Soundstream C#

TPI 2017 N’hairi Ilias

Table des matières

[Analyse préliminaire 2](#_Toc482183577)

[Introduction 2](#_Toc482183578)

[Organisation 2](#_Toc482183579)

[Objectifs 2](#_Toc482183580)

[Analyse 2](#_Toc482183581)

[Analyse concurrentielle 2](#_Toc482183582)

[Cahier des charges détaillé 2](#_Toc482183583)

[Définition de l’audience 2](#_Toc482183584)

[Définition du contenu et des fonctionnalités 2](#_Toc482183585)

[Maquette préliminaire 2](#_Toc482183586)

[Détermination des points faibles 2](#_Toc482183587)

[Stratégie de tests 2](#_Toc482183588)

[Planification détaillée 2](#_Toc482183589)

[Conception 2](#_Toc482183590)

[Analyse de l’environnement 2](#_Toc482183591)

[Détermination de l’arborescence du site et des rubriques 2](#_Toc482183592)

[Définition de la charte graphique 2](#_Toc482183593)

[Maquette graphique 2](#_Toc482183594)

[Conception de la base de données 2](#_Toc482183595)

[Interrogation 2](#_Toc482183596)

[Conception du code 2](#_Toc482183597)

[Ressources 2](#_Toc482183598)

[Réalisation 2](#_Toc482183599)

[Dossier de réalisation 2](#_Toc482183600)

[Description des tests 2](#_Toc482183601)

[Erreurs restantes 2](#_Toc482183602)

[La vie du projet 3](#_Toc482183603)

[Conclusion 3](#_Toc482183604)

[Annexes 3](#_Toc482183605)

[Sources – bibliographie 3](#_Toc482183606)

[Procédure d’installation 3](#_Toc482183607)

# Analyse préliminaire

## Introduction

À la fin de notre formation il nous est demandé de réaliser un travail pratique individuel qui dure 80 heures. J’ai fait le choix de réaliser une application servant à écouter de la musique en streaming et un site web qui va avec. Ce document va présenter le processus d’élaboration de mon application. Il sera en deux parties : la première concernant l’application C# et la deuxième concernant le site web.

## Organisation

N’hairi Ilias, [ilias.nhr@eduge.ch](mailto:ilias.nhr@eduge.ch)

Dans ce projet je m’occuperai de toutes les tâches à réaliser.

Pour une question de sécurité j’ai décidé d’utiliser les serveurs de GitHub avec GitKraken comme client pour réaliser des sauvegardes de mon travail. En plus de la gestion des sauvegardes cela me permettra aussi de gérer les différentes versions de mon projet.

## Objectifs

Voici les objectifs de mon application développée en C# :

* L’utilisateur doit pouvoir créer un compte et/ou se connecter.
* Il pourra aussi écouter de la musique.
* Ajouter/supprimer des morceaux à ses favoris.
* Créer/modifier/supprimer des playlists.
* L’application se connecte directement à la base de données se trouvant sur le même serveur que le serveur web (pour le projet j’hébergerait le serveur en local à l’aide de EasyPhp).

# Analyse

## Analyse concurrentielle

Si nous regardons des applications similaires à celle que je souhaite réaliser nous trouvons que la plus connue est Spotify. Je m’inspirerai donc de certains points de cette application pour la réalisation de mon projet.

## Cahier des charges détaillé

### Définition de l’audience

Mon application est destinée à tout type de personnes et de tout âge car tout le monde peut écouter de la musique.

### Définition du contenu et des fonctionnalités

* L’utilisateur doit pouvoir créer un compte et/ou se connecter.
  + Pour créer un compte il y aura, au lancement de l’application une fenêtre de connexion/inscription ou l’utilisateur à le choix de ce qu’il veut faire.
* Il pourra aussi écouter de la musique.
  + Une fois connecter l’utilisateur pourra effectuer une recherche pour écouter la musique qu’il veut.
* Ajouter/supprimer des morceaux à ses favoris.
  + Si l’utilisateur veut garder une musique dans ses favoris il aura juste à appuyer sur un bouton après avoir sélectionné la musique.
* Créer/modifier/supprimer des playlists.
  + L’utilisateur pourra aussi créer des playlists dans lesquelles il pourra ajouter/supprimer des musiques.

### Maquette préliminaire

Fenêtre de connexion :

L’utilisateur peut choisir se connecter avec ses identifiants ou cliquer sur « Inscription » pour ouvrir la fenêtre d’inscription.

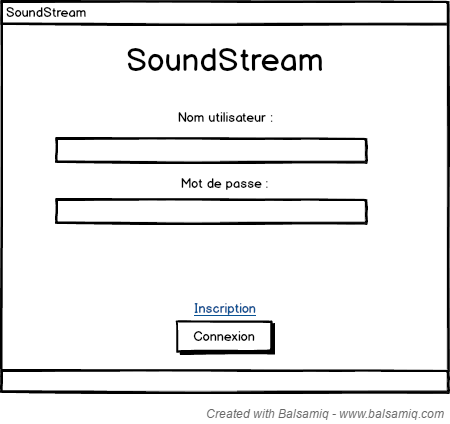


Figure 1 Fenêtre de connexion

Fenêtre inscription :

L’utilisateur peut entrer ses informations pour créer un compte.

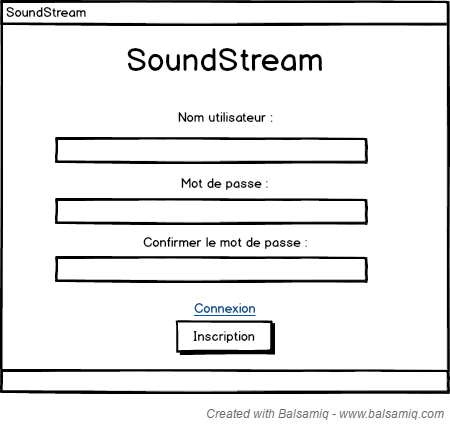


Figure 2 Fenêtre inscription

Fenêtres principales :

Onglet Musiques/Recherche :  
L’utilisateur peut rechercher des musiques, les écouter, changer le volume et choisir ou il se trouve dans le morceaux à l’aide d’une trackbar.

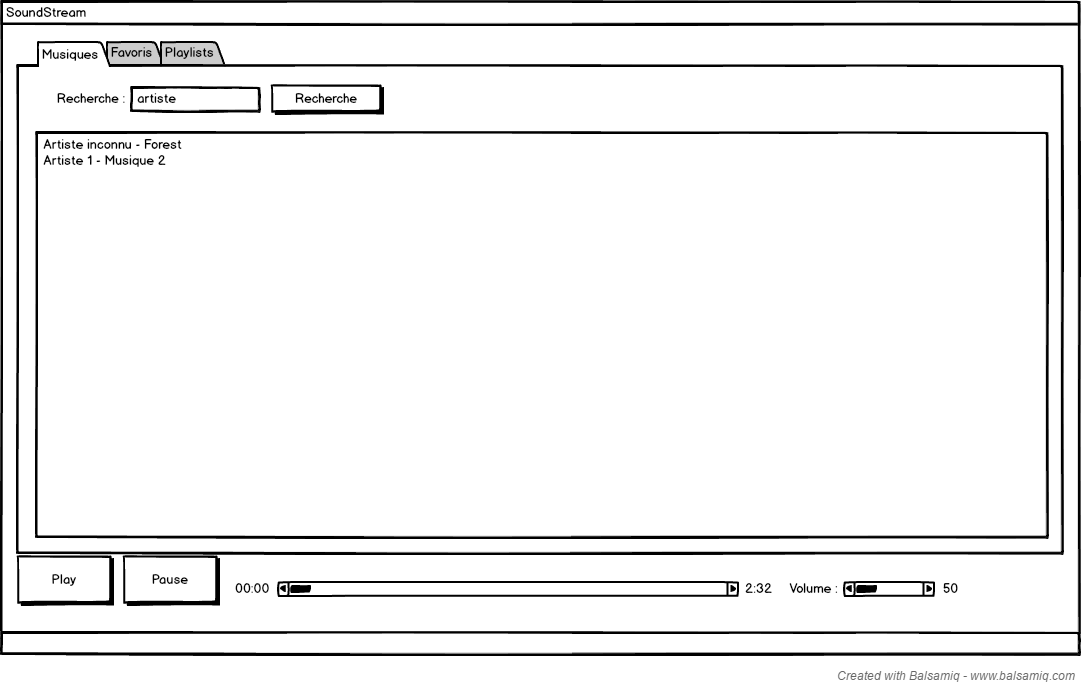


Figure 3 Fenêtre musiques/recherche

Onglet favoris :

L’utilisateur peut consulter ses favoris et les écouter.

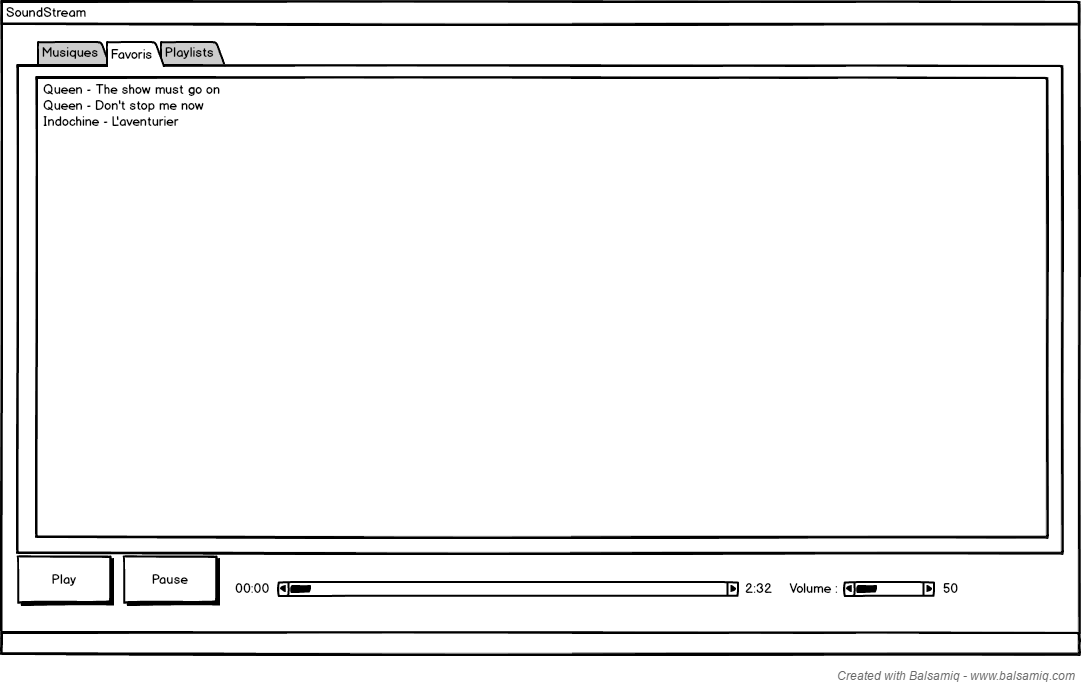


Figure 4 Onglet favoris

Onglet Playlists :  
L’utilisateur peut choisir la playlist qu’il veut consulter avec la liste déroulante puis écouter les musiques qu’il désire.

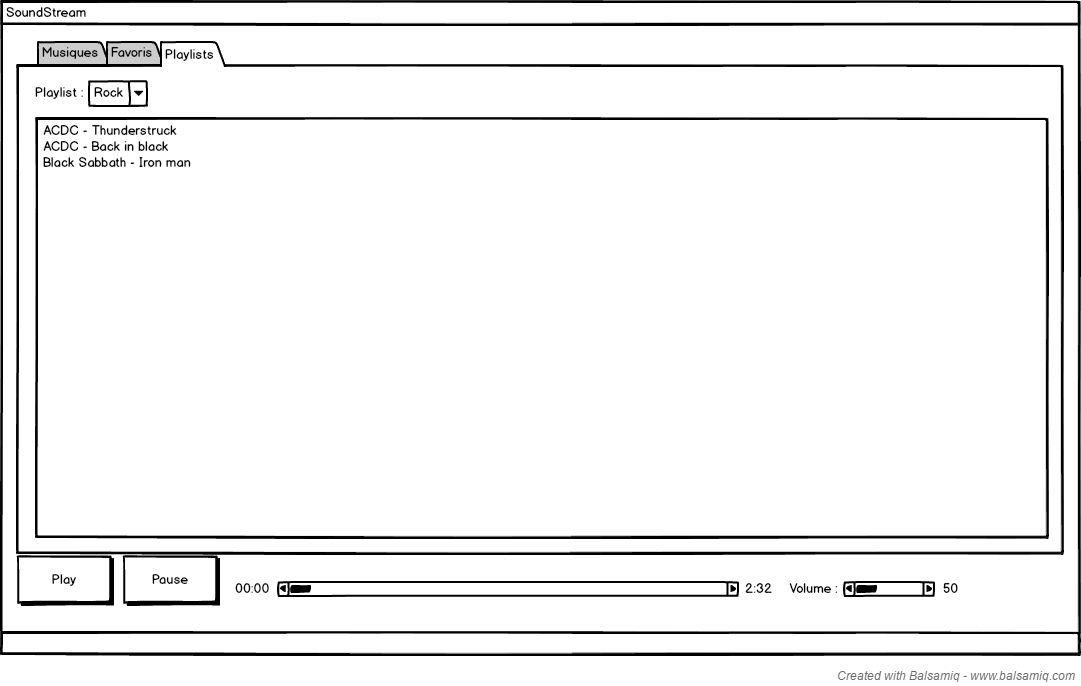


Figure 5 Onglet playlists

### Détermination des points faibles

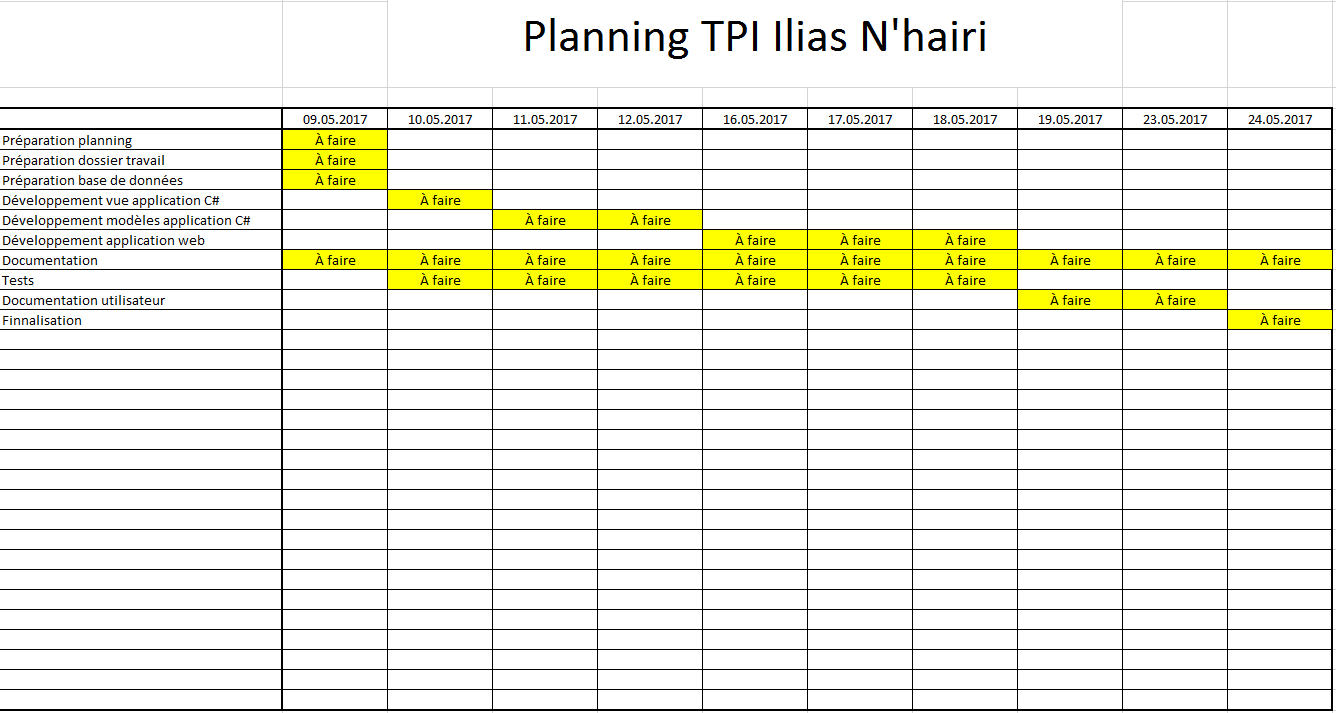
Pour ce projet il faudra que je m’intéresse un peu plus à certaines notions de LINQ afin de pouvoir traiter des requêtes directement dans le code et gagner quelques millisecondes d’exécution.

## Stratégie de tests

Pour tester mon application je vais réaliser des plans de tests ainsi que des rapports. Je donnerai aussi un plan à un de mes collègues pour qu’il le réalise vu qu’il n’a pas connaissance de mon application. Je réaliserai aussi des tests unitaires.

## Planification détaillée

Pour ce projet j’ai réalisé un petit planning pour pouvoir gérer mon temps de travail et rester dans les délais.



# Conception

## Analyse de l’environnement

Pour ce projet le matériel informatique est le suivant :

* Un ordinateur
* Un SSD 330 Go
* Clavier + souris
* Deux écrans

Logiciels utilisés pour ce projet :

* Visual studio 2015
* MS Word 2016
* MS Excel 2016
* Notepad ++
* GitKraken
* Google Chrome
* EasyPhp
* PhpMyAdmin

## Détermination de l’arborescence de l’application

Au lancement l’application l’utilisateur arrive sur la fenêtre de connexion/inscription.

Une fois connecté il arrive sur la fenêtre principale ou il peut écouter de la musique, consulter ses favoris ou ses playlists.



## Définition de la charte graphique

Concernant la charte graphique j’ai choisi de faire quelque chose de sobre et d’intuitif avec mes connaissances. C’est pour ça que j’ai choisi de n’avoir que deux fenêtres afin de ne pas perdre l’utilisateur. Puis pour les couleurs j’ai décidé de rester sur quelque chose de sobre, en l’occurrence du blanc.

## Maquette graphique

Fenêtre de connexion :

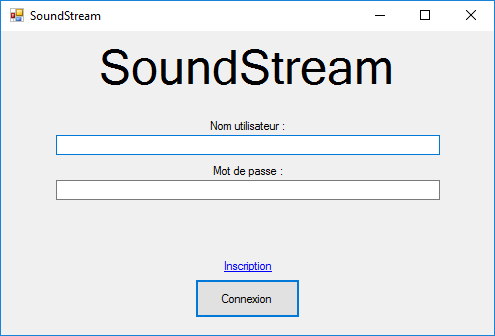


Figure 6 Fenêtre de connexion

Fenêtre d’inscription :

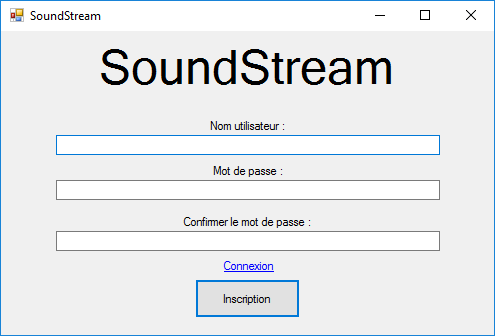


Figure 7 Fenêtre d'inscription

Fenêtre principale (onglet musique/recherche) :

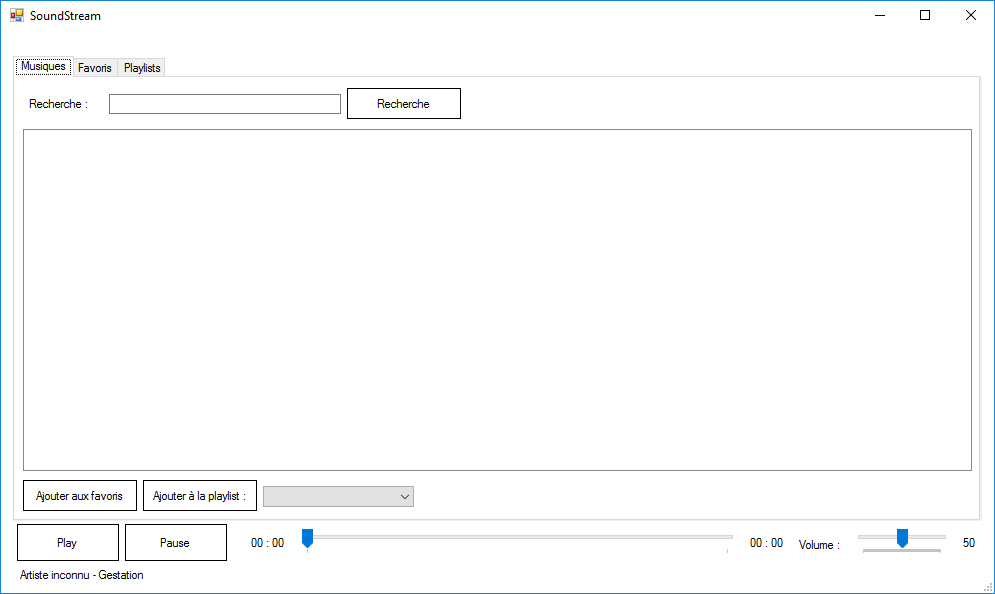


Figure 8 Fenêtre principale (musique/recherche)

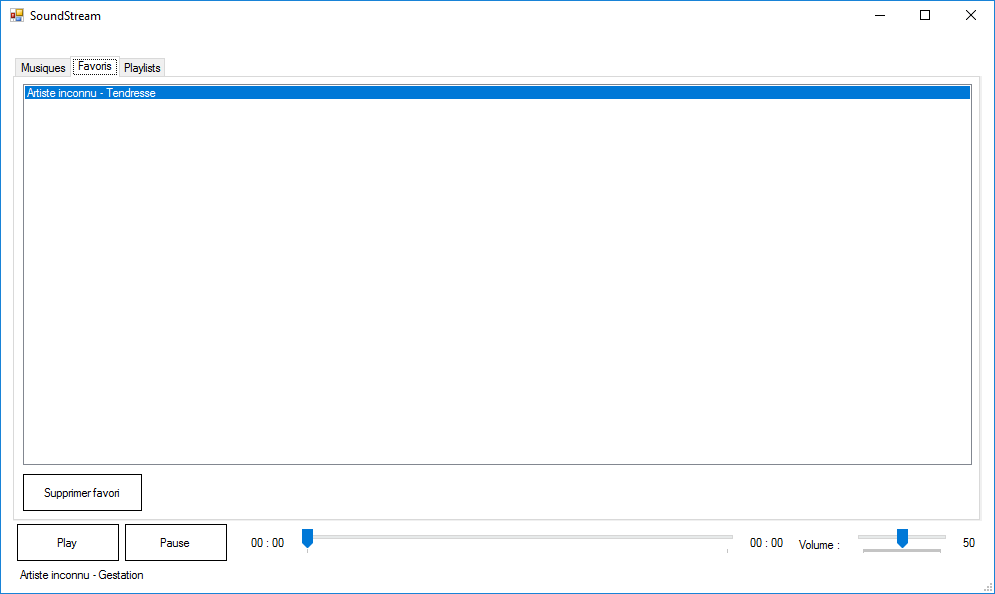
Fenêtre principale (onglet favoris) :  


Figure 9 Fenêtre principale (onglet favoris)

Fenêtre principale (onglet playlists) :

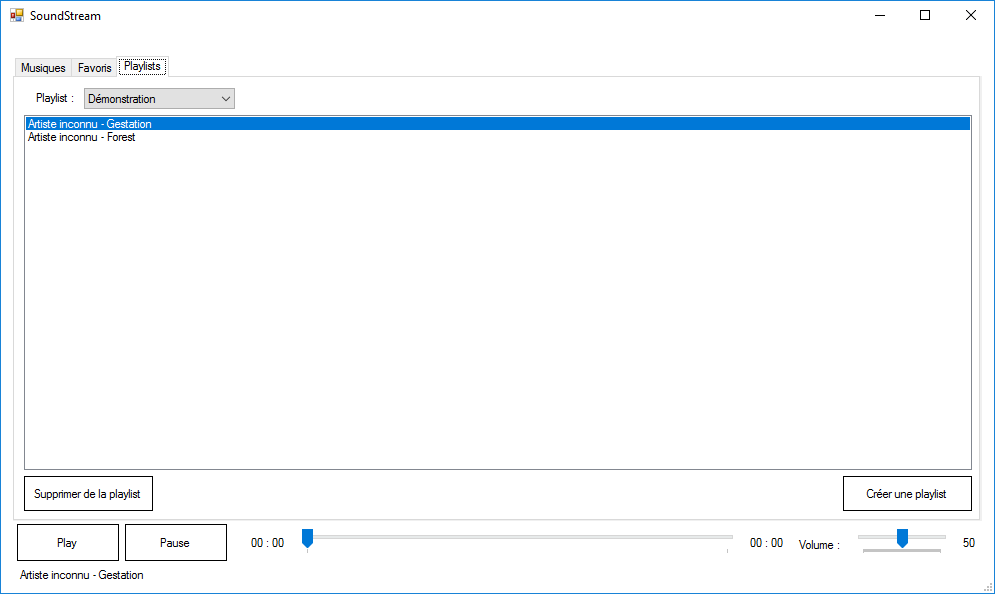


Figure 10 Fenêtre principale (onglet playlists)

## Conception de la base de données

Pour la réalisation de la base de données je me suis demandé ce qui était vraiment important à sauvegarder puis j’ai obtenu le résultat suivant comme premier jet :

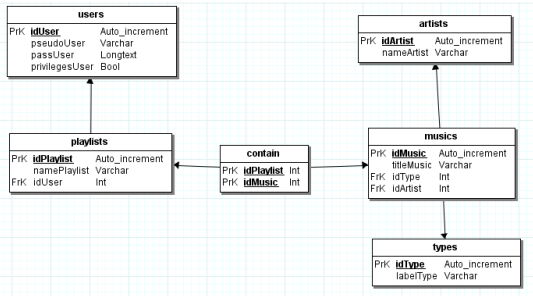


Figure 11 Schéma base de données initial

Cependant après le début du projet je me suis rendu compte qu’il me manquait un élément dans ma base de données pour gérer les favoris des utilisateurs j’ai donc rajouté une table dans ce but. Suite à cette modification voici le schéma final :

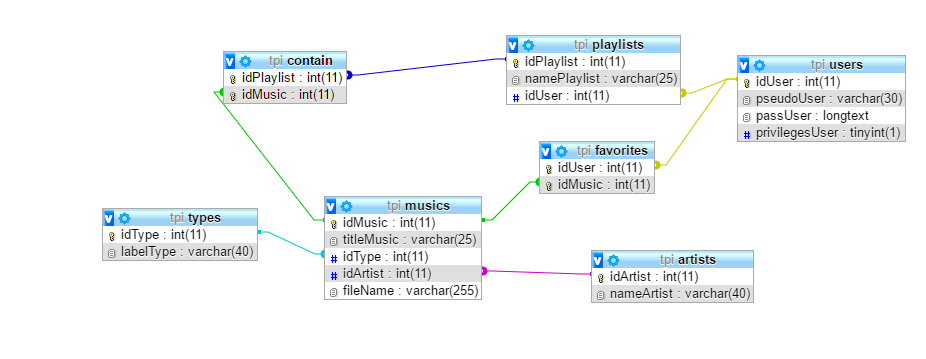


Figure 12 Schéma base de données final

## Interrogation

Pour mon projet j’ai eu besoin de plusieurs requêtes SQL les voici avec des explications :

* SELECT \* FROM users WHERE pseudoUser = @pseudo AND passUser = @pass  
  Vérifier si l’utilisateur a entré les bonnes données pour se connecter
* SELECT \* FROM users WHERE pseudoUser = @pseudo  
  Vérifie si le nom d’utilisateur existe déjà lors de la création d’un compte
* INSERT INTO users (pseudoUser,passUser,privilegesUser) VALUES (@pseudo,@pass,0)  
  Crée un nouveau compte
* SELECT idMusic,titleMusic,labelType,fileName,nameArtist FROM musics,artists,types WHERE musics.idArtist = artists.idArtist AND musics.idType = types.idType AND (musics.titleMusic = @title OR artists.nameArtist = @artist  
  Récupère les informations des musiques après une recherche sur le nom de la musique ou sur l’artiste
* SELECT DISTINCT musics.idMusic,musics.titleMusic,types.labelType,musics.fileName,artists.nameArtist,playlists.namePlaylist,playlists.idPlaylist FROM musics,favorites,contain,users,playlists,types,artists WHERE users.idUser = playlists.idUser AND contain.idPlaylist = playlists.idPlaylist AND contain.idMusic = musics.idMusic AND users.idUser = @id AND types.idType = musics.idType AND musics.idArtist = artists.idArtist  
  Récupère les musiques se trouvant dans des playlists avec le nom de la playlist pour un traitement ultérieur en fonction de l’ID d’un utilisateur
* SELECT musics.idMusic,musics.titleMusic,types.labelType,musics.fileName,artists.nameArtist FROM musics,artists,types,users,favorites WHERE musics.idType = types.idType AND musics.idArtist = artists.idArtist AND users.idUser = favorites.idUser AND musics.idMusic = favorites.idMusic AND users.idUser = @id  
  Récupère les favoris d’un utilisateur en fonction de son ID
* SELECT DISTINCT titleMusic FROM musics  
  Récupère tous les titres pour l’auto complétion de la recherche
* SELECT DISTINCT nameArtist FROM artists  
  Récupère les nom des artistes pour l’auto complétion de la recherche
* INSERT INTO favorites (idUser,idMusic) VALUES (@idUser,@idMusic)  
  Ajoute une musique dans les favoris d’un utilisateur
* INSERT INTO contain (idPlaylist,idMusic) VALUES (@idPlaylist,@idMusic)  
  Ajoute une musique dans la playlist d’un utilisateur
* SELECT playlists.namePlaylist,playlists.idPlaylist FROM playlists,users WHERE users.idUser = @id AND playlists.idUser = users.idUser  
  Récupère les information des playlists (id et nom) afin de les afficher dans les combobox
* DELETE FROM contain WHERE idPlaylist = @idPlaylist AND idMusic = @idMusic  
  Supprime une playlist
* DELETE FROM contain WHERE idPlaylist = @idPlaylist  
  Supprime toutes les musiques d’une playlist
* DELETE FROM playlists WHERE idPlaylist = @idPlaylist  
  Supprime la playlist après avoir supprimer ses musiques avec la requête précédente
* INSERT INTO playlists (namePlaylist,idUser) SELECT @name,@idUser WHERE NOT EXISTS (SELECT \* FROM playlists WHERE namePlaylist = @name AND idUser = @idUser  
  Crée une playlist si l’utilisateur n’en a pas déjà une avec ce nom
* DELETE FROM favorites WHERE idUser = @idUser AND idMusic = @idMusic  
  Supprime une musique des favoris d’un utilisateur

## Conception du code

Dans ce chapitre je vais vous expliquer comment j’ai organisé mon code et expliquer les méthodes que je trouve importantes. Nous allons commencer par les modèles.

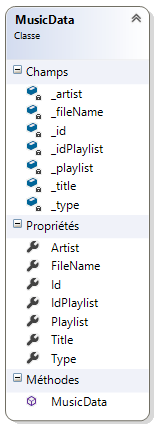
**MusicData**

Figure 13 Classe MusicData

La classe MusicData me sert à stocker les informations sur les musiques. Vous pouvez voir que le type de musique est stocker aussi mais jamais utilisé, je l’ai quand même mis car je trouve que ça serai une très bonne amélioration à faire à l’application. Nous voyons aussi qu’un champ playlist existe il sert à savoir dans quelle playlist la musique se trouve dans un but d’affichage.

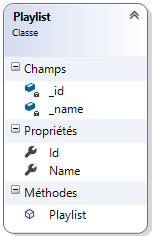
**Playlist**

Figure 14 Classe Playlist

La classe playlist me sert simplement à stocker les informations des playlists.

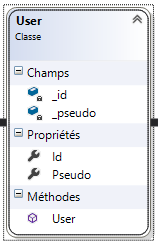
**User**

Figure 15 Classe User

La classe User sert à stocker les informations sur l’utilisateur en vue de réaliser des requêtes.

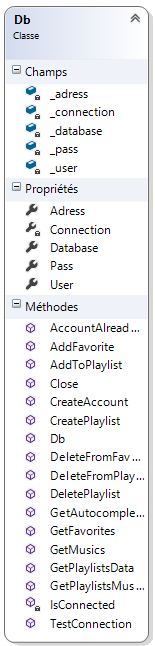
**Db**

Figure 16 Classe Db

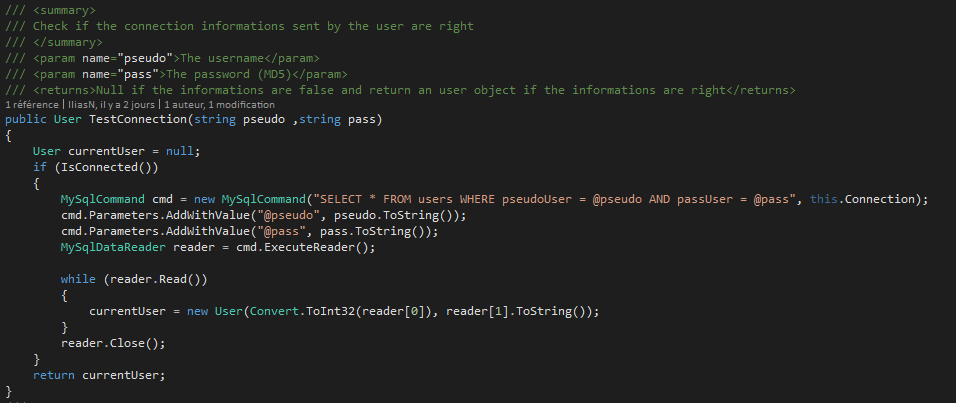
La classe Db est une des plus importantes c’est elle qui s’occupe de la communication avec la base de données. On voit donc que ses propriétés servent à stocker les données de connexion à la base (adresse, nom de la base de données, nom d’utilisateur et mot de passe). Les méthodes ont très souvent la même forme. Pour les méthodes retournant un résultat une liste d’objets (que nous avons vu plus tôt) est retournée.  


Figure 17 Méthode TestConnection

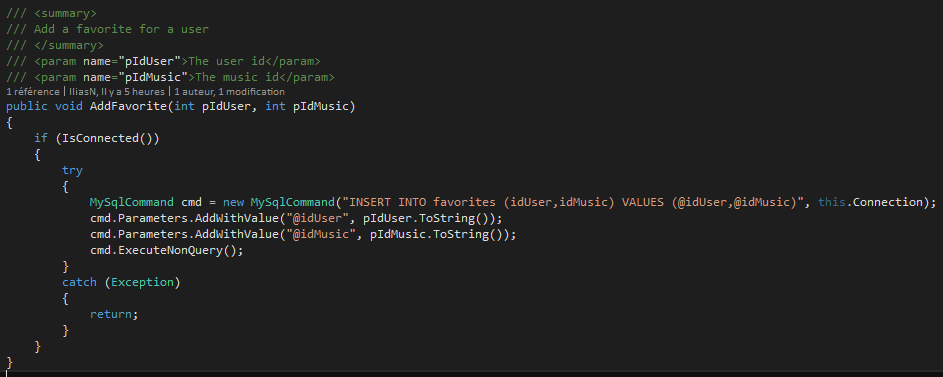
Les requêtes qui ne retournent rien sont sous cette forme :  


Figure 18 Méthode AddFavorite

En général cette classe est plutôt répétitive mais importante.

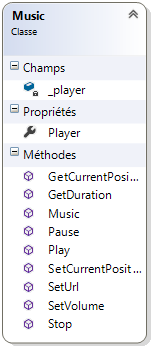
**Music**

Figure 19 Classe Music

Laclasse music est selon moi la plus importante car c’est elle qui s’occupe de la lecture depuis le serveur http. Voici comment elle fonctionne. Tout d’abord un objet « WindowsMediaPlayer » est créer, c’est lui qui va s’occuper de tout la classe est juste là pour l’encapsuler et faciliter son utilisation. J’ai donc créer des méthodes pour gérer le volume, le temps du média, la lecture ainsi que le lien de lecture. Pour lire une musique en ligne il suffit d’indiquer au lecteur le lien et il chargera la musique automatiquement.

Pour le reste de l’application la vue contient un mélange de tous ce que je vous ai montré plus tôt dans de multiples événements. Cependant je vais détailler ce que l’on y trouve.

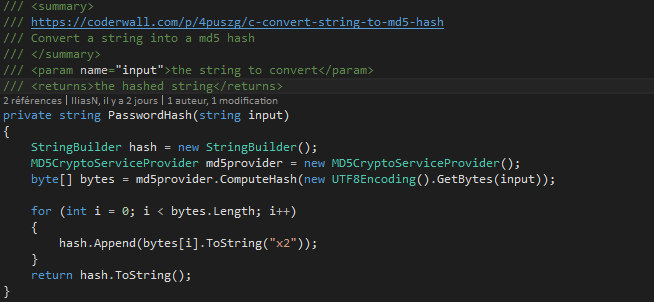
Pour commencer la fenêtre de connexion. Rien de spécial si nous cliquons sur le lien la textbox de confirmation de mot de passe devient visible et nous pouvons nous inscrire. Puis toutes les vérifications de texte sont faites sur les champs pour la connexion ou l’inscription. J’avais besoin d’une méthode pour hasher les mots de passe en Md5 j’ai donc effectué une petite recherche et j’ai trouvé ceci qui marche très bien.  


Figure 20 Méthode PasswordHash

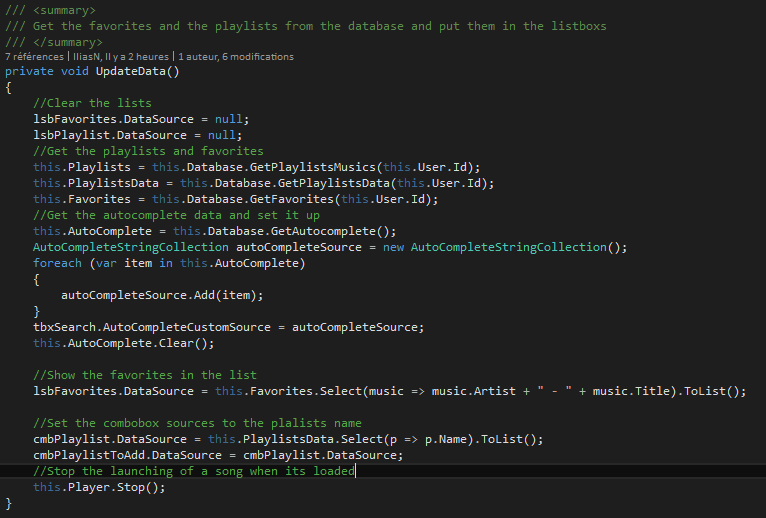
Une fois l’utilisateur connecté la fenêtre principale s’ouvre. Elle récupère l’objet « User » qui sert à garder les informations sur l’utilisateur. Elle récupère aussi la connexion à la base de données. Une fois tout ça fait la méthode « UpdateData » va se charger de récupérer les playlists ainsi que les favoris de l’utilisateur qui s’est connecté. Cette méthode va être utilisée à plusieurs reprise dans le programme, pour actualiser les informations à chaque fois que l’utilisateur modifiera ses favoris ou une playlist.  


Figure 21 Méthode UpdateData

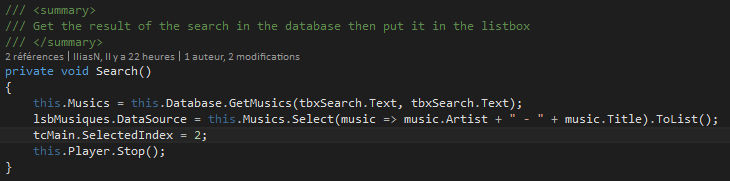
Quand l’utilisateur effectue une recherche c’est la fonction « Search » qui s’en occupe.  


Figure 22 Méthode Search

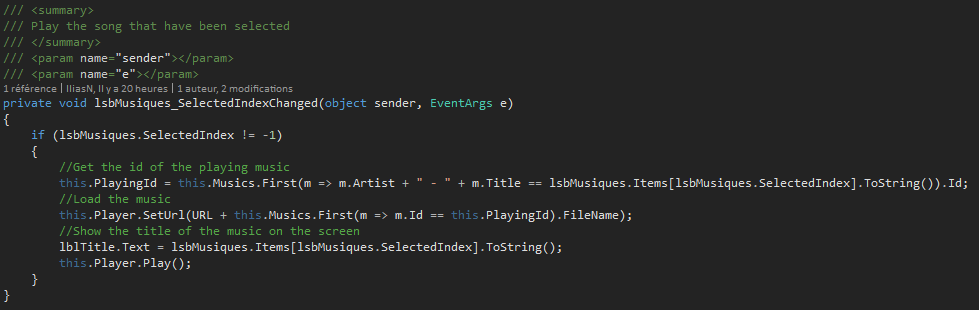
La lecture des musiques se fait automatiquement lorsque l’utilisateur clique sur le nom d’une musique ce qui a pour effet de changer l’id sélectionné et de déclenché l’événement lié. Il y a trois liste (recherche, favoris et playlists) c’est le même principe pour les trois.  


Figure 23 Méthode déclanchement musique

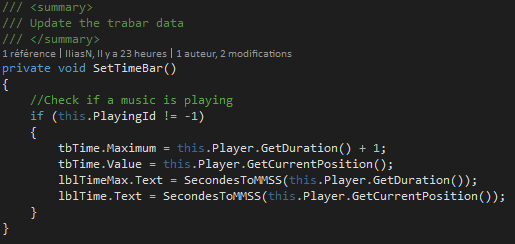
Pour la gestion du temps j’ai utilisé une trackbar qui se règle en fonction de la musique sélectionnée. Avec un timer réglé toutes les 500 millisecondes je mets à jour cette trackbar avec la méthode « SetTimeBar » jumelée avec la méthode « SecondesToMMSS ».  


Figure 24 Méthode SetTimeBar

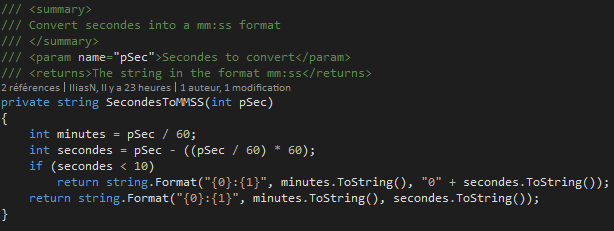
La méthode « SecondesToMMSS » me permet de convertir un entier représentant des secondes en chaine de caractères sous le format mm : ss.  


Figure 25 Méthode SecondesToMMSS

## Ressources

Pour ce projet j’ai eu besoin de une librairie externe et de

# Réalisation

## Dossier de réalisation

## Description des tests

## Erreurs restantes

# La vie du projet

# Conclusion

# Annexes

## Sources – bibliographie

## Procédure d’installation