2.2 Désignations automatiques

Pour rappel, l'algorithme de désignations automatique est construit de façon à prendre deux paramètres nullables qui correspondent aux date de la période pour laquelle effectuer ces désignations.

L'intégration de l'algorithme dans mon interface se fait donc simplement en envoyant ces valeurs via AJAX à une page annexe qui s'occupe de lancer l'algorithme et de renvoyer un tableau des matchs de la période avec les nouvelles désignations. Ce tableau écrase ensuite celui déjà en place sur la page, et permet à son tour grâce aux fonctions Javascript de procéder aux désignations manuelles.

Les désignations automatiques doivent ensuite être validées grâce à un bouton, afin de les envoyer à une page qui s'occupe de les enregistrer dans la base de données.

5.3 Sécurité

Ce projet n'est qu'une branche d'une transition numérique plus importante, et est destiné à un public privé. Il n'a pour but d'être accessible que par les gestionnaires fédéraux grâce à des identifiants privés attribués par la fédération.

Le principal aspect de la sécurité pris en compte concerne les requêtes SQL car la base de données utilisée pour ce projet n'est pas directement liée à celle de la fédération. Ceci a été fait dans le cas d'un futur changement dans la récupération des paramètres pour effectuer ces requêtes. En effet, ces paramètres sont pour l'instant récupérés en interne mais pourraient potentiellement être récupérés par des formulaires.

De plus, les différentes pages permettant le renvoi d'informations via AJAX nettoient les données reçues grâce à la fonction `htmlspecialchars()` pour éviter l'exploitation de la faille XSS. Cette faille permet à un attaquant d'envoyer du code malveillant sur un site, ce qui est contré par cette fonction car celle-ci transforme les balises html en entités correspondantes.

Finalement, aucun formulaire n'est intégré dans l'interface. Dans le cas contraire, il aurait fallut mettre en place un système de token pour éviter une exploitation malveillante de la faille CSRF. Ce système permet en effet de confirmer l'identité de la personne envoyant la requête.

6 Documentation

Afin de faciliter la reprise de cette partie par mon tuteur et mes successeurs, j'ai également écrit et mis en ligne une documentation complète. Celle-ci est écrite en Markdown (.md) et joue le rôle d'un cahier des spécifications techniques, permettant ainsi la compréhension du code en profondeur.

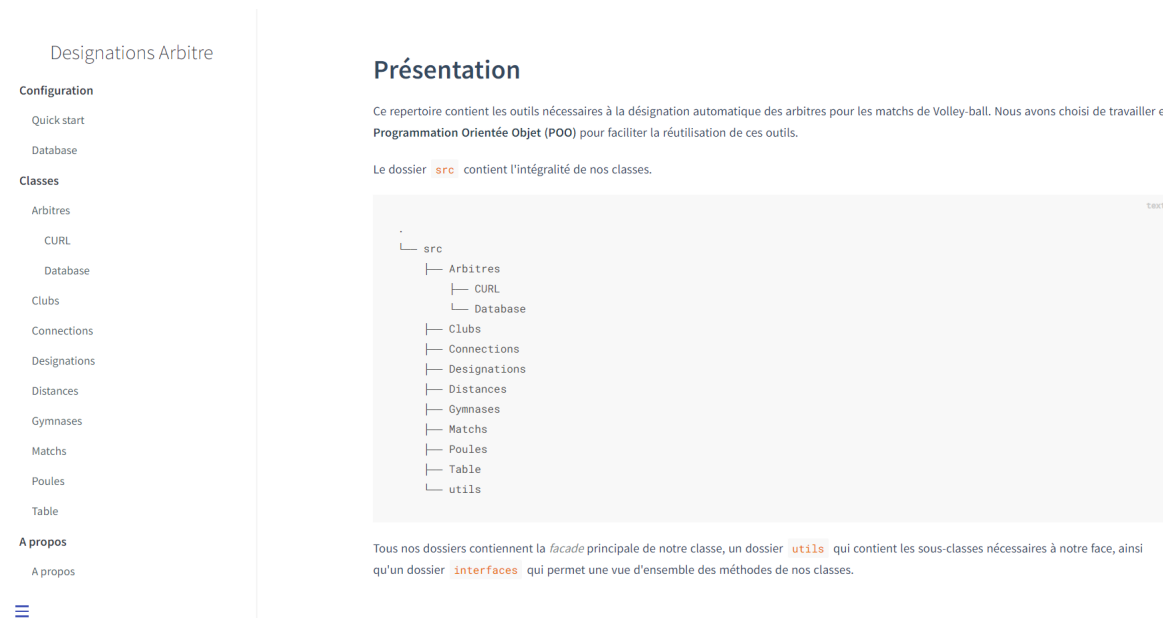


FIGURE 34 – Documentation en ligne pour le projet

La mise en page est gérée par **Docsify**, une bibliothèque Javascript qui permet de convertir les fichiers markdown en pages HTML. Celle-ci permet ainsi de faciliter le rendu de la documentation, ce qui permet de se concentrer sur son contenu.

Pour la mise en ligne, j'ai opté pour le service gratuit de Google Firebase afin d'avoir une première interaction avec cet outil.

7 Conclusion

Même si plusieurs points restent à améliorer, je suis satisfait de ce projet car toutes les demandes ont pu être mises en place. Celui-ci m'a permis de mettre en pratique les connaissances théoriques que j'avais pu acquérir en formation, mais également de prendre conscience d'aspects plus fonctionnels d'une application.

Devoir travailler sur une partie d'un projet plus grand m'a permis de me rendre compte de l'importance de la communication afin de mettre en place un plan de départ optimal, de l'importance de bien documenter son travail, ainsi que des avantages de découper son code pour une meilleure gestion du temps.

Il n'est pas toujours possible de savoir dans quelles conditions seront utilisés les outils que l'on met en place. Il faut donc pouvoir prendre en compte tous les aspects de ceux-ci