

Compte-rendu de projet

Réaliser un site de candidature pour le festival

Année 2 – Groupe 4

Ferdinand Justin

Risse Baptiste

Rhuin Maël

Objectifs :

Le principal objectif de ce projet est la découverte de Framework tel que Flight et de Smarty, le moteur de template. Ainsi, la création, la gestion des données par l'intermédiaire d'une base de données de phpMyAdmin seront vus dans ce projet. Des services d'inscription, de connexion, seront mis en place.

Sujet :

Le sujet de ce projet est de réaliser un site web pour un festival, où les groupes candidats peuvent s'inscrire et déposer leurs candidatures pour participer à l'évènement. Ils pourront accéder à leurs candidatures via leur profil.

Les membres de l'équipe du festival devront avoir la possibilité de consulter toutes les candidatures, en plus de pouvoir consulter en détail celles qui les intéressent.

Sommaire



Base de données



Inscription



Candidater



Liste et candidatures détaillées



Conclusion

Base de données

Afin de réaliser nos tables de la base de données, nous avons suivi le sujet du projet.

Nous avons ainsi créé 6 tables :

candidature, departement, groupe, scene, style_musicaux et utilisateur. Chaque table est nommée en fonction de ce qu'elle contient. Ainsi la table candidature possède toutes les informations concernant les dépôts des groupes, la table style_musicaux contient divers styles de musique et utilisateur contient les informations sur les membres de l'équipe et les responsables de groupe.

#	Nom	Type	#	Nom	Type
1	<u>id</u>	int(255)	1	<u>email</u> 	varchar(255)
2	<u>nom</u>	varchar(255)	2	<u>type</u>	varchar(255)
3	<u>departement</u>	varchar(255)	3	<u>nom</u>	varchar(255)
4	<u>email</u>	varchar(255)	4	<u>prenom</u>	varchar(255)
5	<u>style</u>	varchar(255)	5	<u>adresse</u>	varchar(255)
6	<u>annee</u>	int(255)	6	<u>code_postal</u>	int(255)
7	<u>presentation</u>	text	7	<u>telephone</u>	int(255)
8	<u>experience</u>	text	8	<u>motdepasse</u>	varchar(255)
9	<u>urlgroupe</u>	varchar(500)			
10	<u>soundcloud</u>	varchar(500)			
11	<u>youtube</u>	varchar(500)			
12	<u>association</u>	tinyint(1)			
13	<u>sacem</u>	tinyint(1)			
14	<u>producteur</u>	tinyint(1)			
15	<u>fichier1</u>	varchar(255)			
16	<u>fichier2</u>	varchar(255)			
17	<u>fichier3</u>	varchar(255)			
18	<u>dossier_presse</u>	varchar(255)			
19	<u>photo1</u>	varchar(255)			
20	<u>photo2</u>	varchar(255)			
21	<u>fiche_technique</u>	varchar(255)			
22	<u>doc_sacem</u>	varchar(255)			

Structure des champs des tables candidature et utilisateur. Il s'agit des deux tables les plus importantes de notre base de données.

Pour la table candidature, l'ID du groupe est la clé primaire.

Pour la table utilisateur, l'email est la clé primaire.

Inscription

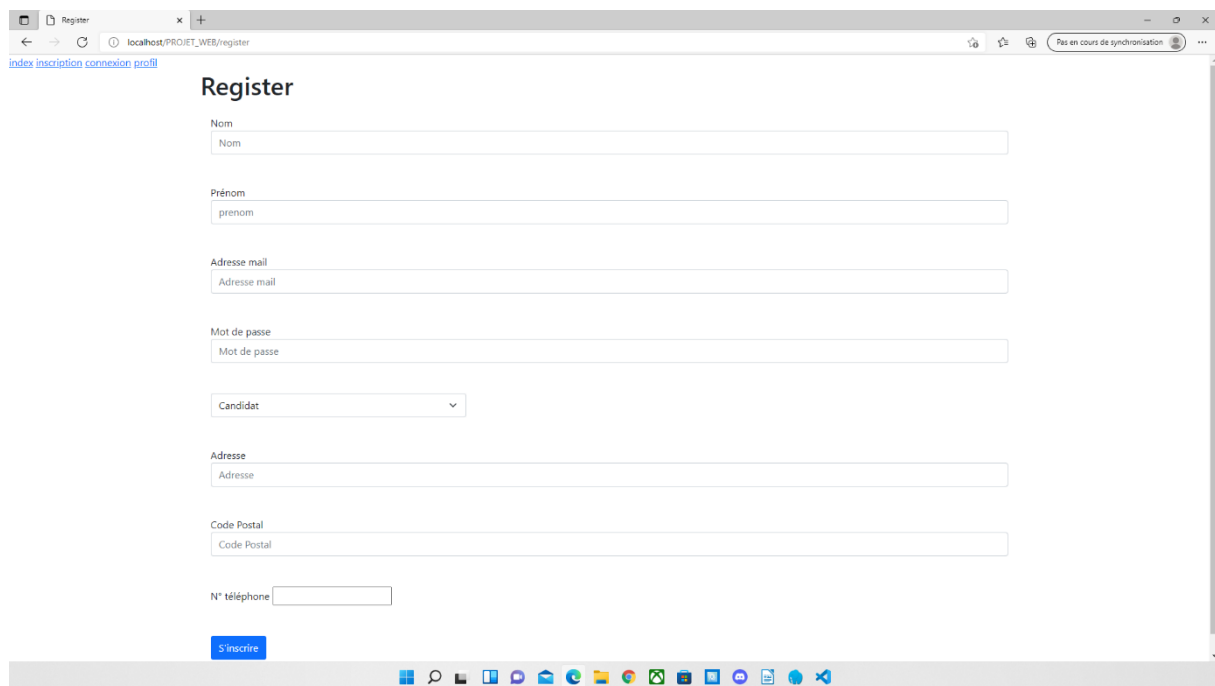
L'objectif ici est de créer une route et une page web qui permet à un utilisateur de s'inscrire sur le site. Si le formulaire est correctement rempli, les données seront stockées dans la table utilisateur. Sinon le site doit gérer les erreurs et les indiquer à l'utilisateur.

Le formulaire est codé dans le template register, on y retrouve toutes les données de la table à taper. La méthode utilisée est la méthode POST, car cela évite de passer toutes les données dans l'URL ce qui est préférable au niveau de la protection des données.

Ensuite nous allons voir la route qui permet de rendre cette page fonctionnelle. La route register avec la méthode GET qui permet d'afficher le template grâce à la commande `Flight::render()`.

```
Flight::route('GET /register', function(){  
    Flight::render("register.tpl",array());  
});
```

Page d'inscription :



The screenshot shows a web browser window with the address bar displaying `localhost/PROJET_WEB/register`. The page title is "Register". The form contains the following fields:

- Nom:
- Prénom:
- Adresse mail:
- Mot de passe:
- Candidat:
- Adresse:
- Code Postal:
- N° téléphone:

At the bottom of the form is a blue button labeled "S'inscrire".

La route register (avec la méthode POST) va nous permettre de vérifier les informations rentré par l'utilisateur.

Tout d'abord dans cette route un tableau associatif message est instancié. Ce tableau permettra par la suite de stocker les messages d'erreurs.

```
$message = array();
```

Après la déclaration des messages on s'occupe de stocker tous les champs du formulaire dans des variables. Cette action se fait avec `Flight::request()->data`, cette « commande » permet d'accéder à un objet, qui peut ensuite être stocké dans une variable. Avec cette partie l'objectif est de récupérer les valeurs du tableau `$_POST` pour les mettre dans des variables.

`$_POST` : est donc un tableau associatif qui contient les valeurs du formulaire rempli par l'utilisateur du site. Par exemple si le client rentre son adresse mail dans le champ email, alors `$_POST['email']` sera égal à l'adresse mail du client.

`Flight::request` : comme vu juste avant cette commande permet de d'accéder à des objets. `Flight::request()` possède également des options, dans ce TP on utilise l'option `data` qui permet de récupérer des données de POST ou des données de JSON. On utilisera donc cette option pour récupérer `$_POST`.

```
$data = Flight::request()->data;  
$nom = $data -> nom;
```

Ici la variable `nom` prend donc la valeur de `$_POST['nom']`.

Après ça on vérifie que toutes les variables sont remplies, donc que l'utilisateur a bien rempli tous les champs du formulaire d'inscription. Utilisation de `empty` qui indique si la variable est vide. De ce fait, si la variable est vide on crée une nouvelle case dans `$message` dans laquelle on va mettre le message d'erreur.

Exemple :

```
if (empty($prenom)){  
    $message['prenom'] = "Veuillez remplir le champs prenom";
```

Après avoir effectué cette vérification pour toutes les variables (`$nom`, `$prenom`, `$email`, `$mdp`, `$role`, `$mdp_res` si responsable, `$adresse`, `$code_postal`, `$telephone`), il faut vérifier que l'adresse mail soit correcte et qu'elle ne se trouve pas déjà dans la table utilisateur.

Tout d'abord, pour vérifier la validité on utilise une fonction qui permet de vérifier si la forme de l'adresse mail rentré est correct.

```
filter_var($email, FILTER_VALIDATE_EMAIL)
```

Ensuite si l'adresse mail est validée, nous pouvons donc vérifier si elle n'est pas encore stockée dans la table. Pour cela on appelle une ligne de la table (toutes les informations sur un client) où l'adresse serait la même que celle stockée dans `$email`, puis on regarde si cette ligne existe. Si cette adresse mail existe déjà dans la table alors on ajoute un nouveau message d'erreur sinon cette adresse peut être utilisée pour créer un compte.

Pour cela la recherche dans la table se fait avec une requête SQL. Nous préparons la requête :

```
$BDD = Flight::get('BDD');
$stmt = $BDD->prepare("select * from utilisateur where email = ?");
```

Ensuite on exécute la commande avec l'adresse mail entré par l'utilisateur :

```
$stmt -> execute(array($email));
```

Un jeu de résultat est donc récupéré (ou pas) et stocker dans \$stmt. Ensuite la méthode fetch() est utilisée sur \$stmt, cette méthode permet de récupérer la première ligne disponible dans le jeu de résultat et donc on peut assigner cette ligne à une variable sous la forme d'un tableau. C'est ce qui est fait, \$user contient donc une ligne de la table ou rien du tout. Si \$user contient quelque chose alors cela veut dire que l'adresse mail existe déjà dans la table et qu'il faut créer un nouveau message d'erreur. Sinon si \$user ne contient rien alors cela veut dire que l'adresse mail n'est pas encore dans la table et peut être utilisée :

```
$user = $stmt->fetch();
```

La prochaine étape est de vérifier que le mot de passe du formulaire soit au minimum de 8 caractères et qu'il contient au moins un chiffre, une majuscule et une minuscule si le champ n'est pas vide bien évidemment. Si le mot de passe respecte cette condition alors il sera bon, mais cela est très dangereux de stocker un mot de passe comme ça dans la table. Pour sécuriser le mot de passe on va le hacher puis le stocker dans la table utilisateur haché. Le principe de hachage sera expliqué dans une partie suivante. Le mot de passe haché est stocké dans la variable \$hashed_password.

On déclare avant trois variables, une qui correspond aux majuscules, une aux minuscules et la dernière aux chiffres.

```
$majuscule = preg_match('@[A-Z]@', $mdp);
$minuscule = preg_match('@[a-z]@', $mdp);
$chiffre = preg_match('@[0-9]@', $mdp);
```

```
if(!$majuscule || !$minuscule || !$chiffre || strlen($mdp) < 8){
    $message['mdp_longueur'] = "Le mot de passe doit être de 8 caractères au minimum contenir au
moins un chiffre, une majuscule et une minuscule";
}
else{
    $hashed_password = password_hash($mdp, PASSWORD_DEFAULT);
}
```

Si le mot de passe ne contient pas au moins un chiffre, une majuscule et une minuscule ou il est inférieur à 8 caractères alors nous créons un nouveau message d'erreur sinon on chiffre le mot de passe pour le stocker dans la base de données.

Quand un utilisateur s'inscrit, il a deux rôles disponibles, candidat ou responsable (du festival). Tout le monde peut s'inscrire comme candidat mais tout le monde ne peut pas s'inscrire comme responsable, c'est pour cela que lorsque ce rôle est sélectionné, un mot de passe

spécial est demandé pour s'inscrire en tant que responsable. Donc lorsque responsable est sélectionné, il faut vérifier si le mot de passe est rempli.

```
if(empty($mdp_res) && $role === "responsable"){
    $message['mdp_res'] = "Veuillez saisir le champ mot de passe responsable";
}
```

Si le champ mot de passe est rempli, il y a une vérification que le mot de passe correspond au bon mot de passe.

```
if($role === "responsable" && $mdp_res != 1234567890){
    $message['erreur_mdp_res'] = "Le mot de passe est incorrect";
}
```

Ensuite sur les champs code postal et téléphone sont des inputs de type number. Pour ces deux champs on vérifie leur longueur. Code postal doit être de 5 ou 6 caractères, sinon un message d'erreur apparaît.

```
if(strlen($code_postal) != 5 && strlen($code_postal) != 6){
    $message['erreur_code_postal'] = "Le code postal est incorrect";
}
```

Et le numéro de téléphone doit être égal à 10 chiffres (taille d'un numéro de téléphone).

```
if(strlen($telephone) != 10){
    $message['erreur_telephone'] = "Le numéro de téléphone est incorrect";
}
```

La fonction **strlen** retourne le nombre de caractères de la variable.

Pour finir cette route si \$message est vide (s'il n'y a pas d'erreur), on peut donc importer toutes les données dans la table utilisateur. Sinon s'il n'est pas vide alors il y a des erreurs qu'il faut indiquer à l'utilisateur.

Commençons de regarder le cas où il n'y a pas d'erreurs. Si ce cas se présente, il faut rentrer d'abord les données dans la table afin de les enregistrer. Comme tout à l'heure on utilise la méthode prepare, execute, mais cette fois-ci on met la commande INSERT INTO dans la requête, afin d'insérer toutes les données dans la table.

```
$stmt = $BDD->prepare("INSERT INTO utilisateur VALUES (:email,:type,:nom,:prenom,:adresse,:code_postal,:telephone,:motdepasse)");
$stmt->execute(array('email'=>$email,'type'=>$role,'nom'=>$nom,'prenom'=>$prenom,'adresse'=>$adresse,'code_postal'=>$code_postal,'telephone'=>$telephone,'motdepasse'=>$hashed_password));
```

La page success est ensuite affichée pour indiquer à l'utilisateur que son inscription a bien été prise en compte.

```
Flight::render("success.tpl", array());
```


En revanche, si \$message n'est pas vide il y a des erreurs, il faut donc les afficher à l'utilisateur. Pour cela on va encore utiliser Flight::render avec comme paramètre register.tpl, le tableau \$message pour afficher les erreurs, \$_POST afin que toutes les valeurs remplit dans le formulaire ne s'efface pas à chaque fois qu'il y a une mise à jour de la page, comme une nouvelle erreur. Ainsi que \$role pour voir afficher le mot de passe responsable quand le rôle responsable est choisi.

Dans le register.tpl, les variables passées dans le render sont utilisé pour l'affichage. Le tableau \$_POST permet de ne pas ressaisir les valeurs dans une page si elle contient des erreurs.

Exemple avec nom :

```
value='{ $valeurs.nom | escape | default: "" }'
```

value est un paramètre de la balise HTML input, avec cette notation si valeurs.nom est vide on ne met rien (la chaîne nulle). \$Valeurs correspond au tableau \$_POST, valeurs a été défini dans le render à la fin de la route POST /register.

Ensuite on a une commande qui permet d'afficher les erreurs du tableau \$message s'il y en a.

Exemple avec prenom :

```
{ $message.prenom | escape | default: "" }
```

Enfin \$role est utilisé pour afficher l'input du mot de passe responsable si ce rôle est choisi par l'utilisateur.

Dans ce template message est aussi utilisé pour afficher l'input en rouge s'il y a une erreur. Si les cases des erreurs en question sont vide on affiche un input classique. Mais s'il y a des erreurs on affiche un input « is-invalid » pour que les contours soient rouge et donc que l'erreur soit plus visuelle.

Exemple avec rôle :

```
{if empty($message.mdp_vide) && empty($message.mdp_longueur)} class="form-control"{else} class="form-control is-invalid" {/if}
```

Illustration de la page d'inscription :

Register

Nom
Ferdinand

Prénom
Justin

Adresse mail
justin.ferdinand@gmail.com

Mot de passe
Mot de passe
Veuillez remplir le champ mot de passe

Candidat

Adresse
Tataouine les bains

Code Postal
Code Postal
Veuillez remplir le champ code postal

N° téléphone
n° telephone
Veuillez remplir le champ n° téléphone

Connexion :

Pour réaliser la page de connexion avec deux routes, les routes login, l'une avec la méthode GET et l'autre avec la méthode POST.

La route login de méthode GET va permettre d'afficher le formulaire de connexion on utilise le même principe que précédemment avec la route register.

```
Flight::route('GET /login', function(){
    Flight::render("login.tpl",array());
});
```

Page de connexion :

Login

Le mot de passe est incorrect

Adresse mail
brianjohnson@gmail.com

Mot de passe

Se connecter

On peut voir que pour se connecter l'adresse mail et le de passe sont demander. Or maintenant il faut vérifier que lorsque l'utilisateur remplit le formulaire, l'adresse mail et le mot de passe correspondent dans la table utilisateur. Il faut aussi vérifier comme avec la route POST /register que les deux champs ont été remplis. Tout ceci nous allons le vérifier dans la route login de méthode POST.

Pour commencer avec la cette route, le tableau `$_POST` qui contient deux valeurs (`$_POST['email']` et `$_POST['mdp']`). On utilise la méthode `Flight::request()`. On obtient donc :

- `$email = $_POST['email']`
- `$mdp = $_POST['mdp']`

On instancie également un tableau associatif `$message` qui comportera tout les messages d'erreurs.

Tout d'abord on vérifie que tous les champs du formulaire ont été rempli avec **empty**, si les champs sont vides on stock le message d'erreur.

Sinon, il faut vérifier que les données dans rentées dans le formulaire correspondent dans la table.

Dans un premier temps on va vérifier si l'adresse mail correspond à une adresse mail dans la table. On utilise une requête SQL qui sélectionne toutes les données d'un utilisateur si l'adresse mail du formulaire correspond avec son adresse mail.

```
$stmt = $BDD->prepare("select * from utilisateur where email = ?");  
$stmt->execute(array($email));
```

Ensuite si l'adresse mail ne correspond à aucune des adresse mail de la table, un nouveau message d'erreur est mis dans `$message`. Sinon si l'adresse mail correspond, il faut maintenant vérifier que le mot de passe correspond également.

Le problème est que les mot de passe qui sont dans la table utilisateur sont chiffrés. Pour régler ce problème, il faut utiliser la fonction `password_verify` qui permet de comparer un mot de passe avec un mot de passe chiffré. Si le mot de passe est incorrect on l'indique à l'utilisateur.

```
if(!password_verify($mdp, $hashed_mdp[0]))
```

password_verify : Cette fonction prend en paramètre un mot de passe de type chaîne de caractère et un mot de passe chiffré, haché. La fonction va comparer les deux mot de passe et retourner un booléen, true si les deux mot de passe sont identiques, false s'ils ne le sont pas.

Pour finir, il faut s'occuper de l'affichage. Donc s'il y a des erreurs (vérification que `$message` est vide ou pas), la procédure est la même que pour la route POST /register : utilisation de `Flight::render` qui prend en paramètres `login.tpl` ainsi que `$message` et `$_POST`.

Par contre si le tableau `$message` est vide on récupère le nom de l'utilisateur dans la ligne de la table qui correspond à l'adresse mail rentrée par l'utilisateur. Pour ça on utilise toujours la méthode `prepare execute`, avec laquelle on va pouvoir récupérer le nom dans la table

correspondant à l'email inscrit dans le formulaire. Le jeu de données est ensuite transformé en tableau par la méthode fetch.

```
$info->execute(array($email));  
$infoUtilisateur = $info->fetchAll();  
$_SESSION = array();
```

Ensuite ces données sont stockées dans le tableau associatif \$_SESSION à l'aide d'un foreach, pour être ensuite afficher plus tard sur la page profil. Nous avons décidé de ne pas prendre l'adresse mail et le mot de passe car les données vont être affichées, or l'adresse mail et le mot de passe sont des données sensibles.

\$_SESSION : ce tableau associatif permet en php de stockés les données d'un compte. C'est ce qu'on appelle une « superglobale », ce qui veut dire que cette variable est disponible dans toutes les pages. Il faut utiliser la commande **session_start()** pour démarrer une session, cela attribut également un numéro de session. Dans le code il est placé au début du fichier index.php. Pour fermer la session on utilise **session_destroy()** et **session_unset()** permet d'effacer les contenu du tableau \$_SESSION. Ces deux commandes seront utilisées plus tard dans le TP.

Pour finir on affiche la route « / », qui affiche index.tpl.

L'affichage des erreurs dans login.tpl est géré de la même manière que dans register.tpl.

Profil :

Une fois connecté l'utilisateur peut voir ses données dans la page profil. Il faut donc modifier légèrement la fin de la route login pour récupérer en plus du nom, la ville et le pays dans le tableau session. Dans la requête SQL on change le select nom pour le select * qui permet de récupérer tous les éléments sur un utilisateur. Ensuite, on utilise un foreach pour récupérer ces données dans le tableau \$_SESSION.

Le but est d'afficher les données de l'utilisateur s'il est connecté sinon si l'utilisateur est déconnecté alors on le renvoi vers la page de connexion. C'est donc le rôle de la route profil qui renvoi la page de connexion si \$_SESSION est vide ou sinon la page profil.

Déconnexion :

La route GET /logout permet de se déconnecter.

```
Flight::route('GET /logout', function(){  
    session_unset();  
    session_destroy();  
    Flight::redirect('/');  
});
```

La commande session_unset() vide le tableau \$_SESSION et session_destroy() détruit la session ouverte par session_start() on réaffiche l'index à la fin.

Candidater

À travers l'implémentation des différents mécanismes qui seront développés dans la suite de cette partie, nous devons permettre à un utilisateur candidat et donc représentant de son groupe de candidater au Festival. Il nous faut donc lui **donner la possibilité d'inscrire des informations** qui sont ici sous différentes formes. Il faut ensuite **vérifier la validité des saisies** et en **informer l'utilisateur** des éventuelles erreurs ou d'une confirmation d'une prise en compte de sa candidature. Dès lors que la saisie est sans erreur il nous faut **enregistrer les différents éléments** de la candidature et ainsi la **rendre disponible aux concernés**.

Les différents groupes de mots en gras ci-dessus sont donc les besoins auquel la **candidater** doit répondre.

« Donner la possibilité d'inscrire des informations »

Quelles informations doivent-être inscrites dans la candidature ?

Table **candidature**

Le formulaire en ligne doit permettre aux candidats de fournir et stocker les informations suivantes :

- Nom du groupe
- Département d'origine (cf. table departement)
- Type de scène (cf. table scene)
- Représentant du groupe (contact principal, lié à l'utilisateur connecté, il peut s'agir du manager)
 - Nom, Prénom, Adresse, Code Postal, email, téléphone
- Style musical (ex : *Rock, Punk, Indie Rock, etc*)
- Année de création
- Présentation du texte (quelques lignes, < 500 caractères)
- Expériences scéniques (quelques lignes, < 500 caractères)
- Site web ou page facebook (url)
- Adresse page soundcloud (url, facultatif)
- Adresse page youtube (url, facultatif)
- Membres du groupe (1 à 8)
 - pour chaque membre : nom, prénom, instruments
- Statut associatif (oui/non)
- Inscrit à la SACEM (oui/non)
- Producteur (oui/non)
- 3 fichiers au format MP3
- un dossier de presse PDF (facultatif)
- 2 photos de groupe avec résolution > 300 dpi
- une fiche technique PDF
- un document SACEM PDF (ou la liste des noms/compositeurs/durée des morceaux de la setlist)

On remarque plusieurs formes d'informations à inscrire que l'on va classer et traduire en outils que l'on pourra implémenter dans notre **.tpl** au langage **HTML**.

Nous avons ici 6 types de saisies :

- Les champs d'entrée de petites chaînes de caractère qu'on traduit par :

```
<input type="text" placeholder="" name="" id="" value="" size="25">
```

La balise **input** permet une saisie utilisateur ici de type texte, selon l'information à recueillir le **nom** et l'**id** prendre le nom de cette information (par exemple : le nom du groupe). **placeholder** permet de « placer » par défaut du contenu pour indiquer à l'utilisateur l'information à rentrer. La **valeur** ne comprendra rien par défaut mais aura dès une première saisie la valeur de l'utilisateur vérifiée (exemple pour une variable nom : `{ $nom | default : " " }`). Enfin **size** fixe une limite en nombre de caractère que l'on peut saisir.

- Les champs d'entrée de texte (paragraphe) qu'on traduit par :

```
<textarea name="" id="" cols="" rows="" maxlength="" placeholder="" value="">Texte</textarea>
```

Ici **name**, **id**, **value** et **placeholder** reprennent et reprendrons la même explication qu'au-dessus et ce pour tous les autres champs de saisies que nous verrons. La balise **textarea** permet la saisie sur plusieurs lignes d'une chaîne de caractère. Sa taille de saisie dépendra de la valeur de **cols** pour le nombre de colonne à afficher, **rows**, le nombre de ligne et **maxlength** qui fixe la limite de caractère pouvant être saisie.

- Les champs d'entrée de nombre qu'on traduit par :

```
<input type="number" min="" max="" name="" id="" value="">
```

La balise **input** est ici de type **number**, elle prend un nombre en entrée d'un minimum **min** et d'un maximum **max**.

- Les champs de sélection qu'on traduit par :

```
<select name="" id="">
  <option value="" selected disabled hidden>Choisissez</option>
  <option value="">Option 2</option>
</select>
```

La balise **select** va ouvrir un champ de sélection où chaque choix possible sera une **option** qui retournera sa valeur et aura son texte (par exemple **Option 2**) pour donner une meilleure compréhension à l'utilisateur. À noter que l'on peut rajouter **selected disabled hidden** pour masquer le choix lors de la sélection, à l'utilisateur. Ainsi cette **option** peut permettre d'indiquer la consigne à suivre pour la sélection (**Option 1**).

- Les champs de case à cocher qu'on traduit par :

```
Oui <input type="radio" name="" id="" value="" checked>
```

Ici la balise **input** est de type **radio**, ce type définit l'input comme un bouton à cocher la valeur de saisie voulue. Ainsi on peut imaginer un deuxième champ **Non** qui retournerait **0** tandis que **Oui** retournerait **1** pour une même valeur définit par l'**id**. L'attribut **checked** permet de cocher par défaut le champ concerné (ici **Oui**).

- Les champs de dépôts qu'on traduit par :

```
<input type="file" name="" id="" accept=".pdf">
```

Ici la balise **input** est de type **file**, elle va prendre en entrée un fichier que l'on stockera temporairement jusqu'au prochain rafraichissement de page ou enregistrement. À noter que pour chaque dépôt il faut spécifier le type de fichier attendu (par exemple ici **.pdf**).

Il ne reste plus qu'à donner à l'utilisateur la possibilité de valider ses saisies pour vérification et enregistrement. Ici nous utiliserons un bouton pour soumettre la candidature :

```
<button name="" id="" type="submit" class="registerbtn"><b>Candidater</b></button>
```

La balise **button** crée un bouton ici de type **submit** pour soumettre. Afin de faire comprendre à l'utilisateur la fonction du bouton, on indique l'action qui sera effectué lors de l'appuie (ici **Candidater**).

On obtient un tableau des saisies classées par type :

<i>input text</i>	<i>textarea</i>	<i>input number</i>	<i>select</i>	<i>input radio</i>	<i>input file</i>
Nom du groupe					Document SACEM du groupe
url du site web du groupe	Présentation du groupe	Année de création du groupe	Type de scene	Statut associatif	Dossier de presse du groupe
url du soundcloud du groupe	Expérience scénique du groupe	Nombre de membre du groupe	Département	Possession document SACEM	Fiche technique du groupe
url de la chaine youtube du groupe			Style musicale	Producteur	Les audios du groupe
					Les photos du groupe

« vérifier la validité des saisies et informer l'utilisateur »

Comment récupérer les valeurs saisies ?

Via nos routes **Flight** que nous avons instancié, la première **GET** permet l'affichage initiale avec les données utiles aux formulaires comme la liste des départements, des types de scène et style musicaux ...La seconde **POST** va récupérer les informations saisies. Ci-après le corps des routes :

```
Flight::route('GET /apply', function() {});  
Flight::route('POST /apply', function() {});
```

On instancie une variable **\$candidature** de type tableau associatif auquel on va attribuer nos saisies acquises par le **POST** via le **Flight ::request** des **data** :

```
$candidature = Flight::request()->data;
```

\$candidature contient donc toutes les valeurs saisies par l'utilisateur. Quand il s'agira de gérer les dépôts nous utiliserons la variable **php \$_FILES** qui stocke en tampon les fichiers déposés et attribut les données dans un tableau associatif.

On initialise un tableau **\$erreur** auquel on ajoutera progressivement les différentes éventuelles erreurs décelées.

Quelles sont les principales vérifications ?

Maintenant que nous avons récupérés les données, il nous faut les vérifier. On conditionne donc selon les types d'entrées. C'est-à-dire que nous allons avoir des contraintes globales mais aussi spécifique.

- Contrainte globale

Pour chacun des champs il faut avoir une valeur, on doit donc faire remarquer quand c'est le cas que l'utilisateur n'a pas complété un champ lorsque sa complétion est obligatoire. Pour cela on utilise la fonction **php empty()** qui retourne vrai si la variable est vide sinon faux :

Donc si le champ est vide on affecte à sa valeur « erreur » le message d'erreur, à noter que la proposition soulignée est effective pour tous les champs du formulaire.

```
if (empty($candidature["nom_groupe"])) $erreurs["nom_groupe"] = "Veuillez renseigner le nom du groupe";
```

Ici l'exemple pour le champ Nom du groupe mais utilisé aussi pour tous les champs de sélection

- Spécifique aux textarea

Pour chaque textarea il faut vérifier si la taille de la chaîne de caractère ne dépasse pas 500 caractères imposés par le sujet. On va utiliser la fonction php `strlen()` qui retourne la taille d'une variable string :

Donc si la taille retourner par `strlen()` dépasse 500 alors on a une erreur.

```
if(strlen($candidature["presentation"]) > 500) $erreurs["presentation"] = "Saisissez au maximum 500 caractères";
```

Ici l'exemple pour le textarea de présentation du groupe

- Spécifique au représentant du groupe

En effet, le représentant est le seul champ qui n'est pas un champ à saisir mais qui si l'utilisateur est connecté se complète automatiquement via un ***SELECT sql*** des données du ***profil*** utilisateur. Il nous faut donc vérifier s'il est connecté ou non. Pour cela on va regarder si par exemple un nom a été instancié dans la variable ***php \$_SESSION***. On utilise pour cela la fonction ***php isset()*** qui regarde si la variable est ***définie*** ou ***non null*** et retourne ***vrai*** dans ce cas sinon ***faux*** :

Donc si la variable nom du représentant est nul on affecte le message erreur :

```
if (!isset($_SESSION["nom"])) $erreurs["representant_groupe"] = "Veuillez vous connecter pour candidater";
```

Ici l'exemple de la session utilisateur

- Spécifique aux url

Dans le cas des url, on vérifie si les chaînes de caractères saisies sont au bon « format » url. Pour cela on implémente la fonction ***php filter_var()*** qui va vérifier la validité d'une variable selon un « filtre » ici ***FILTER_VALIDATE_URL*** :

Donc si l'url n'est pas valide selon le filtre alors on a une erreur.

```
if (!filter_var($candidature["site_web"], FILTER_VALIDATE_URL) === true) $erreurs["site_web"] = "URL invalide";
```

Ici l'exemple de l'url site web ou page facebook

- Spécifique aux cases à cocher

Il nous vérifie pour ces dernières si leur valeur de retour est de ***1*** ou ***0*** sinon c'est que l'utilisateur n'a pas coché. On effectue une condition composée d'un ***ET*** logique pour apprendre si la valeur est différente de 1 et de 0 :

Donc si c'est le cas, il y a erreur.

```
if (($candidature["statut_assoc"] != "1") && ($candidature["statut_assoc"] != "0")) $erreurs["statut_assoc"] = "Merci de cocher une case";
```

Ci-dessus l'exemple pour le statut associatif du groupe

- Spécifique aux dépôts fichiers

Le sujet demande de déposer des fichiers d'un certain type (ou extension). Il faut donc vérifier l'extension du fichier déposer. Pour cela on utilise la fonction **php explode()** qui va nous permettre de découper le **type** stocké dans **\$_FILES** en 2 parties, le **nom** et l'**extension** :

Donc si le fichier a la bonne extension on continue sinon on affecte l'erreur

```
if (explode("/", $_FILES["depot_doc_sacem"]["type"])[1] == "pdf")
{ ... }
else $erreurs["depot_fichier_audio"] = "Veuillez déposer le document SACEM au format pdf";
{ ... }
```

Ici l'exemple pour le dépôt du document SACEM

Il faut aussi vérifier si le fichier est déjà existant au cas où pour éviter d'avoir une erreur de doublon de fichier. On implémente ici la fonction **php file_exists()** qui à un chemin donné regarde si un fichier ou un dossier existe ou non :

Donc si le fichier n'existe pas continue sinon on affecte l'erreur.

```
if (!file_exists($pathCheminDocSACEM))
```

Ici l'exemple pour le dépôt du document SACEM

- Spécifique aux dépôts des photos

Le sujet impose une résolution d'image en dpi de plus de 300 dpi. Pour cela on va récupérer la résolution de la photo via la fonction php **getimagesize()** sur le **tmp_name** de la case **\$_FILES** de notre photo, va retourner en dpi la résolution. On compare cette valeur à 300 :

Donc si la résolution est plus petite que 300 on affecte l'erreur sinon on continue.

```
if (getimagesize($_FILES["depot_photo$i"]["tmp_name"]) > 300)
```

Ici l'exemple pour le dépôt des photos

Après avoir vérifié suivants toutes les conditions, si notre tableau **\$erreur** présente des messages que nous saurons via le retour de :

```
if (empty($erreurs))
```

On indiquera la valeur d'erreur affecté dans les balises **span** correspondant aux valeurs vérifiées :

```
<span>
    <i>{$erreurs.nom_groupe|default:""}</i>
</span>
```

*Ci-dessus l'exemple d'une balise **span** retournant l'éventuelle erreur lié au nom du groupe*

On retournera le formulaire avec les erreurs sans enregistrer la candidature mais en gardant les bonnes saisies actualisées. Cela via un **render** de notre **.tpl** avec les données précédemment recueillies et les données d'erreurs :

```
Flight::render('apply.tpl', array('erreur'=>$erreurs,
'candidature'=>$candidature));
```

« enregistrer les différents éléments »

Après avoir vérifier les données et seulement s'il n'y a pas d'erreur, nous pouvons désormais enregistrer les données et ainsi les rendre accessible. Pour cela nous suivons 4 étapes :

- Etape 1 : On enregistre les données de candidature dans la table candidature

Lors de la vérification, s'il n'y avait pas d'erreurs, la variable a été stocké dans un tableau associatif **\$candidatureAInsert** s'il elle avait reçu ou devait recevoir une modification dans le code (traduction de 0 en Non, ...) sinon elle restait dans **\$candidature**. Il ne suffit juste qu'à préparer une requête **sql INSERT INTO** notre table **candidature** avec pour id le dernier **id récupérer + 1**. Les autres données étant collectées dans **\$candidature** et **\$candidatureAInsert**.

- Etape 2 : On enregistre chacun des membres du groupe dans la table groupe

Ensuite pour le représentant et chacun des membres du groupe, on effectue de même une requête **sql INSERT INTO** notre table groupe avec pour id le dernier **id récupérer + 1** et pour id de groupe **l'id** de la candidature. Les autres données étant collectées dans **\$candidature** et **\$candidatureAInsert**.

- Etape 3 : On enregistre tous les fichiers déposés dans le dossier du groupe dédié localement

Après avoir renommé chaque fichier selon le format suivant :

titre_idcandidature_nomgroupe.extension

(Exemple : le document SACEM pour notre groupe ACDC enregistré ->

docSACEM_1_ACDC.pdf).

On va déplacer le fichier placé dans le tampon et renommé vers le dossier du groupe créé par l'instruction suivante :

```
if (!file_exists($pathCandidatureGroupe)) mkdir($pathCandidatureGroupe);
```

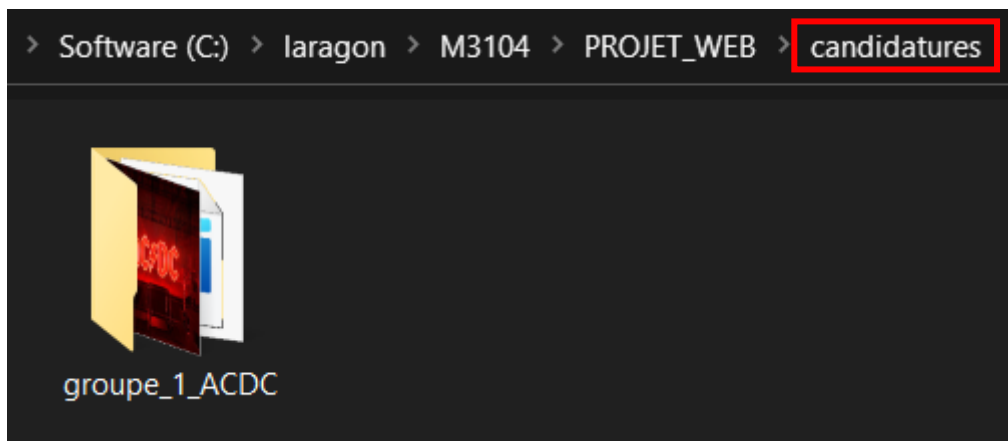
Et donc renommé et déplacé par les instructions qui suivent :

```
$candidatureAInsert["depot_fiche_technique"] = "fichetechnique" . "_" . $id .
"_" . $candidature["nom_groupe"] . "." . explode("/",
$_FILES["depot_fiche_technique"]["type"])[1];
```

```
move_uploaded_file($_FILES["depot_fiche_technique"]["tmp_name"],
$pathCheminFicheTechnique);
```

Ici l'exemple pour le fichier de la fiche technique

.git	20/12/2021 12:27	Dossier de fichiers	
bdd	20/12/2021 12:23	Dossier de fichiers	
candidatures	20/12/2021 12:15	Dossier de fichiers	
css	20/12/2021 09:33	Dossier de fichiers	
includes	20/12/2021 09:33	Dossier de fichiers	
routes	20/12/2021 10:56	Dossier de fichiers	
templates	20/12/2021 11:58	Dossier de fichiers	
templates_c	20/12/2021 12:14	Dossier de fichiers	
.htaccess	20/12/2021 09:33	Fichier HTACCESS	1 Ko
index.php	20/12/2021 10:58	Fichier source PHP	2 Ko
pdo.php	20/12/2021 09:33	Fichier source PHP	1 Ko
README.md	20/12/2021 12:26	Fichier source Mar...	2 Ko



audio1_1_ACDC.mpeg	20/12/2021 12:15	Fichier MPEG	4 100 Ko
audio2_1_ACDC.mpeg	20/12/2021 12:15	Fichier MPEG	3 317 Ko
audio3_1_ACDC.mpeg	20/12/2021 12:15	Fichier MPEG	5 146 Ko
docSACEM_1_ACDC.pdf	20/12/2021 12:15	Foxit PDF Reader ...	208 Ko
dossierpresse_1_ACDC.pdf	20/12/2021 12:15	Foxit PDF Reader ...	1 473 Ko
fichetechnique_1_ACDC.pdf	20/12/2021 12:15	Foxit PDF Reader ...	2 844 Ko
photo1_1_ACDC.jpeg	20/12/2021 12:15	Fichier JPEG	125 Ko
photo2_1_ACDC.jpeg	20/12/2021 12:15	Fichier JPEG	51 Ko

- Etape 4 : On redirige vers une page de confirmation de prise en compte de la candidature

On effectue un **redirect** vers notre route **/success** :

```
Flight::redirect("/success");
```

Rendu sur la page web :

RMFire/IUT/Projet_Web x Candidater x localhost / localhost / projet_w... x +

localhost/Projet_Web/apply

Nom du groupe :

Représentant du groupe :

email : brianjohnson@gmail.com

role : candidat

nom : Johnson

prenom : Brian

adresse : Dunston, Royaume Uni

code_postal : 99999

telephone : 725145689

Renseignez le ou les membres du groupe sans compter le représentant, 8 membres comptant le représentant maximum

Année de création du groupe :

Type de scene :

Département :

Présentation du groupe :

Expérience scénique du groupe :

Style musical du groupe :

Site web du groupe :

Soundcloud du groupe :

Chaine Youtube du groupe :

Statut associatif : ☐ Oui ☐ Non

Liste et candidatures détaillées

Routes des listes :

La route GET /liste permet d'envoyer l'utilisateur sur la page de liste des candidatures. Cette route va permettre aussi de récupérer des données primordiales à stocker dans le cache du navigateur web de l'utilisateur. Il s'agira du contenu des listes.

```
Flight::route('GET /liste', function(){  
    });
```

Dans un premier temps, nous allons déclarer une variable BDD qui appellera la base de données : Ensuite, nous allons vérifier que la session ne soit pas vide, si c'est bien le cas, alors nous allons récupérer le type de l'utilisateur connecté. Ainsi, seul un membre du staff (« responsable » ou « administrateur ») peut accéder à la liste.

```
$requeteTypeSession = $BDD->prepare("SELECT `type` FROM utilisateur WHERE email = ?");  
  
if(($typeConnexion == "responsable") || ($typeConnexion == "administrateur")){
```

L'utilisateur est bien un membre du staff, nous allons pouvoir créer une requête sql :

Nous allons chercher les champs id, nom, departement, style, annee, presentation et experience depuis la table candidature. Ce seront les informations affichées dans la liste.

Cette requête sera exécutée. Afin de récupérer le tableau de données, nous allons créer une variable nommée ListeCandidature. fetchAll() va permettre de tout récupérer en conservant le format des données par l'attribut PDO ::FETCH_ASSOC.

```
$requeteNomCandidatures = $BDD->prepare("SELECT  
id, nom, departement, style, annee, presentation, experience FROM candidature");  
$ListeCandidature = $requeteNomCandidatures->fetchAll(PDO::FETCH_ASSOC);
```

Pour finir, on va assigner cette variable à un nom qui sera utilisé dans le template. C'est la méthode assign. En dernier temps, la méthode render renverra l'utilisateur vers le template liste.tpl avec un tableau array() vide. -> Les données sont assignées dans une vue.

```
Flight::view()->assign('ListeCandidature', $ListeCandidature);  
Flight::render('liste.tpl', array());
```

Contenu du template

Le template est composé d'un seul tableau créé par balises HTML. Il va afficher les données par champs de table. Et ce, pour chaque candidature. Cela est réalisé à l'aide d'une fonction foreach.

```
{foreach from=$ListeCandidature item=$candidature}  
    {$id = $candidature['id']}  
    <tr>  
        <td class = "col">  
            {$candidature['id']}
```

```

        </td>
        <td class = "col">
            <a href="./candidature-{$id}">détails</a>
        </td>
    </tr>

```

Foreach permet de passer par toutes les candidatures enregistrées dans la base de données. L'attribut from permet de récupérer la liste de candidatures de la variable ListeCandidature, elle contient le retour de la requête sql. L'attribut item contient tous les champs de chacune des candidatures.

La balise <tr> permet de créer une ligne et les attributs <td> une colonne d'un tableau.

Chacune des colonnes va contenir un champ par récupération de celui-ci dans la liste.

-> tableau associatif candidature['nomDuChamp'].

Nous pouvons afficher tous les champs recueillis par la requête sql. Enfin, une ligne attire notre attention :

```
<a href="./candidature-{$id}">détails</a>
```

Cette ligne permet d'envoyer l'utilisateur sur une page où apparait une liste détaillée des informations sur cette candidature. L'ID est passé dans le chemin et donc vers la route contenue dans routes.php.

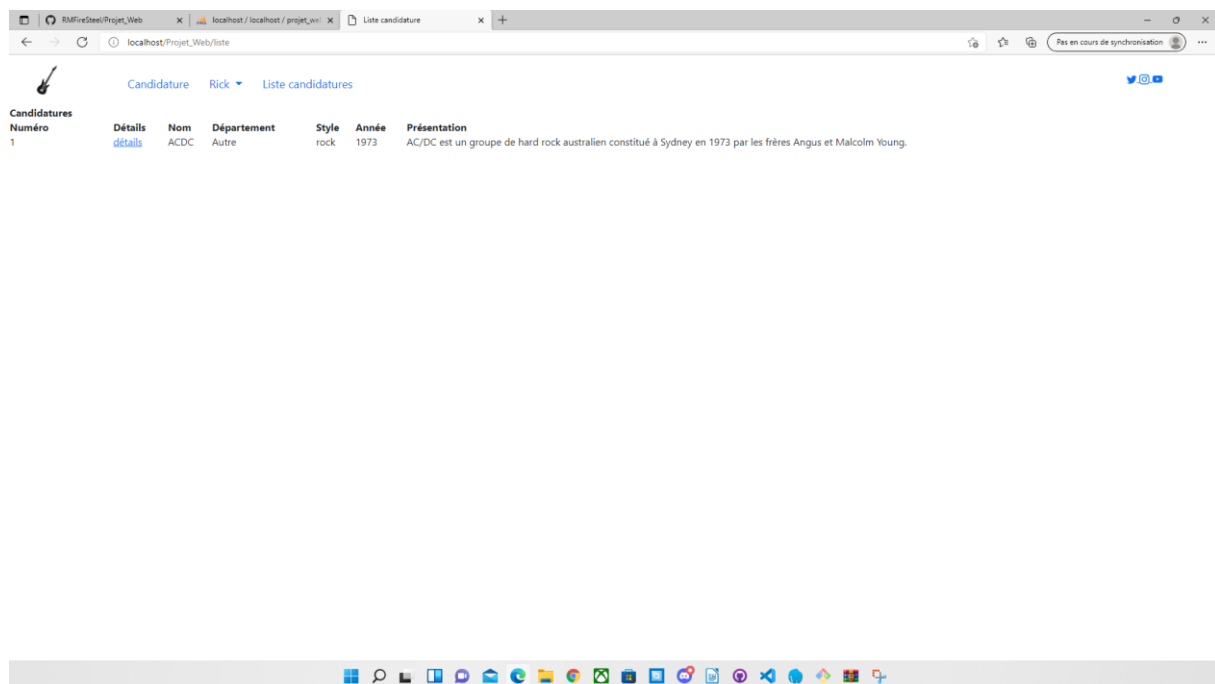
Résultat sur page web :

Affichage avec du css natif

Menu de navigation :

[Accueil](#) [Liste](#)

Numéro	Détails	Nom	Département	Style	Année	Présentation
1	détails	Imagine	Somme	Rock	2007	Imagine est un groupe de rock
2	détails	King	London	Metal	1970	King is a group at the service of her majesty the queen
3	détails	orchestraliss	Rhin	Orchestra	2003	Orchestraliss ist ein Group von Deutschland. Sie spiele Orchestral musiche
4	détails	Coldplay	NewYork	Pop/Rock	2001	Coldplay is a group from Queens in NY

Affichage final avec bootstrap

Ainsi, les champs principaux sont représentés pour chacune des candidatures. Une colonne en particulière attire l'attention -> la colonne Détails composées de liens. C'est ce que nous verrons dans la prochaine partie.

```
<td class="col"> 1 </td>
▼ <td class="col">
  <a href="._/candidature-1">détails</a>
</td>
<td class="col"> Imagine </td>
<td class="col"> Somme </td>
<td class="col"> Rock </td>
<td class="col"> 2007 </td>
▼ <td class="col">
  " Imagine est un groupe de rock "
</td>
```

Voici la page envoyée au lecteur WEB.

Les champs sont remplacés par les chaînes de caractères contenues dans la base de données.

Nous pouvons remarquer que le lien vers les détails a son @id remplacé par l'id du groupe en question.

Ainsi ce lien emmènera sur la route listeDetaillée avec l'id.

Route de la liste détaillée :

```
Flight::route('/candidature-@id', function($id){
});
```

La route de la liste candidature est particulière. Elle permet de renvoyer sur la page détaillée d'une candidature. Le chemin se compose ainsi : /nomDuTemplate-@id. L'ID présent dans le chemin représente le numéro de la candidature qui sera à détailler. Il est récupéré dans le chemin de fichier puisque l'utilisateur a cliqué sur un lien avec l'ID d'un groupe.

Dans le même cas que la route GET /liste, nous allons tester que l'utilisateur connecté soit bien un membre du staff, ou le responsable du groupe et de la candidature :

```
$requeteSessionCandidat = $BDD->prepare(
"SELECT id FROM candidature WHERE email = ?");
```

Nous allons donc récupérer l'id de groupe de la personne connecté

```
$requeteTypeSession = $BDD->prepare(
"SELECT `type` FROM utilisateur WHERE email = ?");
```

Mais aussi le type utilisateur, comme dans la route menant à la liste.

Cette condition vérifie que la personne possède les droits pour accéder au contenu :

```
if(($IdDeConnexion == "{$id}") || ($typeConnexion == "responsable") ||
($typeConnexion == "administrateur")){
```

Une fois vérifier, nous récupérons les données demandées dans la base de données avec les requêtes suivantes :

```
$requeteDetails = $BDD->prepare(
"SELECT id, nom, departement, email, style, annee, presentation,
      experience, urlgroupe, soundcloud FROM candidature WHERE id = ?");
```

Cette requête permet de récupérer les champs de la base de données qui ne sont ni de type booléen, ni de type fichiers. En effet, séparer les 3 types permettra une meilleure gestion des données dans le template.

```
$requeteBool = $BDD->prepare(
"SELECT association, sacem, producteur FROM candidature WHERE id = ?");
```

Cette requête permet de récupérer les champs qui sont de type booléen

-> Association, sacem et producteur.

```
$requeteFichiers = $BDD->prepare(
"SELECT fichier1, fichier2, fichier3, dossier_presse,
      photo1, photo2, fiche_technique, doc_sacem FROM candidature WHERE id = ?");
```

Cette requête permet de récupérer tous les fichiers qui ont été enregistrées lors de l'enregistrement de la candidature.

```
$requeteMembres = $BDD->prepare("SELECT id_membre, nom, prenom, instrument
      FROM groupe WHERE id_groupe = ?");
```

La dernière requête sql va récupérer les id des membres du groupe, ainsi que leurs noms, prénoms, instruments dans la table groupe.

Afin de faire passer toutes ces données, nous allons les assignées dans des variables qui seront récupérables dans le template.

```
Flight::view()->assign('BoolCandidatureGroupe', $CandidatureBool);
Flight::view()->assign('InfosCandidatureGroupe', $CandidatureDetails);
Flight::view()->assign('FichiersCandidatureGroupe', $CandidatureFichiers);
Flight::view()->assign('MembresGroupe', $Membres);
```

```
Flight::render('candidatureDetails.tpl', array());
```

Enfin, l'utilisateur est envoyé sur la page de candidature détaillée.

Contenu du template candidatureDetails

Le début du template commence par ces valeurs assignées, nous réassignons les valeurs récupérées dans la route à de nouvelles. Par défaut, elles seront vides en cas d'erreur.

```
{assign var="Candidature" value=$InfosCandidatureGroupe|default:[]}
{assign var="BoolCandidature" value=$BoolCandidatureGroupe|default:[]}
{assign var="FichiersCandidature" value=$FichiersCandidatureGroupe|default:[]}
{assign var="Membres" value=$MembresGroupe|default:[]}
```

Après le titre, nous allons afficher la photo du groupe, dont le chemin est stocké dans le champ photo1 de la table candidature. -> La chaîne équivaut à « photo1_id_nomDuGroupe.format »

Ainsi, nous nous plaçons dans le dossier candidatures à l'aide de ce lien dynamique, puis dans le dossier concernant le groupe -> de forme « groupe_id_nomDuGroupe ». Dans ce dossier se trouvera notre photo, il suffit donc d'utiliser le champ photo1.

```
<div class="ImageGroupe">

</div>
```

La source de l'image devient : « ./candidatures/groupe_id_nomDuGroupe/photo1_id_nomDuGroupe »

Ensuite, les données de la table candidature sont affichées sous forme de tableau. Les 4 groupes de données sont montrés dans cet ordre :

- Les informations du groupe
- Les informations en booléen du groupe
- Les fichiers donnés par le groupe
- Les informations sur les membres du groupe.

De la même manière que le template Liste, nous allons afficher/générer le tableau de valeurs avec des boucles foreach() qui vont utiliser les variables assignées dans la route.

A l'opposé du tableau du template Liste qui fonctionnait avec une candidature par ligne et les champs en colonne, le tableau de ce template fonctionne avec un champ par ligne (entête + contenu). Il faut donc utiliser 2 boucles foreach() dont la première va découper tous les champs en items numérotés par key. Une deuxième boucle va découper cet item en un entête et le champ.

```
{foreach from=$Candidature key=$key item=$value} //Découpage champ par champ
    {foreach from=$value key=$entete item=$infoProfil} //Découpage des champs
```

Nous allons développer la partie qui récupère les booléens ainsi que les fichiers.

Récupération des booléens :

```
<th class = "infoDetails">
    {if $infoProfilBool == "1"} //Correspond au champ
        Oui
    {/if}
    {if $infoProfilBool == "0"}
        Non
    {/if}
</th>
```

A défaut d'afficher 0 ou 1, nous allons afficher Oui ou non dans la case. Cette condition if permet de réaliser cet affichage. Dans le cas où le groupe n'est pas enregistré à la SACEM, l'entête Sacem apparaîtra dans le tableau mais aucun fichier ne sera présenté.

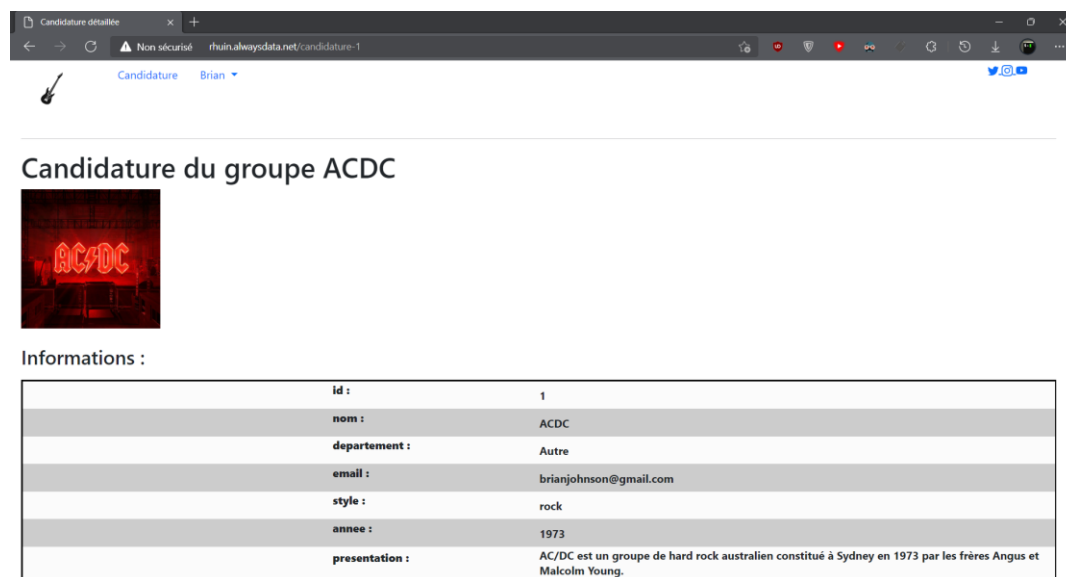
Récupération et ouverture des fichiers dans le navigateur :

```
<a href="./candidatures/groupe_{$Candidature[0].id}_{$Candidature[0].nom}
/{$infoProfilFichier}"
target="_blank"> {enteteFichier}</a>
```

Voici la ligne qui permet de présenter le fichier à visualiser. Tout comme l'image vu plus haut, l'id de la candidature ainsi que le nom du groupe sont utilisés. La variable \$infoProfilFichier de forme suivante :

- Audio1_id_nomDuGroupe
- Audio2_id_nomDuGroupe
- Photo1_id_nomDuGroupe
- Sacem_id_nomDuGroupe
- Dossier_presse_id_nomDuGroupe


L'extension est mise en fin de chemin dans la base de données, ainsi il est possible d'ouvrir tout type de fichier. L'attribut target le permet, il permet la création du nom du fichier visualisé dans l'onglet, que l'on nomme simplement comme il est nommé dans le dossier candidatures/groupe avec la variable \$infoProfilFichier. L'attribut download permet aussi de télécharger le fichier sur l'ordinateur de la personne connecté.

Rendu complet sur page web :


Candidature

Brian

Candidature du groupe ACDC



Informations :

id :	1
nom :	ACDC
departement :	Autre
email :	brianjohnson@gmail.com
style :	rock
annee :	1973
presentation :	AC/DC est un groupe de hard rock australien constitué à Sydney en 1973 par les frères Angus et Malcolm Young.

Candidature détaillée

Non sécurisé | rhain.alwaysdata.net/candidature-1

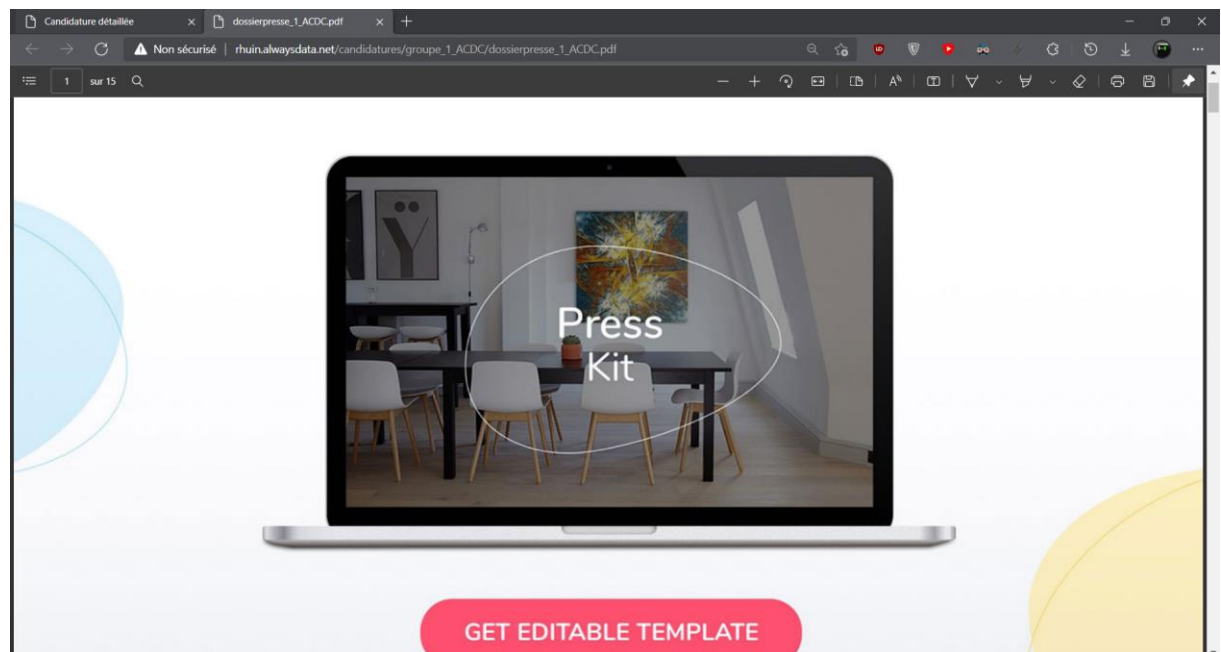
soundcloud :	https://www.youtube.com/channel/UCB0JSO6d5ysH2Mmqz5I9rlw
association :	Oui
sacem :	Oui
producteur :	Oui

Fichiers :

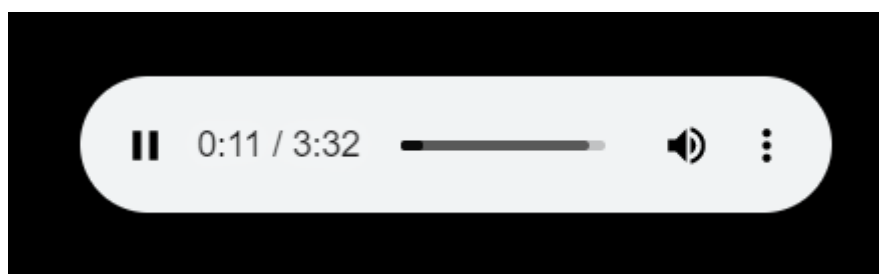
fichier1 :	fichier1
fichier2 :	fichier2
fichier3 :	fichier3
dossier_presse :	dossier_presse
photo1 :	photo1
photo2 :	photo2
fiche_technique :	fiche_technique
doc_sacem :	doc_sacem

Membres :

Numéro	Nom	Prénom	Instrument
1	Johnson	Brian	
2	Young	Angus	guitar solo
3	Young	Malcolm	guitar rythmique
4	Gregg	Paul	basse
5	Chris	Slade	batterie



Fichier pdf ouvert en cliquant sur le lien dossier_presse.



Lecture d'un fichier mp3 ou mp4 dans le navigateur à partir du lien.

Conclusion

Ce TP nous a permis d'apprendre et comprendre le fonctionnement de Smarty et de Flight. Les divers services que nous avons mis en place sont intéressants et sauront se montrer utiles à l'avenir. Durant ce projet, nous avons coopéré dans la conception fonctionnelle du site en partageant équitablement le travail entre nous trois. Bien sûr, l'analyse du sujet et la création de la base de données ont été vus ensemble afin de partir sur une base commune et stable.