

# Sommaire

I - Htaccess	page 2
II - Index.php	page 3
III - Contrôleurs	page 4
Détails BkofficeController	page 5
Détails UnlogController	page 5
Détails DatasController	page 6
IV - Modèles	page 7
Détails Database	page 7
Détails LoginUser	page 8
Détails SendMail	page 9
Détails Datas	page 10
V - Vues	page 11
VI - Js/Jquery/Ajax	page 13
Détails bddFunctions.js	page 13
Détails checkForm.js	page 14
Détails isIntoView.js	page 17
Détails main.js	page 17
Détails sendMail.js	page 18
Détails smoothScroll.js	page 18
Détails storageFunctions.js	page 19
Détails utilities.js	page 20
VII - Base de données	page 22

# I - Htaccess

Le htaccess reformate les urls via RewriteRule pour une utilisation plus simple du site.

Exemple : **site.com/contact/** donnera **site.com/?page=contact**

Cela permet notamment de récupérer des informations et de les orienter plus rapidement avec le routeur index.php.

Veillez à bien supprimer le nom du htaccess une fois sur le serveur.

« **deletename.htaccess** » devient « **. htaccess** ».

Caractères	Signification
#	commentaire htaccess
^	début
\$	fin
(.*)	n'importe quel terme répété autant de fois que nécessaire
\$1, \$2, \$3	variable
[L]	exit/break
[a-zA-Z0-9\~]	toutes lettres minuscules et majuscules de a à z et nombre de 0 à 9 et les tirets

```
1 RewriteEngine On
2 RewriteBase /
3 #^= début //$ = fin#
4 #(.*) = $1 // . veut dire n'importe quoi et * autant de fois que nécessaire#
5 #[L] = "exit/break"#
6 RewriteRule ^css/(.*)$ css/$1 [L]
7 RewriteRule ^js/(.*)$ js/$1 [L]
8 RewriteRule ^img/(.*)$ img/$1 [L]
9 #url type parametre1/param2/param3/ #
10 #[a-zA-Z0-9\~] = toutes lettres minuscule et majuscule de a à z et nombre de 0 a 9 et les - (tirets)#
11 RewriteRule ^([a-zA-Z0-9\~]*)/([a-zA-Z0-9\~]*)/([a-zA-Z0-9\~]*)/$ index.php?c=$1&a=$2&p=$3 [L]
12 #url type parametre1/param2/ #
13 RewriteRule ^([a-zA-Z0-9\~]*)/([a-zA-Z0-9\~]*)/$ index.php?c=$1&a=$2 [L]
14 #url type parametre1/ #
15 RewriteRule ^([a-zA-Z0-9\~]*)/$ index.php?page=$1 [L]
16 #url defaut / #
17 RewriteRule ^(.*)/$ index.php?p=$1 [L]
18 RewriteRule ^(.*)$ $1 [L]
```

## II – Index.php

Index.php est le routeur qui va charger et utiliser les classes php ainsi que les vues html.

### L'index en détail :

Cette ligne démarre la session peut importe où l'on se trouve sur le site.

```
session_start();
```

Ici l'affichage d'erreur php est activé.

Pour les erreurs ajax, utilisez la console du navigateur, dans 'network/réseau' sélectionnez votre requête puis dans XHR sélectionnez 'response/réponse'.

```
ini_set('display_errors',1);
ini_set('display_startup_errors',1);
error_reporting(-1);
```

Définit l'horaire local et la localisation pour toutes les fonctions

```
date_default_timezone_set('Europe/Paris');
setlocale (LC_TIME, 'fr_FR.utf8','fra');
```

Chargement automatique des classes contrôleurs et modèles qui sont appelés via l'url

```
spl_autoload_register(function ($className) {
    if(is_file('controllers/'.ucfirst($className).'.class.php')){
        require_once('controllers/'.ucfirst($className).'.class.php');
    }
    else if(is_file('models/'.lcfirst($className).'.class.php')){
        require_once('models/'.lcfirst($className).'.class.php');
    }
});
```

Si aucune page n'est appelée, la variable **\$page** est la page par défaut

```
$page = $_GET['page']??'home';
```

Le switch oriente vers le contrôleur correspondant et déclenche la méthode souhaitée.

Si aucune page existante n'est trouvée, le routeur renvoi vers la page **erreur** par défaut.

```
switch($page) {
    case 'home':
        return (new HomeController())->ShowPage($page);
        break;
    case 'page1':
        return (new Page1Controller())->ShowPage($page);
        break;
    case 'bkoffice':
        return (new BkofficeController())->ShowPage($page,$_POST);
        break;
    case 'datas':
        return new DatasController($_POST);
        break;
    case 'unlog':
        return (new UnLogController())->UnlogUser();
        break;
    default:
        http_response_code(404);
        return (new ErrorController())->ShowPage();
}
```

## III – Contrôleurs

Les contrôleurs sont indépendants et dédiés à leurs vues ; ils peuvent utiliser n'importe quelles méthodes de n'importe quels modèles.

Le nom d'un contrôleur se compose d'un préfixe suivi de **'Controller.class.php'**.

Le préfixe de chaque contrôleur correspond à la page qui est appelée dans l'url.

Exemple : **'site.com/contact/'** va charger le contrôleur **'ContactController.class.php'**.

Le nom de chaque class dans les contrôleurs possède le même préfixe suivi de **'Controller'**.

Tous les contrôleurs ont une méthode par défaut appelée par le routeur.

La méthode **'ShowPage()'** est présente dans tous les contrôleurs qui appellent des vues, tels que **BkOffice**, **Error** et **Home**.

Les pages créées ont donc besoins de leurs contrôleurs pour être chargées.

Exemple : **'page1.html'** est appelée par la méthode **'ShowPage()'** du contrôleur **'Page1Controller.class.php'** lui-même appelé par l'url **'site.com/page1/'**.

En détails :

```
2
3  class HomeController{
4      public function ShowPage($getPage){
5          $page = $getPage;
6          $pageTitle = ucfirst($getPage);
7          $pageDescription = "Description de la page d'accueil";
8          require_once 'views/layout.html';
9      }
10 }
```

L'argument **\$getPage** est la variable **\$page** envoyée par le routeur :

```
case 'home':
    return (new HomeController())->ShowPage($page);
break;
```

Les variables **\$page**, **\$pageTitle** et **\$pageDescription** sont des variables utilisées par les vues.

**\$page** permet de récupérer le contenu de votre page.

**\$pageTitle** permet de définir un titre à votre page (onglet navigateur).

**\$pageDescription** permet de définir la description metatag de votre page.

Require\_once va charger le **layout** html dans lequel ces variables sont récupérées.

```
require_once 'views/layout.html';
```

### Détails du contrôleur Bkoffice :

```
if($_POST){
    $result = new LoginUser();
    $newLogin = $result->NewLog($_POST["userLog"],$_POST["userPass"]);
    if($newLogin){
        $_SESSION["logged"] = true;
        $_SESSION["pseudo"] = $newLogin[0]["userName"];
        echo json_encode("Success");
    }
    else{
        echo json_encode("Fail");
    }
    die;
}
```

**\$\_POST** vérifie la présence de données de formulaire avant de charger la page.

```
if($_POST){
```

**\$result** est un objet **LoginUser()** qui va permettre de se logger via la base de données.

```
$result = new LoginUser();
```

**\$newLogin** va stocker la réponse de la méthode **NewLog()** dans laquelle on envoie les données de connexion de l'utilisateur.

```
$newLogin = $result->NewLog($_POST["userLog"],$_POST["userPass"]);
```

La condition qui suit vérifie que la connexion est un succès et assigne les données de la base de données récupérées dans des variables sessions puis retourne « Success » à ajax.

Si la connexion a échoué ou que les identifiants sont incorrects, « Fail » est retourné.

```
if($newLogin){
    $_SESSION["logged"] = true;
    $_SESSION["pseudo"] = $newLogin[0]["userName"];
    echo json_encode("Success");
}
else{
    echo json_encode("Fail");
}
```

### Détails du contrôleur Unlog :

Ce contrôleur n'a que pour but de clore la session de l'utilisateur et revenir sur le back office.

```
class UnlogController{
    public function UnlogUser(){
        session_destroy();
        unset($_SESSION);
        header('Location: bkoffice/');
    }
}
```

## Détails du contrôleur Datas :

**DatasController** permet de faire tous les échanges avec la base de données.

Le constructeur reçoit une variable **requeteSelection** depuis **bddFunctions.js** qui va définir et orienter votre requête dans le modèle **datas.class.php**.

```
public function __construct(array $array){
    //en fonction de l'argument on appelle une méthode
    if(isset($array["requeteSelection"])){
        switch($array["requeteSelection"]){
            case "select":
                $this->selectDatas();
                break;
            case "insert":
                $this->insertDatas();
                break;
            case "update":
                $this->updateDatas();
                break;
            case "delete":
                $this->deleteDatas();
                break;
            case "mail":
                $this->SendMail();
                break;
            //gestion images
            case "insertImg":
                $this->insertImgDatas();
                break;
            case "deleteImg":
                $this->deleteImgDatas();
                break;
        }
    }
}
```

Les méthodes sélectionnez créent un nouvel objet **Datas()** et retournent à **bddFunctions.js** le résultat de leurs requêtes, ou si elles ont simplement réussies.

```
private function selectDatas(){
    $result = new Datas();
    echo json_encode($result->selectDatas());
}
```

```
private function updateDatas(){
    $result = new Datas();
    if($result->updateDatas()){
        echo json_encode("Success");
    }
    else{
        echo json_encode("Fail");
    }
}
```

La méthode **SendMail()** permet d'envoyer un mail() php via **sendMail.js**.

```
private function SendMail(){
    $newMail = new SendMail();
    if($newMail->NewMail()){
        echo json_encode("Success");
    }
    else{
        echo json_encode("Fail");
    }
}
```

## IV – Modèles

Les modèles sont les liens directs entre la base de données et le contenu du site.  
Ils sont chargés par le routeur en même temps que les contrôleurs.

### Détail du modèle database :

**\$pdo** stock l'objet PDO lors d'une connexion.

**\$bddHostName** est le nom du serveur.

**\$bddName** le nom de la base de données.

**\$bddUser** le nom de l'utilisateur.

**\$bddPasse** le mot de passe de la base de données.

```
private $pdo;  
private $bddHostName = "localhost";  
private $bddName = "";  
private $bddUser = "";  
private $bddPasse = "";
```

La méthode **query()** est utilisée pour n'importe quelle sélection de données.

Elles sont préparées automatiquement et nécessitent une variable **string** comportant la requête SQL et un tableau contenant les valeurs.

Il est tout à fait possible de faire une requête sans valeurs.

```
//select  
public function query($sql, array $values)  
{  
    $query = $this->pdo->prepare($sql);  
    $query->execute($values);  
    return $query->fetchAll(PDO::FETCH_ASSOC);  
}
```

La méthode **prepare()** est utilisée pour toutes les autres requêtes.

Elles sont préparées automatiquement et nécessitent une variable **string** comportant la requête SQL et un tableau contenant les valeurs.

```
//insert/delete/update  
public function prepare($sql, array $values)  
{  
    $query = $this->pdo->prepare($sql);  
    $query->execute($values);  
    return $this->pdo->lastInsertId();  
}
```

La méthode PDO native **lastInsertId()** permet de retourner la dernière ligne insérée, et/ou de savoir si la requête a été un succès.

Si ce n'est pas le cas, la valeur renvoyée au modèle ne sera pas un nombre.

```
return $this->pdo->lastInsertId();
```

## Détails du modèle LoginUser :

```
public function NewLog($user,$pass){  
    $this->userName = htmlspecialchars($user);  
    $this->userPass = hash('sha512',$pass);  
  
    $bdd = new Database();  
    $this->result = $bdd->query('SELECT * FROM users WHERE userName = ? AND userPass = ?', array($this->userName,$this->userPass));  
  
    return $this->result;  
}
```

Le mot de passe est crypté avec la méthode hash(512) et doit donc être insérée dans la base de données avec ce même format.

```
$this->userPass = hash('sha512',$pass);
```

**\$bdd** est un objet **Database** permettant d'accéder aux méthodes **query()** et **prepare()**.

```
$bdd = new Database();
```

**result** stock le résultat de la requête envoyée à la méthode **query()**.

```
$this->result = $bdd->query
```

La requête SQL nécessaire aux méthodes du modèle **Database** est directement écrite ici. Dans ce cas précis, elle est préparée avec 2 valeurs.

```
('SELECT * FROM users WHERE userName = ? AND userPass = ?',
```

Ces valeurs sont envoyées dans un **array** qui sera récupéré dans la méthode **query()**.

```
array($this->userName,$this->userPass)
```



## Détails du modèle SendMail :

Les mails envoyés avec ce modèle utilisent la méthode **mail()** de php et sont au format html.  
Votre serveur doit donc autoriser l'envoi des mails php.

**\$headers** est le header du mail.

**\$mailto** est le destinataire

**\$mailfrom** est l'expéditeur

**\$message** est le corps du message

**\$messContent** est l'intégralité du html

**\$objet** est l'objet du mail

**\$back\_ligne** est le retour à la ligne au format html

```
private $headers;  
private $mailto;  
private $mailfrom;  
private $message;  
private $messContent;  
private $objet;  
private $back_ligne;
```

Les données **\$\_POST** récupérées ici proviennent de **sendMail.js**

```
$this->message = htmlspecialchars($_POST["mailMessage"]);  
$this->mailto = htmlspecialchars($_POST["mailTo"]);  
$this->mailfrom = htmlspecialchars($_POST["mailFrom"]);  
$this->objet = htmlspecialchars($_POST["mailObject"]);
```

Insertion du **message** dans le corps html.

```
$this->messContent = "<html><body>".$this->message."</body></html>";
```

Création et envoi du mail en suivant les directives php :

<https://www.php.net/manual/fr/function.mail.php>

```
mail($this->mailto,$this->objet,$this->messContent,$this->headers);
```

## Détails du modèle Datas :

Chaque requête vers la base de données est préconstruite et suivent le même schéma.

Les données sont envoyées par ajax depuis **bddFunction.js**

```
public function selectDatas(){
    $datasArray = array ($_POST["element"]); //array ($_POST["val1"], $_POST["val2"])
    $bdd = new Database();
    $this->result = $bdd->query('SELECT * FROM '.$_POST["table"].'', $datasArray);
    return $this->result;
}

public function insertDatas(){
    $datasArray = array ($_POST["element"]); //array ($_POST["val1"], $_POST["val2"])
    $bdd = new Database();
    $this->result = $bdd->prepare('INSERT INTO '.$_POST["table"].'(colonne1, colonne2) VALUES(?,?)', $datasArray);
    return $this->result;
}

public function updateDatas(){
    $datasArray = array ($_POST["element"]); //array ($_POST["val1"], $_POST["val2"])
    $bdd = new Database();
    $this->result = $bdd->prepare('UPDATE '.$_POST["table"].' SET colonne1 = ? WHERE colonne2 = ?', $datasArray);
    return $this->result;
}

public function deleteDatas(){
    $datasArray = array ($_POST["element"]); //array ($_POST["val1"], $_POST["val2"])
    $bdd = new Database();
    $this->result = $bdd->prepare('DELETE FROM '.$_POST["table"].' WHERE dataId = ?', $datasArray);
    return $this->result;
}
```

**\$datasArray** vous permet d'insérer les valeurs nécessaires à votre requête.

Sur le même schéma que **LoginUser**, la requête SQL et les valeurs sont envoyées à la méthode **query()** de **Database** pour **select** et **prepare()** pour les autres méthodes.

```
public function selectDatas(){
    $datasArray = array ($_POST["element"]); //array ($_POST["val1"], $_POST["val2"])
    $bdd = new Database();
    $this->result = $bdd->query('SELECT * FROM '.$_POST["table"].'', $datasArray);
    return $this->result;
}
```

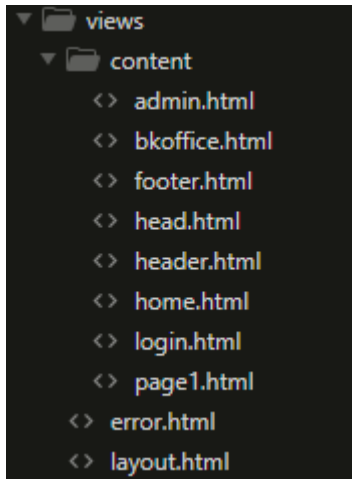
L'insertion et la suppression des images est automatique, vous avez la possibilité d'ajouter des choix de dossiers dans le formulaire de la vue **admin.html** (ne pas les oublier sur le ftp).

```
//gestion images
public function insertImgDatas(){
    $bdd = new Database();

    //verification d'erreur
    $uploadError = 0;
    //construction du chemin final - folderPath = choix radio dans le formulaire
    $target_dir = $_POST["folderPath"]."/";
    //adresse du fichier à uploader
    $target_file = $target_dir . basename($_FILES["imgPath"]["name"]);
    //recuperation de l'extension du fichier
    $imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));
    //vérification de la taille de l'image > 0
    $check = getimagesize($_FILES["imgPath"]["tmp_name"]);
    if($check == false || $target_dir == "0" || $target_dir == "o" || $target_dir == null || $target_dir == "" || empty($target_dir) ) {
        echo "Erreur : Ce n'est pas une image.<br>";
        $uploadError = 1;
    }

    //si il y a une erreur
    if ($uploadError == 1) {
        echo "Le fichier n'a pas été envoyé. <a href='\"index.php?page=bkoffice\"'>Retour</a><br>";
        exit;
    }
    //pas d'erreur
    else {
        //envoi du fichier sur le serveur puis dans la base de données
        if (move_uploaded_file($_FILES["imgPath"]["tmp_name"], $target_file)) {
            $datasArray = array ($target_dir.$FILES["imgPath"]["name"]);
            $this->result = $bdd->prepare('INSERT INTO images(imagePath) VALUES(?)', $datasArray);
        }
        else {
            echo "Une erreur s'est produite lors de l'upload. <a href='\"index.php?page=bkoffice\"'>Retour</a>";
            exit;
        }
    }
    header('location: ../index.php?page=bkoffice');
}
```

# V - Vues



Les vues sont divisées en 3 catégories :

- les layouts
- les contenus communs
- les contenus dédiés.

Les layouts sont les structures de vos pages.

Par défaut c'est **layout.html** qui est chargé pour les pages et **error.html** pour les erreurs 404.

Les contenus communs sont **head.html**, **header.html** et **footer.html**.

Les contenus dédiés sont les fichiers correspondants aux pages/urls et aux sections tel que **admin.html** ou **login.html**.

**Rappel :** Lorsque vous créez une page, le nom html de la page sera le même que l'url, que les préfixes des contrôleurs et des classes des contrôleurs.

Exemple - site.com/mapage/ = mapage.html = MapageController.class.php = MapageController{}

Les layouts sont réduits au strict minimum :

```
<!DOCTYPE html>
<html lang="fr">

  <?php include 'content/head.html'; ?>

  <body id="top">
    <?php include 'content/header.html'; ?>

    <main>
      <?php include 'content/'.$page.'.html'; ?>
    </main>

    <?php include 'content/footer.html' ?>
  </body>
</html>
```

Les **head**, **header** et **footer** sont chargés en même temps que la page et identiques pour toutes les pages.

Pour faire une nouvelle page indépendante il faut créer un nouveau layout tel que **error.html** et changer les ressources includes.

**\$page** correspond au contenu html dédié, tel que **home.html** et **page1.html**.

Modifiez le head dans **head.html**.

**\$pageDescription** récupère la description du contrôleur de la page.

**\$pageTitle** récupère le titre du contrôleur de la page.

**<base>** définit l'URL de base à utiliser pour recomposer toutes les URL relatives.

Note : Liens vers les pages internes **href= "nomDeLaPage/"**

Rappel : remplacer le kit fontawesome par le votre ou celui d'un autre site.

```
<head>
  <meta name="description" content="<?= $pageDescription ?>"></meta>
  <title>SiteName - <?= $pageTitle ?></title>
  <base href="/" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta charset="utf-8" />
  <link rel="icon" href="img/favicon.png"/>
  <!--Changer l'identifiant fontawesome-->
  <script src="https://kit.fontawesome.com/28e1129916.js"></script>
  <link rel="stylesheet" type="text/css" href="css/reset.css"/>
  <link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
```

Construisez votre header dans **header.html**.

```
<header>
  <h1><a href="home/">Nom du site</a></h1>
  <nav id="menuh">
    <ul>
      <li>
        <a href="page1/">page1</a>
      </li>
    </ul>
  </nav>
</header>
```

Construisez votre footer dans **footer.html**.

Les fichiers js sont communs à toutes les pages puisque le layout est unique.

Si vous ne voulez pas charger des fichiers, créez de nouveaux layouts indépendants ou utilisez des exceptions en modifiant les contrôleurs.

Jquery est chargé par défaut.

La variable currentPage récupère la page courante et peut être utilisée dans les fichiers html/js. Vous pouvez évidemment uniquement charger les fichiers utiles à votre projet.

```
<footer>
  <small>
  </small>
</footer>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script type="text/javascript">var currentPage = <?php echo json_encode($page); ?></script>
<script src="js/checkForm.js"></script>
<script src="js/utilities.js"></script>
<script src="js/bddFunctions.js"></script>
<script src="js/smoothScroll.js"></script>
<script src="js/isIntoView.js"></script>
<script src="js/sendMail.js"></script>
<script src="js/storageFunctions.js"></script>
<script src="js/main.js"></script>
```

Le contenu de chaque page est indépendant, vous pouvez le modifier dans le fichier html dédié tels que **home.html**, **admin.html**, **page1.html** ...

## V – JS/Jquery/Ajax

Dans le dossier js se trouvent tous les scripts dédiés au frontend.

Notamment des fonctions de base Math, les requêtes ajax préconstruites, un vérificateur de formulaire dynamique ou encore les fonctions prédéfinies pour le localStorage.

### bddFunctions.js

Vous trouverez dans ce script les 4 fonctions liées au contrôleur **DatasController** et 2 fonctions liées aux images. C'est ici que vos données sont traitées puis envoyées à la base de données. Les 6 fonctions sont identiques excepté leurs **requeteSelection** qui indiquent au contrôleur comment gérer ces données.

Pour plus de clarté les fonctions sont séparées mais peuvent parfaitement être regroupées en une seule avec un argument supplémentaire qui modifiera la valeur de **requeteSelection**. Le champs **data** : est également utilisable en l'état mais peut tout aussi bien remplacer ses variables par un **array()** ou un **form.serialize()**. Tout dépend de votre aisance avec ajax.

```
//select datas
function ShowDatas(table,element){
    let myTable = table;
    let myElement = element;

    $.ajax({
        url : 'datas/',
        type : 'POST',
        data: { requeteSelection : "select", table : myTable, element : myElement},
        dataType: 'json',
        error : function(resultat, statut, erreur){
            console.log('erreur ajax : '+resultat+"//"+statut+"//"+erreur);
            $('#errorForm p').css("color","red");
            $('#errorForm p').html('Echec de l\'envoi.');
```

Note : **\$('#errorForm p').css("color","red");** et **\$('#errorForm p').html('Echec de l\'envoi.');** sont facultatifs et peuvent être supprimés.

Ils désignent un id que vous devez assigner au DOM afin de récupérer et afficher les erreurs.

**function AdminShowImages(table,element)** est appelée au chargement de la page bkoffice pour afficher les images présentes sur le serveur et dans la base de données.

La ligne appelant la fonction est dans le fichier **main.js** (**AdminShowImages("images","");**)

## checkForm.js

Ce fichier permet de vérifier tous vos formulaires avant d'envoyer les données.

### Note sur la création du formulaire :

- les champs à vérifier par la fonction doivent posséder l'attribut **required**.

- le formulaire doit contenir la balise **<fieldset>**.

Dans le cas contraire supprimez **.parent("fieldset")** du vérificateur.

- les input de type checkbox doivent être enfant de leur **<label>** :

**<label>Name<input type="checkbox" name=" " value=" " required></label>**

- indiquer l'id du formulaire dans **<form>** :

**<form id="logForm"></form>**

- indiquer l'url de la page de destination dans le **data-action\_form** du **<button>** :

**< button data-action ="datas/"></button >**

- indiquer la classe **. sendForm** dans le **<button>** :

**< button data-action ="datas/" class="sendForm" ></button>**

### **Formulaire type pour la vérification :**

```
<form id= "monId">
```

```
  <fieldset>
```

```
    <legend>Titre formulaire</ legend >
```

```
    <label for="userLog">Pseudo :</label>
```

```
    <input type="text" name="userLog" id="userLog" required>
```

```
    <label for="userMessage">Message :</label>
```

```
    <textarea type="text" name=" userMessage " id=" userMessage " required>
```

```
  </textarea>
```

```
    <select name="mySelect" required >
```

```
      <option value="">Selectionner</option>
```

```
      <option value="option1">option1</option>
```

```
      <option value="option2">option2</option>
```

```
    </select>
```

```
    <label>
```

```
      Name
```

```
      <input type="checkbox" name="myBox" value=" myBox " required>
```

```
    </label>
```

```
  <div id="errorForm"><p></p></div>
```

```
    <button data-action_form="bkoffice/" type="button" class="sendForm">
```

```
      Connexion
```

```
    </button>
```

```
  </ fieldset >
```

```
</form>
```

### En détails :

La vérification commence au clique sur un button comportant la classe **.sendForm**

```
$('.sendForm').click(function() {
```

La variable **idForm** récupère l'ID du formulaire.

**nbErreurs** sera incrémenté à chaque champs vide.

**actionForm** récupère l'url dans le **button**.

```
let idForm = $(this).parent("fieldset").parent("form").attr("id");
let nbErreurs = 0;
let actionForm = $(this).data("action_form");
```

Pour chaque champs **input** avec l'attribut **required**, la variable **valeur** prend sa valeur puis **valTrim** vérifie qu'elle n'est pas vide ou que ce n'est pas un espace.

Si c'est le cas une erreur est ajoutée et un border rouge est ajouté à l'élément.

```
//vérification des champs input
$('#'+idForm+' input[required]').each(function() {
  let valeur = $(this).val();
  let valTrim = $.trim(valeur);
  if(valTrim.length == 0) {
    nbErreurs++;
    $(this).css('border', '1px solid red');
  }
  else{
    $(this).css('border', '1px solid grey');
  }
});
```

Procédure identique pour **textarea** et **select**

```
//vérification des champs textarea
$('#'+idForm+' textarea[required]').each(function()
  let valeur = $(this).val();
  let valTrim = $.trim(valeur);
  if(valTrim.length == 0) {
    nbErreurs++;
    $(this).css('border', '1px solid red');
  }
  else{
    $(this).css('border', '1px solid grey');
  }
});
```

```
$('#'+idForm+' select[required]').each(function() {
  let valeur = $(this).val();
  let valTrim = $.trim(valeur);
  if(valTrim.length == 0) {
    nbErreurs++;
    $(this).css('border', '1px solid red');
  }
  else{
    $(this).css('border', '1px solid grey');
  }
});
```

Pour les **checkbox**, c'est le **label** qui subit une modification css.

`$(this).prop('checked')` vérifie que chaque **checkbox** obligatoire est cochée.

```
$('#'+idForm+' input[type=checkbox][required]').each(function() {
    let parent = $(this).parent("label");
    if(!$(this).prop('checked')){
        nbErreurs++;
        $(parent).css('border','1px solid red');
    }
    else{
        $(parent).css('border','1px solid transparent')
    }
});
```

L'envoi est identique à ceux traités dans les chapitres précédents.

`url` est récupérée via **actionForm** et le formulaire est sérialisé automatiquement.

```
if(nbErreurs == 0) {
    $('#errorForm p').html('');
    $.ajax({
        url : actionForm,
        type : 'POST',
        data: $('#'+idForm).serialize(),
        dataType: 'json',
        error : function(resultat, statut, erreur){
            console.log('erreur ajax : '+resultat+"//"+statut+"//"+erreur);
            $('#errorForm p').css("color","red");
            $('#errorForm p').html('Echec de l\'envoi.');
```

Ce switch sert à rediriger l'utilisateur sur une page spécifique en cas de succès.

```
switch(actionForm){
    case "bkoffice/":
        window.location.href = 'bkoffice/';
    default:
        $('#errorForm p').html(isDone);
}
```



## isIntoView.js

Cette fonction identifie si les éléments qu'on lui envoie ponctuellement ou à chaque scroll sont visibles ou non dans la fenêtre.

Pour l'utiliser, utilisez simplement **isScrolledIntoView('selecteur')** avec votre élément en argument, ou via `window.scroll` pour une détection automatique.

```
function isScrolledIntoView(elem)
{
    let docViewTop = $(window).scrollTop();
    let docViewBottom = docViewTop + $(window).height();
    let elemTop = $(elem).offset().top;
    let elemBottom = elemTop + $(elem).height();
    return ((elemBottom <= docViewBottom) && (elemTop >= docViewTop));
}

//chaque scroll déclenche cette fonction
$(window).scroll(function() {
    //si l'élément est présent dans la fenêtre
    if(isScrolledIntoView('#top')){
        //do something
    }
});
```

## main.js

Ceci est le script principal où vous ferez appel aux fonctions et construirez vos interactions. On y retrouve la variable **currentPage** qui peut par exemple être utilisée pour changer le css du menu nav dans le header et indiquer où l'utilisateur se trouve sur le site.

Ici elle est utilisée pour détecter la page back office et lancer la fonction qui affichera les images du site dans le back office.

```
$(function() {
    //récupère le nom de la page courante
    if(currentPage == "bkoffice"){
        //change le css du menu en fonction de la page courante
        //$('#header nav ul li:nth-child(1)').addClass('boldMenu');

        //récupération des images dans l'administration
        AdminShowImages("images", "");
    }

    //////////////////////////////////fonctions////////////////////////////////////

});
```

## sendMail.js

La fonction **sendMail** récupère vos données de formulaire puis les envoi à **DatasController**.

```
function SendMail(mailFrom,mailTo,mailMsg,mailObj){
  let myMailFrom = mailFrom;
  let myMailTo = mailTo;
  let myMessage = mailMsg;
  let myObject = mailObj;

  $.ajax({
    url : 'datas/',
    type : 'POST',
    data: { requeteSelection : "mail", mailFrom : myMailFrom, mailTo : myMailTo, mailMessage : myMessage, mailObject : myObject},
    dataType: 'json',
    error : function(resultat, statut, erreur){
      console.log('erreur ajax : '+resultat+'/'+'statut+'/'+'erreur');
      $('#errorForm p').css("color","red");
      $('#errorForm p').html('Echec de l\'envoi.');
```

Pour fonctionner, ces arguments doivent obligatoirement être fournis par votre formulaire.

La méthode **SendMail()** du modèle **Datas** se charge du reste.

Expéditeur, destinataire, message et objet du mail.

```
function SendMail(mailFrom,mailTo,mailMsg,mailObj){
  let myMailFrom = mailFrom;
  let myMailTo = mailTo;
  let myMessage = mailMsg;
  let myObject = mailObj;
```

## smoothScroll.js

Comme son nom l'indique, ce script permet de récupérer les hash des liens pour se déplacer dans votre page de façon fluide.

Exemple : `<a href="#top"></a>`

La valeur **addOffsetTop** est réglable pour faire correspondre la hauteur de votre header par exemple. Elle va être ajoutée ou soustraite à la position finale.

La valeur **speedScroll** est la vitesse de déplacement.

**window.location.hash** est désactivé en cas d'offset personnalisé.

```
$( "a" ).on( 'click', function( event ) {
  let addOffsetTop = 0;
  let speedScroll = 600;
  if ( this.hash !== "" ) {
    // bloquer le changement de page
    //event.preventDefault();
    let hash = this.hash;
    $('html, body').animate({scrollTop: $(hash).offset().top - addOffsetTop}, speedScroll, function(){
      if(addOffsetTop == 0){
        window.location.hash = hash;
      }
    });
  }
});
```

## storageFunctions.js

Si vous avez besoin de stocker des données dans le **localStorage** :

**ToLocalStorage("vosDonnées","nomDansLeLocalStorage") ;**

**vosDonnées** peut être un tableau ou simplement une variable

**nomDansLeLocalStorage** doit être unique pour chaque données stockées

Pour récupérer des données :

**FromLocalStorage("nomDansLeLocalStorage") ;**

La variable de retour est presque toujours un tableau qu'il faut parcourir avec une boucle.

```
function FromLocalStorage(dataName){
    return JSON.parse(window.localStorage.getItem(dataName));
}

//Envoyer vers le localStorage
function ToLocalStorage(myDatas,dataName){
    let dataJson = JSON.stringify(myDatas);
    myDatas = JSON.parse(window.localStorage.getItem(dataName));
    if(myDatas == null){
        myDatas = [];
    }
    window.localStorage.setItem(dataName,dataJson);
}
```

## utilities.js

Utilities contient des fonctions basiques souvent utilisées et permettent donc de gagner du temps dans vos algorithmes.

Obtenir un nombre décimal aléatoire entre 0 et X, arrondie à Y décimales.

Exemple : **GetRandomDecimal(15,2); (peut donner 12,15)**

```
//get random decimal
function GetRandomDecimal(maxInt,maxDec){
    return (Math.random()*maxInt).toFixed(maxDec);
}
```

Obtenir un nombre entier aléatoire entre 0 et X.

Exemple : **GetRandomInt(8); (peut donner 3)**

```
//get random int
function GetRandomInt(maxInt){
    return Math.floor(Math.random()*maxInt);
}
```

Arrondir un décimal à son entier le plus proche.

Exemple : **RoundToInt(38,8254); (donne 39)**

```
//round to int
function RoundToInt(decToRound){
    return Math.round(decToRound);
}
```

Obtenir le nombre entier d'un nombre décimal.

Exemple : **DecimalToInt(38,1254); (donne 38)**

```
//get int from decimal
function DecimalToInt(decToInt){
    return Math.floor(decToInt);
}
```

Obtenir le nombre le plus grand dans un tableau.

Exemple : **GetMax([1, 33, 288,8]); (donne 288)**

```
//GetMax([1, 33, 288,8]);
function GetMax(myArray){
    return Math.max(...myArray);
}
```

Obtenir le nombre le plus petit dans un tableau.

Exemple : **GetMin([1, 33, 288,8]); (donne 1)**

```
//get min
//GetMin([1, 33, 288,8]);
function GetMin(myArray){
    return Math.min(...myArray);
}
```

Savoir si un nombre est positif ou négatif.

Exemple : **IsPositive(-45);** (donne false)

```
function IsPositive(number){
    let intToBool = false;
    if(Math.sign(number) == 1){
        intToBool = true;
    }
    else{
        intToBool = false;
    }
    return intToBool;
}
```

Obtenir la date et l'heure du jour au format Fr ou convertir une date au format aaaa/mm/jj.

Exemple : **GetDate();** (donne par exemple Samedi 22 Février - 18h09min18sec)

Exemple : **GetDate("2015-12-05");** (donne par exemple Mardi 05 Décembre)

```
//get date
function GetDate(setDate){
    //let test = Date(Date.now()); // jour systeme
    let myDate;
    let myTime;
    setDate ? myDate = new Date(setDate) : myDate = new Date();
    let myMonths = new Array('Janvier', 'Fevrier', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Aout', 'Septembre', 'Octobre', 'Novembre', 'Decembre');
    let myDays = new Array('Dimanche', 'Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi');
    let trueHours, trueMinutes, trueSeconds;
    myDate.getHours() < 10 ? trueHours = "0" + myDate.getHours().toString() : trueHours = myDate.getHours().toString();
    myDate.getMinutes() < 10 ? trueMinutes = "0" + myDate.getMinutes().toString() : trueMinutes = myDate.getMinutes().toString();
    myDate.getSeconds() < 10 ? trueSeconds = "0" + myDate.getSeconds().toString() : trueSeconds = myDate.getSeconds().toString();

    setDate ? myTime = "" : myTime = " - " + trueHours + "h" + trueMinutes + "min" + trueSeconds + "s";
    return (myDays[myDate.getDay()] + " " + myDate.getDate() + " " + myMonths[myDate.getMonth()] + " " + myDate.getFullYear() + myTime);
}
```

## VII - Base de données

La base de données est structurellement libre, néanmoins pour profiter pleinement des outils en place dans le template il est important d'importer le fichier « **default\_data.sql** ».

Pour accéder au back office par défaut, rajouter **bkoffice/** à la suite du nom de votre site.  
Exemple : site.fr/**bkoffice/**

Le pseudo par défaut est : admin

Le mot de passe par défaut est : P2JSGQF239

Les mots de passe sont vérifiés avec la méthode hash(sha512) , il est donc obligatoire de générer des mots de passe avec cette méthode dans la base de données.

### 1ere méthode :




Sur <http://phptester.net/> ou un autre site permettant de tester du code php, tapez la ligne **echo hash('sha512','votreMotDePasse');**


**votreMotDePasse** doit être remplacé par le mot de passe désiré.

Une chaîne de caractères sera générée, il suffira alors de la rentrer dans la colonne « **userPass** » de la base de données, exemple :

**86ce4e2762c6faea19a283f6528e269442fa484fc568ac99384486dfac8d237b942cab6506b884f3d391e31d9f0dfdc5273a8eaed9a77f83c9531f16d3ce94fb**


+ Options


	userId	userName	userPass
<div><div><input type="checkbox"/></div><div> Éditer</div><div> Copier</div><div> Supprimer</div></div>	1	admin	86ce4e2762c6faea19a283f6528e269442fa484fc568ac9938...





☐ Tout cocher

Avec la sélection :

 Éditer

 Copier

 Supprimer

 Exporter

### 2° méthode :

Sur n'importe quelle page contenu **html** de votre site (home, page1...), insérez le code suivant comme première ligne de la page avant n'importe quelle autre ligne :

```
<div style="width: 100% !important;background-color:#ff8181;position: fixed;z-index: 1000;font-weight: bold;text-align: center;padding: 5px;border: 2px solid black;color:black;">  
    Clé : <?= hash('sha512','votreMotDePasse') ?>  
</div>
```

La clé sera générée et visible en haut de la page.

