Audio File player

I.FA-P3A

Table des matières

[1 Introduction 3](#_Toc485602625)

[1.1 Motivations 3](#_Toc485602626)

[1.2 Organisation 3](#_Toc485602627)

[2 Cahier des charges 4](#_Toc485602628)

[2.1 Objectif 4](#_Toc485602629)

[2.2 Fonctionnalités principales 4](#_Toc485602630)

[2.3 Modèle conceptuel 4](#_Toc485602631)

[2.4 Planning initial 5](#_Toc485602632)

[2.5 Technologies & logiciels utilisées 5](#_Toc485602633)

[3 Analyse concurrentielle 6](#_Toc485602634)

[3.1 iMusic 6](#_Toc485602635)

[3.2 Google Play Music 7](#_Toc485602636)

[4 Analyse fonctionnelle 8](#_Toc485602637)

[4.1 Plan du site 8](#_Toc485602638)

[4.1.1 Login – inscription 8](#_Toc485602639)

[4.1.2 Home 8](#_Toc485602640)

[4.1.3 Profil 8](#_Toc485602641)

[4.1.4 Amis 8](#_Toc485602642)

[4.1.5 Audiothèque 8](#_Toc485602643)

[4.2 Description de l’interface 9](#_Toc485602644)

[4.2.1 Eléments statiques 9](#_Toc485602645)

[4.2.2 Page de login 10](#_Toc485602646)

[4.2.3 Page d’inscription 10](#_Toc485602647)

[4.2.4 Page d’accueil 11](#_Toc485602648)

[4.2.5 Page des artistes (audiothèque) 11](#_Toc485602649)

[4.2.6 Page des albums (audiothèque) 12](#_Toc485602650)

[4.3 Fonctionnalités détaillées 12](#_Toc485602651)

[4.3.1 Connexion 12](#_Toc485602652)

[4.3.2 Inscription 12](#_Toc485602653)

[4.3.3 Upload de fichiers .mp3 12](#_Toc485602654)

[4.3.4 Vérification des métadonnées ID3 12](#_Toc485602655)

[4.3.5 Affichage de l’audiothèque 13](#_Toc485602656)

[4.3.6 Lecture de fichiers .mp3 13](#_Toc485602657)

[4.3.7 Création de listes de lecture 13](#_Toc485602658)

[4.3.8 Modification de son profil 13](#_Toc485602659)

[4.3.9 Système d’amis 13](#_Toc485602660)

[5 Analyse organique 14](#_Toc485602661)

[5.1 Modèle conceptuel final 14](#_Toc485602662)

[5.2 Modèle physique des données 14](#_Toc485602663)

[5.3 Le choix du single-page 15](#_Toc485602664)

[5.4 Architecture du site 15](#_Toc485602665)

[5.4.1 Routeur 16](#_Toc485602666)

[5.4.2 Contrôleurs 17](#_Toc485602667)

[5.4.3 Vues 17](#_Toc485602668)

[5.4.4 Arborescence 17](#_Toc485602669)

[5.4.5 API jsmediatags 17](#_Toc485602670)

[5.5 Fonctions remarquables 18](#_Toc485602671)

[5.5.1 Upload de fichiers 18](#_Toc485602672)

[5.5.2 Lecture de musique 20](#_Toc485602673)

[5.6 Requêtes SQL notables 22](#_Toc485602674)

[5.6.1 Récupération des artistes (nom, id et nombre d’albums) d’un utilisateur 22](#_Toc485602675)

[5.6.2 Récupération des titres d’un album lié à un utilisateur 22](#_Toc485602676)

[5.6.3 Récupération des amis d’un utilisateur et leur nombre de titres et d’albums 22](#_Toc485602677)

[5.7 Sécurité 23](#_Toc485602678)

[5.7.1 Formulaires 23](#_Toc485602679)

[5.7.2 Upload 24](#_Toc485602680)

[5.7.3 Requêtes SQL 25](#_Toc485602681)

[6 Rapport de tests 26](#_Toc485602682)

[7 Conclusion 26](#_Toc485602683)

[8 Annexes 26](#_Toc485602684)

[9 Sources 26](#_Toc485602685)

# Introduction

Actuellement en **dernière année** de mes 3 années de formation en **informatique** au sein du Centre de Formation Professionnel Technique (CFPT), j’ai pour tâche d’effectuer un **travail de fin de diplôme**. Ce travail, nommé **TPI**, est nécessaire pour l’obtention de mon **Certificat Fédéral de Capacité** (CFC).

## Motivations

Passionné depuis toujours par la musique, j’ai décidé de réaliser, dans le cadre de mon projet de TPI, une application liée à cette passion pour l’art musical : **AudioFile Player**. Cette application est un **lecteur de fichiers audio** qui a une particularité : les fichiers sont sur un **serveur distant**.

**Pourquoi un système de cloud ?** Tout d’abord, afin de **centraliser** en un point central ses données. En effet, aujourd’hui, beaucoup jonglent entre leur ordinateur et leur smartphone pour écouter de la musique. Avoir ses fichiers sur un seul serveur distant permet ainsi de faciliter l’écoute de sa bibliothèque musicale.

De plus, ce système de cloud est également avantageux pour les personnes ayant une audiothèque volumineuse. En effet, sur un périphérique comme le smartphone, le stockage peut vite être saturé par les applications, vidéos etc. Accéder à son audiothèque par le biais du réseau de données peut ainsi être une solution afin **d’économiser de la place** sur son smartphone.

## Organisation

|  |  |
| --- | --- |
| *Élève :*  Michael RAMUSI  [ramusim@protonmail.ch](mailto:ramusim@protonmail.ch) | *Maître d’apprentissage :*  Jasmina TRAVNJAK  [edu-travnjakj@eduge.ch](mailto:edu-travnjakj@eduge.ch) |
| *Experts :*  Eric FEHR  [eric.fehr@skyguide.ch](mailto:eric.fehr@skyguide.ch) | Daniel VANINI  [daniel.vanini@skyguide.ch](mailto:daniel.vanini@skyguide.ch) |

Ce travail de TPI sera réalisé du 6 mai au 19 mai 2017. La réalisation du projet se déroule sur 10 jours de 8h pour un total de 80 heures de travail.

# Cahier des charges

## Objectif

L’objectif de ce projet est de créer une **application web** qui permet à un utilisateur de **déposer des fichiers musicaux** afin qu’il puisse avoir accès à ceux-ci partout, à la manière d’un **cloud**.

## Fonctionnalités principales

* Fonctions d’inscription et de connexion
* Dépôt de fichiers au format .mp3
* Vérification des métadonnées (ID3) des fichiers déposés
* Lecture des fichiers de l’audiothèque de l’utilisateur
  + Play, pause, suivant, précédent
* Affichage des musiques selon plusieurs tris
  + Par artiste
  + Par album
  + Par titre
* Création de listes de lecture (playlist)
* Fonctionnalités sociales :
  + Modification de profil
  + Ajout d’amis

## Modèle conceptuel

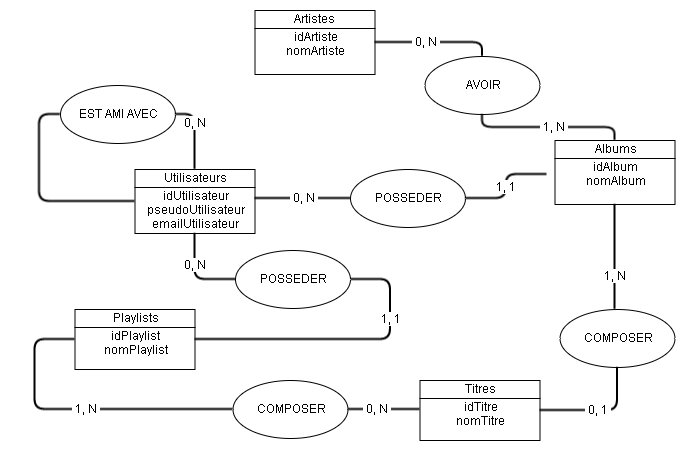


Figure 1: MCD AudioFile Player

Ceci est le modèle conceptuel de données défini lors de la rédaction du cahier des charges. Le travail se base sur ce modèle pour la réalisation de la base de données en MySQL.

## Planning initial

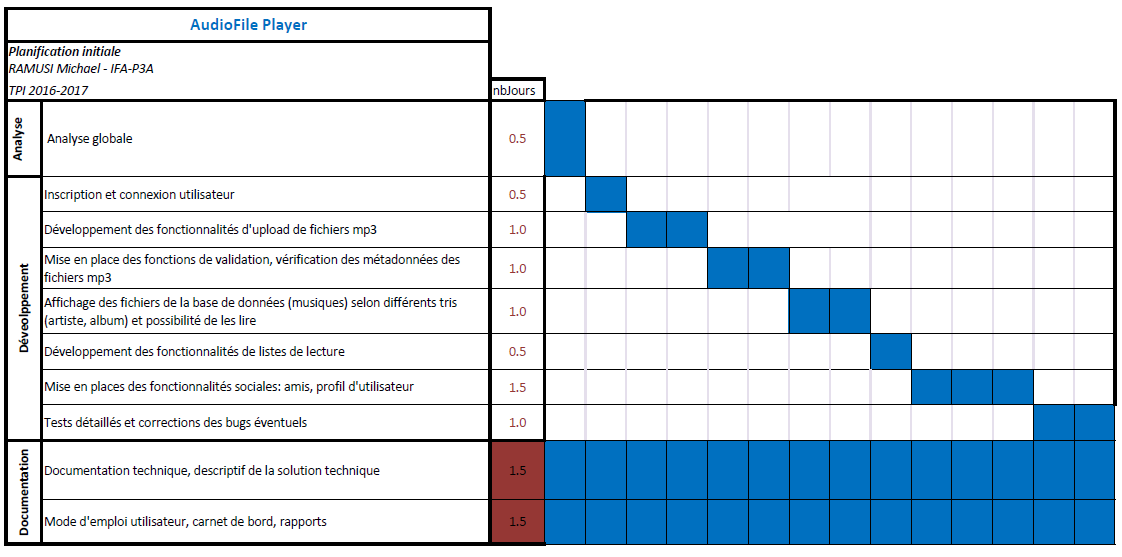


Figure 2: planning initial

Ceci est le déroulement des tâches nécessaires à la réalisation du projet. Comme énoncé plus haut, le travail est réalisé en 80 heures / 10 jours.

## Technologies & logiciels utilisées

Pour la réalisation de cette application j’ai décidé d’utiliser des technologies **web** par **préférence personnelle** d’abord mais également pour faciliter un éventuel passage de l’application sur une **plateforme mobile**:

* PHP
* JavaScript (jQuery)
* HTML/CSS
* MySQL

Les logiciels utilisés :

* IDE : *NetBeans IDE*
* Serveur local : *EasyPHP*
* Navigateur : *Google Chrome & Mozilla Firefox*
* Gestion de version : *GIT*
* Maquette : *Balsamiq*

# Analyse concurrentielle

## iMusic

L’application **iMusic** est un outil puissant pour gérer sa collection de musiques. En plus de pouvoir déposer ses propres titres, l’application permet d’écouter des musiques à partir d’autres logiciels comme Spotify ou YouTube. iMusic propose également sa propre librairie musicale, afin de pouvoir écouter de la musique populaire sans avoir à la télécharger.

Egalement, l’application permet une **synchronisation** **locale** entre différents périphériques. Cette synchronisation locale diffère donc de la lecture en streaming de mon application.

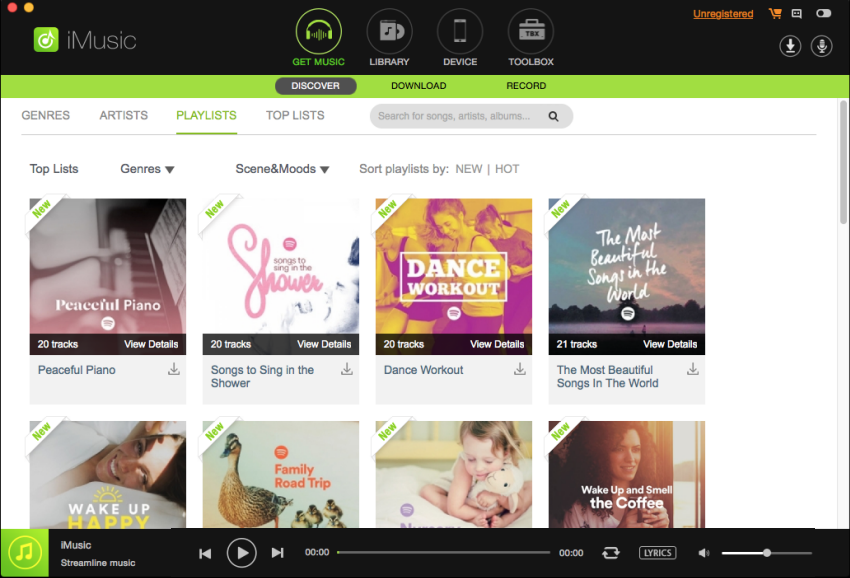


Figure 3: iMusic

## Google Play Music

L’application musicale signée Google. C’est une application robuste qui permet de lire ses fichiers musicaux mais également d’aller les chercher sur le magasin de titres de chez Google.

La force de l’application est de pouvoir **accéder à une bibliothèque** **depuis plusieurs périphériques** grâce aux serveurs de Google qui offre la possibilité d’avoir **20'000 titres disponibles en streaming.** L’application ne gère cependant pas la **validation des métadonnées du fichier**, celle-ci doit donc se faire en amont.

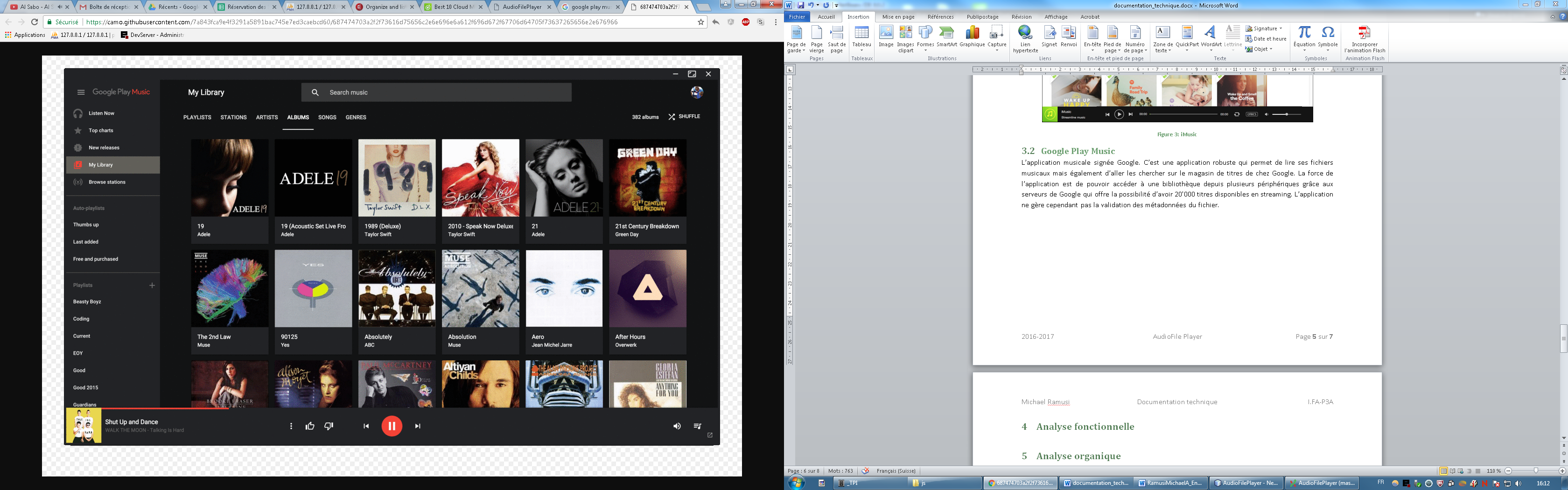


Figure 4: Google Play Music

# Analyse fonctionnelle

## Plan du site

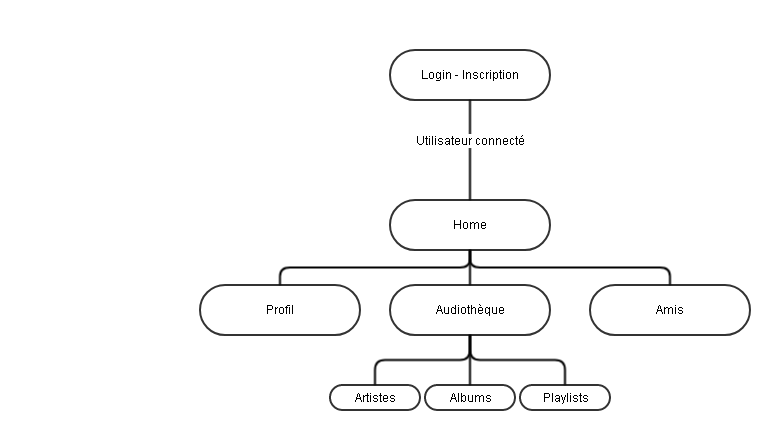


Figure 5: diagramme plan du site

### Login – inscription

Cette page est celle qui s’affiche au lancement du site. Elle permettra à l’utilisateur de faire deux actions : la première est la **connexion**, la deuxième est **l’inscription** au site.

Une fois la connexion réussie, l’utilisateur est redirigé sur Home.

### Home

La page Home est la première page qui est affiché à l’utilisateur lorsqu’il est connecté. Sur cette page, des **informations concernant l’audiothèque** de l’utilisateur sont affichés.

### Profil

Cette page est celle qui s’occupe de la **gestion du profil de l’utilisateur**. Ce dernier pourra effectuer un changement de son avatar et son pseudo, seules possibilités envisagées à l’heure actuelle.

### Amis

Amis est la page des **fonctions sociales**. Sur cette page, l’utilisateur peut ajouter des amis, c’est-à-dire d’autres utilisateurs, et afficher la liste de ses amis. Il peut également voir la liste des albums de ses amis.

### Audiothèque

Les sous-pages d’Audiothèque, comme son nom l’indique, s’occupent de **gérer la collection de titres musicaux** de l’utilisateur de plusieurs façons **:** tri par artiste, tri par album et gestion des playlists.

#### Artistes

Sous-page d’ « Audiothèque », Artistes **affiche la liste des artistes** représentés dans la collection musicale de l’utilisateur.

#### Albums

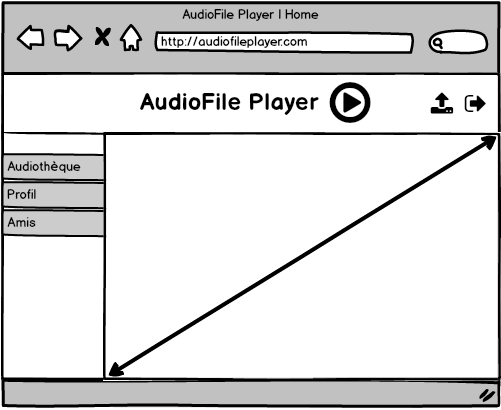
Sous-page d’ « Audiothèque », Albums **affiche la liste des albums** représentés dans la collection musicale de l’utilisateur.

#### Playlists

C’est sur cette page que l’utilisateur peut **gérer ses listes de lecture** : création de liste, suppression de liste et affichage de liste.

## Description de l’interface

### Eléments statiques



Eléments dynamiques

Figure 6: Maquette éléments statiques/dynamiques

Les éléments statiques représentent les composants du site qui ne varieront pas lors de la navigation entre les pages. Ces éléments sont : la bannière du site, la navigation du site. **Ce squelette est utilisé après la connexion de l’utilisateur.**

Bannière :

* Titre / logo du site
* Bouton d’upload de fichiers
* Bouton de déconnexion

Navigation :

* Pages du site

### Page de login

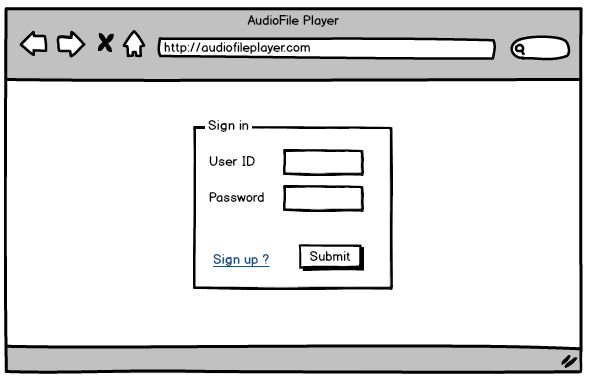


Figure 7: maquette page de login

Composants notables :

* Fenêtre modale statique
* Formulaire
* Lien

La page de login est composée d’une fenêtre modale statique placée au centre de l’écran. Cette fenêtre affiche un formulaire contenant les deux champs de base permettant la connexion : champ du nom de l’utilisateur, champ du mot de passe.

En dessous de ces champs se trouvent le bouton d’envoi du formulaire ainsi qu’un lien vers la page d’inscription.

### Page d’inscription

Composants notables :

* Fenêtre modale statique
* Formulaire
* Lien

La page d’inscription est sensiblement la même que celle de connexion : un champ de confirmation du mot de passe est rajouté. Le lien, quant à lui, redirige vers la page de connexion.

### Page d’accueil

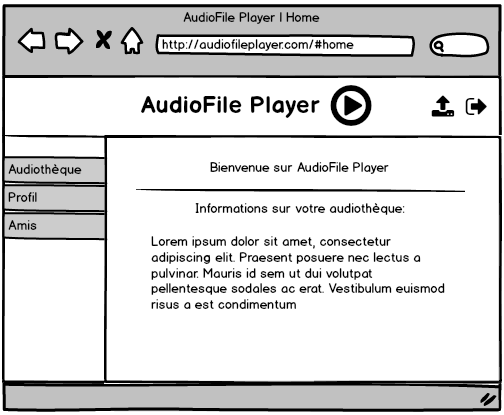


Figure 8: maquette homepage

Composants notables :

* Labels d’affichage

La page d’accueil comporte exclusivement des composants textes d’affichage. Les différentes informations affichées à l’utilisateur concernent son audiothèque : nombre d’artistes, d’albums, de titres etc.

### Page des artistes (audiothèque)

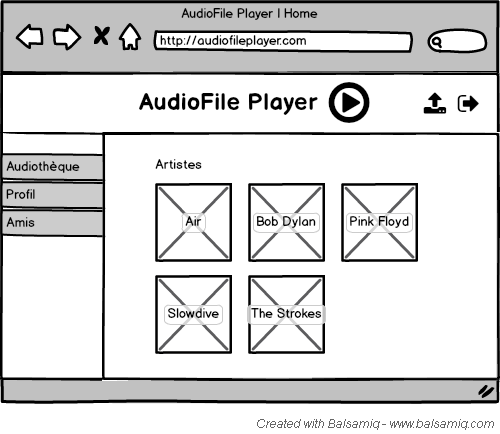


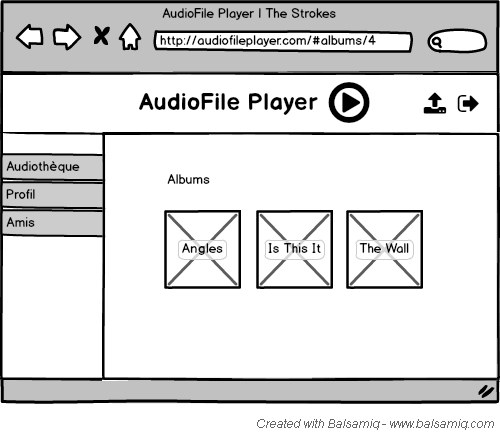
Figure 9: maquette page des artistes

Composants notables :

* Tuiles clickables

La page des artistes affiche tous les artistes que l’utilisateur a dans son audiothèque. L’affichage se fait par un système de tuile. Appuyer sur une tuile envoie sur la page albums de l’artiste.

### Page des albums (audiothèque)



Composants notables :

* Tuiles clickables

La page des albums affiche tous les albums que l’utilisateur a dans son audiothèque. L’affichage se fait par un système de tuile. Appuyer sur une tuile envoie sur la page des titres de l’album.

## Fonctionnalités détaillées

Dans ce chapitre, les fonctionnalités de l’application seront détaillées.

### Connexion

Lors de l’arrivé d’un visiteur sur l’application, c’est la page de connexion qui s’affiche en premier lieu. Ici, l’utilisateur se trouve face à un formulaire classique avec comme champs son identifiant ainsi que son mot de passe.

Il a ainsi la possibilité d’effectuer une identification afin d’avoir accès à son audiothèque personnelle.

### Inscription

Le visiteur de l’application a la possibilité de s’inscrire sur le site. Pour cela, il doit remplir le formulaire avec les valeurs adéquates. Après cela, l’utilisateur est ajouté dans la base de données de l’application et peut désormais accéder au reste du site.

### Upload de fichiers .mp3

AudioFile Player permet d’accéder à ses fichiers .mp3 n’importe où. Pour cela, l’utilisateur doit passer par une étape de transfert de fichiers avec le serveur.

L’utilisateur peut ainsi choisir des fichiers sur son ordinateur qui seront ensuite transférés sur le serveur de l’application.

### Vérification des métadonnées ID3

Un fichier .mp3 possède des métadonnées, des informations qui lui sont propres. Ces données sont : le nom de l’artiste, le nom de l’album, le nom de la piste, le nom du genre, entres autres.

Pour s’assurer d’avoir des données cohérentes autant pour l’utilisateur que pour l’application, il est nécessaire que ces métadonnées soient remplies.

Actuellement,le standard en vigueur pour les fichiers mp3 est l’ID3. L’application se sert de cela pour vérifier si les métadonnées du titre uploadé ne sont pas vides.

Si les métadonnées sont déjà remplies, l’upload se fait. Sinon, l’utilisateur est invité à modifier les fichiers incriminés et cela directement dans l’application. Une fois que le formulaire de modification des fichiers est valide, l’application accepte l’upload.

### Affichage de l’audiothèque

L’application permet à un utilisateur d’afficher les musiques qu’il a précédemment transférées sur le serveur. L’utilisateur peut afficher ses musiques de plusieurs manières.

Dans un premier temps, il peut afficher une liste des artistes qui composent son audiothèque. Dans un deuxième temps, il peut afficher une liste de ses albums et finalement, il peut afficher une liste comportant la totalité des titres qu’il possède.

Il y a également d’autres chemins possibles. Par exemple, si l’utilisateur sélectionne un artiste parmi sa liste d’artistes, tous les albums de l’artiste apparaissent. Puis, s’il sélectionne un album parmi cette liste des albums, c’est tous les titres de l’album en question qui s’affichent.

L’application permet aussi à l’utilisateur de voir toutes ses listes de lectures.

### Lecture de fichiers .mp3

L’application permet à l’utilisateur connecté de lire les fichiers mp3 qui composent son audiothèque. L’utilisateur peut effectuer plusieurs actions sur un fichier : lecture, morceau suivant ou précédant, pause et contrôler le volume sonore.

### Création de listes de lecture

L’application permet à un utilisateur de créer des listes de lecture contenant les titres qu’il souhaite. Il peut nommer sa liste de lecture puis y ajouter des titres.

### Modification de son profil

L’utilisateur peut modifier son profil : il peut tout d’abord changer de pseudo, puis également changer d’avatar.

### Système d’amis

Un utilisateur connecté peut ajouter d’autres utilisateurs en tant qu’amis. Avoir un ami de voir le nombre de titres qu’il possède dans son audiothèque mais également de voir la liste des albums que l’ami possède.

# Analyse organique

## Modèle conceptuel final

Le MCD servant de base à l’élaboration de la base de données a subi **quelques modifications** depuis la définition du cahier des charges. Le changement s’est opéré au niveau de la relation entre l’utilisateur et son audiothèque. Avant, la relation se faisait entre l’entité utilisateurs et l’entité albums. **Désormais, cette relation se fait entre l’entité utilisateurs et titres**, pour des raisons sémantiques puisque qu’un utilisateur possède en réalité des fichiers (=des titres) et non des albums directement.

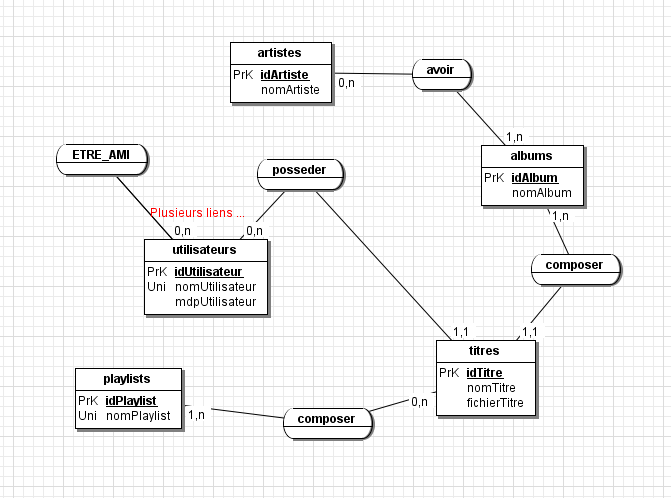


Figure 10: MCD final

## Modèle physique des données

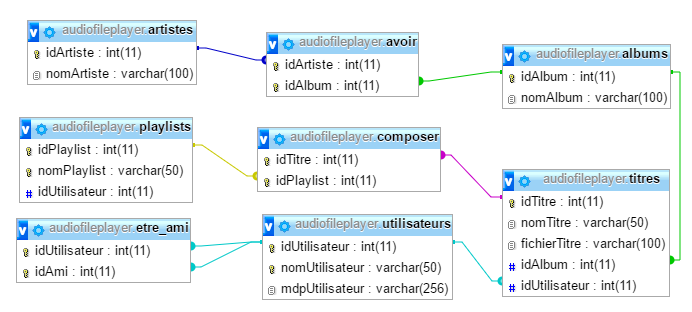


Figure 11: Modèle physique des données

## Le choix du single-page

La structure de l’application est une notion importante lors de la réalisation d’un projet. Chaque structure, design patterns ont leurs forces et faiblesses. Pour la réalisation de mon application de streaming audio, **il était nécessaire d’avoir une application qui soit exempte de rafraîchissement de page**, faute de quoi la musique en cours de lecture serait tout simplement coupée, arrêtée. Ainsi, mon choix s’est porté sur la conception d’une application web dite « monopage » ou, en anglais, **single-page application (SPA)**.

Ce type d’application permet d’éviter le rechargement d’une page à chaque action de l’utilisateur car une SPA créer les vues dynamiquement et les affichent sans recharger la page.

Evidemment, une application moderne ne peut se faire sans une base de données. L’application AudioFile Player a également besoin d’upload une certaine quantité de fichiers, d’où l’importance d’avoir un serveur distant. Pour communiquer avec ce serveur sans rechargement de page, **l’application utilise des requêtes HTTP avec AJAX** (Asynchronous JavaScript and XML).

## Architecture du site

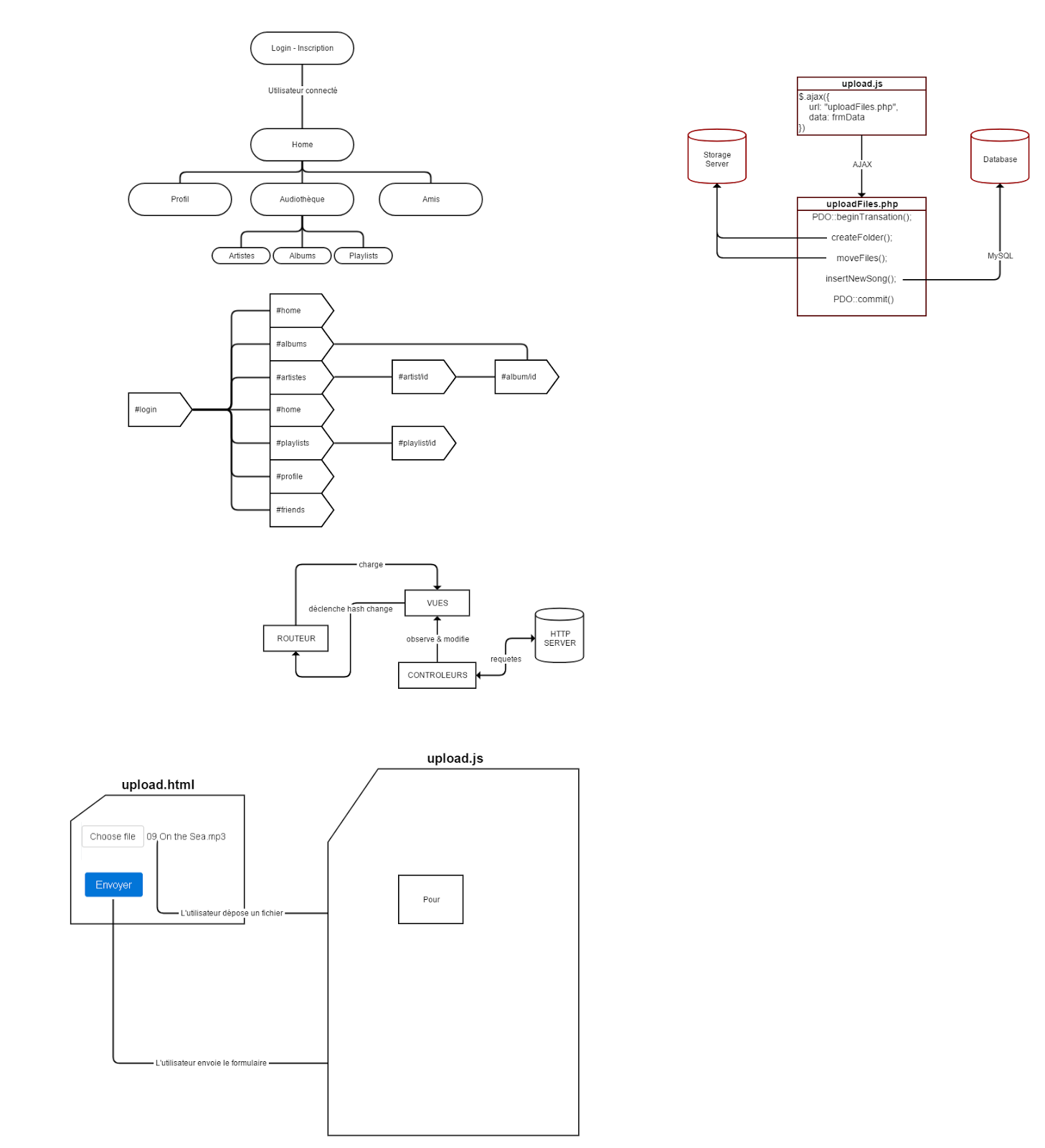


Figure 12: architecture du site

L’architecture du site peut se diviser en 4 éléments :

1. Le routeur
2. Les contrôleurs
3. Les vues
4. Le serveur

Le routeur « écoute » le changement d’url pour générer la vue correspondante. Chaque vue est rattachée à son contrôleur qui s’occupe quant à lui d’observer puis modifier la vue actuelle selon les actions qui lui sont demandés. Le contrôleur s’occupe également de communiquer avec le serveur par le biais de requêtes AJAX.

### Routeur

Dans l’optique de créer une SPA, il était nécessaire de gérer le routage des « pages ». Pages dans le sens, pages vues par l’utilisateur de l’application. Le routage est donc le système qui va s’occuper de donner à l’utilisateur la page qui correspond à l’action qu’il fait. La route est codée en JavaScript et se trouve sur la page **routes.js.**

#### Fonctionnement

Pour pouvoir effectuer un routage, il est nécessaire d’avoir un événement qui détecte un changement de « page ». Dans la plupart des SPA, le changement de « page » se fait par l’intermédiaire d’un changement d’une partie de l’URL. Ce changement se détecte en général d’après une ancre (anchor) particulière, la plupart du temps le symbole ‘#’ suivi d’un mot-clé (ex : #accueil, #commentaires, #faq etc.).

Pour détecter ce changement en JavaScript, **un événement existe déjà nativement : « hashchange »**. Cet événement permet donc de vérifier les changements du hash de l’url, c’est-à-dire la chaine de caractères qui suit ‘#’.

Egalement, il faut définir les différentes routes possibles pour l’application. En règle générale, il est judicieux d’avoir une route par vue afin d’améliorer la cohérence et la lisibilité du code. Dans le cas d’AudioFilePlayer, les routes principales sont :

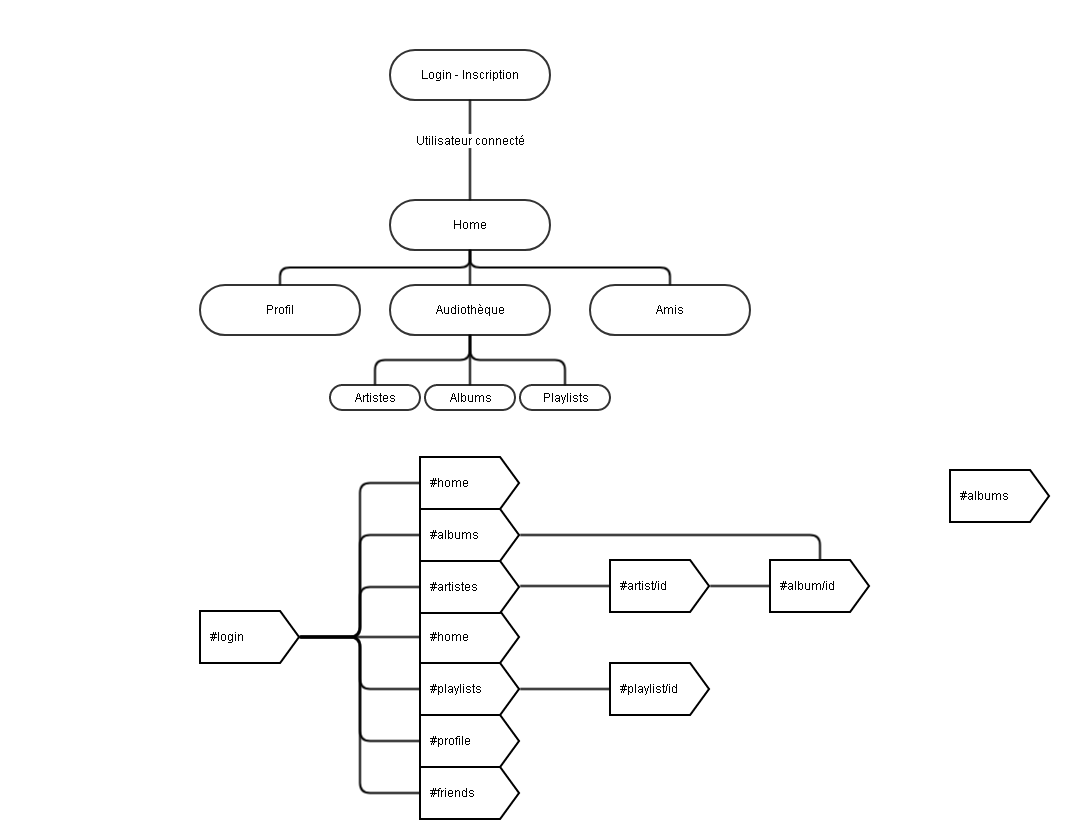


Figure 13: routes de l'application

La première route est ainsi #login, qui comme son nom l’indique s’occupe du login/inscription. Une fois l’action de connexion effectuée, on change le hash du site pour ainsi rediriger l’utilisateur sur une nouvelle route. Les routes qui suivent #login sont les différentes « pages » que l’utilisateur peut voir.

De plus, ce système de route permet de gérer des espèces de « sous-pages ». Ainsi, #artists affiche la liste des artistes quand #artist/id affiche les albums d’un artiste en particulier, grâce à l’id qui suit la barre oblique.

### Contrôleurs

Les contrôleurs de l’application sont les pages qui s’occupent de la logique, du traitement de l’application. Il existe autant de contrôleurs que de vues. Chaque contrôleur s’occupe du traitement au sein de la vue qu’il contrôle. C’est dans les contrôleurs que les appels au serveur se font, grâce aux requêtes asynchrones en AJAX qui vont aller chercher les données que retournent les fichiers PHP.

### Vues

Les vues de l’application sont des fichiers html. Ces derniers sont chargés au fil des actions de l’utilisateur par le routeur (routes.js) qui s’occupe de charger la vue adéquate selon la route qu’emprunte l’utilisateur.

### Arborescence

### API jsmediatags

## Fonctions remarquables

### Upload de fichiers

#### Côté front-end

L’upload de fichiers se fait en plusieurs étapes. Tout d’abord, il y a toute la validation des métadonnées des fichiers .mp3 qui se fait côté front-end. Cette étape est cruciale car l’application a besoin de données cohérentes avant de déplacer les fichiers sur le serveur.

Le traitement des données sur la partie front-end s’occupe ainsi de parcourir les différents fichiers déposés pour détecter si un des fichiers possède des métadonnées manquantes.

Après l’analyse des fichiers, l’utilisateur est renvoyé sur une autre page si la détection a trouvé des métadonnées manquantes, afin de modifier lesdites métadonnées.

Au contraire, si la détection se termine sans déceler de métadonnées manquantes, l’application envoie ces données au serveur qui va s’occuper de la suite du processus (transferts de fichiers, ajout dans la BD).

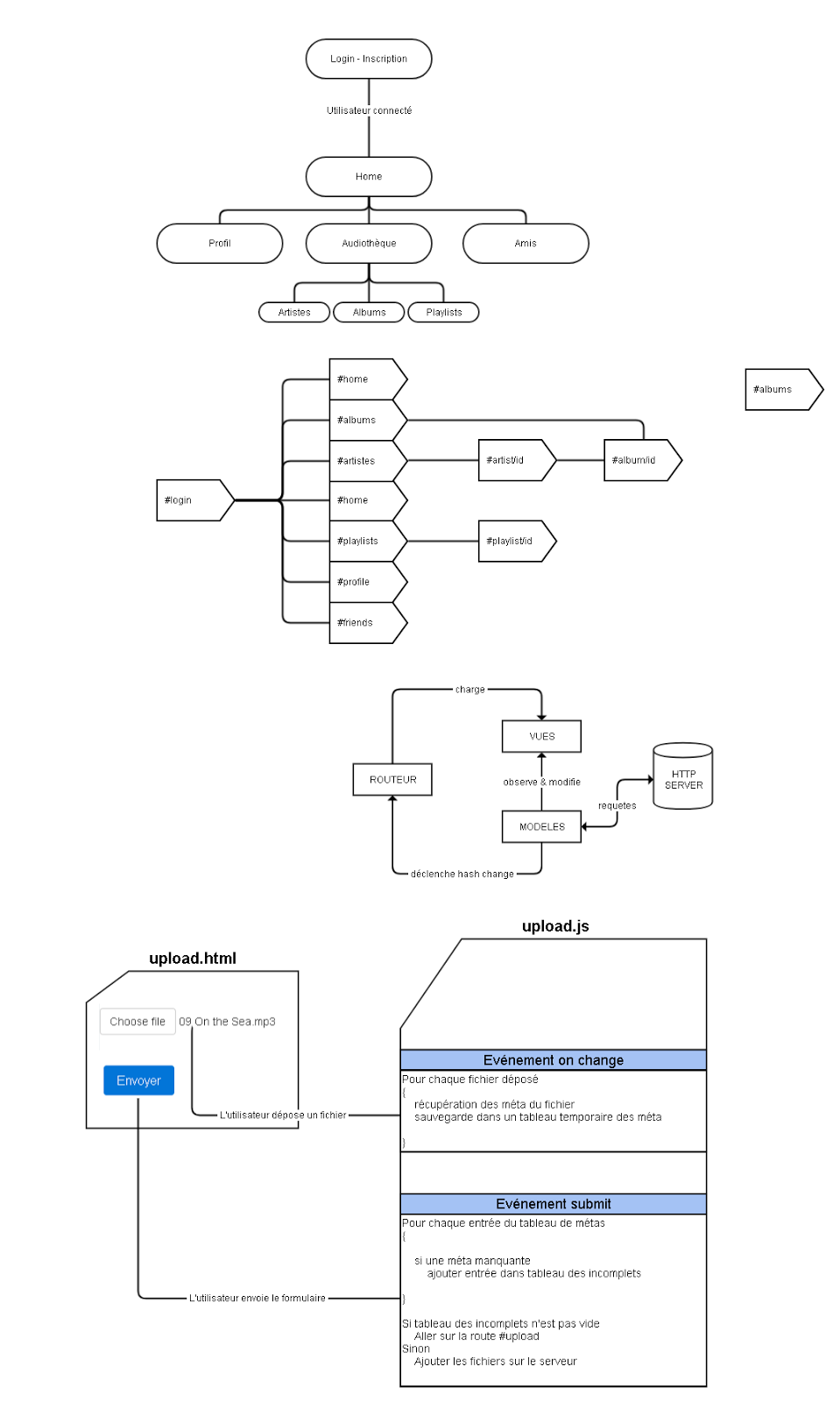


Figure 14: traitement des fichiers côté front-end

#### Côté back-end

Une fois les fichiers validés par le front-end, le serveur s’occupe tout d’abord du transfert des fichiers musicaux. Pour cela, il récupère en premier lieu l’emplacement temporaire de ceux-ci puis il récupère les métadonnées que le front-end lui a données par le biais d’un appel AJAX. L’utilisation d’AJAX permet de communiquer avec le serveur de manière asynchrone, évitant ainsi de recharger la page.

Grâce aux données récupérées, le serveur lance un appel à la base de données afin de pouvoir insérer les données de chaque fichier en les reliant à l’utilisateur par son id dans la base.

Avec ces données, il créée les dossiers adéquats pour pouvoir y mettre les fichiers .mp3. L’arborescence se fait ainsi : **uploads/idUtilisateur/nomArtiste/nomAlbum/titre.mp3**

Afin de conserver l’intégrité des données ; la création des dossiers, les déplacements des fichiers sur ces derniers, et l’insertion des informations concernant ces fichiers dans la base de données sont réalisés grâce à une transaction.

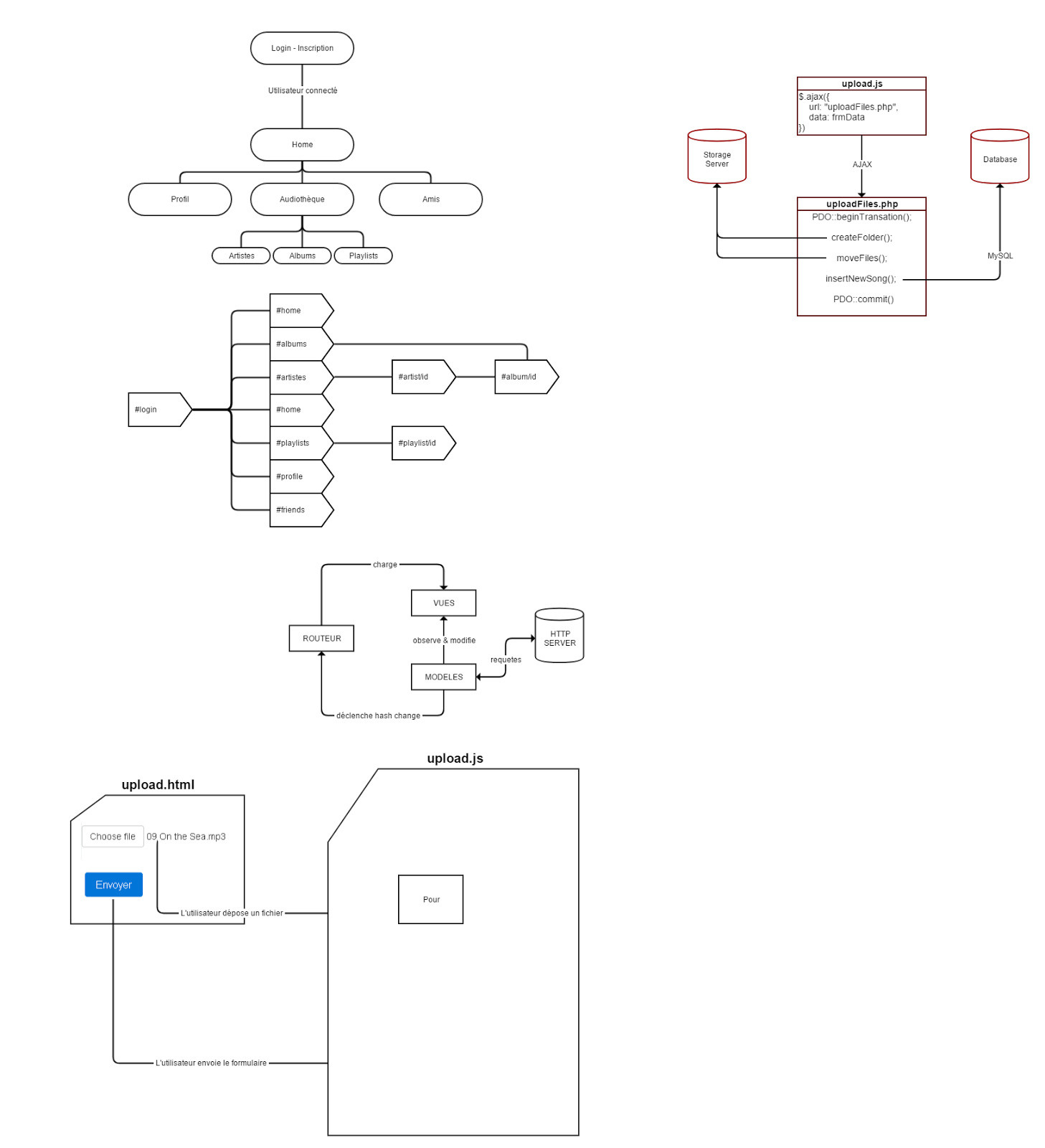


Figure 15: déroulement de l'upload

### Lecture de musique

La lecture de la musique se fait côté front-end, en JavaScript.

#### HTML5 audio

Avec l’arrivée de la version 5 d’HTML, une révision majeure du format de données célèbre, une pléthore de nouveaux éléments, balises ont pu être rendus disponibles.

Pour mon application, afin de pouvoir lire les fichiers musicaux, j’utilise l’élément « audio ». Cet élément HTML5 facilite grandement la lecture de musique grâce à l’implémentation d’attributs utiles et clairs.

AudioFile Player utilise plusieurs événements propres à « audio » pour gérer la lecture :

**play**: événement qui se déclenche lorsqu’un fichier est lancé. Utilisé pour afficher les informations de la musique et également l’image si elle est présente.

**ended** : événement qui se déclenche lorsque la musique en cours de lecture arrive à son terme. Cet événement est utilisé pour lancer la prochaine musique.

**timeupdate**: événement qui se déclenche à chaque fois que l’attribut currentTime s’actualise, c’est-à-dire tout au long de lecture de la musique (fréquence dépend du système). Cet événement permet d’afficher la barre de lecture avec le temps actuel et la durée totale de la musique lue.

#### Class Player

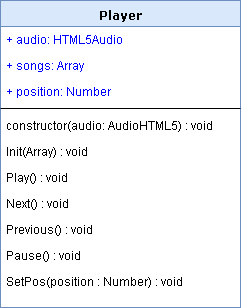


Figure 16: diagramme classe Player

La classe Player est une classe en JavaScript. L’application utilise cette classe pour gérer la lecture de la musique de l’utilisateur.

Propriétés

**audio**: HTML5 audio

Représente en fait un élément audio HTML5. C’est lui qui permet de lire un fichier musical.

**songs**: Array

Le tableau d’objets JSON contenant les informations des musiques. C’est au sein de l’objet JSON que « audio » récupère les informations permettant à ce dernier de trouver l’emplacement du fichier musical.

**position**:Number

Représente une valeur numérique qui est l’index dans songs de la musique à lire.

Méthodes

**Constructor ( audio**:AudioHTML5 **)**

La méthode constructor est, comme son nom l’indique, le constructeur de la classe. Il a besoin en paramètre d’un élément audio. Cet élément peut être une balise existante du DOM (ex : <audio>) mais dans le cas d’AudioFilePlayer, l’élément est généré dans l’initialisation des variables globalesavec document.createElement('audio'), une fonction javascript. Ainsi, la propriété audio devient égale au paramètre passé au constructeur.

Egalement, le constructeur initialise la propriété **songs** en un tableau vide [] et la propriété **position** à 0.

**Init ( songs**:Array **)**

La méthode Init permet d’attribuer une valeur à la propriété songs afin d’avoir un tableau contenant les différents objets JSON représentant les musiques à lire.

**SetPos ( position** : Number **)**

Cette méthode permet de définir l’index de la propriété songs, c’est-à-dire sélectionner la musique à lire dans le tableau de musiques. Grâce à la définition de cet index, la méthode trouve l’emplacement du fichier musical.

**Play ( )**

Déclenche la fonction .play() de la propriété audio, ce qui lance la lecture de la musique.

**Pause ( )**

Déclenche la fonction .pause() de la propriété audio, ce qui met la lecture en pause.

**Next ( )**

Cette méthode permet de lire la prochaine musique qui se trouve dans le tableau de musiques actuellement traité. Si la musique en cours de lecture est la dernière, c’est la première qui sera lue.

**Previous ( )**

Cette méthode permet de lire la musique précédente qui se trouve dans le tableau de musiques actuellement traité.

## Requêtes SQL notables

### Récupération des artistes (nom, id et nombre d’albums) d’un utilisateur

|  |
| --- |
| SELECT a.nomArtiste, a.idArtiste, COUNT(\*) nbAlbums  FROM artistes a  INNER JOIN avoir b ON a.idArtiste = b.idArtiste  INNER JOIN albums c ON b.idAlbum = c.idAlbum  INNER JOIN (  SELECT idAlbum  FROM albums  natural join titres  natural join utilisateurs  WHERE idUtilisateur = :idUser  GROUPBY idAlbum  ) d ON c.idAlbum = d.idAlbum  GROUP BY a.nomArtiste  ORDER BY a.nomArtiste |

### Récupération des titres d’un album lié à un utilisateur

|  |
| --- |
| SELECT `idTitre`, `nomTitre`, `fichierTitre`, a.nomAlbum, art.nomArtiste, idUtilisateur  FROM `titres` t  INNER JOIN albums a ON t.idAlbum = a.idAlbum  INNER JOIN avoir v on a.idAlbum = v.idAlbum  INNER JOIN artistes art ON v.idArtiste = art.idArtiste  WHERE idUtilisateur = :idUser  [AND](http://127.0.0.1/modules/phpmyadmin414x150916125807/url.php?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.5%2Fen%2Flogical-operators.html&server=0&token=f769a0b167ed8e9e4d3ef62c55f5e651#operator_and) t.idAlbum = :idAlbum |

### Récupération des amis d’un utilisateur et leur nombre de titres et d’albums

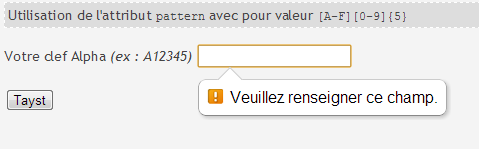
|  |
| --- |
| SELECT e.idAmi, u.nomUtilisateur nom, (  SELECT [COUNT](http://127.0.0.1/modules/phpmyadmin414x150916125807/url.php?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.5%2Fen%2Fgroup-by-functions.html&server=0&token=f769a0b167ed8e9e4d3ef62c55f5e651#function_count)(\*)  FROM titres t  WHERE t.idUtilisateur = e.idAmi  ) as nbTitres, (  SELECT [COUNT](http://127.0.0.1/modules/phpmyadmin414x150916125807/url.php?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.5%2Fen%2Fgroup-by-functions.html&server=0&token=f769a0b167ed8e9e4d3ef62c55f5e651#function_count)(DISTINCT idAlbum)  FROM titres t  WHERE t.idUtilisateur = e.idAmi  ) as nbAlbums  FROM etre\_ami e  INNER JOIN utilisateurs u ON e.idAmi = u.idUtilisateur  WHERE e.idUtilisateur = :idUser |

## Sécurité

La sécurité est cruciale sur une application comme AudioFile Player. En effet, en plus d’avoir des formulaires comme la plupart des applications web d’aujourd’hui, celle-ci se base sur de l’upload de fichiers qui peut s’avérer être le meilleur moyen pour mettre en péril le serveur et ses données.

La sécurité du site se fait sur deux couches : d’abord côté utilisateur en JavaScript. Puis dans un second temps, au niveau du serveur grâce au PHP.

### Formulaires

L’application possède plusieurs formulaires.

Tout d’abord, il y a les formulaires de login et d’inscription. La première couche sécuritaire est en HTML5, avec les attributs « required » des champs de type texte. Cet attribut empêche l’utilisateur de laisser le champ vide. Egalement, l’application utilise des champs comme « password » qui cache le contenu du champ.

Figure 17: attribut required

Cette couche n’est utile que pour se protéger des utilisateurs inattentifs. Pour se protéger de l’utilisateur malveillant, il a fallu mettre en place une validation par l’utilisation d’expressions régulières. Il y en a plusieurs pour différents scénarios mais elles ont en commun d’empêcher l’utilisateur de faire de l’injection de code, en filtrant les caractères problématiques comme les balises ouvrantes/fermantes etc.

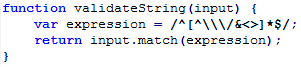


Figure 18: regExp caractères spéciaux JS

Au niveau du serveur, une deuxième validation se fait. Le PHP offre une multitude de possibilités pour valider, filtrer des données. L’application AudioFile Player utilise les fonctions filter\_input, qui permettent de facilement vérifier une variable externe (d’une communication HTTP POST ou GET par exemple). Cette fonction permet de spécifier des filtres de validation ou de nettoyages afin de sécuriser ce que l’on reçoit.



Figure 19: filter\_input PHP

La figure 16 est un exemple de filter\_input : elle retourne la valeur de la variable « user » dans $\_POST nettoyée grâce au filtre spécifié. Dans le cas présent, FILTER\_SANITIZE\_STRING nettoie la chaine en supprimant les balises, et en supprimant ou encodant les caractères spéciaux.

### Upload

L’application possède deux champs de type « file », c’est-à-dire des champs de dépôt de fichiers. Le premier est celui de l’upload des fichiers .mp3 et le deuxième est celui qui s’occupe de l’image de profil de l’utilisateur.

La première étape de validation se fait au niveau du client. D’une part avec les attributs HTML « accept » afin d’empêcher à un utilisateur inattentif de déposer le mauvais type de fichier.



Figure 20: exemple attribut accept

D’autre part, en JavaScript, par une validation de la taille du fichier. En effet, il n’est pas souhaitable d’envoyer des fichiers trop volumineux en traitement côté serveur d’où cette première vérification côté client. Pour les fichiers mp3, j’utilise une fonction qui additionne la taille de chaque fichier déposé et qui la compare ensuite à une valeur de référence.

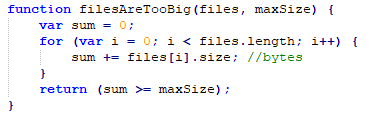


Figure 21: fonction validation taille JS

Du côté du serveur, une autre vérification s’effectue. Etant un langage serveur, le PHP possède des fonctions puissantes pour valider et vérifier des fichiers. C’est également là où la sécurité doit être la plus importante. Pour l’upload de fichiers .mp3, la première vérification est celle sur le type de fichier que le serveur reçoit.

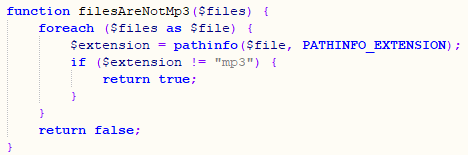


Figure 22: validation extension PHP

La fonction filesAreNotMp3() permet de vérifier si les fichiers sont bien des fichiers .mp3. La fonction parcourt chaque entrée du tableau passé en paramètre. Ensuite, grâce à la fonction native PHP pathinfo(), on récupère l’extension du fichier puis si cette extension diffère de « mp3 », on retourne vrai : un des fichiers n’est pas un fichier mp3.

La vérification de la taille totale des fichiers déposés s’opère également sur le serveur, dans le cas où celle côté client serait outrepassée. A noter que la taille totale des fichiers énoncée doit être plus petite que celle imposé par la configuration du serveur afin d’éviter une erreur.

### Requêtes SQL

L’application est également sécurisée au niveau des appels à la base de données. Toutes les requêtes du site sont basées sur l’objet PDO (Php Data Object), qui permet une couche d’abstraction aux fonctions d’accès aux bases de données. Cela signifie simplement que PDO s’occupe de diriger la requête adéquate selon le type de SGBD indiqué.

La vraie force de l’objet PDO est d’avoir plusieurs méthodes de protection contre les injections SQL, c’est-à-dire faire passer du code SQL depuis les champs d’un formulaire côté client.

L’application AudioFile Player utilise donc des requêtes préparées, avec des méthodes comme bindParam() afin d’envoyer à la requête des données sûres. Cette dernière lie des paramètres à un nom de variable spécifique.

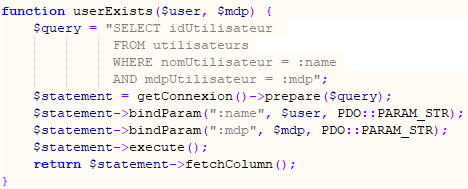


Figure 23: PDO::bindParam()

Sur la requête de la figure 20, il est possible de voir la méthode bindParam en pratique. Le premier paramètre, ici « :name », est le nom du marqueur à remplacer. Ce marqueur se retrouve dans la chaine de la requête SQL.

Quand la requête SQL s’effectuera, le marqueur :name sera remplacé par le contenu de la variable passé dans le deuxième paramètre, ici « $user ». Enfin, le dernier paramètre est le type de valeur à envoyer, en l’occurrence ce sont des constantes de PDO : ici « PARAM\_STR » pour une chaine de caractères.

# Rapport de tests

|  |  |  |  |
| --- | --- | --- | --- |
| Route | Id du test | Description | Résultat |
| #login | LOG01 | Sur la page de connexion, le champ du nom de l’utilisateur s’affiche en clair. |  |
|  | LOG02 | Sur la page de connexion, le champ du mot de passe de l’utilisateur est caché. |  |
|  | LOG03 | Sur la page de connexion, lorsque les champs sont laissés vides et que l’on appuie sur le bouton valider, l’application notifie que les champs sont obligatoires. |  |
|  | LOG04 | Sur la page de connexion, lorsque l’on met des identifiants erronés, un message d’erreur apparait. |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Conclusion

Planning initial, réel

En conclusion, ce travail de fin de diplôme s’est très bien déroulé, malgré quelques embuches comme il est possible de le constater sur le planning réel. En effet, certaines tâches ont pris plus de temps que prévu, alors que d’autres moins. La tâche qui m’a posé le plus de problèmes est sans aucun doute l’upload de fichiers mp3, de par le nombre de cas d’utilisations possibles. Particulièrement, un bug dû au système asynchrone de la lecture de tags qui faussaient les informations du fichier et qui m’a pris un certain temps. A contrario, les tâches finales comme les pages de profil et des amis ont été plus rapides à mettre en place, un rythme, une logique de codage s’étant installé.

Personnellement, j’ai trouvé ce travail enrichissant. Tout d’abord, de par sa nature : devoir travailler sur le même projet une dizaine de jours sans s’arrêter m’a permis de beaucoup m’intéresser au travail que j’accomplissais. Tout le long, une sensation de vouloir bien faire les choses perdure et je trouve que le résultat final est gratifiant lorsque l’on a fait tous ces efforts. Egalement, un travail de cette envergure permet de mettre en pratique nos compétences et connaissances au mieux et permet d’apprendre à être plus autonome. En effet, le fait d’avoir une certaine liberté pendant la journée est une chose positive de ce travail car il permet d’adapter son rythme de travail à notre forme du moment, et donc de maximiser notre rendement.

Pour terminer, ce travail m’a beaucoup apporté dans le sens où il m’a permis de mieux appréhender le futur et de confirmer mon désir de travailler dans le domaine de l’informatique.

# Annexes

# Sources

Le fond d’écran utilisé est libre de droit

Notes diverses :

A METTRE DANS MANUEL UTILISATEUR : CONFIGURER PHP POUR ACCEPTER FICHIERS VOLUMINEUX