Soundstream C#

TPI 2017 N’hairi Ilias

Table des matières

[Analyse préliminaire 2](#_Toc482367344)

[Introduction 2](#_Toc482367345)

[Organisation 2](#_Toc482367346)

[Objectifs 2](#_Toc482367347)

[Analyse 3](#_Toc482367348)

[Analyse concurrentielle 3](#_Toc482367349)

[Cahier des charges détaillé 3](#_Toc482367350)

[Définition de l’audience 3](#_Toc482367351)

[Définition du contenu et des fonctionnalités 3](#_Toc482367352)

[Maquette préliminaire 3](#_Toc482367353)

[Détermination des points faibles 7](#_Toc482367354)

[Stratégie de tests 7](#_Toc482367355)

[Planification détaillée 7](#_Toc482367356)

[Conception 8](#_Toc482367357)

[Analyse de l’environnement 8](#_Toc482367358)

[Détermination de l’arborescence de l’application 8](#_Toc482367359)

[Définition de la charte graphique 9](#_Toc482367360)

[Maquette graphique 9](#_Toc482367361)

[Conception de la base de données 12](#_Toc482367362)

[Interrogation 12](#_Toc482367363)

[Conception du code 14](#_Toc482367364)

[Ressources 21](#_Toc482367365)

[Réalisation 22](#_Toc482367366)

[Dossier de réalisation 22](#_Toc482367367)

[Description des tests 22](#_Toc482367368)

[Erreurs restantes 23](#_Toc482367369)

[La vie du projet 23](#_Toc482367370)

[Conclusion 23](#_Toc482367371)

[Annexes 24](#_Toc482367372)

[Sources – bibliographie 24](#_Toc482367373)

[Procédure d’installation 24](#_Toc482367374)

# Analyse préliminaire

## Introduction

Pourquoi ce projet ?

À la fin de notre formation il nous est demandé de réaliser un travail pratique individuel qui dure 80 heures.

But :

J’ai fait le choix de réaliser une application servant à écouter de la musique en streaming grâce à un serveur http jumelé avec une base de données. Un site web sera aussi codé en parallèle utilisant la même base de données.

Objectifs de ce document :

* Présenter le processus d’élaboration de mon application
* Documenter la conception de la base de données
* Documenter la conception du code de l’application
* Recenser les informations nécessaires à la reprise du projet

## Organisation

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

Maitre d’apprentissage : Garcia Francisco, [edu-garciafr@eduge.ch](mailto:edu-garciafr@eduge.ch)

Experts :

* Murisier Serge, [SMURISIER@cross-systems.com](mailto:SMURISIER@cross-systems.com)
* Court Pascal, [experts.tpige@outlook.com](mailto:experts.tpige@outlook.com)

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 les 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’hébergerai 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 connecté 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 de l’application C#

Fenêtre de connexion :

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

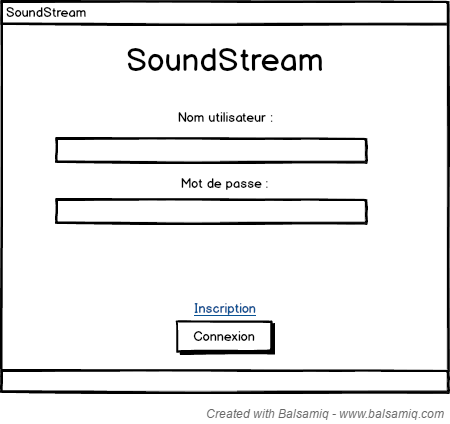


Figure Fenêtre de connexion

Fenêtre inscription :

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

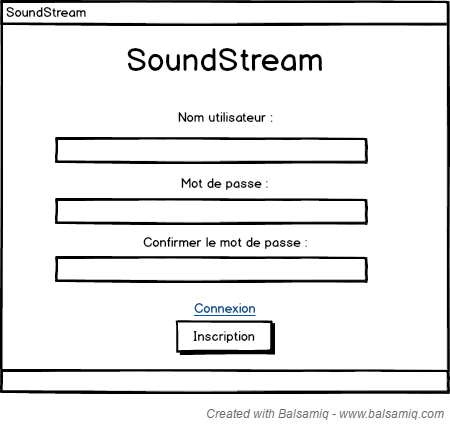


Figure Fenêtre inscription

Fenêtres principales :

Onglet Musiques/Recherche :  
L’utilisateur peut rechercher des musiques et les écouter. Il peut aussi changer le volume et avancer/reculer dans le morceau.

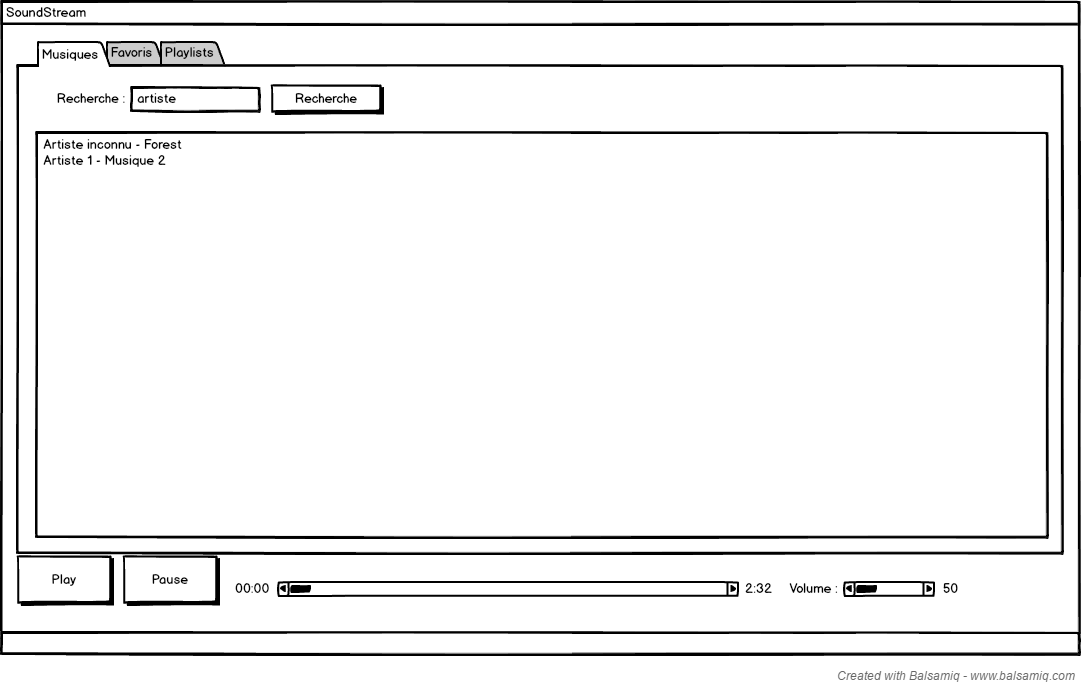


Figure Fenêtre musiques/recherche

Onglet favoris :

L’utilisateur peut consulter ses favoris et les écouter.

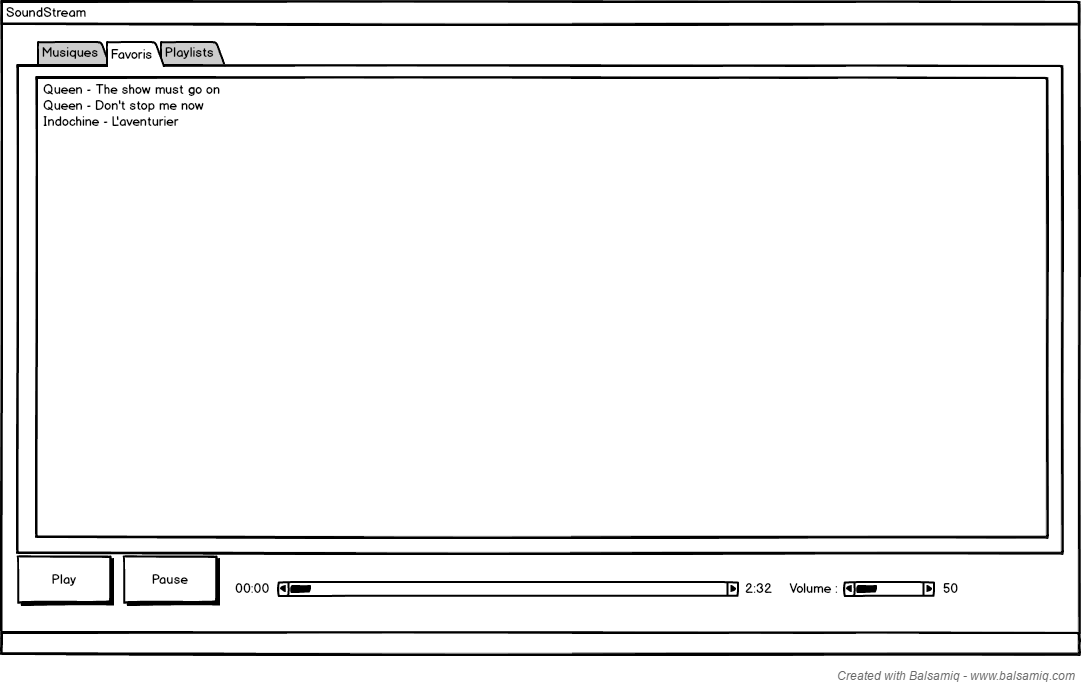


Figure Onglet favoris

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

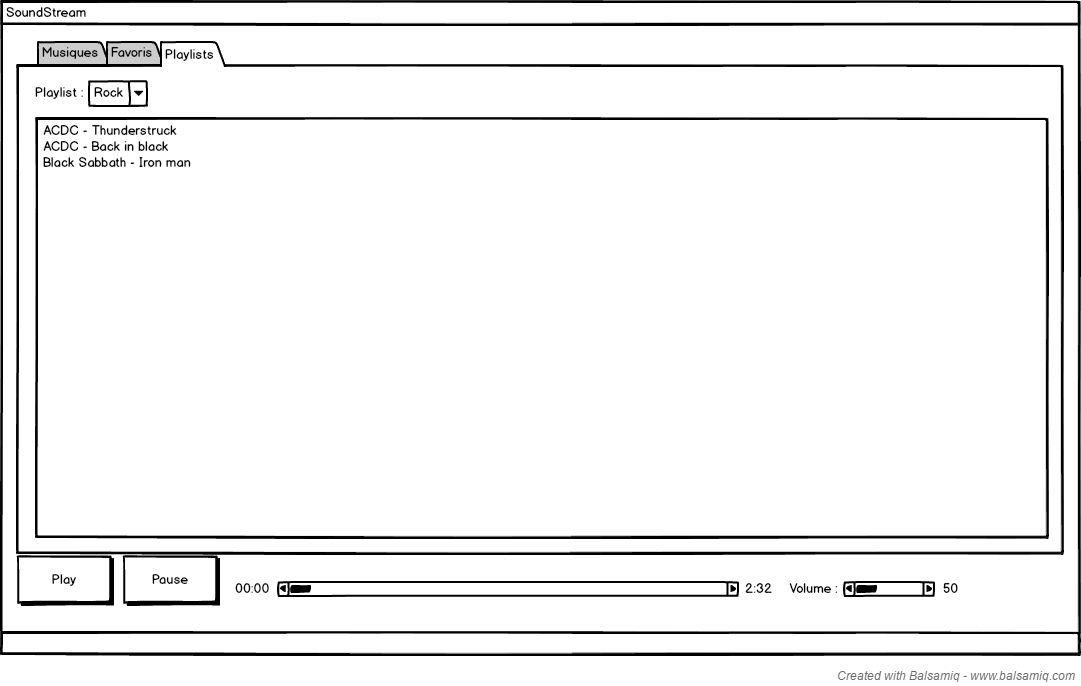


Figure Onglet playlists

### Détermination des points faibles

Pour ce projet il faudra que je m’intéresse un peu plus à certaines notions de LINQ. Je pourrai ainsi 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 suive. Vu qu’il n’a pas connaissance de mon application, il risque de tomber sur des erreurs. Je réaliserai aussi des tests unitaires. Je ferai aussi tester mon application par quelqu’un d’externe au monde de l’informatique pour avoir un avis plus général sur mon application.

## Planification détaillée

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

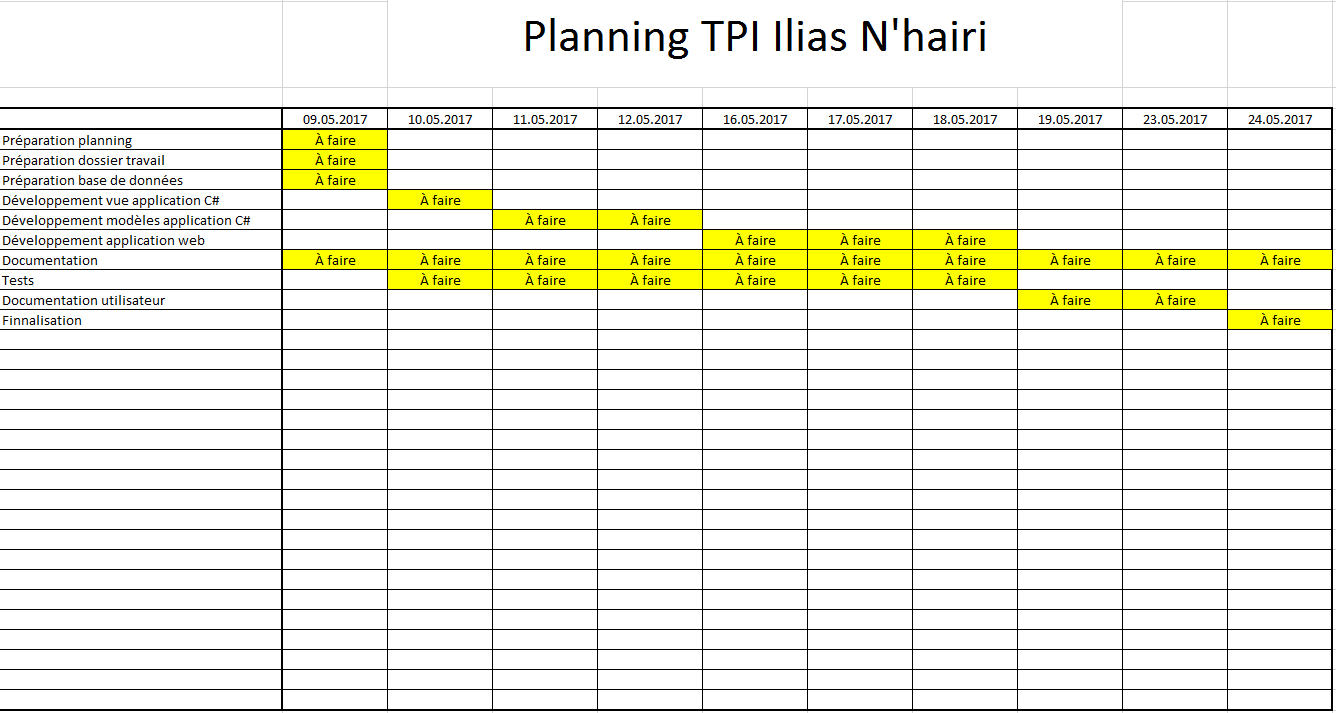


Figure Planning initial

# 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 où il peut écouter de la musique, consulter ses favoris ou ses playlists.



Figure Arborescence

## 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

Nous allons voir le résultat final des fenêtres de l’applications. Nous verrons que des modifications ont été apportées par rapport aux maquettes vues précédemment.

Fenêtre de connexion :

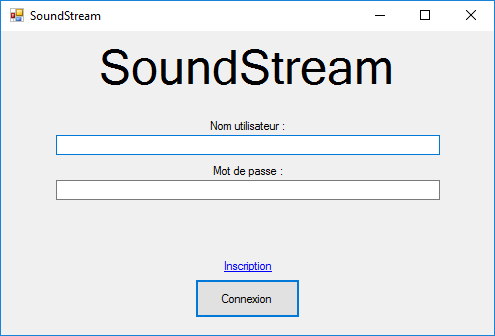


Figure Fenêtre de connexion

Fenêtre d’inscription :

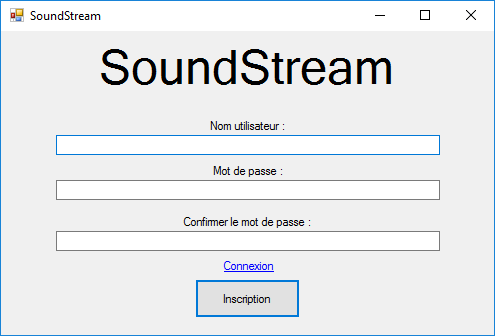


Figure Fenêtre d'inscription

Fenêtre principale (onglet Playlists) :

Nous pouvons voir que j’ai choisi de mettre l’onglet favoris en premier pour des raisons d’ergonomie. De plus j’ai ajouté des contrôles pour gérer les playlists qui n’étaient pas sur les maquettes.

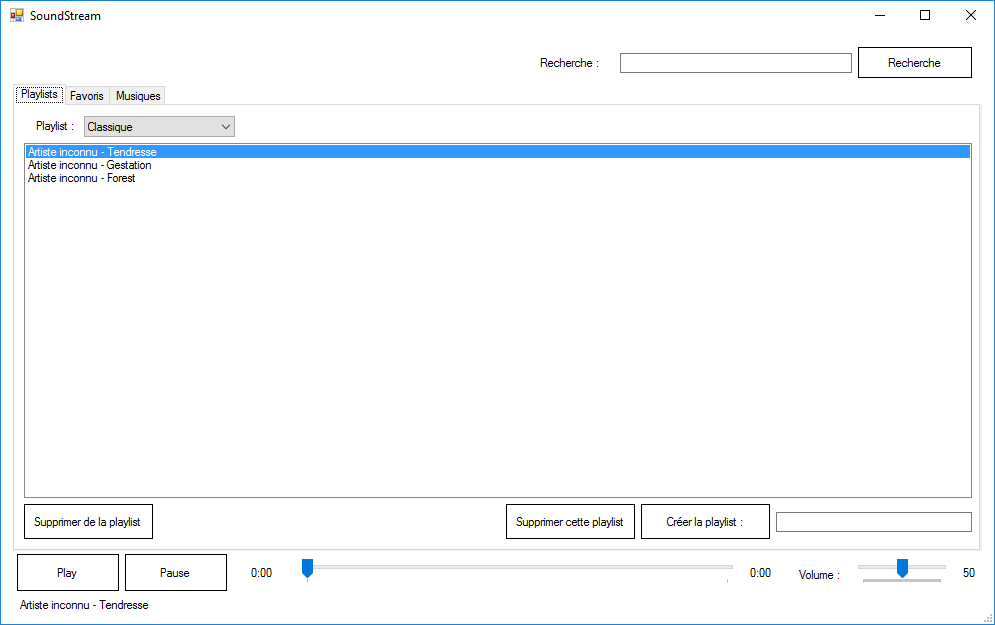


Figure Fenêtre principale (Playlists)

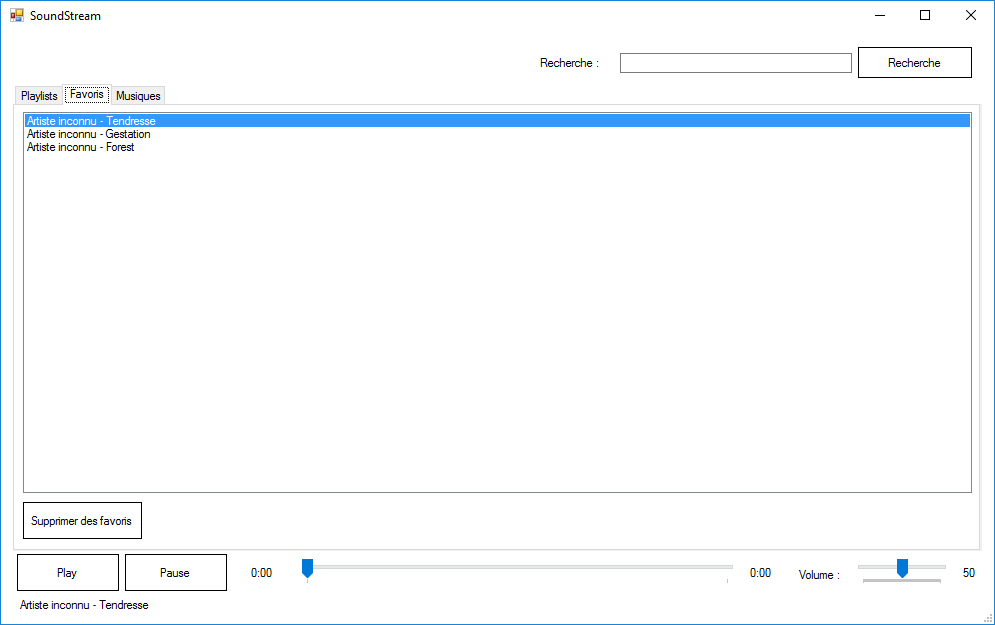
Fenêtre principale (onglet favoris) :  
Ici aussi des modifications ont été apportées par rapport aux maquettes préliminaires. L’ajout du bouton de suppression par exemple.  


Figure Fenêtre principale (onglet favoris)

Fenêtre principale (onglet Musiques/recherche) :

J’ai choisi de mettre la barre de recherche en dehors de l’onglet pour qu’elle soit accessible de n’importe où. J’ai aussi ajouté les boutons d’ajout aux playlists/favoris.

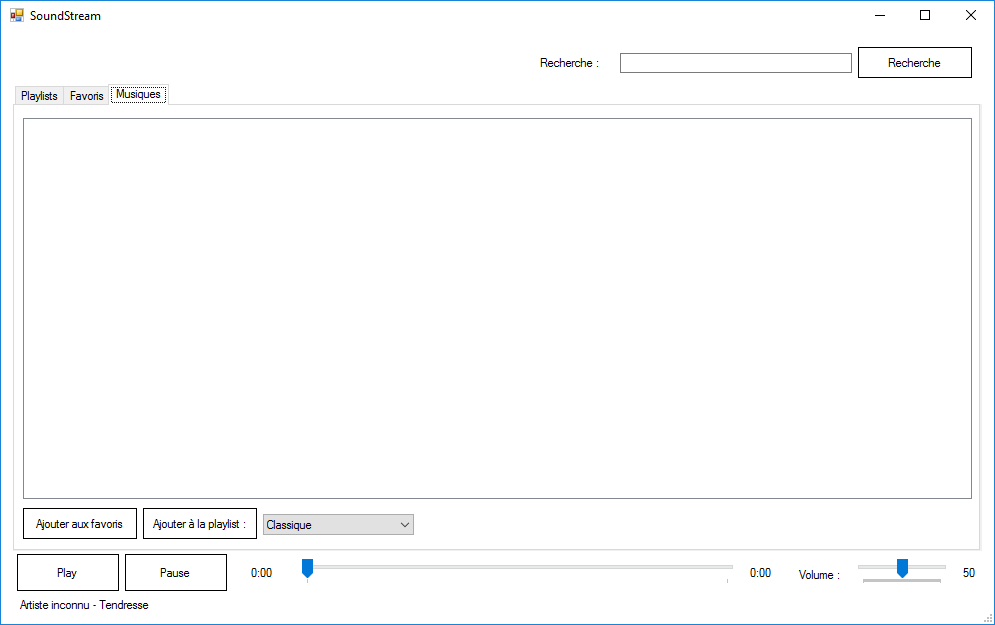


Figure Fenêtre principale (onglet Musiques/recherche)

## 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. J’ai obtenu le résultat suivant comme premier jet :

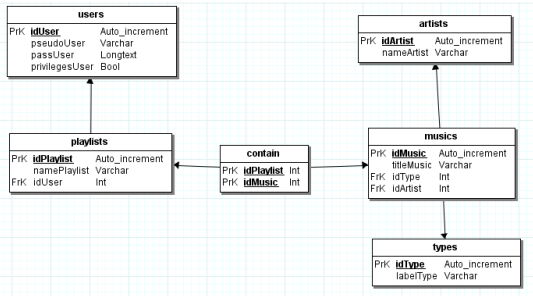


Figure Schéma base de données initial

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

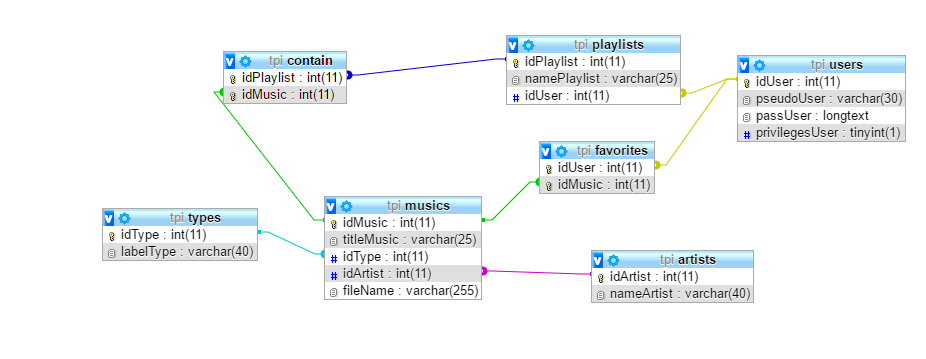


Figure Schéma base de données final

## 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  
  Vérifie 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 supprimé 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.

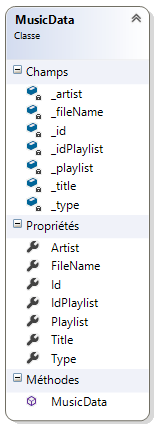
**MusicData**

Figure Classe MusicData

La classe MusicData me sert à stocker les informations sur les musiques. Vous pouvez voir que le type de musique est stocké aussi mais jamais utilisé. Je l’ai quand même mis, car je trouve que ça serait 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.

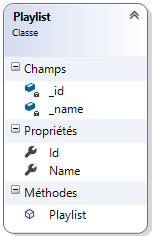
**Playlist**

Figure Classe Playlist

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

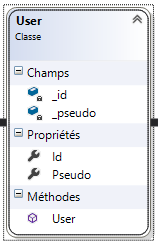
**User**

Figure Classe User

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

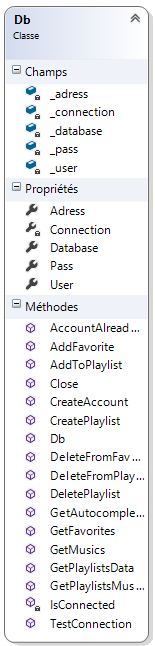
**Db**

Figure 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 Méthode TestConnection

Les requêtes qui ne retournent rien sont sous cette forme :  


Figure Méthode AddFavorite

En général cette classe est plutôt répétitive mais importante.

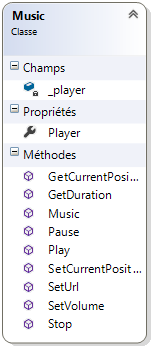
**Music**

Figure 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 créé. C’est lui qui va s’occuper de tout. La classe est juste là pour l’encapsuler et faciliter son utilisation. J’ai donc créé 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 hacher (crypter de manière pratiquement impossible à inverser) les mots de passe en Md5. J’ai donc effectué une petite recherche, sur google, et j’ai trouvé ceci qui marche très bien.  


Figure 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 reprises dans le programme, pour actualiser les informations à chaque fois que l’utilisateur modifiera ses favoris ou une playlist.  


Figure Méthode UpdateData

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


Figure 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 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 Méthode SetTimeBar

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


Figure Méthode SecondesToMMSS

## Ressources

Pour ce projet j’ai eu besoin d’une librairie externe pour la connexion à la base de données. J’ai donc choisi d’utiliser la librairie de MySQL. Car c’est celle que nous avons étudiée et je la connaissais déjà.

<https://www.nuget.org/packages/MySql.Data>

<https://dev.mysql.com/downloads/connector/net/>

J’ai aussi récupérer la fonction servant au hashage des mots de passe en md5 sur le web.  


Figure Méthode hashage

<https://coderwall.com/p/4puszg/c-convert-string-to-md5-hash>

Pour la lecture des musiques j’ai utilisé la librairie WindowsMediaPlayer qui est très facile à prendre en main et très bien documentée.

<https://msdn.microsoft.com/en-us/library/windows/desktop/dd564585(v=vs.85).aspx>

# Réalisation

## Dossier de réalisation

Mon dossier de réalisation est séparé en plusieurs dossier. Au premier niveau nous trouvons ces séparations puis à l’intérieur les fichiers spécifiques à chaque section.

Fichiers :

* BDD
  + Les fichiers .sql de backup
  + Fichier Mcd (jMerise)
* C#
  + Le projet Visual studio
* Documentation
  + La documentation technique pour le projet C# et web
  + La documentation utilisateur
  + Les maquettes
* Web
  + Les fichier web
  + Les musiques dans le dossier « Music »
* Planning.xls

Système d’exploitation :

Ce projet a été réalisé sous Windows 10 entreprise 64x version 1607.

Matériel :

* HP compaq elite 8300 convertible microtower
* SSD Samsung 330 Go
* Clavier + souris HP
* 2x HP Compaq LA 2306x

Version :

SoundStream V1.0

Librairies externes :

* MySQL Connector/Net

## Description des tests

J’ai opté pour plusieurs stratégies de tests. Premièrement j’ai demandé à mes collègues de tester mon application. Plusieurs erreurs ont été trouvé suite à cette méthode :

* Erreur lors de la suppression d’une playlist si la liste déroulante des playlists est vide
* Erreur lors de la suppression d’une musique d’une playlist si la liste déroulante des playlists est vide
* Création de playlists avec le même nom possible (ne devrait pas l’être)

Grâce à la collaboration de mes collègues j’ai pu corriger ces bugs, que je n’avais pas remarqué tout seul, immédiatement.

Deuxièmement j’ai fait quelques tests unitaires pour la fonction de conversion des secondes en chaine de caractères (voir fig. 25).

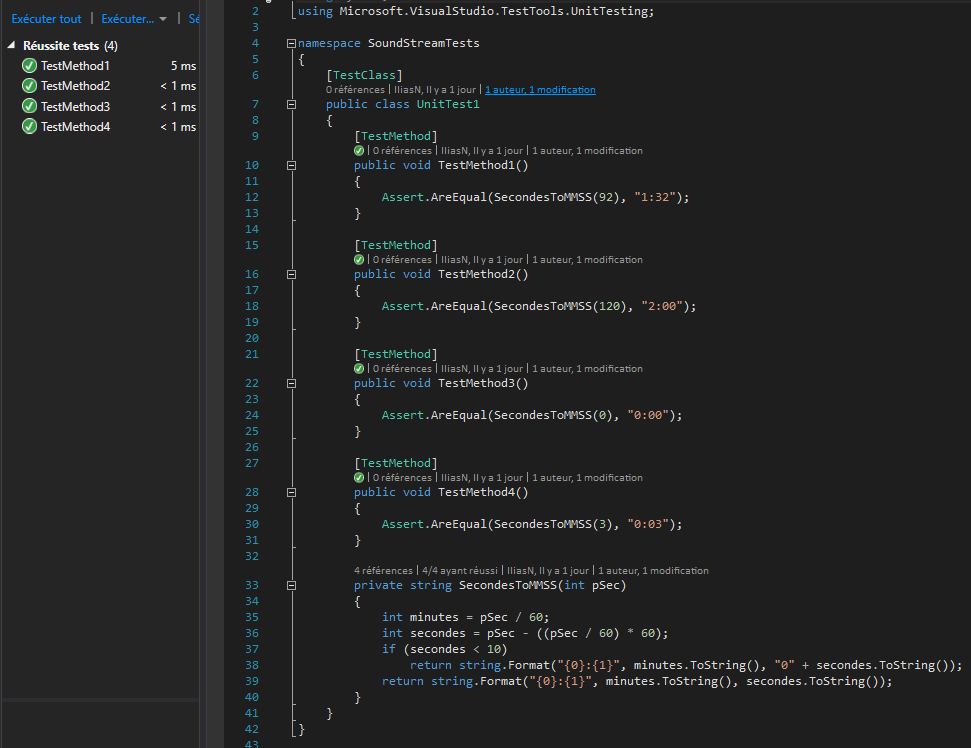


Figure Tests unitaires de SecondesToMMSS

Pour finir j’ai réalisé un plan de tests ainsi que qu’un rapport de test. Le plan de tests doit être réalisé avec la base de donnée version 2 (tpi\_v2.sql).

Plan de tests :

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro test | Auteur | Scénario | Résultat attendu |
| 1 | Ilias N'hairi | Double clic sur l'exécutable. | L'application se lance sans message d'erreur |
| 2 | Ilias N'hairi | L'utilisateur clique sur "Inscription" crée le compte : "TestRapport"  avec le mot de passe "Super" | Un message confirme à l'utilisateur que  le compte a bien été crée |
| 3 | Ilias N'hairi | L'utilisateur se connecte avec le compte qu'il vient de créer | Le fenêtre principale de l'application s'ouvre  sans message d'erreur |
| 4 | Ilias N'hairi | L'utilisateur crée une playlist à l'aide du champ se trouvant à côté du bouton "Créer la playlist : " avec comme nom : "Playlist Test" | Le nom de la playlist doit s'afficher en haut de l'onglet |
| 5 | Ilias N'hairi | L'utilisateur tape : "artiste" dans la barre de recherche puis appuie sur "Enter" | 1. une liste d'auto complétion s'affiche  2. l'onglet musiques s'affiche et présente le résultat de la recherche |
| 6 | Ilias N'hairi | L'utilisateur clique sur le premier morceau puis  clique sur "Ajouter aux favoris" et "Ajouter à la playlist:" | La musique doit commencer, le nom de la musique s'affiche en bas à gauche de la fenêtre, le trackbar du  temps doit s'actualiser ainsi que les labels de temps |
| 7 | Ilias N'hairi | L'utilisateur clique sur l'onglet "Favoris" | Le morceau qu'il vient d'ajouter doit s'y trouver |
| 8 | Ilias N'hairi | L'utilisateur clique sur le morceau puis clique sur  "Supprimer des favoris" | Le morceau ne doit plus se trouver dans les favoris |
| 9 | Ilias N'hairi | L'utilisateur clique sur l'onglet "Playlists" | Le morceau qu'il vient d'ajouter doit s'y trouver |
| 10 | Ilias N'hairi | L'utilisateur clique sur le morceau puis clique sur "Supprimer de la playlist" | Le morceau ne doit plus se trouver dans la playlist |
| 11 | Ilias N'hairi | L'utilisateur clique sur "Supprimer cette playlist" puis sur "Oui" | Un message de confirmation doit s'afficher, puis une fois celui-ci validé la liste déroulante en haut de l'onglet doit être vide |
| 12 | Ilias N'hairi | L'utilisateur lance un morceau puis clique sur "Pause" | La musique doit commencer et s'arrêter au moment où l'utilisateur clique sur "Pause" |
| 13 | Ilias N'hairi | L'utilisateur lance un morceau puis modifie le volume grâce à la barre en bas à droite de l'écran | Le volume doit changer |
| 14 | Ilias N'hairi | L'utilisateur lance un morceau puis modifie temps du morceau grâce à la barre en bas de l'écran | Le morceau doit continuer de jouer là ou l'utilisateur a mis le curseur |
| 15 | Ilias N'hairi | Fermer l'application grâce à la croix en haut à droite de la fenêtre. | L'application se ferme sans messages d'erreur |

Rapports de tests :

Réalisé par Ilias N’hairi le 12.05.2017 :

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro de test | Auteur | Résultat | Fiche anomalie |
| 1 | N'hairi Ilias | OK | / |
| 2 | N'hairi Ilias | OK | / |
| 3 | N'hairi Ilias | OK | / |
| 4 | N'hairi Ilias | OK | / |
| 5 | N'hairi Ilias | OK | / |
| 6 | N'hairi Ilias | OK | / |
| 7 | N'hairi Ilias | OK | / |
| 8 | N'hairi Ilias | OK | / |
| 9 | N'hairi Ilias | OK | / |
| 10 | N'hairi Ilias | OK | / |
| 11 | N'hairi Ilias | OK | / |
| 12 | N'hairi Ilias | OK | / |
| 13 | N'hairi Ilias | OK | / |
| 14 | N'hairi Ilias | OK | / |
| 15 | N'hairi Ilias | OK | / |

Réalisé par Dylan Schito le 12.05.2017 :

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro de test | Auteur | Résultat | Fiche anomalie |
| 1 | Schito Dylan | OK | / |
| 2 | Schito Dylan | OK | / |
| 3 | Schito Dylan | OK | / |
| 4 | Schito Dylan | OK | / |
| 5 | Schito Dylan | OK | / |
| 6 | Schito Dylan | OK | / |
| 7 | Schito Dylan | OK | / |
| 8 | Schito Dylan | OK | / |
| 9 | Schito Dylan | OK | / |
| 10 | Schito Dylan | OK | / |
| 11 | Schito Dylan | OK | / |
| 12 | Schito Dylan | OK | / |
| 13 | Schito Dylan | OK | / |
| 14 | Schito Dylan | OK | / |
| 15 | Schito Dylan | OK | / |

## Erreurs restantes

Je n’ai pour l’instant pas trouvé d’erreurs restantes.

# La vie du projet

On peut voir que durant le projet quelques modifications ont été apportés au planning.



Figure Planning final

Le projet s’est cependant déroulé sans soucis majeur.

# Annexes

## Sources – bibliographie

Liens utiles :

* <https://dev.mysql.com>
* <https://docs.microsoft.com/fr-fr/dotnet/articles/csharp/language-reference/index>
* <https://www.developpez.net/>
* <https://coderwall.com>
* [www.google.ch](http://www.google.ch)
* <https://github.com>

Remerciement Dylan Schito pour les tests.