LES PRINCIPES DE LA CRÉATION DU SITE WEB M2L

==========

I. La création d'une API REST

1. Qu'est-ce qu'une API REST

L'acronyme **API** signifie "**Application Programming Interface**" en Anglais et peut être traduit simplement par "**Interface de Programmation d'Application**".

Pour rester simple, il s'agit de **créer un accès à une application** et permettre de venir y **consommer des données ou des fonctionnalités**. Ainsi, vous pouvez, avec une autre application ou un site, venir interroger l'application et utiliser ses données.
L'acronyme REST, quant à lui, signifie "**REpresentative State Transfer**".

Nous allons traiter ici les bases de la création d'une API REST, en utilisant l'exemple de l'accès à une base de données de l'ensemble des objets concernant notre projet M2L.

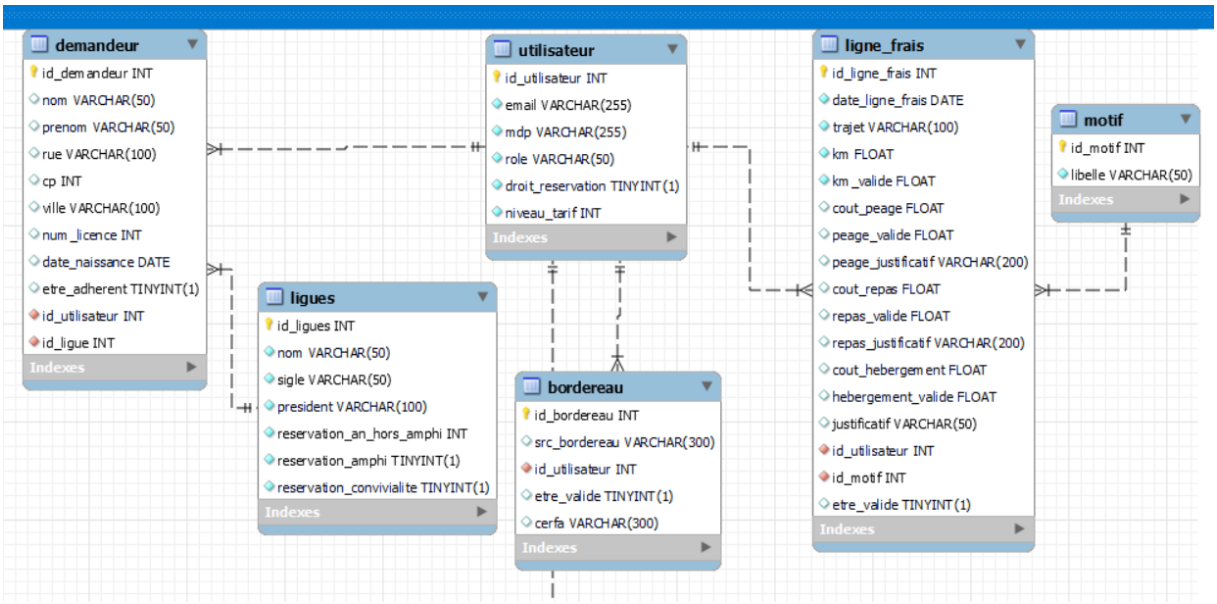
Lors de la création de notre API, nous devons garder en tête les critères qui en font une API REST. Une API REST se doit d'être :

- ✓ **Sans état** : le serveur ne fait aucune relation entre les différents appels d'un même client. Il ne connaît pas l'état du client entre ces transactions
- ✓ **"Cacheable"** : le client doit être capable de garder nos données en cache pour optimiser les transactions
- ✓ **Orienté client-serveur** : Il nous faut une architecture client-serveur
- ✓ **Avec une interface uniforme** : ceci permet à tout composant qui comprend le protocole HTTP de communiquer avec votre API
- ✓ **Avec un système de couches** : avec des serveurs intermédiaires, le client final ne doit pas savoir s'il est connecté au serveur principal ou à un serveur intermédiaire.

2. La base de données

Pour notre API, nous allons utiliser une base de données contenant les tables : utilisateur, motif, ligues, ligne_frais, bordereau.

Sa modélisation est la suivante :



3. Les étapes principales

a. Création de la connexion

Dans notre projet, nous avons créé un dossier « config » et nous y avons ajouter un fichier qui s'appelle « database.php » qui contient une classe « Database » : il nous permet d'effectuer la connexion.

```php
<?php
class Database{

    // specify your own database credentials
    private $host = "localhost";
    private $db_name = "projet_db";
    private $username = "root";
    private $password = "";
    private $port = "3306";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;

        try{
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name. ";port=".$this->port,
            $this->conn->exec("set names utf8");
        }catch(PDOException $exception){
            echo "Echec de connexion: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>
```

b. Création des modèles (objets)

Une fois la base de données créée et la connexion fonctionnelle, nous allons créer nos modèles permettant d'effectuer les différentes tâches associées.

Pour ce faire, nous avons créé un dossier « objects » (*M2L/api/objects*) et nous y avons créer un fichier pour chacune de nos tables.

Par exemple, le fichier « motif.php » contiendra la classe "Motif" qui aura comme propriétés les différents champs de la table "motif" et la connexion à la base de données.

```php
<?php
// object
class Motif {

    // database connection and table name
    private $conn;
    private $table_name = "motif";

    // object properties
    public $id;
    public $libelle;


    // constructor
    public function __construct($db){
        $this->conn = $db;
    }
```

c. Création des méthodes nécessaires dans chaque objet (modèle)

Nous allons maintenant ajouter les différentes méthodes CRUD (Create, Read, Update, Delete) qui nous seront nécessaires pour interagir avec la base de données. Et aussi les autres méthodes qui soutiennent la manipulation de l'utilisateur.

Create

La méthode "Create" va nous servir à créer un motif. Nous allons l'appeler "create" mais elle peut prendre le nom qui vous convient.

Cette méthode utilisera les propriétés du « motif » qui auront préalablement été définies et les écrira en base de données.

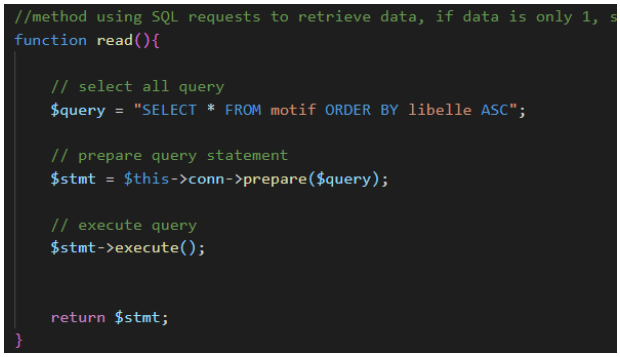
```php
// create new motif record
function create(){

    // insert query
    $query = "INSERT INTO motif
            SET
                libelle = :libelle";

    // prepare the query
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->libelle=htmlspecialchars(strip_tags($this->libelle));


    // bind the values
    $stmt->bindParam(':libelle', $this->libelle);


    // execute the query, also check if query was successful
    if($stmt->execute()){
        return true;
    }

    return false;
}
```

Read

La méthode "Read" va nous servir à lire dans la table "motif". Nous allons en préparer 2, l'une pour lire la liste entière, l'autre pour lire un seul motif.

Ces méthodes utiliseront les informations transmises dans l'instance du motif, si nécessaire :

- Charger la liste de tous les motifs



```php
//method using SQL requests to retrieve data, if data is only 1, s
function read(){

    // select all query
    $query = "SELECT * FROM motif ORDER BY libelle ASC";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // execute query
    $stmt->execute();


    return $stmt;
}
```

- Charger un seul motif

```php
function readOne(){

    // query to read single record
    $query = "SELECT * FROM motif WHERE id_motif = :id LIMIT 0,1";

    // prepare query statement
    $stmt = $this->conn->prepare( $query );

    // execute query
    $stmt->execute(array(':id'=>$this->id));

    // get retrieved row
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    if(!$row)
    {
        return false;
    }else{

    // set values to object properties
    $this->id = $row['id_motif'];
    $this->libelle = $row['libelle'];
    return true;
    }

}
```

Update

La méthode "Update" va nous servir à modifier un motif.

Cette méthode utilisera les propriétés du motif qui auront préalablement été définies et les modifiera en base de données.

```php
// update a motif record
public function update(){

    // query
    $query = "UPDATE motif
            SET
                libelle = :libelle
            WHERE id_motif = :id";

    // prepare the query
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->libelle=htmlspecialchars(strip_tags($this->libelle));


    // bind the values from the form
    $stmt->bindParam(':libelle', $this->libelle);

    // unique ID of record to be edited
    $stmt->bindParam(':id', $this->id);

    // execute the query
    if($stmt->execute()){
        return true;
    }

    return false;
}
```

Delete

La méthode "Delete" va nous servir à supprimer un motif. Cette méthode utilisera l'id de l'objet pour interagir.

```php
// delete motif
function delete(){

    // delete query
    $query = "DELETE FROM motif WHERE id_motif = ?";

    // prepare query
    $stmt = $this->conn->prepare($query);

    // sanitize
    $this->id=htmlspecialchars(strip_tags($this->id));

    // bind id of record to delete
    $stmt->bindParam(1, $this->id);

    // execute query
    if($stmt->execute()){
        return true;
    }

    return false;
}
// search motif
```

Search

Cette méthode nous permet de chercher des motifs relatifs aux mots clés saisis dans la barre de recherche. Nous allons l'appeler « search ».

```php
// search motif
function search($keywords){

    // select all query using LIKE which match case-insensitive by default, do LIKE BINARY for
    $query = "SELECT * FROM
                " . $this->table_name . "
            WHERE
                libelle LIKE ?
            ORDER BY
                libelle ASC";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // sanitize
    $keywords=htmlspecialchars(strip_tags($keywords));
    $keywords = "%{$keywords}%"; // search for all data contains keywords in ANY POSITION! */

    // bind
    $stmt->bindParam(1, $keywords);

    // execute query
    $stmt->execute();

    return  $stmt;
}
```

ReadPaging

Cela va nous permettre d'afficher la liste des objets plus lisible en la divisant en plusieurs pages numérotées de 5 objets au max par page.

```php
// read resultat with pagination
public function readPaging($from_record_num, $records_per_page){

    // select query
    $query = "SELECT * FROM motif ORDER BY libelle ASC LIMIT ?,?";
    // prepare query statement
    $stmt = $this->conn->prepare( $query );

    // bind variable values
    $stmt->bindParam(1, $from_record_num, PDO::PARAM_INT);
    $stmt->bindParam(2, $records_per_page, PDO::PARAM_INT);

    // execute query
    $stmt->execute();

    // return values from database
    return $stmt;
}
```

Count

Elle compte le nombre total des lignes d'objet de la liste et aide la méthode « readPaging » à afficher la liste.

```php
// used for paging
public function count(){
   $query = "SELECT COUNT(*) as total_rows FROM " . $this->table_name . "";

   $stmt = $this->conn->prepare( $query );
   $stmt->execute();
   $row = $stmt->fetch(PDO::FETCH_ASSOC);

   return $row['total_rows'];
}
```

d. Création de l'API

Lors de la création des fichiers de notre API, nous allons devoir prendre en compte une contrainte REST concernant la méthode utilisée pour effectuer nos différentes requêtes HTTP.

En effet, le standard est très strict. Il indique le rôle exact de chaque méthode HTTP dans notre API. Ainsi :

GET sera exclusivement utilisé pour lire des données

POST servira uniquement à créer des données

PUT servira uniquement à mettre à jour des données

DELETE sera réservé à la suppression de données

Nous devrons donc toujours indiquer quelle méthode sera utilisée en entête et vérifier que celle-ci est effectivement utilisée.

Lire des données :

Commençons par la lecture des données. Nous allons tout d'abord créer le fichier permettant d'obtenir la liste complète des motifs.

Nous appellerons ce fichier « read.php », il se trouvera dans le dossier "motif" (*M2L/api/motif*), comme tous les fichiers concernant les motifs.

Tout d'abord, en début du fichier, nous définirons les entêtes HTTP nécessaires au bon fonctionnement de l'API.

```php
// required header
header("Access-Control-Allow-Origin: http://localhost/M2L/");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Headers: access");
header("Access-Control-Allow-Methods: GET");
header("Access-Control-Allow-Credentials: true");
```

Une fois la méthode vérifiée, nous allons accéder aux données en instanciant la base de données et l'objet "Motif". La classe « Utilities » est une classe créée dans le fichier « utilities.php » (*M2L/api/shared/*) qui contient la méthode « getPaging » pour calculer le nombre total des pages d'une liste des objets. Il nous rend un tableau contenant : les numéros de page (1,2,3,…), l'url de la page, la page actuelle, la dernière page de la liste afin que l'utilisateur puisse les voir en cliquant sur le numéro de la page.

```php
// include database and object files
include_once '../config/core.php';
include_once '../shared/utilities.php';
include_once '../config/database.php';
include_once '../objects/motif.php';

// utilities
$utilities = new Utilities();

// instantiate database and motif object
$database = new Database();
$db = $database->getConnection();

// initialize object
$motif = new Motif($db);
```

Récupération des données et envoi :

L'accès aux données étant créé, nous allons appeler la méthode permettant de lire la liste des motifs en les divisant en pages, puis créer le tableau json à envoyer.

```php
// query motifs
$stmt = $motif->readPaging($from_record_num, $records_per_page);
$num = $stmt->rowCount();

// check if more than 0 record found
if($num>0){

    // declare motifs array as empty
    $motifs_arr=array();
    $motifs_arr["records"]=array();
    $motifs_arr["paging"]=array();

    // retrieve our table contents
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){

        // extract row : this will make $row['name'] to just $name only
        extract($row);

        $motif_item=array(
            "id" => $id_motif,
            "libelle" => $libelle
        );

        array_push($motifs_arr["records"], $motif_item);
    }


    // include paging
    $total_rows=$motif->count();
    $page_url="{$home_url}motif/read_paging_motifs.php?";
    $paging=$utilities->getPaging($page, $total_rows, $records_per_page, $page_url);
    $motifs_arr["paging"]=$paging;

    // set response code - 200 OK
    http_response_code(200);

    // make it json format
    echo json_encode($motifs_arr);
}
```

En cas d'échec :

La réponse 404 s'affichera dans la console et enverra à l'utilisateur un tableau json qui contient un message.

```php
else{

    // set response code - 404 Not found
    http_response_code(404);

    // tell the motif motifs does not exist
    echo json_encode(
        array("message" => "Aucun motif trouvé.")
    );
}
```

Lire une seule donnée :

Sur le même modèle nous allons créer un fichier permettant de lire la table "motif" mais qui ne retournera qu'un seul motif en fonction de son id.

Les entêtes, la vérification de la méthode et l'accès aux données seront strictement identiques. La différence se fera sur le fait que nous devrons recevoir l'id de l'élément à lire, celui-ci étant transmis en json.

Traiter les données reçues :

Nous allons recevoir des données en json, il sera nécessaire de les traiter afin de décider si nous allons plus loin dans le traitement ou si nous stoppons. Pour récupérer les données reçues, nous allons devoir utiliser la méthode php "file_get_contents", puis attribuer cette donnée (ou ces données) aux attributs de l'objet.

```php
// get motif id _ use POST method to inject into body of request
$data = json_decode(file_get_contents("php://input"));

// set motif id to be deleted
$motif->id = $data->id;
```

Pour cela, nous pouvons utiliser la variable super globale $_GET pour récupérer la donnée (les données) envoyée(s) par la requête GET HTTP définie dans le fichier js.

```php
// set ID property of record to read
$motif->id = isset($_GET['id']) ? $_GET['id'] : die();
```

L'accès aux données étant créé et l'id étant existant, nous allons appeler la méthode permettant de lire un motif puis créer le tableau json à envoyer.

```php
// read the details of motif to be edited
$result=$motif->readOne();

if($result){
    // create array
    $motif_arr = array(
        "id" => $motif->id,
        "libelle" => $motif->libelle
    );

    // set response code - 200 OK
    http_response_code(200);

    // make it json format
    echo json_encode($motif_arr);
}

else{
    // set response code - 404 Not found
    http_response_code(404);

    // tell client that the motif does not exist
    echo json_encode(array("message" => "Le motif n'existe pas."));
}
?>
```

Créer un élément :

Une API REST peut également nous permettre de créer des éléments. Dans ce cas, la méthode autorisée est la méthode "POST".

Le fonctionnement global sera identique, à la différence près que nous allons recevoir les données détaillées de l'élément à créer et que nous devrons les traiter.

Les entêtes :

Nous définirons les entêtes HTTP nécessaires au bon fonctionnement de l'API, dont l'une sera légèrement différente : celle concernant la méthode.

```php
<?php
// required headers
header("Access-Control-Allow-Origin: http://localhost/M2L/");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
```

Accès aux données :

Une fois la méthode vérifiée, nous allons instancier la base de données et l'objet "Motif".

```php
// files needed to connect to database
include_once '../config/database.php';
include_once '../objects/motif.php';

// get database connection
$database = new Database();
$db = $database->getConnection();

// instantiate object
$motif = new Motif($db);
```

Traiter les données reçues :

Pour récupérer les données reçues, nous allons devoir utiliser de nouveau la méthode php "file_get_contents", puis définir la valeur (ou les valeurs) de chaque propriété de l'objet. Enfin nous allons vérifier si les données ne sont pas vides et si la méthode de l'objet est bien effectuée avec succès :

```php
// get posted data
//file_get_contents() to read the contents of a file into a string and json_deco
$data = json_decode(file_get_contents("php://input"));

// set motif property values
$motif->libelle = $data->libelle;

// create the motif
if(
    !empty($motif->libelle) &&
    $motif->create()
){

    // set response code
    http_response_code(200);

    // display message: motif was created
    echo json_encode(array("message" => "Le motif a été créé."));
}
```

Sinon, une réponse s'affichera dans la console avec un message de json :

```php
// message if unable to create motif
else{

    // set response code
    http_response_code(400);

    // display message: unable to create motif
    echo json_encode(array("message" => "Impossible de créer le motif ."));
}
?>
```

Supprimer un élément :

Dans ce cas, la méthode utilisée est la méthode "POST" car, en théorie, vous pouvez utiliser différentes méthodes POST, GET, DELETE, PUT pour réaliser différents actions, mais, en

pratique, la plupart des navigateurs ne prennent en charge que HTTP GET et POST, donc, nous allons utiliser la méthode « POST » pour supprimer un élément.

```php
// required headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-

// include database and object file
include_once '../config/database.php';
include_once '../objects/motif.php';

// get database connection
$database = new Database();
$db = $database->getConnection();

// prepare motif object
$motif = new Motif($db);

// get motif id _ use POST method to inject into body of request
$data = json_decode(file_get_contents("php://input"));

// set motif id to be deleted
$motif->id = $data->id;

// delete the motif
if($motif->delete()){

    // set response code - 200 ok
    http_response_code(200);

    // tell client
    echo json_encode(array("message" => "Le motif a été supprimé ."));
}
```

Le processus est toujours le même : les entêtes, la création des objets, la récupération des données, l'attribution de ces données aux propriétés de l'objet, l'utilisation de la méthode delete() propre à l'objet et la réponse si succès. Sinon :

```php
// if unable to delete the motif
else{

    // set response code - 503 service unavailable
    http_response_code(503);

    // tell the motif
    echo json_encode(array("message" => "Impossible de supprimer le motif."));
}
?>
```

Modifier un élément :

C'est pareil pour la modification d'un élément en utilisant la méthode « POST » et la méthode propre à l'objet : update().

```php
// required headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Header

// files needed to connect to database
include_once '../config/database.php';
include_once '../objects/motif.php';

// get database connection
$database = new Database();
$db = $database->getConnection();

// instantiate motif object with empty property values
$motif = new Motif($db);

// get input data posted, turn it into JSON string then convert it into PHP var
$data = json_decode(file_get_contents("php://input"));


// set motif property values according to posted data
$motif->id = $data->id;
$motif->libelle = $data->libelle;


// update the motif record
if($motif->update()){

    // set response code
    http_response_code(200);

    // response in json format
    echo json_encode(array("message" => "Le motif a été mis à jour."));
}
```

II. Création de l'interface d'utilisateur

Le corp (body) de notre page « index.html » est divisé en 2 parties principales :

La barre de navigation :

Elle contient toutes les options possibles pour que l'utilisateur puisse interagir avec la base de données selon son rôle. À défaut, elle n'affiche que 2 options : « Accueil » et « S'inscrire ». Les autres (« se déconnecter » et « Update compte ») sont cachées par défaut, elles apparaissent quand c'est indiqué dans le contexte.

D'ailleurs, il existe aussi un vide avec l'id « rôle-menu » qui affichera toutes les autres options appartenant à l'utilisateur.

```html
<div class="collapse navbar-collapse" id="navbarNavAltMarkup">
    <div class="navbar-nav"id="float_left">
        <a class="nav-item nav-link" href="#" id='home'>Accueil</a>
        <a style="display:none;" class="nav-item nav-link" href="#" id='update_account'>Update Compte</a>
        <!--add menu according to the role of user-->
        <div class="navbar-nav" id="role-menu"></div>

        <a class="nav-item nav-link" href="#" id='logout'>Se déconnecter</a>
        <!--<a class="nav-item nav-link" href="#" id='login'>Se connecter</a>-->
        <a class="nav-item nav-link" href="#" id='sign_up'>S'inscrire</a>

    </div>
    <div class="navbar-nav" id="float_right">
        <a class="nav-item nav-link" href="#"><i class="bi bi-person-circle"></i></a>
        <a class="nav-item nav-link" href="#" id='nom_user'></a>
    </div>
</div>
```

Les options selon le rôle :

```
// if the user is logged in
function showLoggedInMenu(role){
    // hide login and sign up from navbar & show logout button
    $("#sign_up").hide();//#login,
    $("#logout").show();
    $('#update_account').show();

    // according to the role to add menu to navbar_ can add more functions for reservation
    const menuAdmin = ` <a class="nav-item nav-link" href="#" id='gerer-user'>Utilisateurs</a>
                        <a class="nav-item nav-link" href="#" id='gerer-demandeur'>Demandeurs</a>
                        <a class="nav-item nav-link" href="#" id='gerer-ligue'>Ligues</a>
                        <a class="nav-item nav-link" href="#" id='gerer-bordereau'>Bordereaux</a>
                        <a class="nav-item nav-link" href="#" id='gerer-ligne-frais'>Lignes Frais</a>
                        <a class="nav-item nav-link" href="#" id='gerer-motif'>Motifs</a>
                        `;

    const menuUser = `<a class="d-none nav-item nav-link" href="#" id='demander-adhesion'>Demander d'adhésion</a>
                      <a class="d-none nav-item nav-link" href="#" id='gerer-demande'>Gérer la Demande</a>`;

    const menuTresor = `<a class="nav-item nav-link" href="#" id='consulter-bordereau'>Consulter Bordereaux</a>
                        <a class="nav-item nav-link" href="#" id='gerer-ligne-frais'>Gérer Lignes Frais</a>`;

    const menuAdherent = `<a class="nav-item nav-link" href="#" id='gerer-ligne'>Gérer Lignes Frais</a>`;

    switch (role) {
        case 'admin':
            $('#role-menu').append(menuAdmin);
            break;
        case 'user':
            $('#role-menu').append(menuUser);
            break;
        case 'tresorier':
            $('#role-menu').append(menuTresor);
            break;
        case 'adherent':
            $('#role-menu').append(menuAdherent);
            break;
    }
```

Le contenu de la page :

Le contenu du site s'affichera en 3 parties comme dans la photo :

```
<!-- where prompt / messages will appear -->
<div id="response"></div>

<h2 id='page-title'></h2>
<!-- where main content will appear -->
<div id="content">
```

« **response** » : pour afficher le message correspondant à chaque action de l'utilisateur. Par défaut, il est vide.

« **page-title »** : il change selon le titre du contenu à afficher. Par défaut, il est vide.

« **content** » : il contient le résultat retourné par la base de données via l'API. Par défaut, il affiche le contenu de la page d'accueil du site de Maison des ligues de Lorraine.

III. Création des fonctions en Javascript pour interagir avec chaque action de l'utilisateur (onClick) en utilisant l'API :

Chaque action de l'utilisateur est définie par un clic sur un bouton, une image ou une (un élément HTML) de notre page HTML. Elles sont toutes stockées dans le fichier « page_principale.js » (*M2L/assets/js/*). Chaque élément cliquable est défini par un « id » ou une « class » qui fonctionne selon les instructions qu'il contient.

Par exemple :

```
// show sign up / registration form
$(document).on('click', '#sign_up', function(){
    changePageTitle("S'inscrire");
    clearResponse();
    showSignUpForm();
});
```

En cliquant sur le bouton défini par l'id « sign_up », le système effectuera 3 fonctions :

- changePageTitle (« S'inscrire ») : il va appeler la fonction changePageTitle(), définie dans « M2L/assets/js/for_page/page.js », qui permet d'ajouter le texte « S'inscrire » dans l'élément HTML <h2> avec l'id « page-title » de la page principale de notre site, et en même temps, il change aussi le tag <tittle> dans <head> de notre site en « M2L – S'inscrire »

```
// change page title
function changePageTitle(page_title){

    // change page title
    $('#page-title').text(page_title);

    // change title tag
    document.title=`M2L - ${page_title}`;
}
```

Toutes les fonctions écrites en langage javascript qui supportent la page_principale.js sont classées en différents dossiers stockés dans M2L/assets/js/ (à découvrir !) selon l'objet qu'il sert.

En principe, chaque clic de l'utilisateur générera un morceau de HTML qui traitera les données extraites de notre base de données et les affichera comme le contenu de notre page « index.html » ou comme une réponse en cas d'échec.

Pour interagir avec notre API, on utilise les méthodes Ajax de jQuery :

- $.getJSON() est utilisée pour obtenir des données JSON à l'aide d'une requête AJAX HTTP GET. Les données envoyées au serveur sont ajoutées à l'URL sous forme de chaîne de requête. Si la valeur du paramètre data est un objet brut, elle est convertie en chaîne et encodée en URL avant d'être ajoutée à l'URL.

Il s'agit d'une fonction Ajax abrégée, qui équivaut à :

```
$.ajax({
  dataType: "json",
  url: url,
  data: data,
  success: success
});
```

- ☛ $.post() est utilisée pour envoyer des données au serveur à l'aide d'une requête HTTP POST. En principe, si la valeur du paramètre data est un objet brut, elle est convertie en chaîne et encodée en URL avant d'être ajoutée à l'URL.

Il s'agit d'une fonction Ajax abrégée, qui équivaut à :

```
$.ajax({
  type: "POST",
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```