Gestion audio



Eric-Nicolas Preisig / CIN4A

ETML - Lausanne

110 heures

**Chef de Projet**

M. Bertrand Sahli – bertrand.sahli@vd.ch

**Experts**

M. Sylvain Rollinet – sylvain.rollinet@gmail.com

M. Jonathan Melly – jonathan.melly@vd.educanet2.ch

Table des matières

[1 Spécifications 4](#_Toc484552264)

[1.1 Titre 4](#_Toc484552265)

[1.2 Description 4](#_Toc484552266)

[1.3 Matériel et logiciels à disposition 4](#_Toc484552267)

[1.4 Prérequis 5](#_Toc484552268)

[1.5 Cahier des charges 5](#_Toc484552269)

[1.6 Les points suivants seront évalués 5](#_Toc484552270)

[1.7 Validation et conditions de réussite 5](#_Toc484552271)

[2 Planification Initiale 6](#_Toc484552272)

[3 Analyse 7](#_Toc484552273)

[3.1 Opportunités 7](#_Toc484552274)

[3.2 Document d’analyse et conception 8](#_Toc484552275)

[3.2.1 Maquette 8](#_Toc484552276)

[3.2.2 Architecture 13](#_Toc484552277)

[3.2.3 Librairies utilisées 14](#_Toc484552278)

[3.2.4 Structure du code 16](#_Toc484552279)

[3.2.5 Base de données 18](#_Toc484552280)

[3.2.6 Schéma évènementiel 20](#_Toc484552281)

[3.3 Conception des tests 27](#_Toc484552282)

[4 Planification détaillée 29](#_Toc484552283)

[5 Réalisation 31](#_Toc484552284)

[5.1 Dossier de Réalisation 31](#_Toc484552285)

[5.1.1 MPD 31](#_Toc484552286)

[5.1.1 Dossiers exclus 31](#_Toc484552287)

[5.1.2 Lecture des radios 31](#_Toc484552288)

[5.1.3 Stream radio 32](#_Toc484552289)

[5.2 Modifications 32](#_Toc484552290)

[5.3 MDP final 33](#_Toc484552291)

[5.4 MVVM 34](#_Toc484552292)

[5.5 Extrait de code 35](#_Toc484552293)

[6 Tests 36](#_Toc484552294)

[6.1 Dossier des tests 36](#_Toc484552295)

[6.2 Tests unitaires 39](#_Toc484552296)

[7 Conclusion 40](#_Toc484552297)

[7.1 Bilan des fonctionnalités demandées 40](#_Toc484552298)

[7.2 Bilan de la planification 41](#_Toc484552299)

[7.3 Bilan personnel 42](#_Toc484552300)

[7.4 Remerciement 42](#_Toc484552301)

[8 Webographie 42](#_Toc484552302)

[9 Divers 43](#_Toc484552303)

[9.1 Journal de travail 43](#_Toc484552304)

[10 Annexes 45](#_Toc484552305)

[10.1 Cahier des charges 45](#_Toc484552306)

[10.2 Gantt 48](#_Toc484552307)

# Spécifications

Le candidat réalise un travail personnel sur la base d’un cahier des charges reçu le 1er jour.

Le cahier des charges est approuvé par i — CQ VD. Il est en outre présenté, commenté et discuté avec le candidat. Par sa signature, le candidat accepte le travail proposé.

Le candidat a connaissance de la feuille d’appréciation avant de débuter le travail.

Le candidat est entièrement responsable de la sécurité de ses données.

En cas de problèmes graves, le candidat avertit au plus vite les deux experts et son chef de projet.

Le candidat a la possibilité d’obtenir de l’aide, mais doit le mentionner dans son dossier de projet.

À la fin du délai imparti pour la réalisation du TPI, le candidat doit transmettre par courrier électronique le dossier de projet aux deux experts et au chef de projet. En parallèle, une copie papier du rapport doit être fournie sans délai en trois exemplaires. Cette dernière doit être en tout point identique à la version électronique.

## Titre

Gestionnaire de playlists audio et de flux radios

## Description

Développer une application desktop en C# selon le pattern MVVM afin de gérer des playlists audios et des flux radios publics diffusés sur Internet. Il s’agira notamment :

– d’assurer l’installation de l’application sur les OS Microsoft 7 à 10.

– d’exploiter les données audios locales (sans les dupliquer).

– de créer un lecteur audio « user-friendly ».

– de stocker les données de l’application dans une base de données SQLite.

## Matériel et logiciels à disposition

PC standard de l’ETML (Windows 7 – 64 bits)

Player de machines virtuelles (vmware ou équivalent)

Suite Microsoft Office 2013

IDE C# (Visual Studio Community 2015 ou 2017, SQLite Precompiled Binaries for Windows)

## Prérequis

Compétences élémentaires en base de données (SQL) et en programmation (C#) 🡺Modules 103, 104, 105, 303.

Compétences bureautiques 🡺Modules 301 et 302

## Cahier des charges

Cahier des charges complet en annexe

## Les points suivants seront évalués

* La méthodologie de travail ainsi que la conformité des tâches réalisées par rapport au planning initial.
* Le choix des « métadonnées » nécessaires à l’indexation des chansons.
* La pertinence du modèle de données réalisé.
* La mise en œuvre de tests (notamment les tests unitaires) pour vérifier les fonctionnalités implémentées.
* La journalisation des tâches ainsi que la rédaction régulière de la documentation.

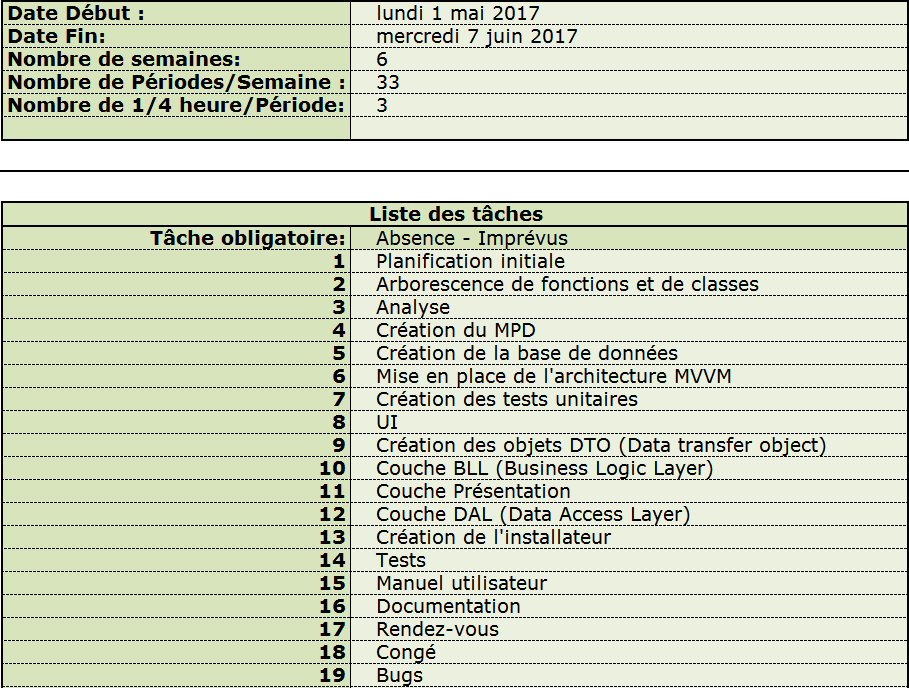
## Validation et conditions de réussite

La grille d’évaluation définit les critères généraux selon lesquels le travail du candidat sera évalué (documentation, journal de travail, respect des normes, qualité…).

En plus de cela, le travail sera évalué sur les trois points spécifiques suivants :

* Le point spécifique n° 1 🡺L’extraction de métadonnées de divers format audio.
* Le point spécifique n° 2 🡺La recherche de Webradio.
* Le point spécifique n° 3 🡺Le (re) scan des contenus audio.

# Planification Initiale



*Gantt disponible en annexe*

# Analyse

## Opportunités

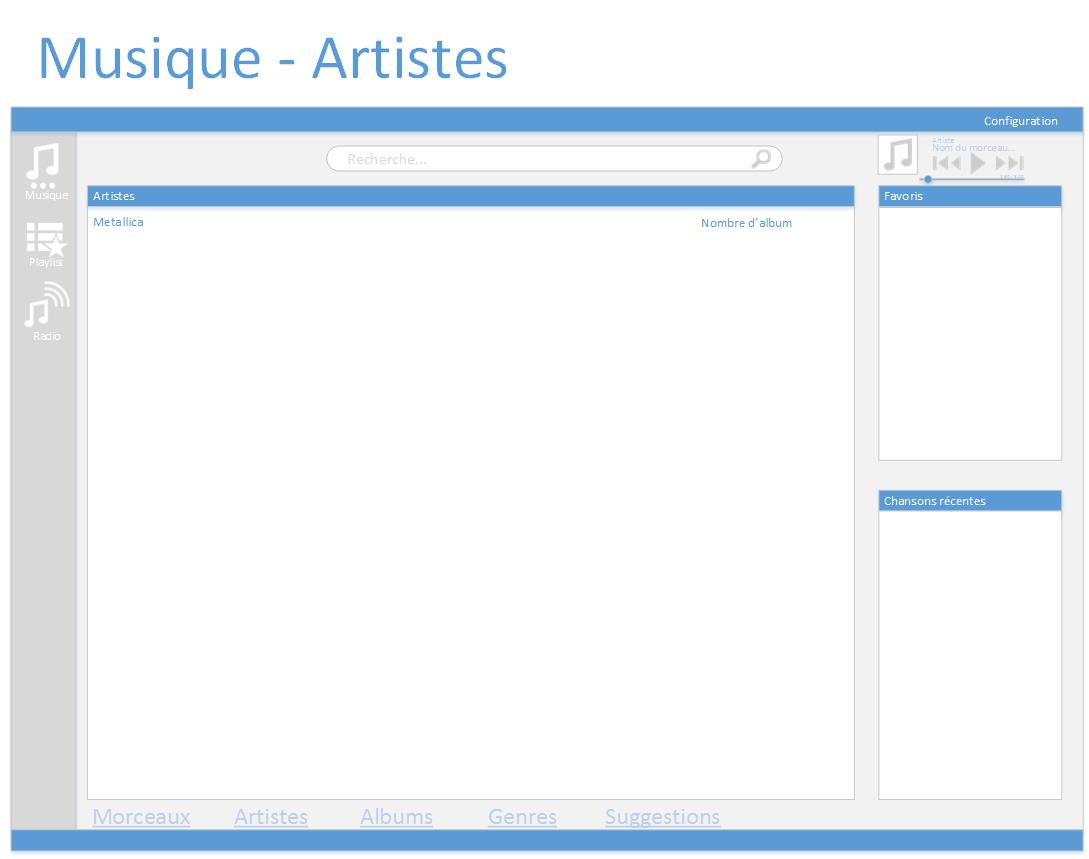
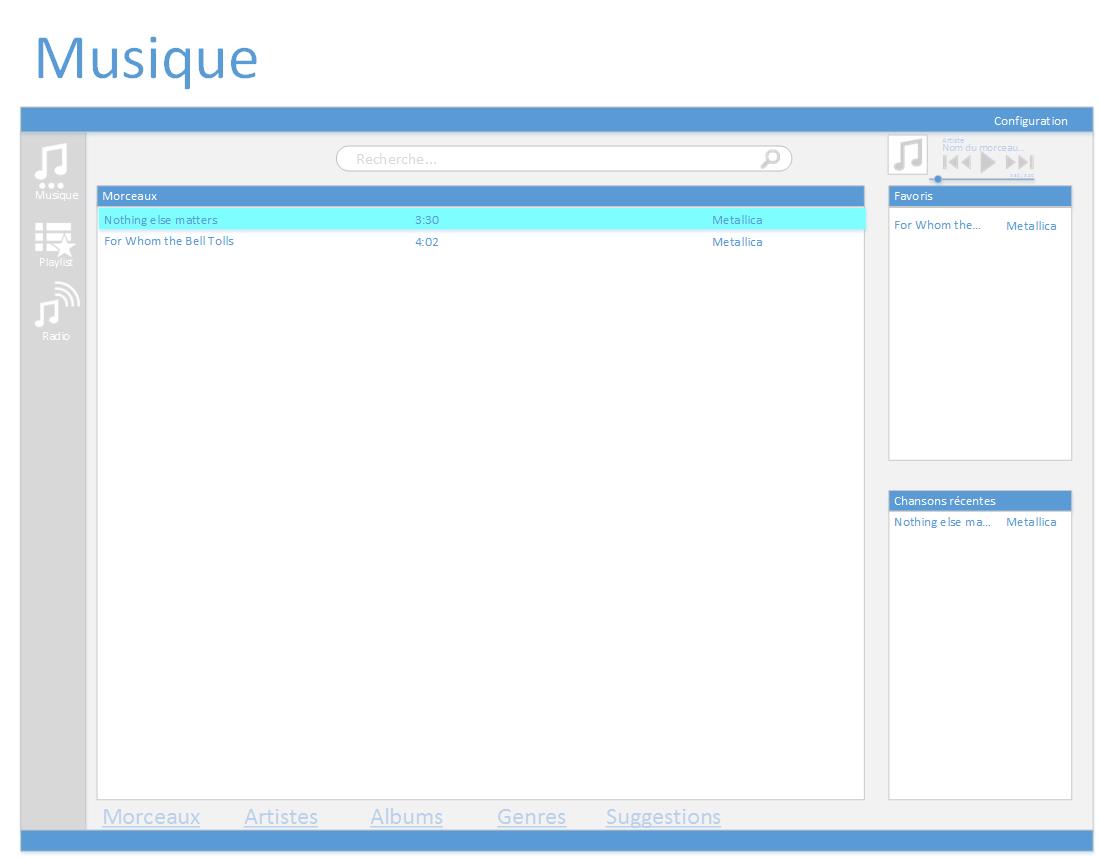
Ce projet me permettra de développer mes capacités en développement .NET. Il me donnera l’opportunité d’approfondir mes connaissances dans l’architecture MVVM.

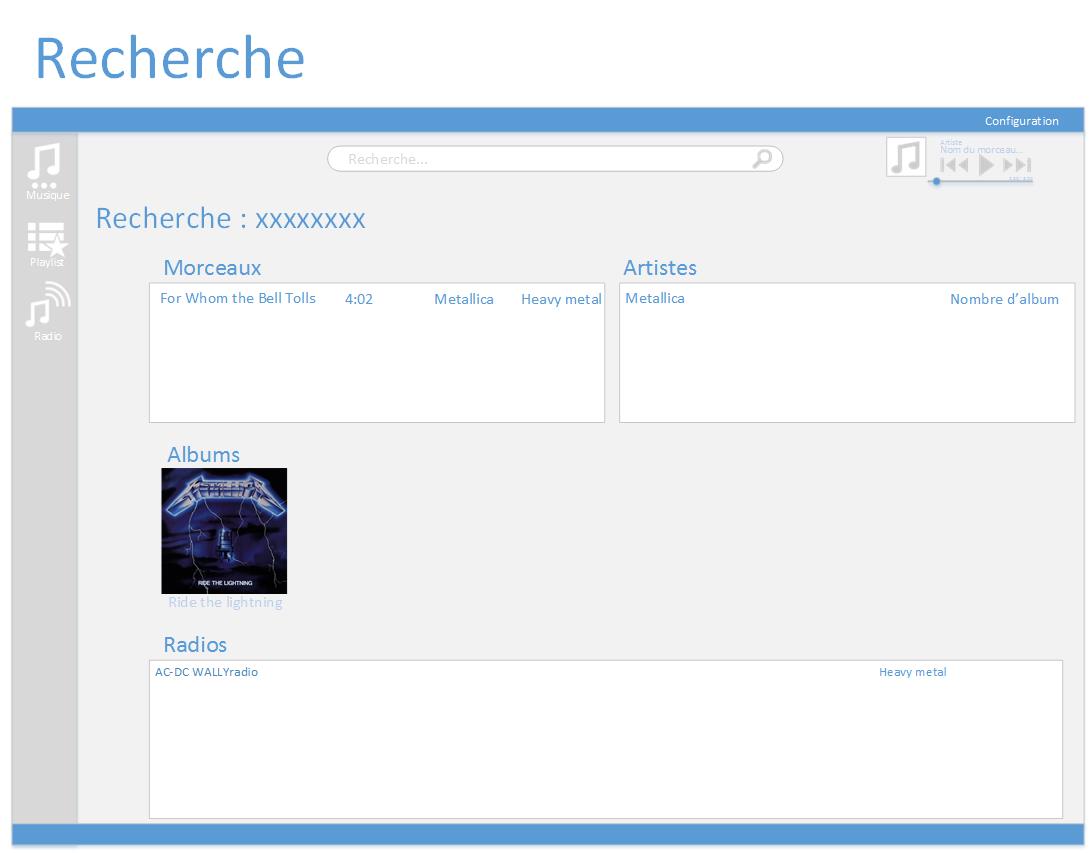
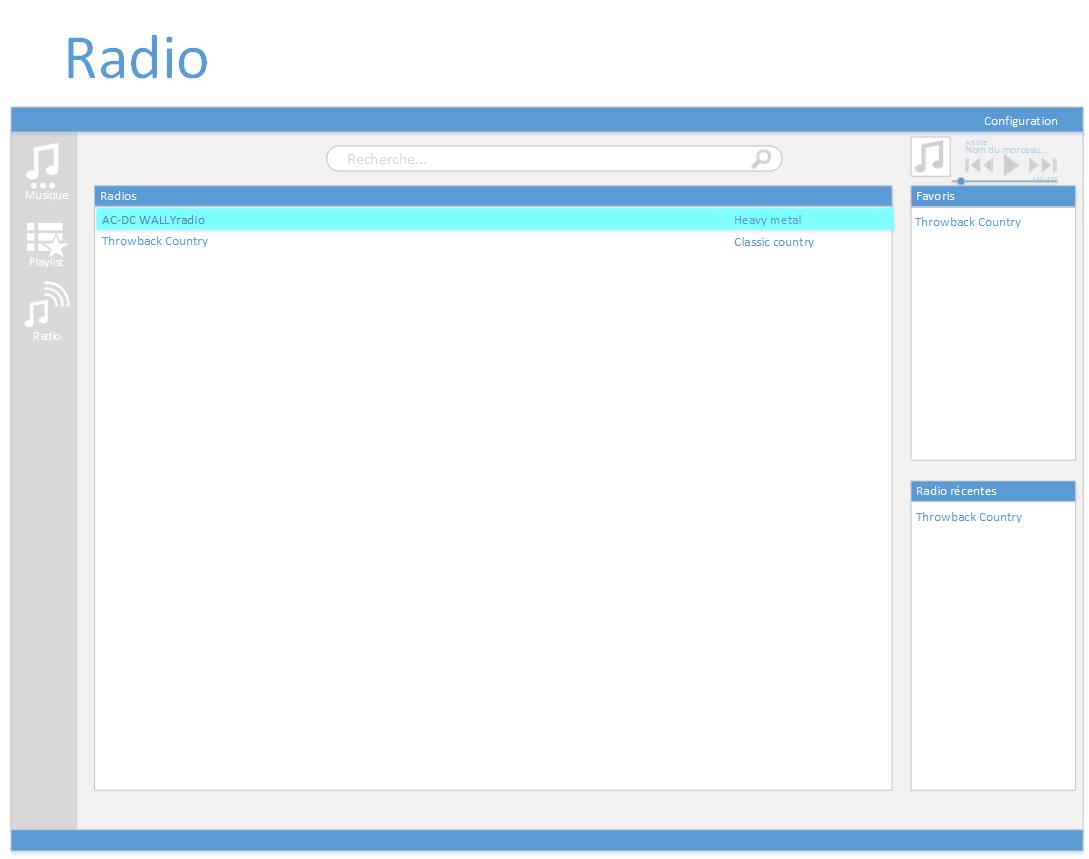
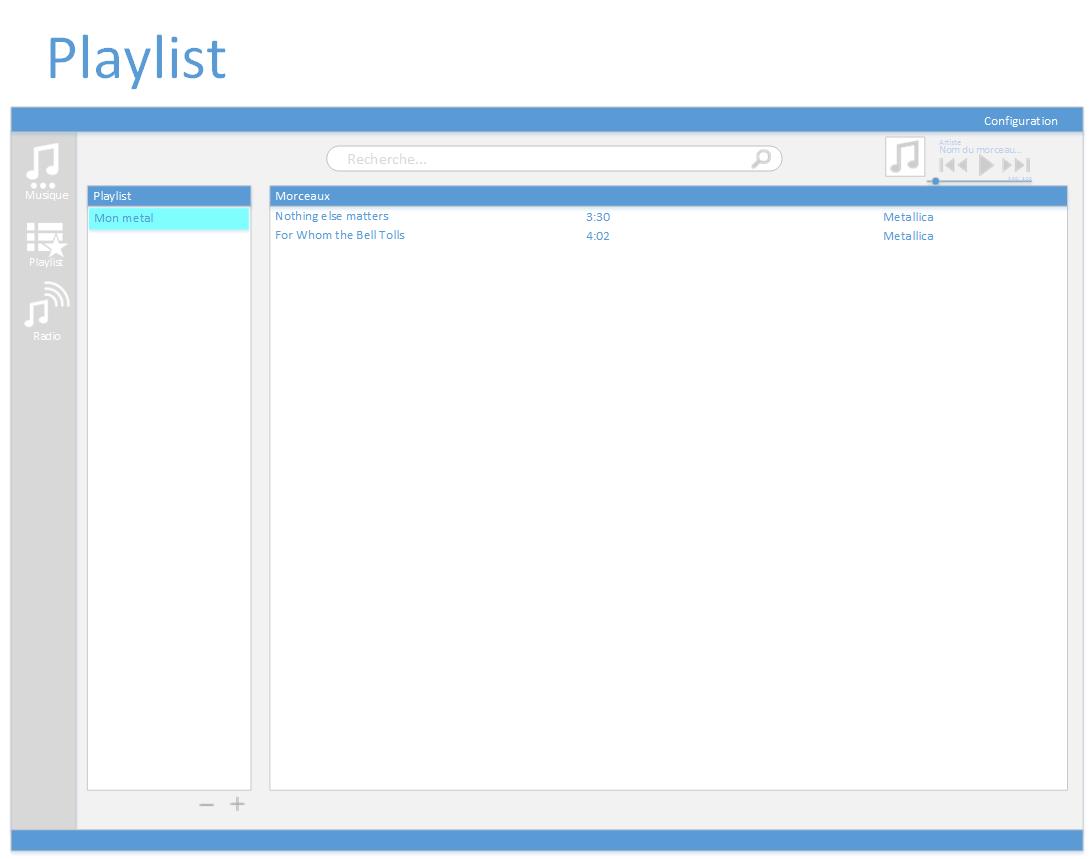
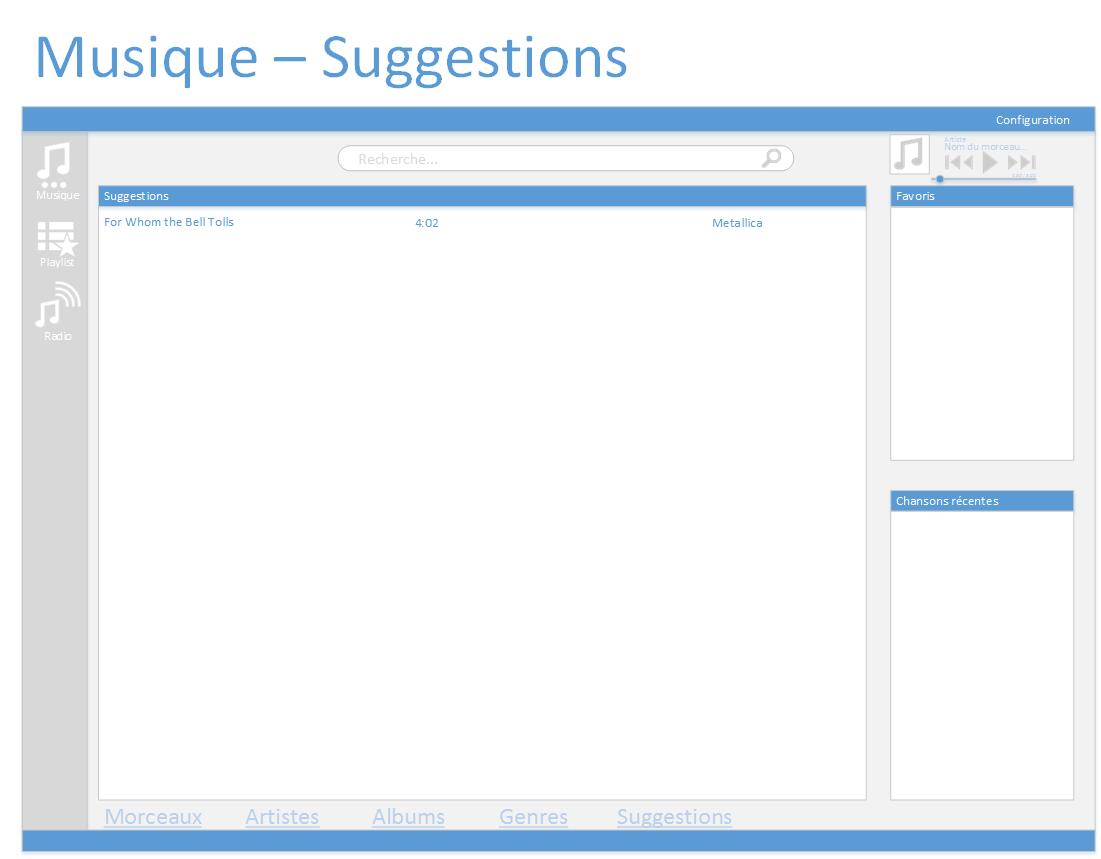
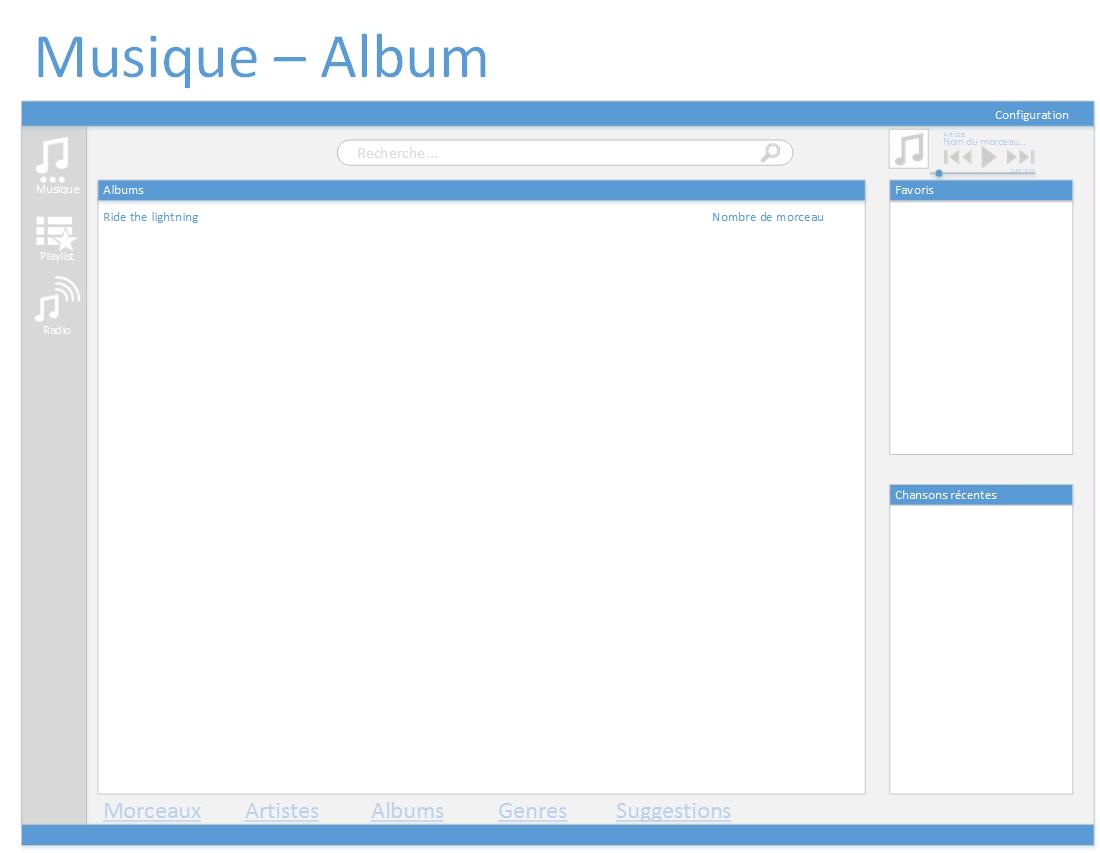
La principale difficulté sera de fournir une interface simple et ergonomique. Je pense donc partir sur un design flat et minimaliste, permettant ainsi une interface simple et intuitive. J’utiliserais pour cela la librairie Mahapps qui me permettra de fournir une interface épurée dans le style de Windows 8.

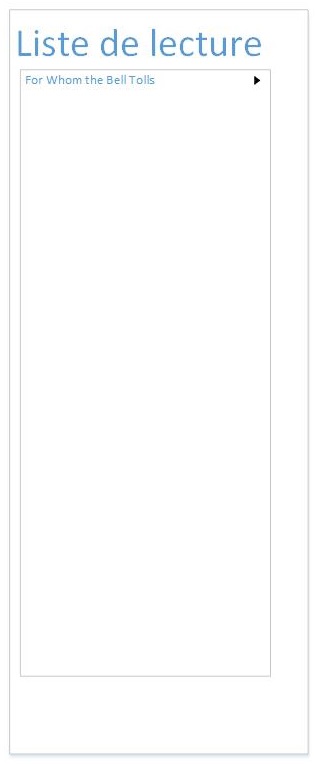
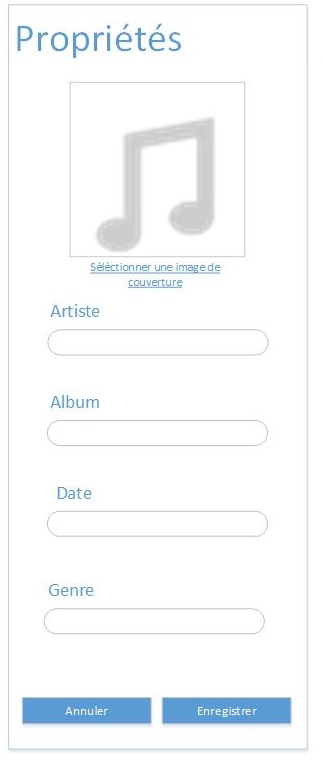
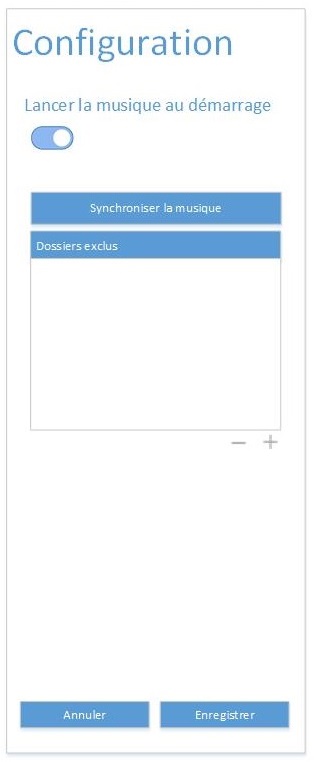
La sauvegarde du contexte de l’application sera un point difficile. Il y a plusieurs options pour pouvoir garder le contexte applicatif après la fermeture de l’application. Je pense créer une classe qui contient toutes les données à sauvegarder. Celle-ci sera enregistrée dans la base de données. À l’ouverture de l’application, je n’aurais alors plus qu’à interpréter les données de la classe pour les restaurer au bon endroit.

## Document d’analyse et conception

### Maquette







### Architecture

Je vais utiliser une architecture en multicouche d’une profondeur de 3 couches. Il y aura :

La couche DAL, qui s’occupera de récupérer les données depuis la base de données ainsi que depuis l’API de Web radio.

La couche BLL, qui se chargera de formater les données brutes reçues par la couche DAL

La couche Présentation, servira d’interface humain-machine elle aura la charge d’afficher les données à l’utilisateur ainsi que de récupérer ses entrées.

Quant au design paterne, j’utiliserai MVVM. Les view et les viewModel seront inclus dans la couche de Présentation. Le modèle sera l’ensemble de la couche DAL et BLL combiné.

Une partie DTO sera commune à toutes les couches, elle aura pour rôle de fournir le modèle des objets. Si une méthode est commune à toutes les couches, elle pourra être placée dans la zone « Méthode commune ».

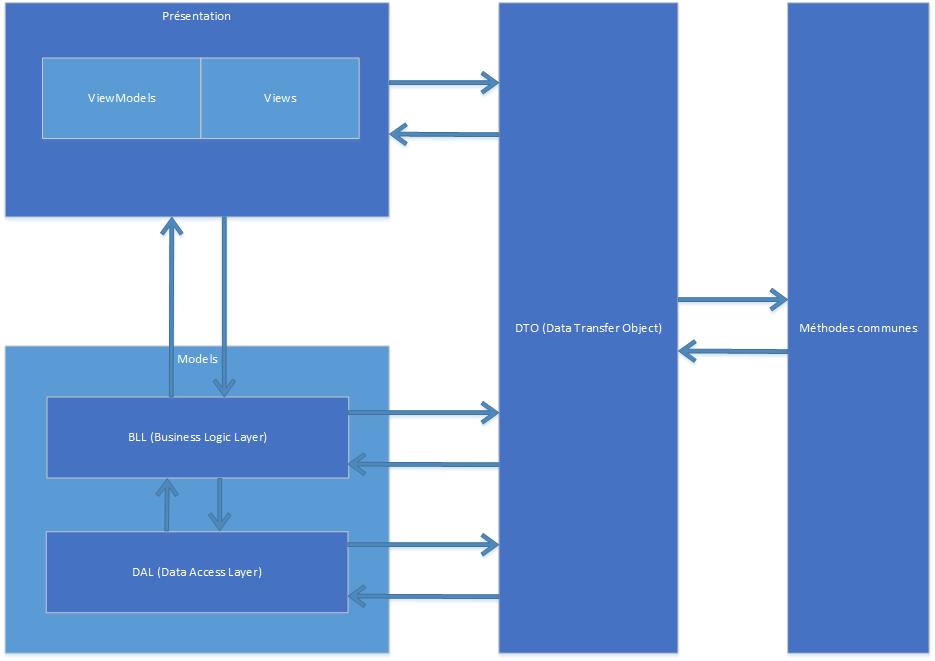


Figure 1 Illustration de l’architecture

### Librairies utilisées

***API ShoutCast***

ShoutCast est une plateforme de radio amateur qui propose plus de 50 000 radios. Elle propose aussi une API que j’utiliserais pour gérer la partie radio de l’application.

Il faut savoir que l’API Shoutcast n’est pas 100% sûre, et que parfois les données qu’elle fournit ne sont pas valides. Soit la radio a expiré, ou sois un problème passager empêche la bonne récupération du stream

***MvvmLight***

Permets la simplification des fonctionnalités propre à MVVM, comme le « Messenger », les « Command », et les mises à jour de la vue

***EntityFramwork***

ORM (Object Relational Mapping). L’ORM permet de simplifier l’utilisation des entités de la base de données en les faisant passer pour des objets en définissant une correspondance entre eux.

***Mahapps.Metro***

Ajoute des éléments graphiques pour rendre l’interface plus moderne

***NAudio***

Permet la lecture des fichiers et flux audio, j’ai choisi cette librairie, car elle est reconnue comme étant une bonne librairie. Elle acceptait tous les formats de fichier dont je pensais avoir besoin.

***NUnit***

Ajoute des fonctionnalités aux tests unitaires

***System.Data.SQLite***

Permet l’utilisation d’une base de données SQLite. Cette librairie inclus les binaires précompilés.

***SQLite.CodeFirst***

Permet d’utiliser les actions de EntityFramwork avec une base de données SQLite

### Structure du code

Le code sera structuré de manière suivante :

1. **DAL** *Projet Data access layer* 
   1. **API** *Dossier*
      1. **Shoutcast** *Contient toutes les fonctions concernant l’API de ShoutCast*
   2. **Database** *Dossier*
      1. **Configuration** *Contient la configuration de la base de données, ainsi que le SEED*
      2. **DbContext** *Crée un contexte de la base de données en utilisant une connexion string*
      3. **Repository** *Class générique permettant d’effectuer les actions de base sur la base de données (CRUD)*
2. **BLL** *Projet Business logic layer* 
   1. **AlbumData** *Permet d’effectuer des actions sur les données relatives aux albums*
   2. **ArtistData** *Permet d’effectuer des actions sur les données relatives aux artistes*
   3. **RadioData** *Permet d’effectuer des actions sur les données relatives aux radios*
   4. **TrackData** *Permet d’effectuer des actions sur les données relatives aux morceaux*
   5. **SearchData** *Permet d’effectuer des actions de recherches*
   6. **FavoriteData** *Permet d’effectuer des actions sur les données relatives aux favoris*
   7. **GeneralData** *Permet d’effectuer des actions sur les données relatives aux fonctionnalités générales de l’application*
3. **Presentation** *Projet Presentation* 
   1. **Helper** *Dossier contenant les classes utiles à la logique de l’application*
      1. **Context** *Gère le contexte applicatif du lecteur, contiendra par exemple la liste de lecture*
      2. **MusicPlayer** *Fonctions relative au lecteur de musique (play, pause, etc.)*
      3. **MusicSync** *Fonctions relative à la synchronisation des pistes musicales depuis l’ordinateur*
      4. **RightClick** *Fonctions relative au clic droit*
   2. **View** *Dossier contenant les vues de l’application*
      1. **Flyout** *Dossier contant les flyout de l’application*
         1. **MusicFlyoutView** *Flyout qui contiendra une liste soit d’album, soit d’artistes, soit de morceaux*
         2. **PropertyFlyoutView** *Flyout contenant les propriétés d’un morceau*
         3. **ReadingFlyoutView** *Flyout contenant la liste de lecture*
         4. **RunningFlyoutView** *Flyout contenant la vue du morceau en cours*
         5. **SettingFlyoutView** *Flyout contenant les options de l’application*
      2. **List** *Dossier contenant les listes de l’application*
         1. **AlbumListView** *Liste d’albums*
         2. **ArtistListView** *Liste d’artistes*
         3. **FavoriteListView** *Liste des favoris*
         4. **GenreListView** *Liste des genres*
         5. **PlaylistListView** *Liste des playlists*
         6. **ReadingListView** *Liste des morceaux en lecture*
         7. **RecentListView** *Liste des dernières radios écoutées*
         8. **SuggestionListView** *Liste de suggestion*
         9. **TrackListView** *Liste de morceaux*
         10. **RadioListView** *Liste de radios*
      3. **MusicView** Vues de la page musique
      4. **PlaylistView** Vues de la page playlist
      5. **RadioView** Vues de la page radio
      6. **SearchView** Vues de la page recherche
      7. **SmallPlayerView** Vues du petit lecteur de musique
   3. **ViewModel** *Dossier contenant les viewmodels*
      1. **MainViewModel** *ViewModel parent de tous les autres ViewModels, il est utile en cas de partage entre ViewModels*
      2. **MainWindowViewModel** *Gère l’affichage de la fenêtre principale*
      3. **MusicViewModel** *Gère l’affichage de toutes les vues musiques (morceaux, artistes, albums et genre)*
      4. **PlayerViewModel** *Gère l’affichage du petit player, ainsi que l’affichage du grand player dans la vue de la musique en cours*
      5. **PlaylistViewModel** *Gère l’affichage de la vue playlist*
      6. **PropertyFlyoutViewModel** *Gère l’affichage de la vue des propriétés*
      7. **RadioViewModel** *Gère l’affichage de la vue radio*
      8. **SettingFlyoutViewModel** *Gère l’affichage de la vue de configuration*
      9. **ViewModelLocator** *ViewModel locator, fait des singletons des ViewModels si l’on veut l’utiliser plusieurs fois avec le même contexte*
   4. **MainWindow** *Vue de la fenêtre layout, qui sera active sur toutes les vues*
4. **DTO** *Projet Data access object*
   1. **Entity** *Dossier contenant toutes les entités qui seront dans la base de données*
      1. **Album** *Table Album*
      2. **Artist** *Table Album*
      3. **Track** *Table Album*
      4. **BaseEntity** *Contient les propriétés communes à toutes les tables*
      5. **Context** *Table Context*
      6. **Playlist** *Table Playlists*
      7. **ExcludeFolder** *Table ExcludeFolder*
      8. **Genre** *Table Genre*
      9. **Radio** *Table Radio*
   2. **Audio** *Contient les propriétés communes entre la table Track et la table Radio*
5. **Shared** *Projet qui contient les classes susceptibles d’être utilisées dans plusieurs layers*
   1. **MusicFile** *Effectue des transformations sur les fichiers audios*

### Base de données

***Normes ETML***

Je n’ai pas respecté les normes de codage ETML au niveau de la base de données, car avec l’utilisation d’EntityFramework, la base de données devrait au final correspondre à ma structure de données dans le DTO. Certaines classes étant héritées, les noms des entités ne correspondraient plus au nom de leur table.

***MCD***

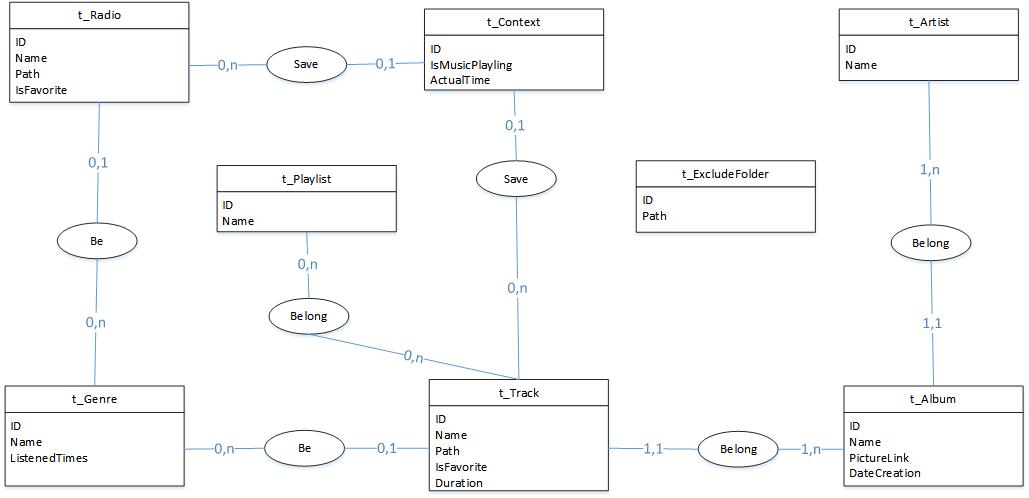
******

Figure 2 Modèle conceptuel de données

***MLD***

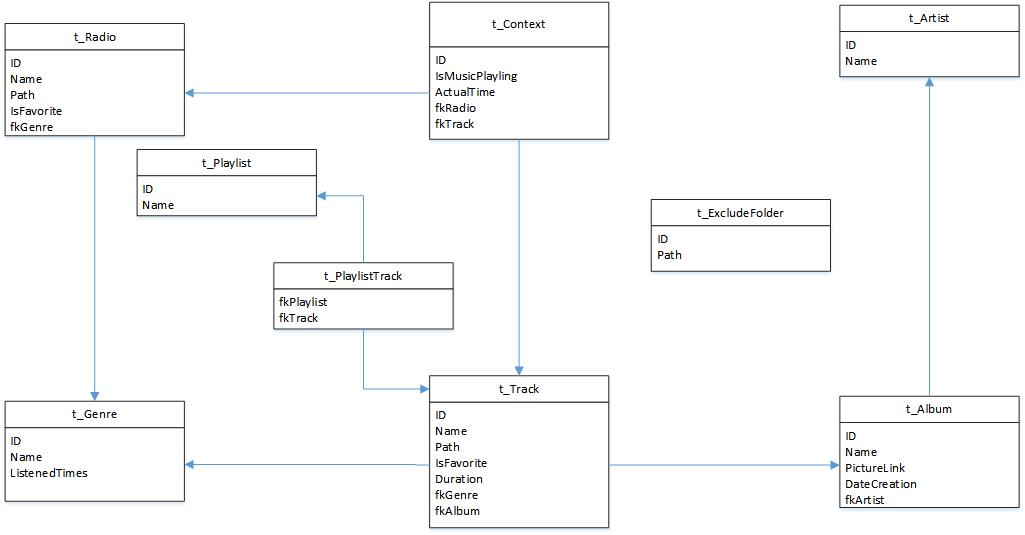
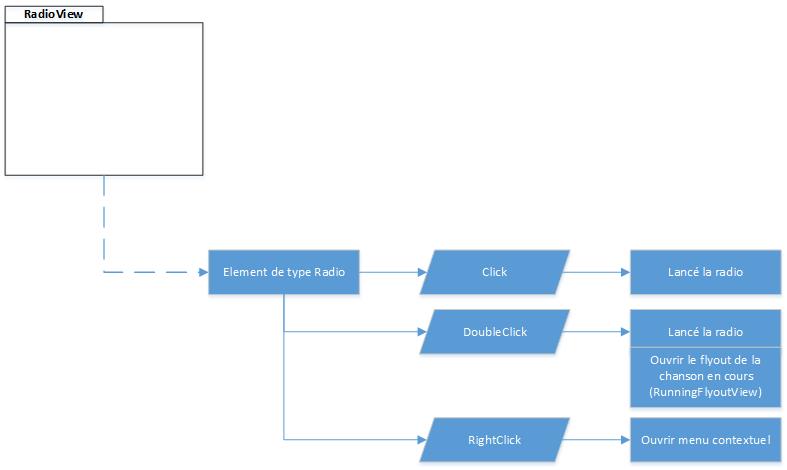
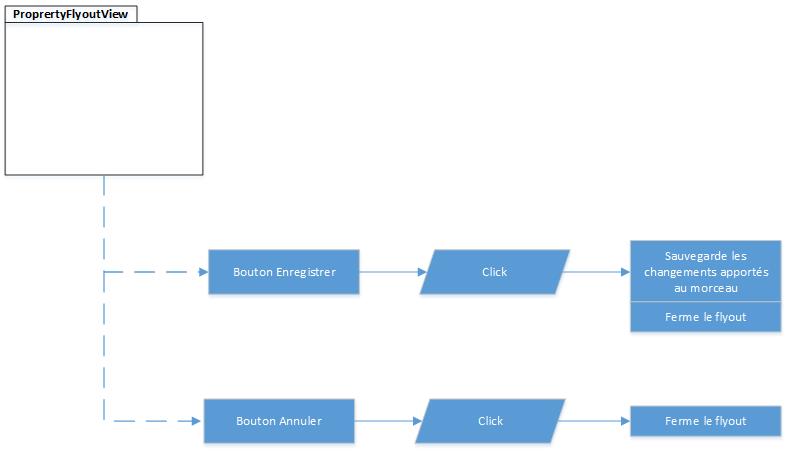
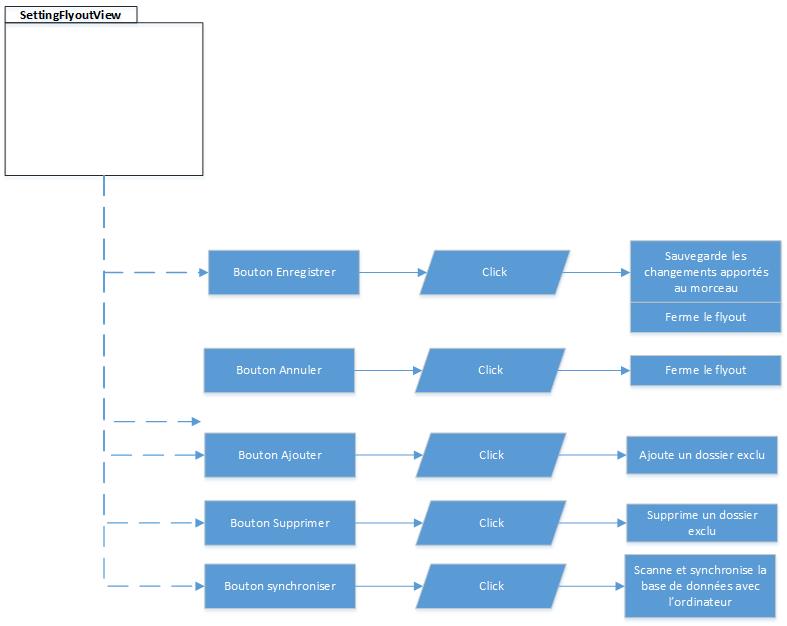
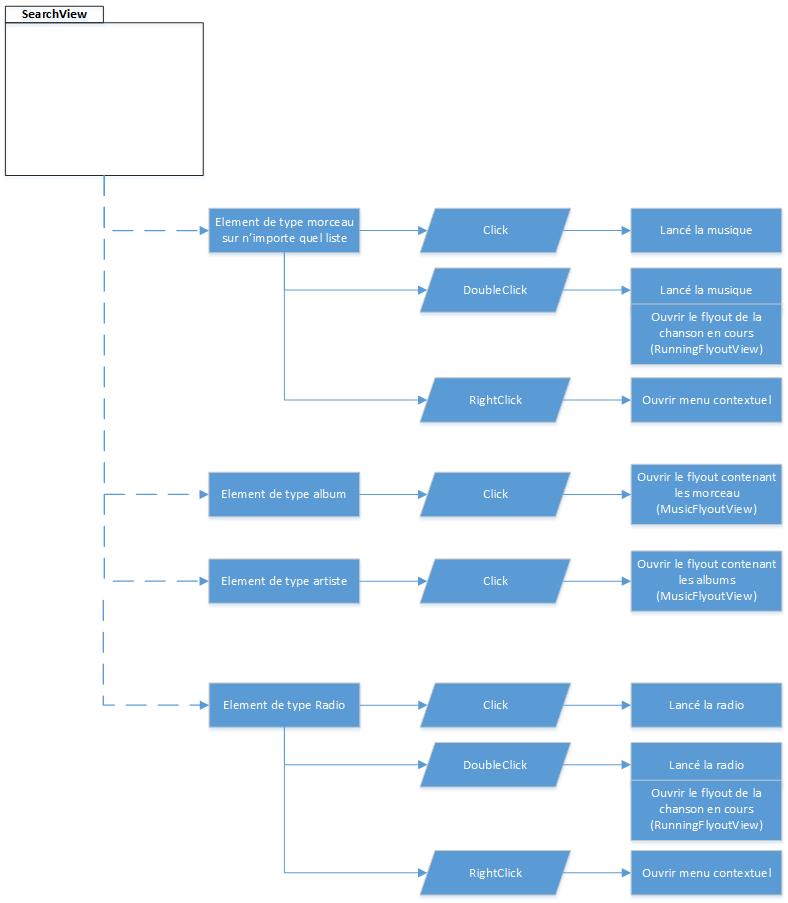
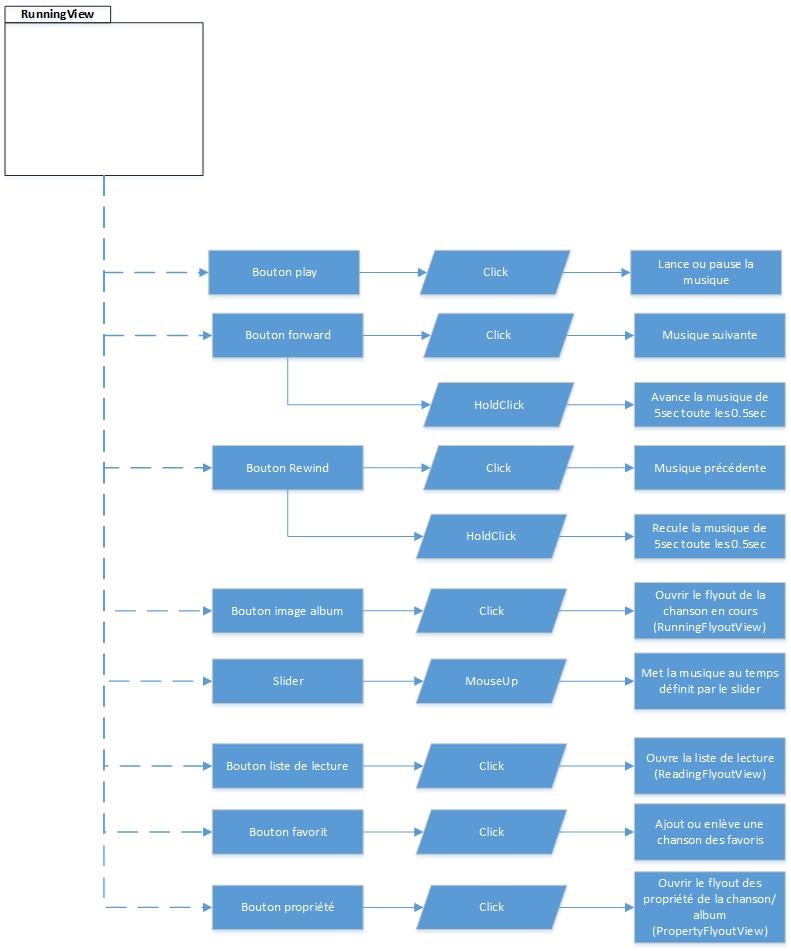
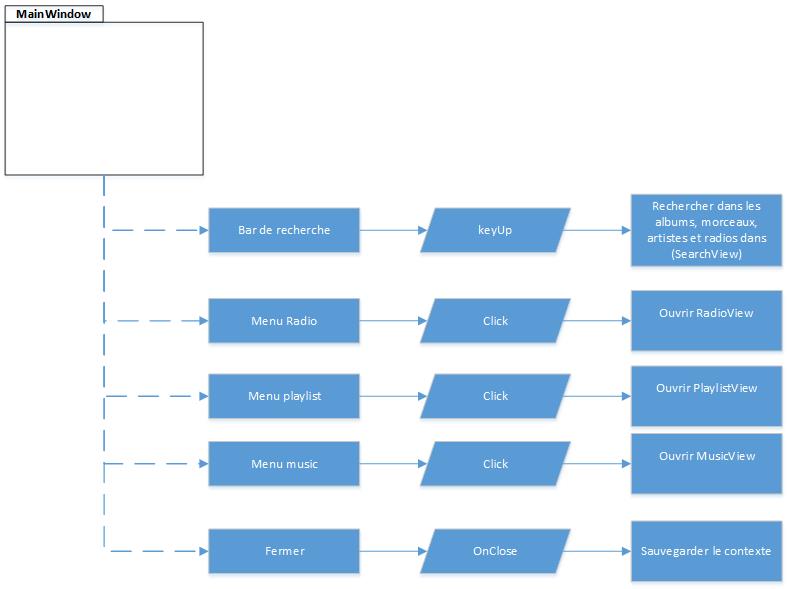
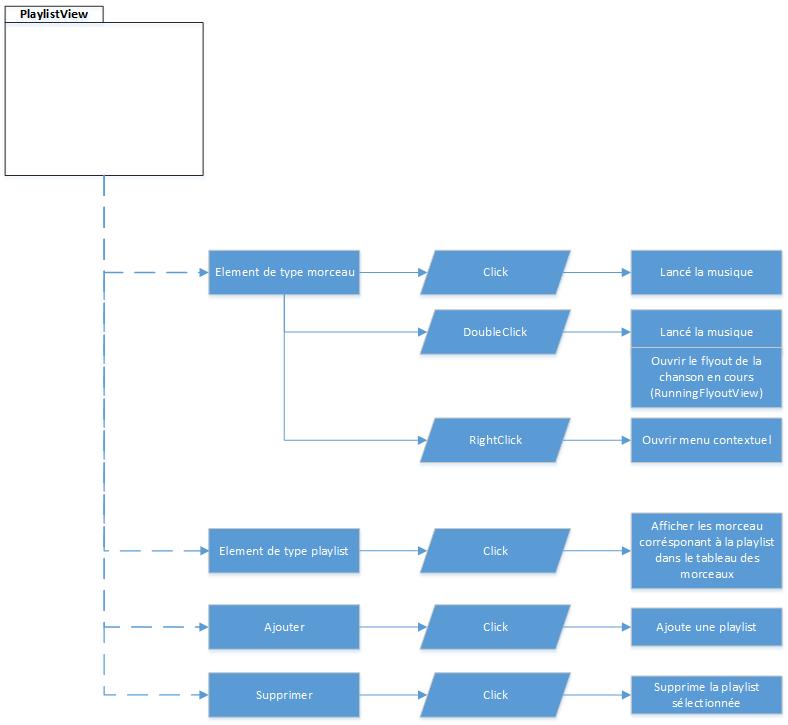
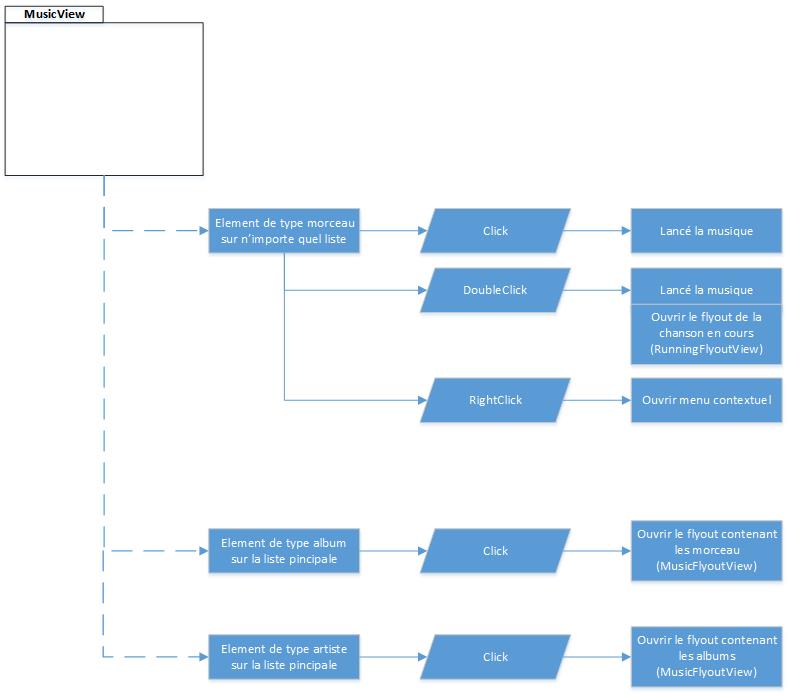
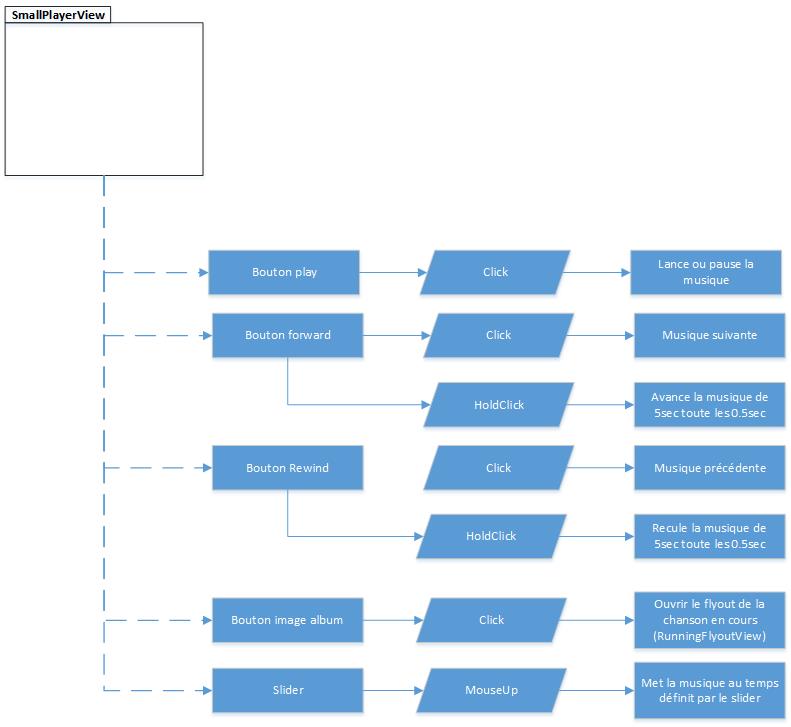


Figure 3 Modèle logique de données

### Schéma évènementiel

## Conception des tests

Les tests seront effectués grâce à un projet de tests unitaire créé dans la solution. Je n’utilise que de la musique classique pour tester mon application (fichiers non soumis à la loi du copyright).

Suite aux conseils de monsieur Sahli, l’application devra supporter au moins le MP3, le WAV et le WMA. Ces trois formats permettront de couvrir 90 %[[1]](#footnote-1) des flux renvoyés par les Web radio.

Je pratique le développement de type TDD (Test Driven Development), donc tous les tests sont écrits avant la création de l’application. Tous les tests, au départ, sont échoués. Plus le développement avance, plus les tests devront se passer correctement.

|  |  |
| --- | --- |
| **AudioTest** |  |
| LoadData() | Crée une classe artiste, album et morceau. |
| MusicTestFormat() | Teste les 3 formats obligatoires en essayant de les jouer (WAV, MP3, WMA) |
| MusicOperationTrack() | Teste les opérations telles que l’avance rapide, la pause |
| MusicOperationRadio() | Teste les opérations telles que la pause |
| CheckTrackInfo() | Vérifie que les données reçues par la fonction sont identiques aux données du fichier de musique |
| FavoriteAddRemovTrackAndRadio() | Ajoute puis supprime une radio et un morceau de la liste des favoris |
| CreatePlaylist() | Création d’une playlist |
| CheckPlaylist | Teste les opérations de bases sur la playlist (précédent, suivant) ; ajoute puis supprime un morceau de la playlist ; supprime la playlist. |

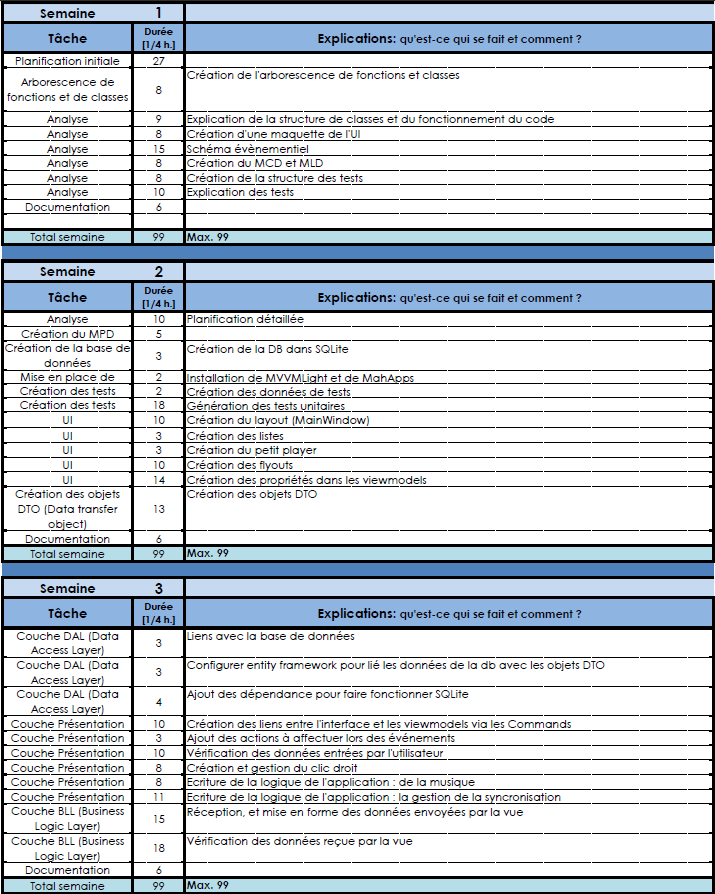
|  |  |
| --- | --- |
| **ContextTest** |  |
| CreateAndSetContext() | Crée un contexte puis essaie de l’attribuer à l’application. |
| ReadingListOperation() | Teste la liste de lecture avec les opérations de base telles que suivant et précédent ; teste l’ajout et la suppression d’élément dans la liste de lecture. |

|  |  |
| --- | --- |
| **DatabaseTest** |  |
| CheckConnection() | Teste la connexion à la base de données |

|  |  |
| --- | --- |
| **RadioTest** |  |
| CheckAPIConnection() | Vérifie que la connexion avec l’API s’effectue correctement |
| ChackApiData() | Teste que les données reçues par l’API sont correctes |
| CheckRadioInfo() | Vérifie que les données de l’élément radio correspondent aux données attendues |

|  |  |
| --- | --- |
| **SyncTest** |  |
| AnalyseFolderTree() | Vérifie que l’arborescence de fichier contient et analyse bien tous les dossiers |
| SyncFolder() | Tente de synchroniser un dossier test |

# H:\TPI\PlanificationDétailée\planif2.PNGPlanification détaillée



# Réalisation

## Dossier de Réalisation

### MPD

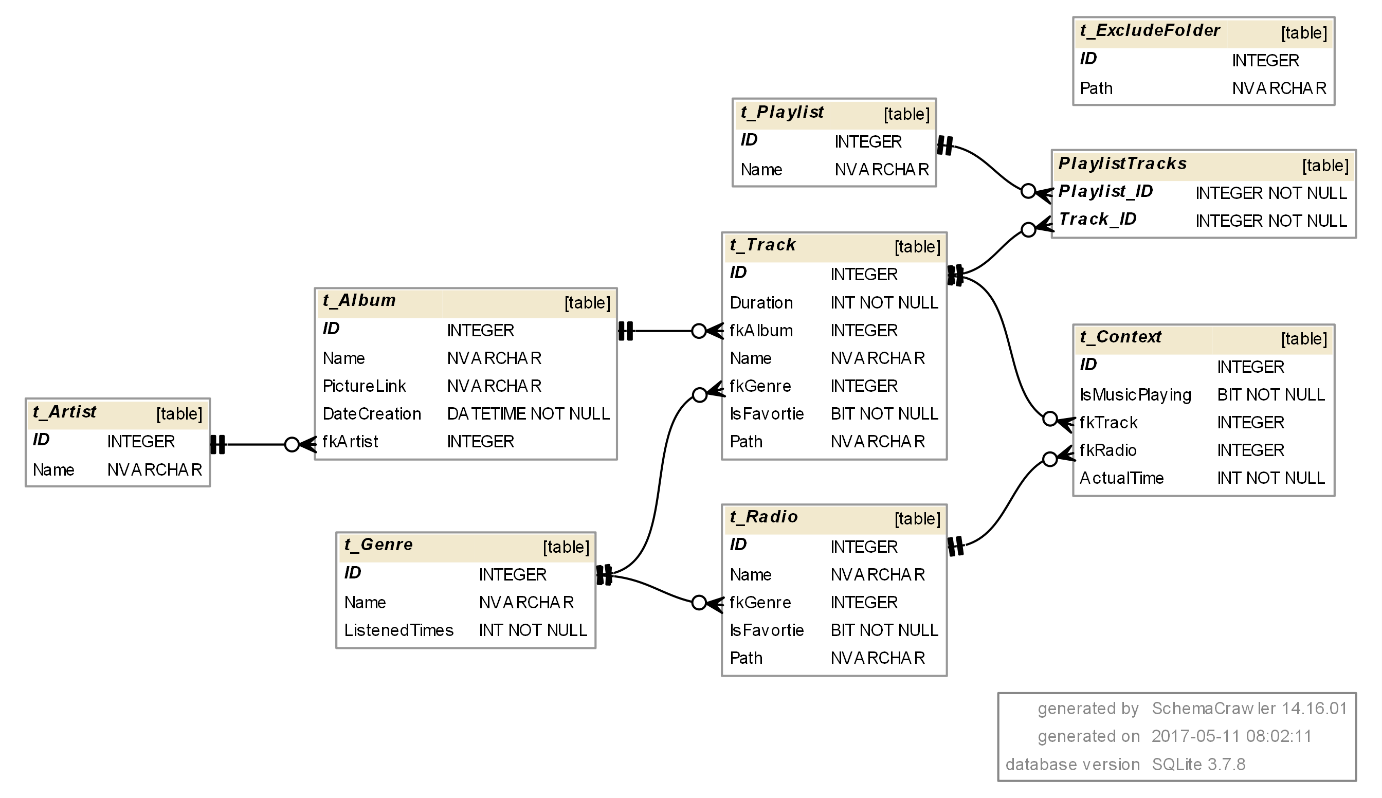


Figure 4 Model physique de données

### Dossiers exclus

Au début de mon application, j’étais parti sur le fait de gérer la synchronisation par rapport aux dossiers exclus. Après mûre réflexion, nous avons décidé d’inclure les dossiers à analyser plutôt qu’à les exclure.

### Lecture des radios

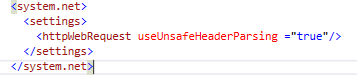
Pour la lecture de radio, j’ai rencontré un problème lors de la lecture des flux AACP, ces flux étant encodés en AAC (format que NAudio ne supporte pas).

Pour lire les flux AAC, j’avais l’option d’utiliser la fonction utilisant la DLL Média Fondation, celle-ci est une librairie créée par Microsoft utilisant les DLL de Windows Media Player. Cependant, bien qu’ayant une fonction permettant de lire des streams, celle-ci dispose le stream après la première itération, ce qui me laissait avec seulement quelque seconde de musique.

Pour résoudre ce problème, moi et Monsieur Melly avons choisi la solution de créer un flux alternatif encodé en MPEG *(format mp3)* à partir du flux AAC. Pour ce faire, j’ai utilisé la DLL LAME, permettant de convertir un flux AAC en MPEG.

### Stream radio

J’ai rencontré une difficulté toute particulière lors de la lecture des streams radio. Les streams utilisant la version 2.x de Shoutcast étaient lus sans problème. Tandis que ceux utilisant la version 1 me retournaient une erreur de type : « Serveur violation ».

Ce problème était dû à un problème de header. Pour résoudre ce problème, j’ai ajouté cette ligne pour empêcher que l’erreur survienne

Synchronisation*.*

La synchronisation sur un ordinateur de moyen de gamme prend environ 20 secondes pour 300 morceaux. D’une synchronisation pour simplement mettre à jour les fichiers, le temps passe à 15 secondes.

## Modifications

**Modification de la structure de la base de données**

17.05.2017 : j’ai supprimé le dossier commun de la gestion du clic droit. J’ai trouvé plus pratique de le gérer directement dans les vues concernées.

17.05.2017 : J’ai créé un fichier PlaylistData pour gérer les playlists plus simplement. Précédemment, les playlists étaient gérées directement dans le TrackData ce qui pouvait porter à confusion.

18.05.2017 : J’ai dû modifier la structure de mon élément radio en ajoutant notamment les données venues de l’API (description, le format, l’URL du logo et l’ID Shoutcast).

19.05.2017 : Ajout de la donnée : LastListen, dans le modèle des radios pour avoir la date de dernière lecture. Celle-ci sera utile lors de la récupération des radios récente.

## C:\Users\preisiger\Desktop\graph.pngMDP final

Figure 5 Model physique de données à la fin du projet

Sur le MPD final, nous pouvons voir quelques changements, notamment la table « t\_Radio » qui se retrouve bien plus fournie.

La table t\_ExcludeFolder qui est devenue t\_IncludeFolder.

La table t\_Context a également gagné l’attribut « IsMusicPlaylingOnStrat » qui représente le choix de si oui ou non la musique doit directement se lancer au démarrage.

Il y a également les entités listenedTimes dans « t\_Track » et « t \_Genre », qui ne sont pas utilisées, mais que j’ai mises pour respecter le cahier des charges. Dans le futur, si elles sont utilisées, elles serviront à traquer les habitudes de l’utilisateur**.**

## MVVM

J’ai choisi le pattern MVVM, car il permet de travailler sur les vues indépendamment des ViewModel. Les vues sont alors indépendantes, donc si un autre logiciel utilise la même vue, je n’aurai qu’à faire un simple copier/coller. La logique elle aussi est bien plus réutilisable, car non polluée par les éléments de la vue.

Le paterne me permet également d’utiliser la même logique sur plusieurs vues différentes et vice-versa. Cet avantage me permet d’économiser énormément de lignes de codes.

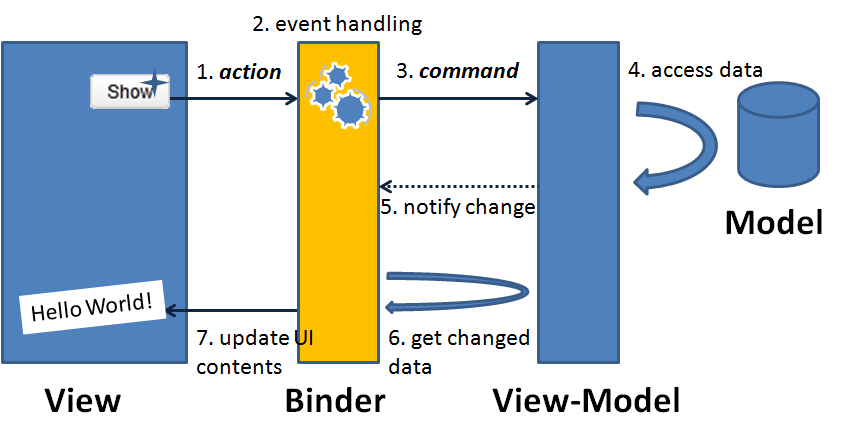
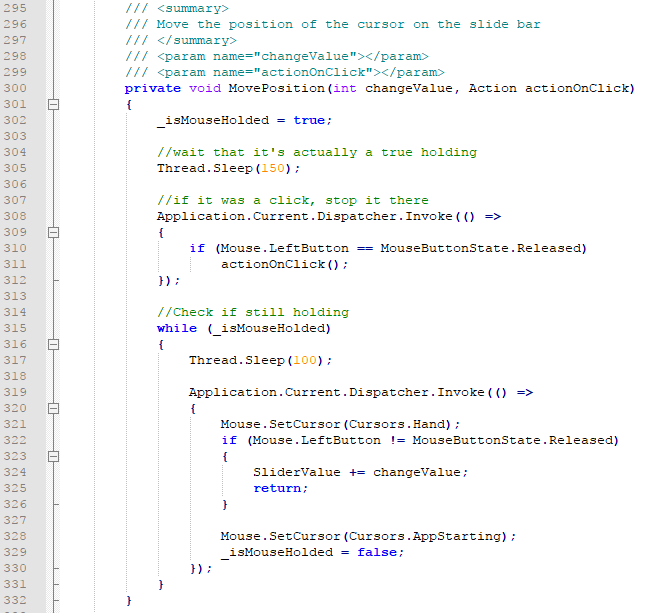


Figure 6 Schéma MVVM <https://www.zkoss.org/_w/images/f/fb/SmallTalk_MVVM_HELLO_FLOW.png>

## Extrait de code



Cette fonction bouge le curseur du slider sur la barre du temps. Elle est appelée lors que l’utilisateur appuie en continu sur avance rapide ou retour rapide.

À la ligne 305, on attend 150 ms pour faire la différence entre un clic et un appui long. Ensuite, de la ligne 308 à 312, on vérifie le statut du clic, 150 ms plus tard. Si le clic est relâché, lancer la fonction assignée au clic (musique suivante ou précédente). Sinon tant que le clic est enfoncé, chaque 100 ms, on avance ou recule la musique d’un certain temps donné.

Le Application.Current.Dispatcher sert à exécuter la commande dans le thread principal de l’application. Donc même en étant dans un thread différent, les éléments visuels sont accessibles et modifiables.

# Tests

## Dossier des tests

|  |  |
| --- | --- |
| ***Version de l’application testée*** | 1.0.0 |
| ***Date du test*** | 29.05.2017 |
| ***Nom du testeur*** | Eric Preisig |

Scénario 1 : Synchronisation

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | La musique des dossiers cibles se synchronise | Lors du premier démarrage, on demande à l’utilisateur de choisir s’il veut synchroniser son répertoire de musique |
| Quand | Lors du premier démarrage ou lors du clic sur le bouton synchroniser depuis la configuration | |
| Résultat | Les musiques qui ne sont plus dans les dossiers cibles se suppriment et les dossiers cibles sont analysés récursivement. Vérifie aussi que les fichiers déjà présents dans la base de données existent toujours. | |

Scénario 2 : Lecture de musique

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | Lit une musique |  |
| Quand | Lorsque l’utilisateur clique sur un morceau depuis la page musique | |
| Résultat | La musique se lance, le volet de la musique s’ouvre et la liste de lecture est mise à jour. | |

Remarque :

|  |
| --- |
| Si la musique a été déplacée, le logiciel avertit l’utilisateur, puis supprime la musique de la base de données ainsi que de la liste de lecture |

Scénario 3 : Lecture de radio

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | Lit une radio |  |
| Quand | Lorsque l’utilisateur clique sur une radio depuis la page radio | |
| Résultat | La radio se lance | |

Remarque :

|  |
| --- |
| Si la musique a été déplacée, le logiciel avertit l’utilisateur, puis supprime la musique de la base de données ainsi que de la liste de lecture |

Scénario 4 : Recherche

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | La recherche de morceaux, artistes, albums et radios |  |
| Quand | L’utilisateur appuie sur la touche « enter » après avoir écrit un mot dans la barre de recherche | |
| Résultat | La recherche s’effectue, elle est rapide. La recherche des radios ne bloque pas l’application et apparaît peu après le reste. | |

Remarque :

|  |
| --- |
| La recherche est rapide, mais elle peut devenir plus lente selon le nombre de fichiers. J’ai testé la recherche avec environ 4000 fichiers, avec un résultat prenant moins de 2 secondes. |

Scénario 5 : L’application démarre

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | Le contexte se récupère correctement |  |
| Quand | Quand l’application s’ouvre | |
| Résultat | Si l’utilisateur écoutait un morceau, celui-ci se lance correctement et reviens au moment précédent l’extinction du logiciel.  Si l’utilisateur écoutait une radio, celle-ci se lance correctement | |

Remarque :

|  |
| --- |
| Si une erreur intervient pendant la récupération (la radio ou le morceau n’existe plus, la radio est indisponible…), le contexte est alors supprimé |

Scénario 6 : Suppression

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | Suppression d’un morceau |  |
| Quand | Quand l’utilisateur effectue un clic droit sur un morceau et le supprime | |
| Résultat | S’il le supprime de disque dur, le fichier est alors supprimé du disque. La liste des morceaux se met à jour, ainsi que la liste de lecture. En cas de suppression de la bibliothèque, le fichier est simplement supprimé de la bibliothèque de l’application (ainsi que des listes qui utilisaient le morceau) | |

Remarque :

|  |
| --- |
| Si le fichier n’est pas supprimable, une erreur se produit |

Scénario 6 : Mise en favoris

|  |  |  |
| --- | --- | --- |
| Étape | Description | Remarque |
| Quoi | Met un morceau ou une radio en favoris |  |
| Quand | Lorsque l’utilisateur clique sur l’étoile | |
| Résultat | Met le morceau ou radio en favoris, et l’affiche dans la liste des favoris de sa vue. | |

## Tests unitaires

J’ai dû supprimer plusieurs tests, car la manière dont ils fonctionnaient n’était pas compatible avec la manière dont le logiciel fonctionne. Je pensais pouvoir créer des ViewModel dans les tests unitaires, malheureusement, je ne peux pas. Aussi toutes les interactions avec la musique sont bloquées (play/pause, etc.).

J’ai aussi supprimé toutes les interactions avec la base de données dans les tests, et essayé de les adapter grâce à du code en dur.

# Conclusion

## Bilan des fonctionnalités demandées

|  |  |
| --- | --- |
| Fonctionnalité | Résultat |
| *Synchronisation au premier lancement* | OK |
| *Habitudes d’écoutes de l’utilisateur* | OK |
| *Synchroniser à la demande* | OK |
| *10 dernière radio écoutée* | OK |
| *Rechercher des chansons* | OK |
| *Concevoir des playlists* | OK |
| *Jouer des chansons (play, pause, slider, suivant, précédent)* | OK |
| *Lecture continue, aléatoire, répétée* | OK |
| *Recherche des radios sur Internet* | OK |
| *Information sur la chanson en cours* | OK |
| *Lire un flux de radio* | OK |
| *Reprendre le dernier contexte applicatif* | OK |

Suite aux tests de M.Sahli, une liste de fonctionnalité et de changement pratique a été faite. Si je devais continuer le logiciel, j’améliorerais ces points précis.

|  |
| --- |
| Remarques et amélioration |
| Ne pas bloquer l’application lors de la synchronisation et permettre à l’utilisateur de voir les morceaux synchronisés |
| Se souvenir des tris effectués sur les vues lors du changement de vue |
| Prendre en compte les tris pendant la création de la liste de lecture |
| Afficher le morceau en cours en surbrillance et déplacer le scroll à celui-ci |

## Bilan de la planification

Voici la liste de tâches qui m’ont pris significativement plus ou significativement moins de temps que planifier. (Le temps est en quart d’heure).

|  |  |  |  |
| --- | --- | --- | --- |
| Tâche | Prévus | Effectuer | Raison |
| Planification initiale | 27 | 14 | J’ai fini plus tôt mon analyse initiale.  Ce qui m’a permis de commencer mon analyse plus tôt |
| Création de la base de données | 3 | 0 | J’ai configuré EntityFramwork pour la créer automatiquement |
| Mise en place de l’architecture MVVM | 2 | 0 | Je l’ai fait durant la création de l’arborescence de classe |
| Création de tests unitaire | 20 | 5 | Je n’avais pas fait beaucoup de tests unitaires auparavant, donc j’ai surestimé le temps |
| UI | 40 | 23 | J’ai utilisé plusieurs fois les mêmes vues, j’ai donc gagné beaucoup de temps |
| BLL  Présentation | 72  80 | 44  104 | J’ai placé plusieurs fonctions du BLL à la couche présentation |
| Installateur | 20 | 5 | L’utilisation de ClickOnce (installateur intégré à Visual Studio) m’a permis de créer l’installateur rapidement.  Cependant, j’ai eu beaucoup de bugs à résoudre par rapport à celui-ci |
| Manuel utilisateur | 27 | 9 | J’ai surestimé le temps de travail que me prendrait la documentation utilisateur |

## Bilan personnel

Si le projet était à refaire, je ne changerais que l’installeur : ClickOnce, car bien que très pratique au niveau du déploiement (il se dépolit très en un clic, et n’as pas besoin des droits administrateurs pour s’installer). Il introduit énormément de bugs, notamment au niveau des dépendances dont le programme a besoin.

Ce projet m’a permis d’apprendre à gérer des flux audios en continu. Il m’a aussi appris à utiliser la librairie NAudio, celle-ci pouvant être très utile dans d’autre projet contenant de la musique.

Si ce projet était à continuer, j’ajouterais en priorité une fonction permettant de remplir automatiquement les métadonnées des morceaux. Permettant ainsi à l’utilisateur d’éviter la tâche fastidieuse de les remplir à la main.

Ensuite, je ferais une option permettant le partage de musique entre utilisateurs, car cette option me semble inédite dans le milieu des logiciels audio.

## Remerciement

Je remercie **M. Sahli**, mon maître de projet, qui en plus de m’assister tout au long du TPI, a aussi testé mon application.

Je remercie **M. Melly**, qui m’a aidé à résoudre un problème qui empêchait la lecture de certains flux radio.

Je remercie **Eric Taylor** qui m’a fourni un ensemble de musique libre de droits. Ce qui m’a permis de tester mon application.

# Webographie

Mes sources principales lors de ce TPI étaient :

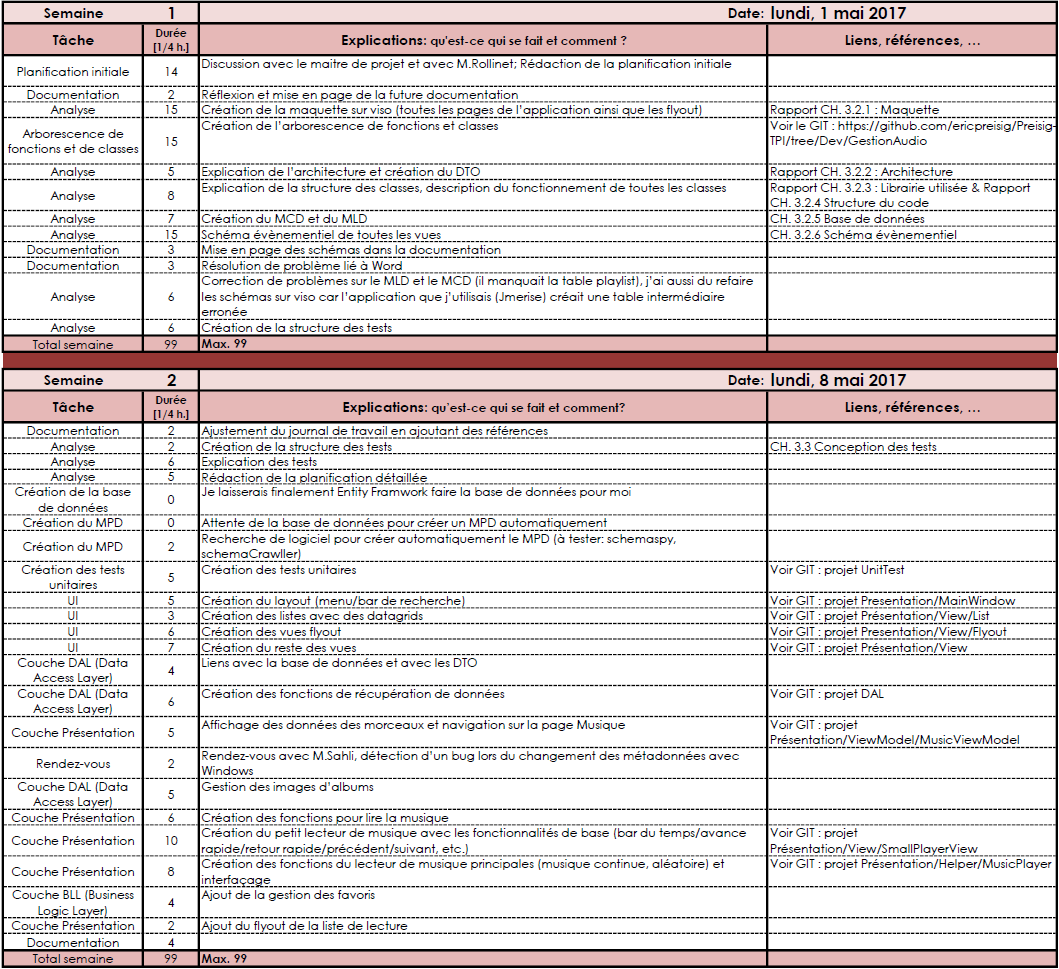
**Stack Overflow**: <https://stackoverflow.com>

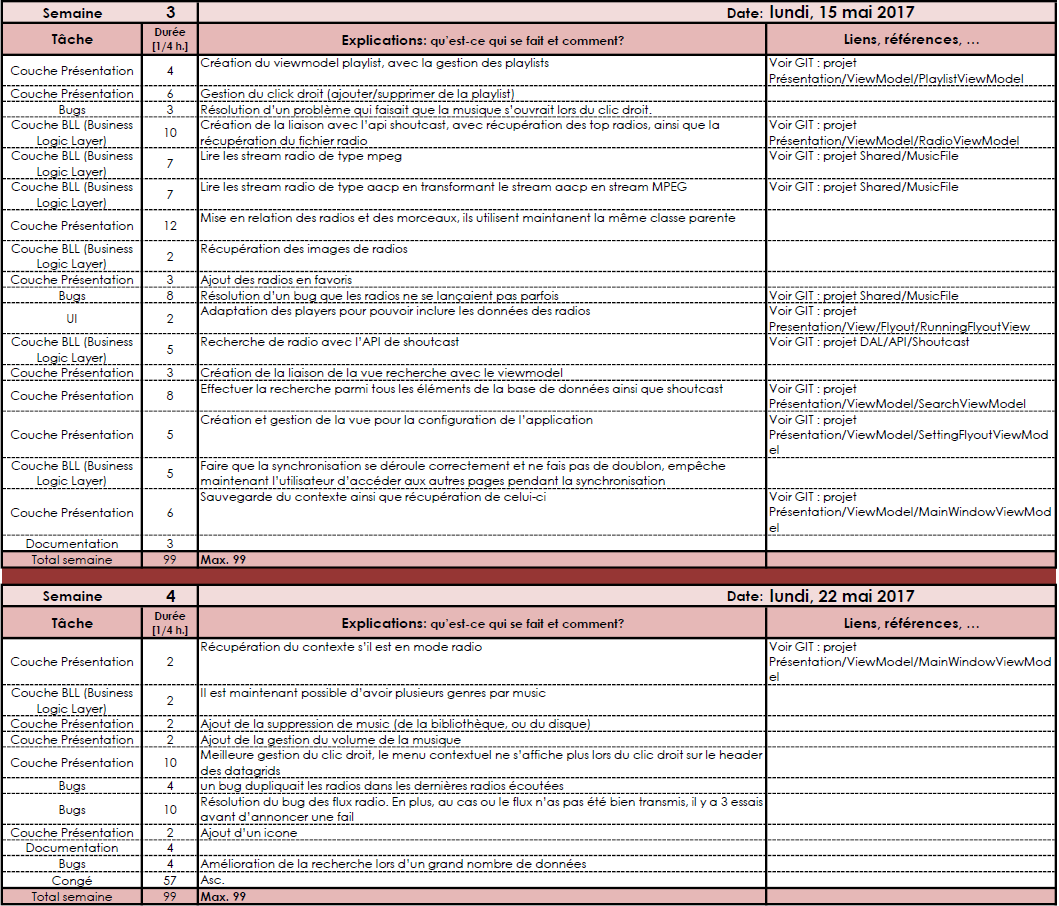
**Microsoft MSDN**: <https://msdn.microsoft.com>

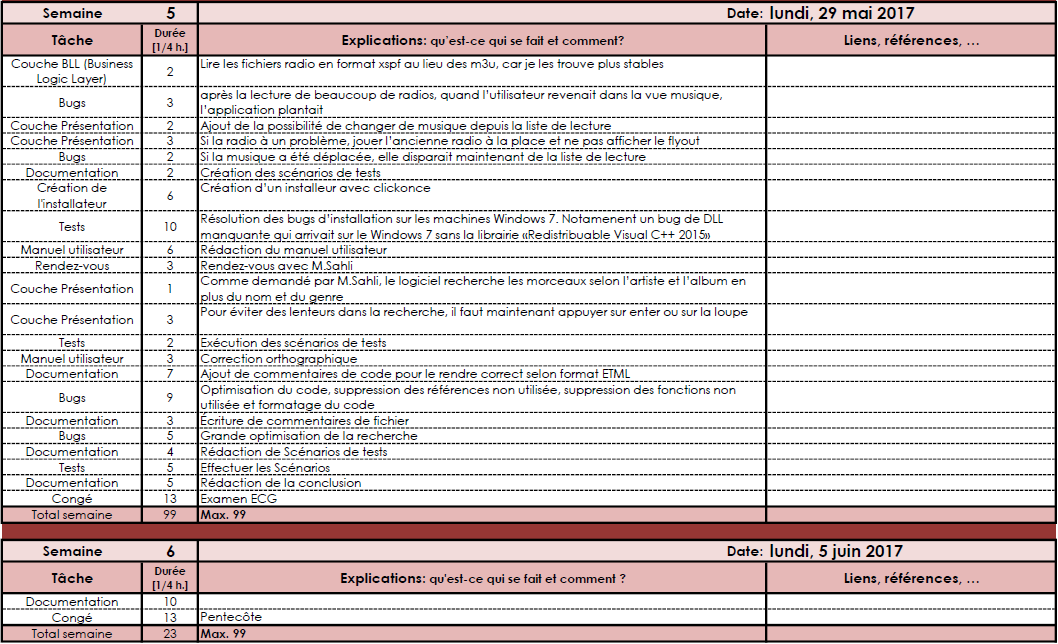
**Microsoft MSDN forum** : <https://social.msdn.microsoft.com>

# Divers

## Journal de travail

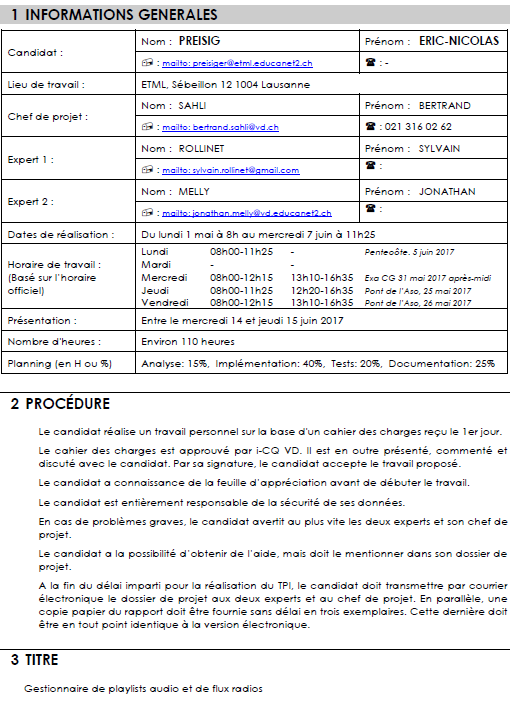


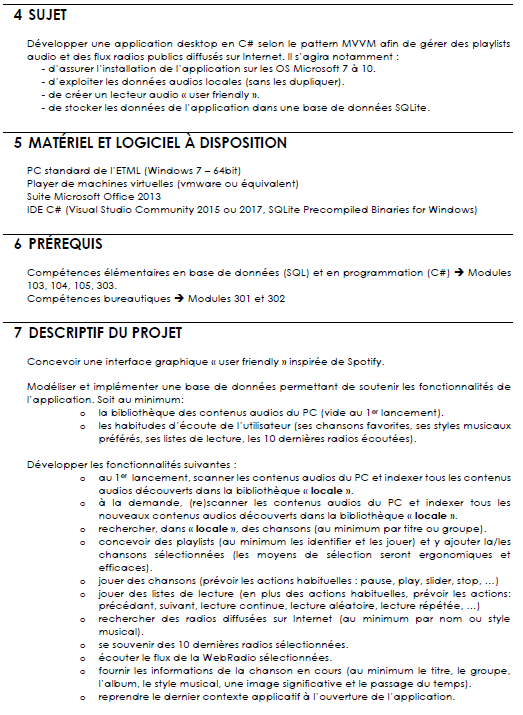


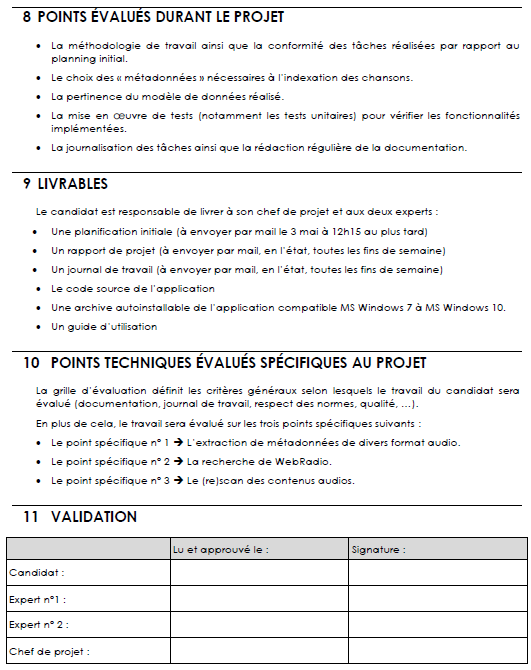


# Annexes

## Cahier des charges



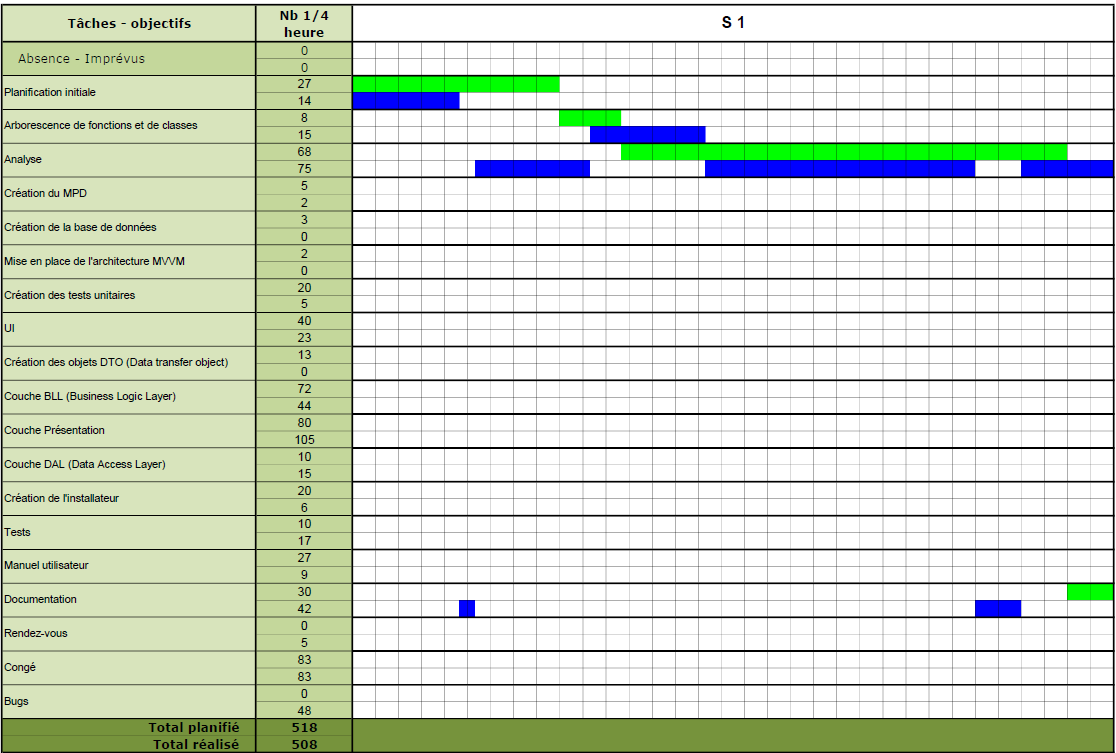


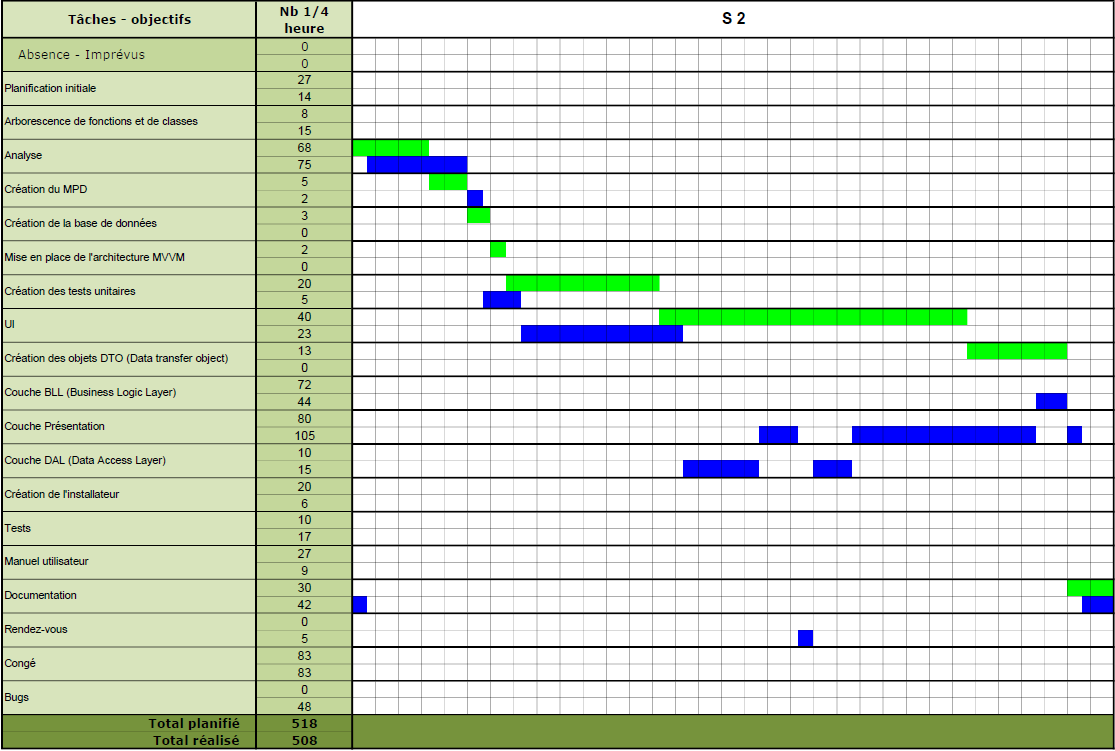


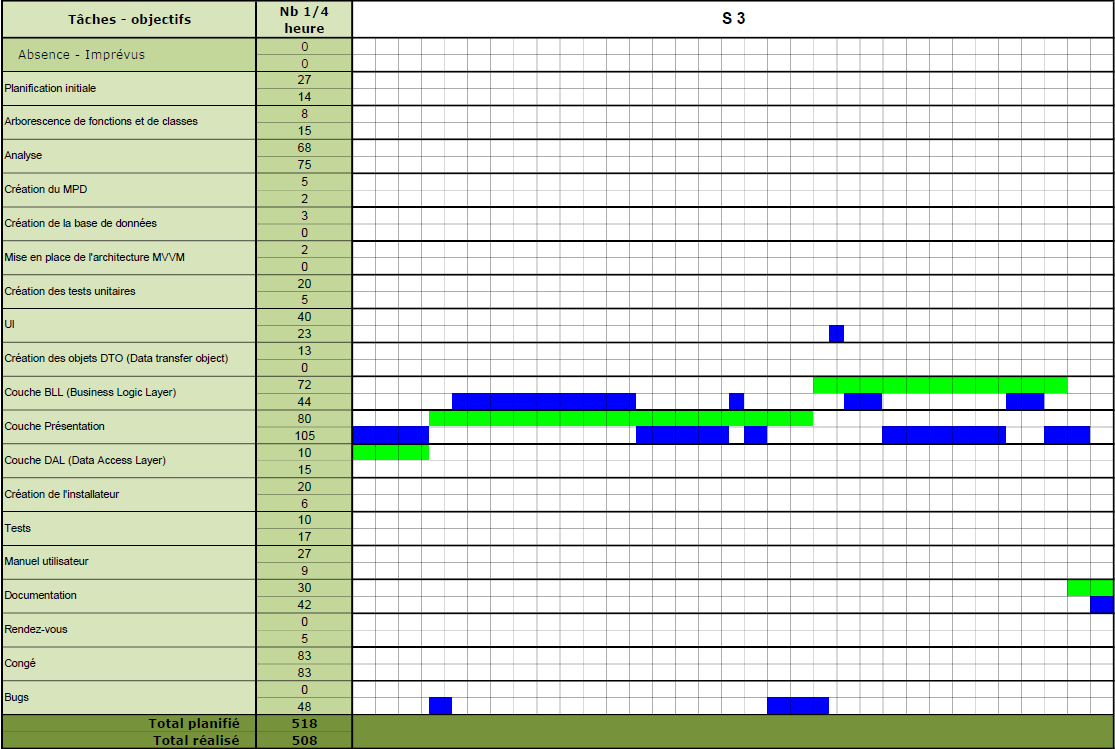
## Gantt

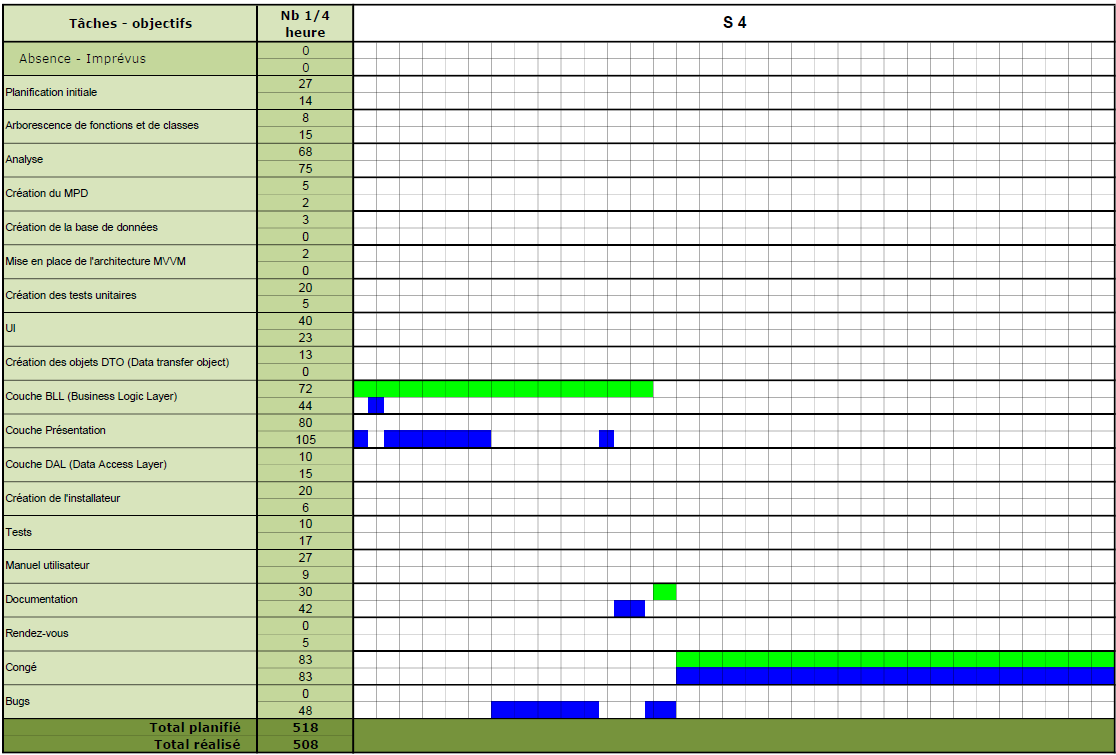
*En vert : les périodes planifiées*

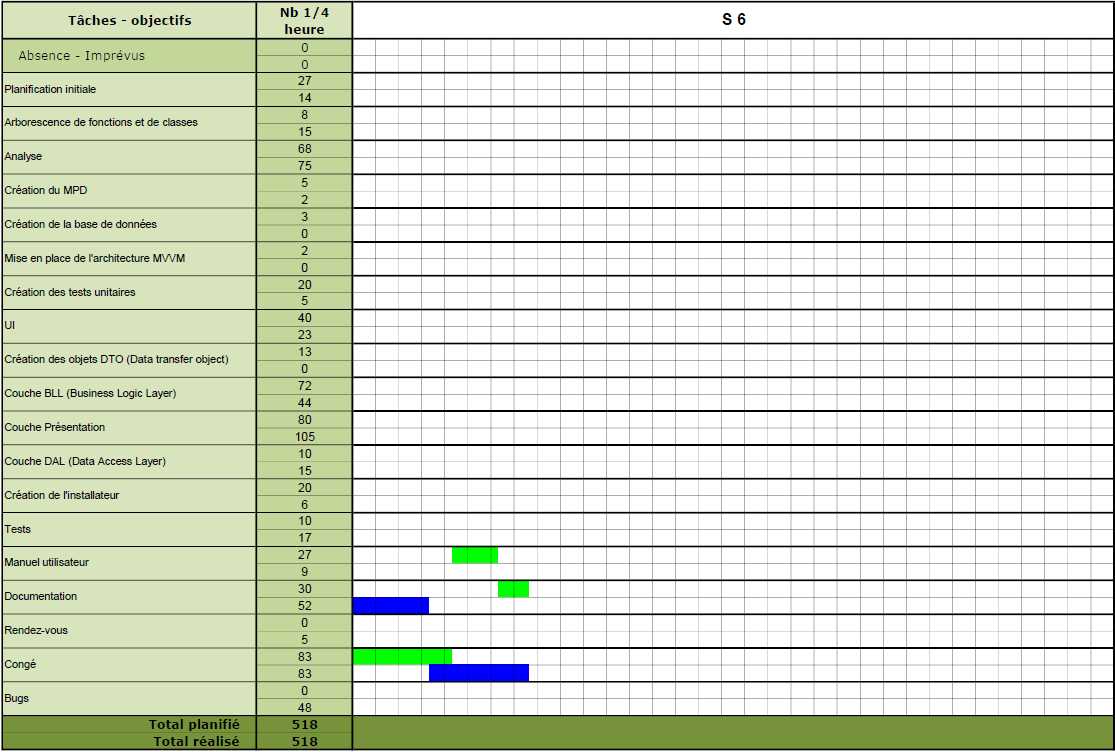
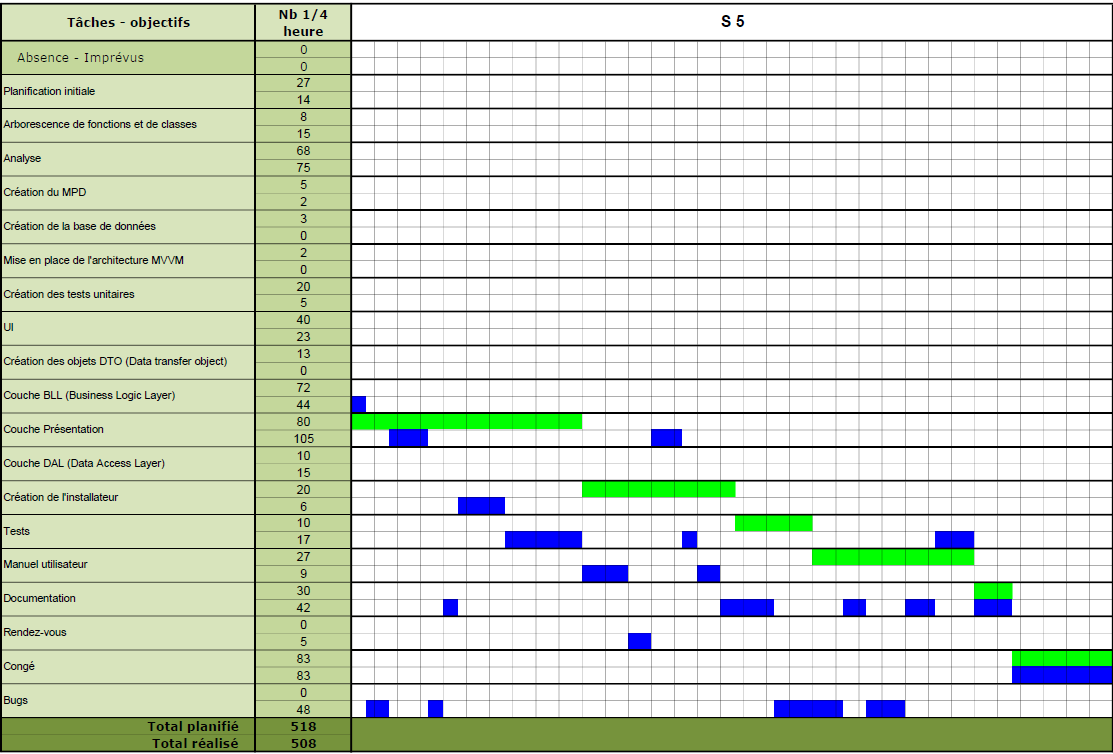
*En bleu : les périodes effectuées*











1. Selon M.Sahli [↑](#footnote-ref-1)