|  |
| --- |
| CFPT-I |
| WebradioManager |
| Travail de diplôme 2014 |
|  |
| **Simon Menetrey – T.IN E2A** |
| **16/04/2014** |

|  |
| --- |
|  |

# Résumé

## Français

## English

Contenu

[1 Résumé 1](#_Toc387323319)

[1.1 Français 1](#_Toc387323320)

[1.2 English 1](#_Toc387323321)

[2 Introduction 7](#_Toc387323322)

[2.1 Mise en situation 7](#_Toc387323323)

[2.2 Sujet 7](#_Toc387323324)

[2.3 Qu’est-ce qu’une webradio ? 7](#_Toc387323325)

[2.4 Pourquoi ce sujet ? 7](#_Toc387323326)

[2.5 Termes 8](#_Toc387323327)

[3 Cahier des charges 9](#_Toc387323328)

[4 Analyse fonctionnelle 10](#_Toc387323329)

[4.1 Schéma de l’application 10](#_Toc387323330)

[4.2 Interface principale 11](#_Toc387323331)

[4.2.1 Fenêtre d’administration 11](#_Toc387323332)

[4.2.2 Menu principale 11](#_Toc387323333)

[4.3 Gestion de plusieurs webradio 12](#_Toc387323334)

[4.3.1 Création d’une webradio 13](#_Toc387323335)

[4.3.2 Sélection d’une webradio 13](#_Toc387323336)

[4.3.3 Fermeture d’une webradio 13](#_Toc387323337)

[4.4 Serveur de diffusion interne et externe 14](#_Toc387323338)

[4.5 Onglet « Status » 14](#_Toc387323339)

[4.6 Gestion des musiques/pubs 15](#_Toc387323340)

[4.6.1 Affichage 15](#_Toc387323341)

[4.6.2 Importer et indexer 15](#_Toc387323342)

[4.6.3 Ajout à une playlist 16](#_Toc387323343)

[4.6.4 Supprimer 16](#_Toc387323344)

[4.7 Gestion des listes de lecture 17](#_Toc387323345)

[4.7.1 Listes de lecture musicales 17](#_Toc387323346)

[4.7.2 Listes de lecture publicitaires 17](#_Toc387323347)

[4.7.3 Création 17](#_Toc387323348)

[4.7.4 Génération automatique 17](#_Toc387323349)

[4.7.5 Affichage du contenu d’une playlist 18](#_Toc387323350)

[4.7.6 Retirer d’une playlist 18](#_Toc387323351)

[4.7.7 Recherche 18](#_Toc387323352)

[4.8 Gestion des horaires 18](#_Toc387323353)

[4.8.1 Événement 19](#_Toc387323354)

[4.8.2 Événement périodique 19](#_Toc387323355)

[4.8.3 Remplissage manuel 19](#_Toc387323356)

[4.8.4 Remplissage automatique 20](#_Toc387323357)

[4.8.5 Modification d’un événement 20](#_Toc387323358)

[4.8.6 Suppression d’un événement 20](#_Toc387323359)

[4.9 Gestion des transcoders 21](#_Toc387323360)

[4.9.1 Création 21](#_Toc387323361)

[4.9.2 Affichage 22](#_Toc387323362)

[4.9.3 Modification 22](#_Toc387323363)

[4.9.4 Contrôles 22](#_Toc387323364)

[4.9.5 Historique de diffusion 22](#_Toc387323365)

[4.10 Gestion du serveur 22](#_Toc387323366)

[4.10.1 Contrôles 23](#_Toc387323367)

[4.10.2 Log 23](#_Toc387323368)

[4.10.3 Configuration 23](#_Toc387323369)

[4.10.4 Interface web 23](#_Toc387323370)

[4.10.5 Administration web 24](#_Toc387323371)

[5 Analyse organique 25](#_Toc387323372)

[5.1 Environnement 25](#_Toc387323373)

[5.2 Diagramme de classes 25](#_Toc387323374)

[5.2.1 Diagramme 25](#_Toc387323375)

[5.2.2 Observateurs/Sujet 25](#_Toc387323376)

[5.2.3 AudioType et StreamType 26](#_Toc387323377)

[5.2.4 Classes abstraites 26](#_Toc387323378)

[5.2.5 Stockage des webradios 26](#_Toc387323379)

[5.3 Base de données 26](#_Toc387323380)

[5.3.1 SQLite 26](#_Toc387323381)

[5.3.2 Utilisation 26](#_Toc387323382)

[5.3.3 Suppression en cascade 27](#_Toc387323383)

[5.3.4 Schéma 27](#_Toc387323384)

[5.3.5 Twebradio 28](#_Toc387323385)

[5.3.6 Tsever 28](#_Toc387323386)

[5.3.7 Tcalendar 28](#_Toc387323387)

[5.3.8 Tcalendarevent 28](#_Toc387323388)

[5.3.9 Tplaylist 29](#_Toc387323389)

[5.3.10 Taudiotype 29](#_Toc387323390)

[5.3.11 Tmusic 29](#_Toc387323391)

[5.3.12 Tgender 29](#_Toc387323392)

[5.3.13 Tplaylist\_has\_music 29](#_Toc387323393)

[5.3.14 Thistory 30](#_Toc387323394)

[5.3.15 Ttranscoder 30](#_Toc387323395)

[5.4 Schéma de diffusion 31](#_Toc387323396)

[5.4.1 Principe de base 31](#_Toc387323397)

[5.4.2 Infomaniak 31](#_Toc387323398)

[5.5 ShoutCast 32](#_Toc387323399)

[5.5.1 Présentation 32](#_Toc387323400)

[5.5.2 Pourquoi cet outil ? 32](#_Toc387323401)

[5.5.3 Serveur 33](#_Toc387323402)

[5.5.4 Transcoder 33](#_Toc387323403)

[5.5.5 Schéma de fonctionnement résumé 33](#_Toc387323404)

[5.6 Structures des dossiers/fichiers 34](#_Toc387323405)

[5.6.1 Schéma 34](#_Toc387323406)

[5.6.2 Exécutables Shoutcast 35](#_Toc387323407)

[5.7 Initialisation de l’application 36](#_Toc387323408)

[5.8 Gestion des processus 37](#_Toc387323409)

[5.9 Webradio 37](#_Toc387323410)

[5.9.1 Classes associée 37](#_Toc387323411)

[5.9.2 Affichage des webradios disponibles 38](#_Toc387323412)

[5.9.3 Création 38](#_Toc387323413)

[5.9.4 Chargement 39](#_Toc387323414)

[5.9.5 Duplication 40](#_Toc387323415)

[5.9.6 Suppression 40](#_Toc387323416)

[5.9.7 Génération des configurations 40](#_Toc387323417)

[5.10 Bibliothèque 41](#_Toc387323418)

[5.10.1 Classes utilisées 41](#_Toc387323419)

[5.10.2 MP3 41](#_Toc387323420)

[5.10.3 Importation 41](#_Toc387323421)

[5.10.4 Tags ID3 42](#_Toc387323422)

[5.10.5 Analyse des tags 43](#_Toc387323423)

[5.10.6 Indexation 43](#_Toc387323424)

[5.10.7 Suppression 43](#_Toc387323425)

[5.10.8 Vérification des données 44](#_Toc387323426)

[5.10.9 Recherche 44](#_Toc387323427)

[5.11 Listes de lecture 46](#_Toc387323428)

[5.11.1 Classes utilisées 46](#_Toc387323429)

[5.11.2 Génération de configuration 46](#_Toc387323430)

[5.11.3 Création d’une playlist 47](#_Toc387323431)

[5.11.4 Génération automatique 47](#_Toc387323432)

[5.11.5 Ajout à une playlist 48](#_Toc387323433)

[5.11.6 Retirer d’une playlist 49](#_Toc387323434)

[5.11.7 Affichage du contenu 49](#_Toc387323435)

[5.11.8 Suppression d’une playlist 50](#_Toc387323436)

[5.12 Grille horaire 51](#_Toc387323437)

[5.12.1 Outil utilisé 51](#_Toc387323438)

[5.12.2 Gestion du calendrier par ShoutCAST 51](#_Toc387323439)

[5.12.3 Classes utilisées 52](#_Toc387323440)

[5.12.4 Génération de configuration 53](#_Toc387323441)

[5.12.5 Affichage du calendrier 53](#_Toc387323442)

[5.12.6 Sélection depuis le calendrier 55](#_Toc387323443)

[5.12.7 Création d’un événement 55](#_Toc387323444)

[5.12.8 Modification d’un événement 56](#_Toc387323445)

[5.12.9 Suppression d’un événement 57](#_Toc387323446)

[5.12.10 Génération automatique 57](#_Toc387323447)

[5.13 Transcoder 57](#_Toc387323448)

[5.13.1 Classes utilisées 57](#_Toc387323449)

[5.13.2 Définition des bitrates, taux d’échantillonnage et type d’encoder 57](#_Toc387323450)

[5.13.3 Fichier de configuration et log 57](#_Toc387323451)

[5.13.4 Licence MP3 58](#_Toc387323452)

[5.13.5 Création d’un transcoder 58](#_Toc387323453)

[5.13.6 Affichage d’un transcoder 58](#_Toc387323454)

[5.13.7 Modification d’un transcoder 58](#_Toc387323455)

[5.13.8 Suppression d’un transcoder 58](#_Toc387323456)

[5.13.9 Exécution et fermeture 58](#_Toc387323457)

[5.13.10 Gestion des processus 59](#_Toc387323458)

[5.13.11 Statut et logs 59](#_Toc387323459)

[5.13.12 Administration web (weblet) 59](#_Toc387323460)

[5.13.13 Historique 59](#_Toc387323461)

[5.14 Serveur de diffusion interne 60](#_Toc387323462)

[5.14.1 Classe utilisée 60](#_Toc387323463)

[5.14.2 Outil utilisé 60](#_Toc387323464)

[5.14.3 Configuration 61](#_Toc387323465)

[5.14.4 Mise à jour de la configuration 61](#_Toc387323466)

[5.14.5 Exécution et fermeture 62](#_Toc387323467)

[5.14.6 Affichage des interfaces web 62](#_Toc387323468)

[5.14.7 Log 62](#_Toc387323469)

[6 Gestion des processus 63](#_Toc387323470)

[7 Tests 63](#_Toc387323471)

[8 Plannings 64](#_Toc387323472)

[8.1 Prévu 64](#_Toc387323473)

[8.2 Final 64](#_Toc387323474)

[9 Apports personnels 65](#_Toc387323475)

[10 Conclusion 66](#_Toc387323476)

[11 Améliorations possibles 67](#_Toc387323477)

[12 Références 67](#_Toc387323478)

[13 Annexes 67](#_Toc387323479)

# Introduction

## Mise en situation

Je m’appelle Simon Menetrey, j’ai 20 ans et je suis actuellement en dernière année de formation technicien ES en informatique. J’ai effectué un CFC d’informaticien en 4 ans avant de commencer ma formation actuelle.

Cette documentation concerne mon travail de diplôme réalisé pour ma dernière année en tant que technicien ES en informatique. Ce travail à pour sujet la création d’un gestionnaire de webradio.

Mon professeur de diplôme, Monsieur Garcia, m’a mis en contact avec une entreprise (KTFM) qui recherche un programme de gestion pour leur webradio. En effet, actuellement, c’est un tiers qui s’occupe de la diffusion de leur webradio via des émissions préalablement enregistrées dans leurs studios. Dans l’état actuel, l’entreprise n’a pas un contrôle direct sur la diffusion de son contenu et elle désirerait pouvoir gérer elle-même l’intégralité de leur webradio. Ainsi, elle supprimera un intermédiaire et aura pleinement contrôle de la diffusion.   
En plus de cela, KTFM a aussi besoin de pouvoir avoir une traçabilité des morceaux qu’elle diffuse avec des informations précises afin de pouvoir faciliter le paiement des droits d’auteur à la Suisa[[1]](#footnote-1).  
J’ai donc réalisé le cahier des charges avec KTFM afin de répondre au mieux à leurs besoins.



## Sujet

Gestionnaire de webradio :

* Permettre de diffuser directement depuis le logiciel
* Gérer les morceaux à diffuser/listes de lecture
* Gérer les plages horaires
* Historique de diffusion

## Qu’est-ce qu’une webradio ?

Une webradio est une radio diffusée sur internet via la technologie de lecture en continu. Cette technologie fournit ce que l’on appelle un « flux » que les auditeurs écoutent via leur lecteur multimédia préféré ou via un site web.

## Pourquoi ce sujet ?

Je suis passionné de musique. J’ai aussi toujours recherché un outil simple et gratuit pour gérer une webradio et sa diffusion. J’ai donc imaginé une application, tout-en-un, remplissant ce besoin. J’ai donc l’espoir que mon projet me sera autant utile à moi qu’à l’entreprise KTFM ainsi que de potentielles futures entreprises intéressées.

## Termes

Voici une liste de termes qui seront utilisés dans cette documentation :

* Playlist : Liste de lecture
* Stream : Flux
* Log : Journal des événements

# Cahier des charges

Ce projet a pour but la création d’un gestionnaire de webradio (de type shoutCAST) complet. Les principales fonctionnalités sont les suivantes :

* Possibilité de gérer plusieurs webradio indépendamment
* Gestion des playlist, horaires et pubs
  + Génération automatique de playlist
  + Génération au format xml
* Gestion des serveurs de diffusions distants et transcoder interne.
  + Diffusion de la webradio
* Indexation des fichiers musicaux (tags, chemin sur le disque dur)
* Historique des morceaux joués
  + Génération d’un compte-rendu afin de facilité la gestion des droits d’auteurs
* Serveurs de diffusion interne (local)

Options :

* Si serveur de diffusion interne activé : Serveur WEB interne contenant un mini-site
* Modifier les informations des musiques indéxées.

Plus de détails dans le résumé et le *mindmap*.

Analyses

# Analyse fonctionnelle

## Schéma de l’application



Figure 1 - Schéma application

Lors du lancement de l’application, la fenêtre de sélection de webradio se lance. L’utilisateur choisit la webradio qu’il veut gérer, puis une fenêtre d’administration s’ouvre avec les informations de la webradio sélectionnée. Il peut à tout moment réafficher la fenêtre de sélection et ouvrir une nouvelle fenêtre d’administration avec une autre webradio à gérer.

La diffusion de webradio est possible via des serveurs distants ou alors via le serveur interne (local) à l’application. Il y en a 1 par webradio.

## Interface principale

### Fenêtre d’administration



Figure 2 - Interface principale AdminView

Cette fenêtre est l’interface principale de l’application. Elle est associée à une webradio. Comme montré dans le schéma de l’application, il est possible d’avoir plusieurs fenêtres d’administration ouvertes simultanément, une pour chaque webradio lancée. Les différents onglets sont décrient dans les chapitre suivants.

### Menu principale

* File
  + Generate all configs = Génère la configuration de chaque élément de la webradio

TODO : complétement en fonction du menu choisi

## Gestion de plusieurs webradio



Figure 3 - Interface gestion des webradios SelectionView

La fenêtre de gestion des webradios s’affiche au démarrage de l’application pour sélectionner ou créer une webradio. Lorsque l’utilisateur clique sur « Open », la fenêtre principale s’ouvre avec les données de la webradio sélectionnée. Il peut aussi utiliser un double clique pour ouvrir une webradio.  
Le bouton « Duplicate » crée une copie exacte de la webradio sélectionnée avec un suffix « copy » à son nom. Enfin, le bouton « Delete » supprime la webradio sélectionnée.   
Le fenêtre n’est pas redimensionnable et ne peut être maximisée.

Concernant la création, la partie inférieure de la fenêtre propose la création d’une nouvelle webradio. L’utilisateur lui donne un nom puis clique sur le bouton « Create ». Attention : Les webradios doivent avoir des noms différents. Un message d’erreur notifie l’utilisateur si une webradio a déjà le même nom.

L’utilisateur peut gérer autant de webradio en même tant qu’il le souhaite, en effet, à tous moments, il peut ouvrir la fenêtre de gestion des webradios et en sélectionner une autre. Cela ouvrira une nouvelle fenêtre principale, sans pour autant fermer les autres déjà ouvertes, avec les informations de la webradio fraichement sélectionnée.

Chaque webradio possède ses propres listes de lecture, sa propre grille horaire ainsi que ses propre transcoders. Un transcoder est un outils qui permet d’envoyer un flux musical à un serveur de diffusion. Pour plus d’informations, rendez-vous au chapitre concernant la [diffusion](#_Schéma_de_diffusion).



Figure 4 - Schéma webradios

### Création d’une webradio

L’utilisateur entre le nom de la future webradio dans le champ correspondant, puis clique sur « create » pour créer la webradio et ouvrir la fenêtre d’administration liée à la nouvelle webradio fraichement créé. Le nom d’une webradio ne peut pas dépasser 255 caractères.

### Sélection d’une webradio

L’utilisateur sélectionne une webradio parmi la liste proposée, puis clique sur « open » pour ouvrir une fenêtre d’administration liée la webradio sélectionnée.

### Fermeture d’une webradio

Lors de la fermeture d’une fenêtre d’administration de webradio, l’utilisateur doit confirmer si il est sur de vouloir fermer la fenêtre en sachant que cela arrêtera le serveur de diffusion et les transcoders.

## Serveur de diffusion interne et externe

Il faut bien différencier les serveurs de diffusion qui sont externes et ceux internes. Les externes sont des serveurs hébergés par un provider[[2]](#footnote-2) (exemple : infomaniak[[3]](#footnote-3) pour KTFM). L’application va donc se servir de ces serveurs pour diffuser la webradio.  
Les serveurs internes sont des serveurs de diffusion hébergés dans l’application elle-même.

## Onglet « Status »

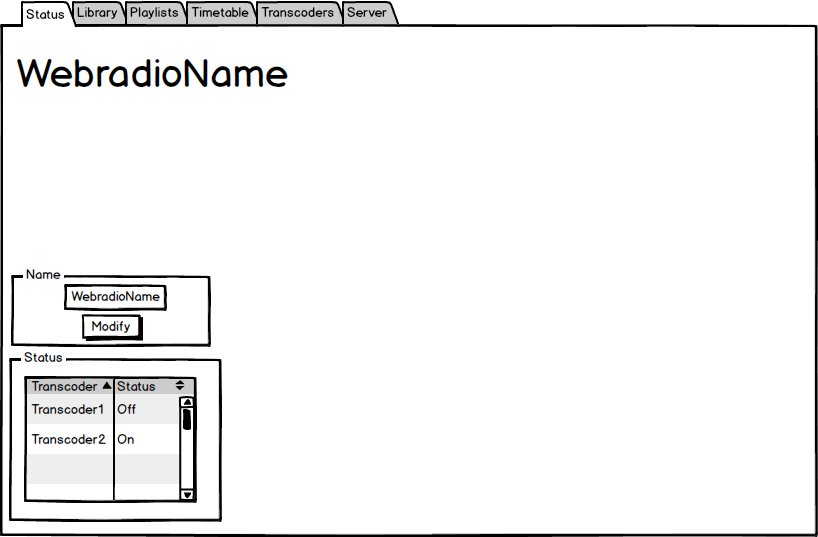


Figure 5 - Onglet "status"

L’onglet de statut est la page d’accueil de la webradio. L’utilisateur peut y modifier le nom de la webradio via le champ en dessous du nom.  
Un tableau affiche l’état des différents trancoders de la webradio actuelle.

## Gestion des musiques/pubs



Figure 6 - Bibliothèque de musiques

Les musiques sont indexées pour former une bibliothèque commune dans l’application. Elle est commune car elle est accessible depuis n’importe quelle webradio créé. L’interface est doublée : Une partie pour les musiques et une autre pour les pubs. Elles sont identiques.

### Affichage

La partie principale de cette interface est l’affichage du contenu de la bibliothèque dans la partie inférieure. Il est affiché sous la forme d’une liste à entrées. Il est possible de sélectionner un ou plusieurs éléments.

### Importer et indexer

L’utilisateur peut importer des fichiers musicaux dans sa bibliothèque. Le bouton « Import from folder… » ouvre une boite de dialogue où l’utilisateur peut sélectionner le dossier à analyser afin d’importer les fichiers musicaux qui y sont présents. C’est la partie dite « d’indexation ». Cette indexation peut être récursive, c’est-à-dire que les sous-dossiers du dossier sélectionné vont être analysés aussi. Une boite de dialogue demande donc à l’utilisateur s’il veut effectuer une analyse récursive.   
Le bouton « Import from files… » permet d’importer un ou plusieurs fichiers sélectionnés manuellement. La sélection s’effectue via une boite de dialogue Windows standard.  
A gauche de ces 2 boutons, une barre de recherche permet d’effectuer une recherche dans la bibliothèque de l’application.  
Le bouton de droite « Delete selected » va supprimer les éléments sélectionnés dans la liste de morceaux affichés en dessous.

### Ajout à une playlist

Pour ajouter des morceaux à une playlist, l’utilisateur peut en sélectionner autant qu’il le souhaite dans la liste d’affichage puis sélectionner une playlist via le menu déroulant situé à droite du bouton « Add selected to ». Ce dernier permet donc de confirmer l’ajout à une playlist.

### Supprimer

L’utilisateur a la possibilité de supprimer une ou plusieurs musique/pub d’un seul coup grâce à la sélection multiple.

## Gestion des listes de lecture

La gestion des différentes listes de lecture se fait dans l’onglet « playlists ».



Figure 7 - Onglet "playlists"

### Listes de lecture musicales

Une liste de lecture musicale contient exclusivement des musiques. Son type est « music ».

### Listes de lecture publicitaires

Une liste de lecture publicitaire contient des pubs, des annonces ou des jingles. Son type est « ad ».

### Création

La création d’une playlist se fait via le cadre situé en haut à gauche. Il lui faut un nom et un type. La playlist fraichement créé est ensuite ajouté à l’une des 2 listes situées à gauche en fonction de son type. L’utilisateur ne peut pas créer de playlist avec le même nom par type de playlist (« Music » ou « Ad ») et par webradio.

### Génération automatique

L’utilisateur a la possibilité de générer automatiquement une playlist en définissant une durée (minimum 1 minute et maximum 500 minutes), un nom, un genre musical et un type. Si le type « Ad » est choisi, le menu déroulant « Gender » est désactivé (une pub ne peut pas avoir de genre musical).  
La liste de genre affiche les genres disponibles dans la bibliothèque de l’application. Le logiciel va prendre différentes musiques ou publicité afin de remplir le temps voulu. L’algorithme fait en sorte de s’approcher le plus possible de la durée demandée mais plus celle-ci est longue, plus cela sera possible d’être précis.

### Affichage du contenu d’une playlist

Quand une playlist est sélectionnée (musique ou pub), son contenu est affiché dans la partie centrale de la vue. Les informations des morceaux sont affichées de même manière que pour l’onglet « [Library](#_Gestion_des_musiques/pubs)». Au-dessus de cette partie, des informations concernant la playlist sont affichées telle que le temps total de lecture, le nombre de morceaux présents etc.

### Retirer d’une playlist

L’utilisateur peut sélectionner un ou plusieurs morceaux dans la partie affichant le contenant de la playlist afin de les retirer de cette dernière. Une fois la sélection faite, le bouton « Remove selected » supprime le/les morceau(x) et le nouveau contenu de la playlist est affiché.

### Recherche

L’utilisateur a la possibilité de recherche parmi les morceaux d’une playlist sélectionnée via le champ de recherche au-dessus de l’affichage du contenu de la playlist.

## Gestion des horaires

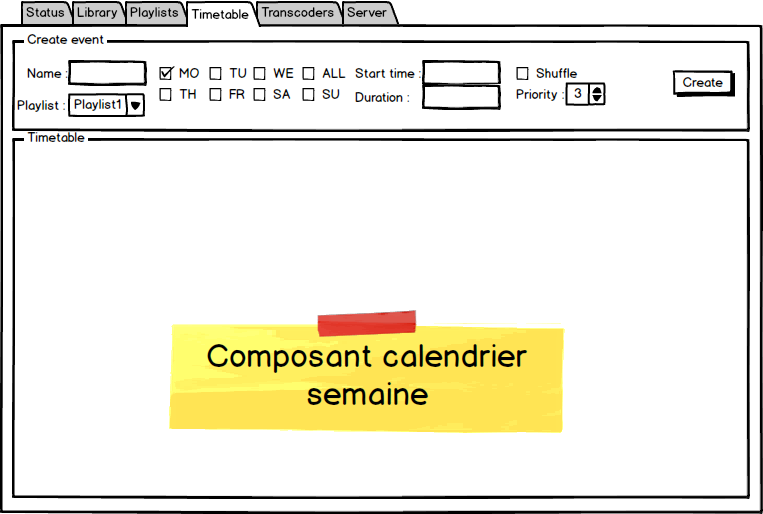


Figure 8 - Onglet "timetable"



Figure 9 - Calendrier

### Événement

Un événement est un élément du calendrier. Il est composé des informations suivantes :

* Un nom
* Une playlist (nom de la playlist qui sera jouée à l’événement)
* Un ou plusieurs jours où la playlist sera jouée
* Une heure de début (identique pour chaque jour sélectionné)
* Une durée de lecture (la playlist est jouée en boucle pendant le temps défini)
* Une option pour jouer la playlist en mode aléatoire ou non
* Une priorité (dans le cas où plusieurs événements se superposent, celui avec la plus grande priorité sera joué). De 0 à 100.

### Événement périodique

Un événement périodique se produit à intervalles réguliers.

TODO

### Remplissage manuel

L’utilisateur peut utiliser le formulaire de création d’événement en le remplissant à la main ou alors il a la possibilité de sélectionner sur le calendrier, la plage horaire où il désire crée un événement.



Figure 10 - Sélection multiple calendrier

La partie bleu foncée est la sélection faite par l’utilisateur. Dans ce cas, l’heure de début sera 17h30 et sa durée sera 2h car la sélection finie à 19h30. Ces informations sont automatiquement affichées dans le formulaire de création. Evidemment, le reste des informations demandées sera à remplir à la main par l’utilisateur.

L’utilisateur peut aussi remplir manuellement les informations concernant le début et la durée d’un événement. La case à cocher « All » permet de cocher tous les jours de la semaine ou au contraire de les décocher. La case « shuffle » permet de choisir si l’événement lira la liste de lecture de façon aléatoire. « Priority » défini le niveau de priorité de l’événement par rapport à un autre qui serai prévu au même moment.

### Remplissage automatique

TODO

### Modification d’un événement

L’utilisateur peut modifier le début, la durée et les jours d’un événement. Pour se faire, il peut déplacer les « zones » (éléments) d’un événement pour le placer ou bon lui semble. Il existe malgré cela des règles :

* Il ne peut pas y avoir plus de un élément d’un événement dans le même jour.
* Tous les éléments commencent et finissent à la même heure. Si un élément est changé dans sa durée, cela sera appliqué à tous les autres éléments de cet événement.

L’utilisateur doit redémarrer ses transcoders pour que la nouvelle configuration soit prise en compte.

### Suppression d’un événement

Pour supprimer un événement, l’utilisateur doit faire clic droit sur l’événement en question. Une boite de dialogue lui demande s’il est sûr de vouloir supprimer cet événement. La suppression d’un élément de l’événement entraine la suppression complète de ce dernier.

## Gestion des transcoders

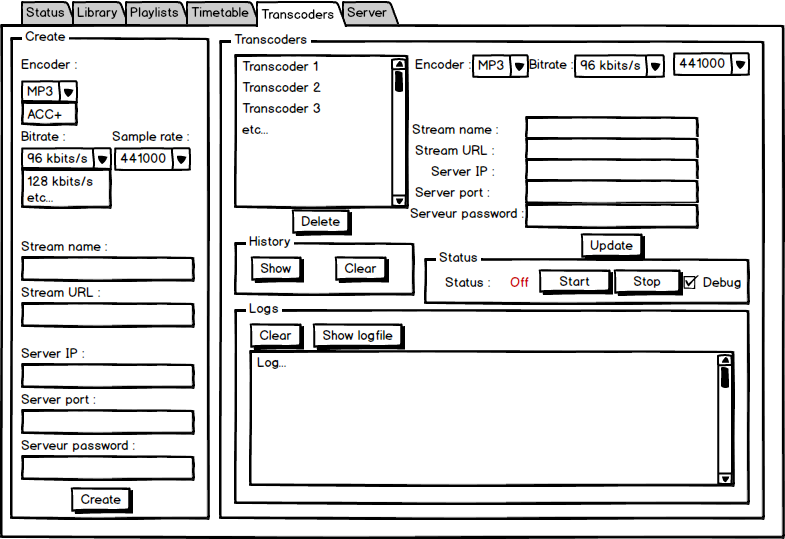


Figure 11 - Onglet "Transcoders"

Cet onglet permet de gérer les différents transcoders d’une webradio. Ces derniers servent à diffuser le flux audio vers un serveur de diffusion. Ils utilisent les playlists et le calendrier configuré par l’utilisateur.

### Création

La partie de gauche offre la possibilité de créer un nouveau transcoder avec ses différents réglages. Pour des mesures de simplicités, le minimum requis est demandé à l’utilisateur. Les informations suivantes sont demandées :

* Le type de flux (mp3 ou ACC+)
* Le bitrate[[4]](#footnote-4) du flux
* Le sample rate[[5]](#footnote-5)
* Le nom du flux (qui sert de nom pour le transcoder
* L’URL du flux (par exemple : site web de la webradio) : pas obligatoire
* Adresse IP du serveur
* Port du serveur
* Mot de passe du serveur

### Affichage

Dans la partie droite, la liste des transcoders de la webradio est affichée. L’utilisateur peut cliquer et sélectionner un des transcoder afin de la supprimer à l’aide du bouton « Delete ». Les informations du transcoder sélectionné sont affichées à droite de la liste, dans les champs prévus à cet effet. Le statut (on ou off) est affiché en dessous de ces dernières. Tout en bas, le log du transcoder est affiché et peut être effacé à l’aide du bouton « Clear ».

### Modification

Les informations affichées dans les champs à droite peuvent être modifiées puis enregistrées (bouton « Update ») afin de mettre à jour les paramètres du transcoder sélectionné.

### Contrôles

La partie « status » permet de contrôler le transcoder sélectionné. La case à cocher « Debug » permet de lancer le serveur avec son log dans une fenêtre console.

Si une erreur est détectée lors du lancement du transcoder, l’utilisateur en est informé. Il est possible que le transcoder en question soit déjà lancé dans un autre processus ou alors que le fichier exécutable ne soit plus présent.

### Historique de diffusion

L’historique de diffusion est enregistré (quelle musique est passé à quelle heure) et l’utilisateur peut l’afficher avec le bouton « show » sous la forme d’un fichier texte qui s’ouvrira dans le programme par défaut de l’ordinateur. Il est aussi possible de vider l’historique avec le bouton « clear ».

## Gestion du serveur

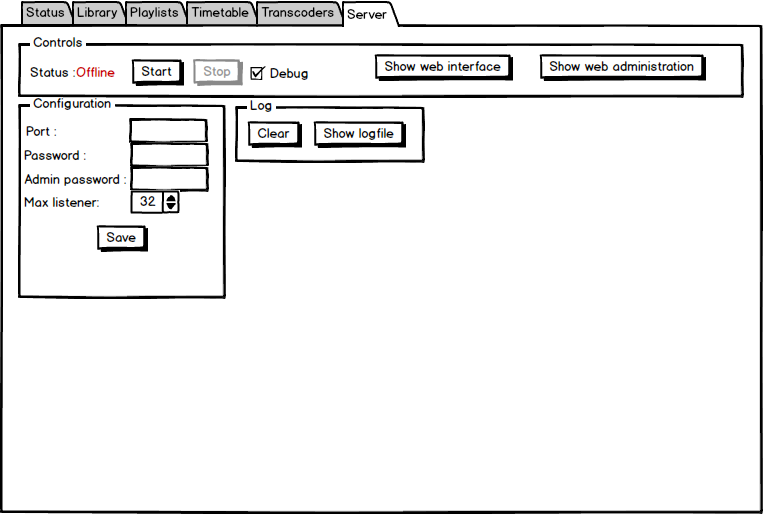


Figure 12 - Onglet "server"

L’onglet serveur propose un serveur de diffusion interne (local) pour la webradio. Bien entendu, l’utilisateur doit créer un transcoder qui s’y connectera afin de diffusion le flux audio.

### Contrôles

La partie « controls » permet de démarrer ou arrêter le serveur de la webradio via les boutons « start » et « stop ». La case à cocher « Debug » permet de lancer le serveur avec son log dans une fenêtre console. A sa droite, un bouton permet de lancer l’interface web et un autre l’administration web via le navigateur par défaut de l’utilisateur.

Si une erreur est détectée lors du lancement du serveur, l’utilisateur en est informé. Il est possible que le serveur soit déjà lancé dans un autre processus ou alors que le fichier exécutable ne soit plus présent.

### Log

Si l’utilisateur lance le serveur en mode « debug », il peut avoir un log sous forme de programme console qui apparaît dans une fenêtre minimisée. Le bouton « Show logfile » permet d’afficher le fichier de log du serveur dans le bloc note.

### Configuration

Différents éléments sont configurables pour le serveur :

* Port : le port de connexion au serveur pour le transcoder (Défaut : 8000)
* Password : Le mot de passe de connexion au serveur pour le transcoder
* Admin password : Le mot de passe pour l’accès à l’administration web du serveur
* Max listener : Le nombre maximum de connexion au serveur (connexion client/auditeur). Par défaut : 1. Maximum : 2000.

Attention : Les 2 mots de passe doivent être différents !

### Interface web

Une interface web est fournie avec le serveur ShoutCAST. Elle permet de voir différentes informations sur le serveur et le stream. Il est aussi possible de télécharger le fichier permettant d’écouter le flux directement sur un lecteur multimédia.



Figure 13 - Interface web ShoutCAST serveur

Un historique des morceaux joué par le serveur est aussi disponible.

### Administration web

Via l’interface web décrite précédemment, il est possible d’accéder à une section d’administration. Pour se faire, il faut cliquer sur le lien « Admin login » puis entrer le nom d’utilisateur « admin » et le mot de passe configuré pour le serveur.



Figure 14 - Administration web ShoutCAST serveur

Pour l’administrateur, il est possible de :

* Voir la liste des « listeners » (clients qui écoutent la webradio depuis ce serveur)
* Voir le log du serveur
* Gérer une liste d’adresses IP bannies
* Gérer une liste d’adresses IP qui disposent d’un accès réservé (si une adresse réservée désire écouter le webradio mais que le serveur est plein, un client sera éjecté du serveur pour laisser une place au client disposant d’une adresse réservée)

Tous ces services sont fournis par l’interface d’administration web.

# Analyse organique

## Environnement

* Langage : C#/.NET
* IDE : Visual Studio 2013
* OS : Windows 7 64 bits

## Diagramme de classes

### Diagramme

TODO : PAS FINI



Figure 15 - Diagramme de classes

### Observateurs/Sujet

TODO : excplicastiondu principe

TODO : expliquer le paramètre « id » qui permet d’update que les fenetre coincernée par la wberadio qui a changée

### AudioType et StreamType

TODO : expliquer que ces enum ont des valeur prise des id de la base de données

### Classes abstraites

TODO : explication du principe

TODO : explication de l’instanciation

### Stockage des webradios

TODO : explication des webraido dans le model sous form de dictionnary avec id

## Base de données

La base de données est présente pour sauvegarder les diverses informations de l’application. La plupart des paramètres sont stockées dans des fichiers texte sous forme de configuration utilisable par les différents transcoders et servers.

### SQLite



Figure 16 - Logo SQLite

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL. SQLite implémente en grande partie le standard SQL-92 et des propriétés ACID.

Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme.

Source : Wikipédia (<http://fr.wikipedia.org/wiki/SQLite>)

Pour mon projet, j’utilise le logiciel SQLite Studio pour éditer ma base de données et y entrer des données de test : <http://sqlitestudio.pl/>

### Utilisation

Une bibliothèque sous forme d’une DLL[[6]](#footnote-6) est disponible pour .NET (C#) ici : <https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki>

Une classe fourni par le site web suivant : <http://www.dreamincode.net/forums/topic/157830-using-sqlite-with-c%23/#/> permet l’interaction avec un fichier SQLite. Cette classe prendra la place de BddControls dans le diagramme de classe de l’application.

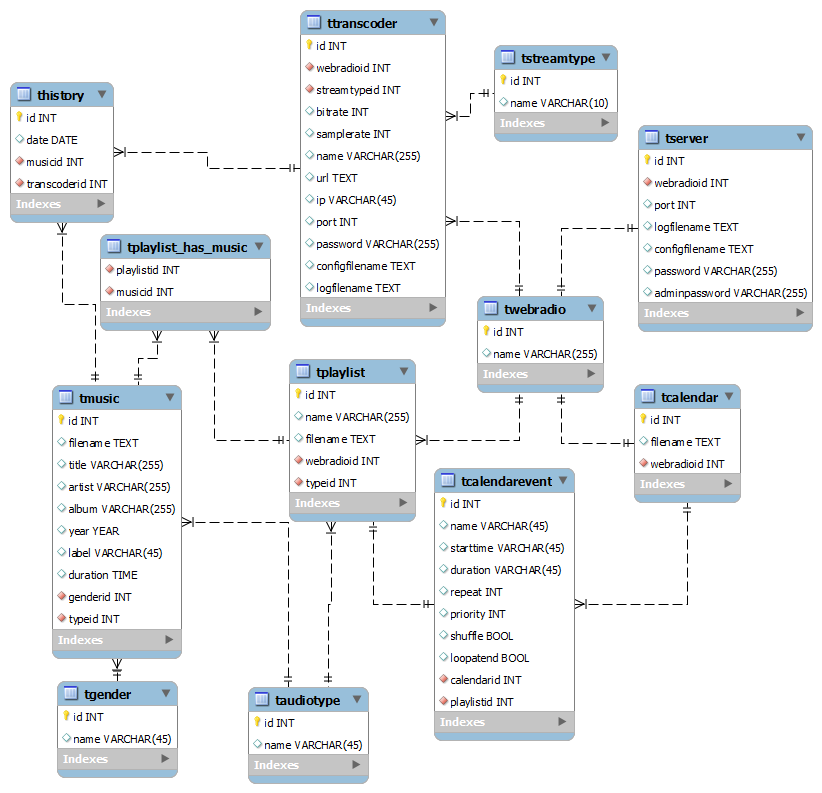
La classe Bdd utilise BddControls afin d’effectuer des requêtes sur la base de données. Bdd propose des méthodes simples comme par exemple « AddWebradio ». Ainsi la partie traitement des données se fait dans Bdd et la partie exécution dans BddControls.

La base de données sert principalement à la sauvegarde des données qui sont récupérées dans le modèle à chaque démarrage de l’application.

### Suppression en cascade

SQLite permet la suppression en cascade. C’est-à-dire, lorsqu’un enregistrement est supprimé, toutes les références à ce dernier via des clés étrangères sont supprimée automatiquement.

### Schéma



### Twebradio

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’une webradio |
| Name | Varchar(255) | Nom de la webradio |

### Tsever

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un serveur |
| Webradioid | Int | Clé étrangère (radio possédant ce serveur) |
| Port | Int | Numéro du port d’écoute du serveur0 |
| Logfilename | Text | Chemin vers le fichier de log du serveur |
| Configfilename | Text | Chemin vers le fichier de configuration du serveur |
| Password | Varchar(255) | Mot de passe de connexion au serveur (pour une source) |
| Adminpassword | Varchar(255) | Mot de passe de l’administration web du serveur |
| Maxlistener | Int | Le nombre maximal d’auditeur sur le serveur |

### Tcalendar

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un calendrier |
| Filename | Text | Chemin vers le fichier XML du calendrier |
| Webradioid | Int | Clé étrangère (radio possédant ce calendrier) |

### Tcalendarevent

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un événement |
| Name | Varchar(45) | Nom de l’événement |
| Starttime | Varchar(45) | Heure du commencement de l’événement |
| Duration | Varchar(45) | Durée de l’événement |
| Repeat | Int | Valeur de répétition de l’événement (voir chapitre [Grille horaire](#_Grille_horaire)) |
| Priority | Int | Priorité de l’événement |
| Shuffle | Bool | Défini si l’événement lit la playlist de façon aléatoire |
| Loopatend | Bool | Défini si la playlist recommence une fois que tous les morceaux sont écoutés |
| Calendarid | int | Clé étrangère (Calendrier possédant cet événement) |
| Playlistid | Int | Clé étrangère (Playlist jouée par cet événement) |

### Tplaylist

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’une playlist |
| Name | Varchar(255) | Nom de la playlist |
| Filename | Text | Chemin vers le fichier de la playlist |
| Webradioid | Int | Clé étrangère (webradio possédant cette playlist) |
| Typeid | Int | Clé étrangère (le type de la playlist) |

### Taudiotype

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un type audio |
| Name | Varchar(45) | Nom du type audio |

Cette table est lié à l’enum « AudioType ».

### Tmusic

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’une musique |
| Filename | Text | Chemin vers le fichier musical |
| Title | Varchar(255) | Titre du morceau |
| Artist | Varchar(255) | Artiste du morceau |
| Album | Varchar(255) | Album du morceau |
| Year | Year | Année du morceau |
| Label | Varchar(45) | Label du morceau |
| Duration | Time | Durée du morceau |
| Genderid | Int | Clé étrangère (genre du morceau) |
| Typeid | Int | Clé étrangère (type du morceau) |

### Tgender

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un genre musical |
| Name | Varchar(45) | Nom du genre |

### Tplaylist\_has\_music

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Playlistid | Int | Clé étrangère (Playlist concernée) |
| Musicid | Int | Clé étrangère (Musique concernée, qui fait partit de la playlist ayant l’identifiant Playlistid) |

### Thistory

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un élément d’historique |
| Date | Date | Date de l’événement dans l’historique |
| Musicid | Int | Clé étrangère (Musique jouée) |
| Transcoderid | Int | Clé étrangère (Transcoder ayant joué cette musique) |

### Ttranscoder

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| Id | Int | Identifiant unique d’un transcoder |
| Webradioid | Int | Clé étrangère (Webradio possédant ce transcoder) |
| Streamtypeid | Int | Clé étrangère (Type du transcoder MP3 ou autre) |
| Bitrate | Int | Débit binaire du flux (en bits/s) |
| Samplerate | Int | Taux d’échantillonnage du flux |
| Name | Varchar(255) | Nom du flux |
| Url | Text | URL concernant le flux (site web par exemple) |
| Ip | Varchar(45) | Adresse IP du serveur de diffusion |
| Port | Int | Port du serveur de diffusion |
| Password | Varchar(255) | Mot de passe du serveur de diffusion |
| Configfilename | Text | Chemin vers le fichier de configuration du transcoder |
| Logfilename | Text | Chemin vers le fichier de log du transcoder |

## Schéma de diffusion

### Principe de base



Figure 17 - Principe de base diffusion

Ce type de diffusion est appelé « client-serveur ». C’est le principe de base de streaming audio/vidéo. Dans cette application, la diffusion est possible sur des serveurs distants ou sur un serveur interne (local).

### Infomaniak



Figure 18 - Schéma de diffusion



Figure 19 - Schéma de diffusion 2

Ces 2 schémas montrent le fonctionnement de la diffusion des webradio via infomaniak. Le logiciel avec les transcoders sont les « sources audio ». Infomaniak demande seulement un flux audio, peut importer le logiciel utilisé pour le diffuser. C’est ensuite leurs serveurs qui s’occuperont de diffuser le flux aux différents auditeurs.

## ShoutCast



Figure 20 - Logo Schoutcast

### Présentation

SHOUTcast est le nom d'un protocole et d'un serveur de diffusion pour webradio ou pour webtv. Il a été créé par la société Nullsoft en même temps que le logiciel client Winamp pour l'écoute. Le protocole s'appuie sur deux protocoles, HTTP et ICY pour supporter les ID tag (« title streaming »).

Source : Wikipédia (<http://fr.wikipedia.org/wiki/SHOUTcast>)

SHOUTcast a été racheté récemment par la société Radionomy (<http://www.pcworld.fr/business/actualites,societe-belge-radionomy-confirme-rachat-winamp-shoutcast,545553,1.htm>). Cela a pour répercussion que les fichiers ne sont plus disponibles sur le site web de Shoutcast. Mais malgré cela, les fichiers de la version 2 sont toujours disponibles sur le forum de Winamp[[7]](#footnote-7). Ce sont donc ces derniers qui seront utilisé pour le projet.

### Pourquoi cet outil ?

J’ai choisi Shoutcast car il propose des outils en ligne de commande ainsi qu’une gestion facilitée via des fichiers XML[[8]](#footnote-8) ou texte basiques. Cela permet de pouvoir interagir plus facilement avec via des applications externes telle que la mienne.

### Serveur

Shoutcast, parmi ses outils, propose un serveur en ligne de commande qui sera utilisé dans ce projet. Il permet de diffuser un flux qu’il reçoit et qui est envoyé par, par exemple, un [transcoder](#_Transcoder). Ce flux est distribué aux clients (auditeurs dans ce cas) qui désire écouter la webradio.

Ce serveur propose aussi une interface web pour visualiser ton statut (données sur le flux actuel, musique en cours etc.) et d’administrer (il faut être authentifié en tant qu’administrateur avec le mot de passe défini dans la configuration du serveur) le serveur. Ces détails sont expliqués dans [l’analyse fonctionnelle.](#_Interface_web)

Plus de détails dans la partie consacrée au [serveur interne de diffusion](#_Serveur_de_diffusion).

### Transcoder

Tout comme le serveur, le transcoder est un outil fourni par Shoutcast en ligne de commande. Il permet de diffuser un flux sur un serveur de diffusion. Ce flux peut être en MP3 ou AAC+. Il donne aussi la possibilité de créer des playlist et de les agencées dans un calendrier XML. Les détails concernant ce système sont expliqués dans la suite de cette analyse organique.

Il gère de façon indépendante les horaires, les priorités entre les playlists et la lecture des fichiers musicaux. Le programme de ce projet va s’occuper de générer les fichiers nécessaires au transcoder en fonction des paramètres définis par l’utilisateur ainsi que d’afficher ces informations de façon visuelle (exemple : calendrier) afin de faciliter la manipulation et la configuration.

### Schéma de fonctionnement résumé



Figure 21 - Schéma shoutcast

## Structures des dossiers/fichiers

### Schéma

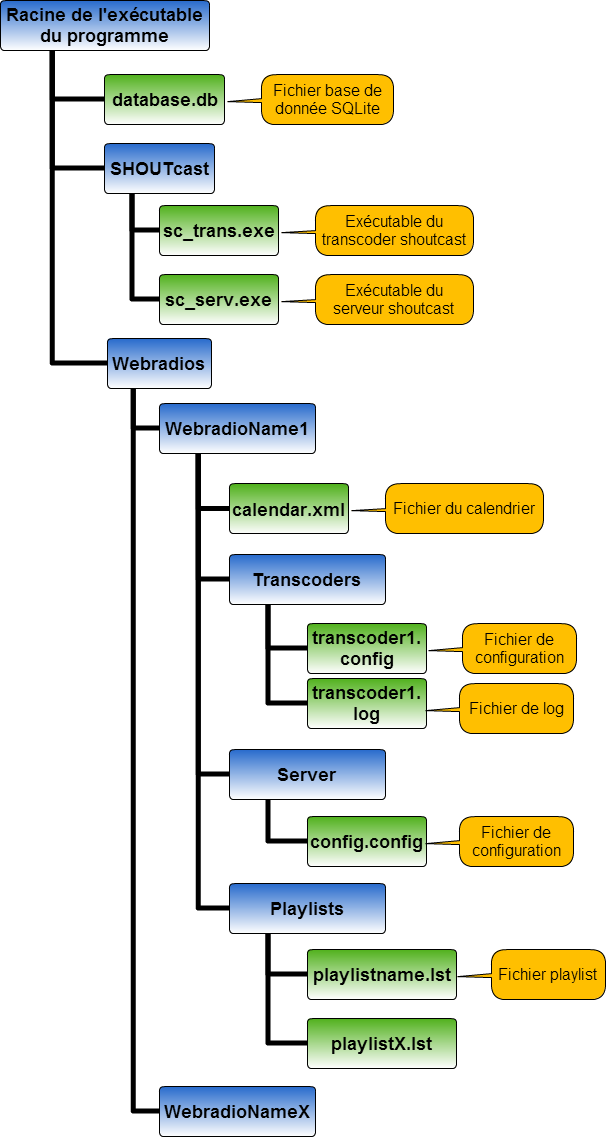


Figure 22 - Schéma structure des fichiers/dossiers

Ce schéma décrit l’organisation des fichiers dans le dossier de l’application. La base de données contient les informations pour les transcoders et les serveurs ainsi que le chemin vers les différents fichiers de ces dernières (configuration, calendrier etc.). Cela permet à l’application de savoir où trouver les fichiers lors des traitements.

### Exécutables Shoutcast

Attention : Il n’y a pas un exécutable de transcodeur par transcoder de webradio ni un exécutable de serveur par webradio. Il existe seulement un seul et unique exécutable transcoder et serveur dans le dossier « shoutcast ». Ces exécutables peuvent être lancés avec le chemin vers un fichier de configuration en paramètre. Par exemple : A chaque fois qu’un transcoder devra se lancer, une nouvelle instance de l’exécutable sc\_trans.exe présent dans le dossier « shoutcast » sera lancée dans un nouveau processus et utilisera le fichier de configuration du transcoder en question. Cela a été décidé car si une mise à jour des exécutables doit être faite, seuls les 2 exécutables du dossier « shoutcast » seront mise à jour.

Voici ce principe illustré :

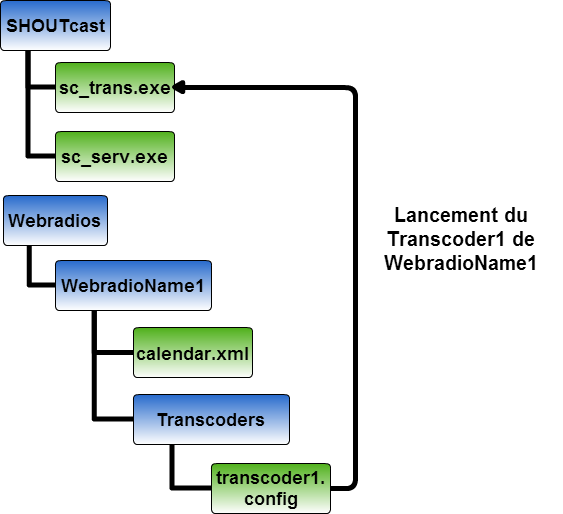


Figure 23 - Exemple lancement transcoder

## Initialisation de l’application

La première fenêtre à se lancer est la SelectionView. C’est elle qui va appeler les différentes méthodes du modèle (via son contrôleur) servant à l’initialisation.  
C’est le classe Bdd qui s’occupe du traitement des données pour les passer ensuite au modèle pour que ce dernier remplisse ses champs « webradios » et « library ». Le diagramme de séquence suivant explique le déroulement de façon simplifiée.



Figure 24 - Diagramme de séquence initialisation application

A la fin de se diagramme, SelectionView est affichée à l’utilisateur avec les informations recueillies avec UpdateView(). Cette méthode met à jour la vue avec les informations disponibles dans le modèle la concernant.

Le modèle vérifie avant tout, que les dossiers de « base » (webradios, shoutcast) sont présents. Si ce n’est pas le cas, il les crée. Cette vérification est effectuée dans la méthode « LoadWebradios » du modèle.

Le modèle est donc rempli au démarrage de l’application. Toutes les informations contenues dans la base de données sont récupérées, traitées et ajoutées au modèle. Cela permet d’éviter un nombre de requêtes inutiles vers la base de données et de travailler avec les informations stockées en mémoire.

Les 2 méthodes « LoadWebradios() » et « LoadLibrary() » de la classe Bdd s’occupe du traitement des informations. La première charge toutes les informations de façon hiérarchique (une webradio a un calendrier, qui lui dispose d’événement etc…) pour chaque webradio de la base de données. La 2ème charge les musiques présentes dans la bibliothèque ainsi que les playlist qui leur sont associées.

## Gestion des processus

TODO : explication classe Process C# et la gestion ( un processus par classes en aillant besoin, exemple : server)

TODO : tout s’eteins a la fermture

TODO : POUR CHAQUE CHAPITRE, EXPLIQUER LA CLASSE ASSOCIEE

## Webradio

### Classes associée



Figure 25 - Classe "Webradio"

TODO : mettre ajour l’image en fin de rappoort + expliquer le ToString overrid

### Affichage des webradios disponibles

Les webradios disponibles sont affiché dans la ListBox centrale de la fenêtre SelectionView. Pour la remplir, cella est effectué dans la méthode « UpdateView() » qui va récupérer les webradios du model et remplir la liste « d’items » avec les objets de type Webradio obtenus. C’est la méthode « ToString » de la classe Webradio qui va être appelée pour afficher le nom de la webradio dans la liste.

### Création

Une webradio ne peut pas avoir un nom qui dépasse 255 caractères. Pour se faire, la limitation est directement configuré dans la propriété « MaxLength » du TextBox permettant à l’utilisateur de nommer sa webradio.



Figure 26 - Schéma création webradio

Comme présenté dans le schéma ci-dessus, la création se divise en 5 étapes, ces dernières s’effectuent dans le modèle et la méthode « CreateWebradio() » :

* 1 : Instanciation d’un nouvelle objet de type Webradio avec le nom donné en paramètre à la méthode « CreateWebradio() ».  
  Instanciation des objets nécessaires à la webradio (WebradioServer, WebradioCalendar et les différentes listes d’objet telle que la propriété Playlists). Seul la propriété « id » de l’objet webradio n’est pas rempli car il s’agit de son identifiant dans la base de données, il sera donc rempli par la suite.
* 2 : L’objet créé est passé à la classe Bdd qui s’occupe de l’ajout de toutes ces informations dans la base de données via sa méthode « AddWebradio() » qui retourne l’identifiant qui a été attribué à cette nouvelle webradio par la base de données. Cet id est récupéré dans la méthode « CreateWebradio() » précédente et ajouté à l’objet webradio.
* 3 : L’objet webradio final est ajouté à la liste d’objets de type « Webradio » du modèle.
* 4 : Création des différents sous-dossiers pour stocker les fichiers de la nouvelle webradio.
* 5 : La méthode « GenerateConfigFiles() » est appelée afin de créer les fichiers de configuration nécessaires à la webradio. Plus d’information sur cette méthode dans [ce chapitre](#_Génération_des_configurations).

### Chargement

Une webradio est chargée lorsqu’elle est sélectionnée via SelectionView. Une fenêtre de type AdminView est ensuite ouverte avec les informations de la webradio sélectionnée. Ces informations viennent du modèle. Voici le diagramme de séquence pour la création d’une nouvelle instance d’une AdminView :



Figure 27 - Diagramme de séquence instanciation AdminView

Après cette initialisation, le nouvel AdminController est ajouté à la liste d’observateurs du modèle.

A la création de la fenêtre, cette dernière appel la méthode « CheckFolders » de son contrôleur. Cette méthode va faire de même sur le modèle. En fin de compte, cette méthode vérifie la présence de tous les dossiers nécessaire à la webradio lancé avec AdminView (server,playlists et transcoders). Si un d’eux n’est pas présent, il est créé.

Comme montré dans la diagramme, la méthode « UpdateView() » de AdminView est appelé afin de charger les informations dans ses différents composants. La vue contient l’id de la webradio qui lui est attribuée. C’est avec cet id qu’elle va aller récupérer la webradio en question dans le modèle via son contrôleur.

En ce qui concerne le remplissage des différentes ComboBox et ListBox, leurs Items ne seront pas de simples chaînes de caractères mais des objets complets. Par exemple, les ComboBox affichant les playlists disponible comme dans l’onglet « Library », sont remplis avec des objet de type Playlist. Il est important que les classes ajoutées à des composants de ces types implémente une méthode « ToString() » car c’est celle qui est appelée par le composant lorsqu’il affiche, sous forme de chaîne de caractère, les éléments de sa liste. Cette méthode « override » celle héritée par la classe parente « Object ». De cette façon, il est facile de personnalisée les informations retournée par classe quand un composant utilise sa méthode ToString.

Le fait d’utiliser directement des objets dans les ListBox ou ComboBox permet de garder un pointeur sur les objets présent dans le modèle. Cela permet de manipuler un objet et ses modification seront répercutées partout où il est utilisé.

Le calendrier est un composant spécial et il est aussi mis à jour lors de l’UpdateView. Pour plus d’information sur son fonctionnement, rendez-vous [au chapitre le concernant](#_Grille_horaire).

### Duplication

TODO : depuis model -> dis a la bdd de dupliquer et duplique dans son model

Todo : Cloneable . Un clone() par sous object de webradio ? Modifier la méthode « AddWebradio » pour parcourir tous les elements de la webradio donnée ? et pas que nle calendrier et le server

### Suppression

La suppression d’une webradio s’effectue via son identifiant. La méthode « DeleteWebradio() » du modèle va en premier temps supprimer le webradio de la base de données, puis de sa liste (dictionnaire) de webradios. La suppression dans la liste se fait via la méthode « Remove » proposée par les listes de type Dictionnary qui prend l’identifiant de la webradio à supprimer.

### Génération des configurations

La classe Webradio dispose d’une méthode « GenerateConfigFiles() » qui appel la méthode « GenerateConfigFile() » de chacun de ses membres (Server, Playlists etc.) ayant besoin d’un fichier de configuration.

Ces méthodes suppriment le fichier de configuration existant (si il y en a un) et en crée un nouveau avec les informations contenues dans les champs de la classes en question. En fonction de la classe, le type de fichier de configuration sera différent (simple fichier texte ou fichier XML).

La génération de configuration est appelée lors de la sauvegarde ou le changement d’informations depuis l’AdminView mais encore lors de la création d’une nouvelle webradio. Plus d’informations pour la génération de configuration de chaque composant du programme dans la suite de la documentation.

## Bibliothèque

### Classes utilisées



Figure 28 - Classes bibliothèque

### MP3

A l’heure actuelle, seuls les fichiers MP3 peuvent être importés. Le raison principale est que le flux de sortie de la webradio sera de toute façon compressé donc il n’y a pas d’intérêt à importer des fichiers de meilleure qualité.

TODO : explication algo de compression

### Importation

Depuis l’onglet « library », l’utilisateur choisi si il veut importer un dossier ou des fichiers à la section « music » ou « ad ». Chaque bouton contient une valeur dans sa propriété « tag » qui est soit « Music » ou soit « Ad » car les événements « OnClick » des boutons (regroupé par type d’importation, c’est-à-dire, par dossier ou fichiers) pointe sur une même méthode et cette propriété « tag » permet de différencier si le bouton cliqué est dans la section « music » ou « ad ».  
 Dans le cas d’un dossier, le traitement va rechercher tous les fichiers MP3 contenus dans le dossier sélectionné. Si l’utilisateur a désiré d’importer de façon récursive, les sous-dossiers du dossier sélectionné seront aussi analysés.   
La méthode statique « GetFiles » de la classe Directory permet de récupérer un tableau de string contenant les « filename » (chemin absolu vers les fichiers) des fichiers correspondant au pattern donné en paramètre dans le dossier spécifié. Ce pattern se présente sous la forme : « \*.mp3 » pour récupérer seulement les fichiers dont l’extension est « mp3 ». Une option permet de faire cette recherche de façon récursive. Voici un exemple de code pour des fichiers mp3 et récursivement :

Directory.GetFiles(FBD.SelectedPath, "\*.mp3", SearchOption.AllDirectories);

Pour une recherche non-récursive, l’option « SearchOption » doit être changée en : SearchOption.TopDirectoryOnly.

La vue va ensuite passer le tableau de string au modèle via son contrôleur. Le modèle va effectuer le traitement (analyse de tags ID3 et ajout à la base de donnée/modèle) avec sa méthode « ImportFilesToLibrary ». Cette méthode est générique, il suffit de lui envoyer un tableau de filenames pour fonctionner. Cela permet qu’une importation par fichiers ou dossier puisse utiliser la même méthode. Voici le schéma récapitulatif :



Figure 29 - Schéma importation fichier

La partie « analyse de tag » est expliquée plus précisément [ici](#_Analyse_des_tags). Concernant l’ajout à la base de donnée, [ce chapitre](#_Indexation) décrit le procéder.

### Tags ID3

ID3 est le nom des métadonnées pouvant être insérées dans un fichier audio comme MP3. Ces métadonnées permettent d'avoir des informations sur le contenu du fichier comme le titre, le nom de l'interprète, les commentaires, ou encore la date de sortie.

Source : Wikipédia

### Analyse des tags

La récupération des tags ID3 est effectuée à l’aide de la bibliothèque « TagLib-Sharp » : <https://github.com/mono/taglib-sharp>

Le fonctionement de cette bibliothèque est simple. Pour analyser et récupérer les tags ID3 d’un fichier, elle fournit une classe File (à ne pas confondre avec la classe File fourni par .NET) qui propose une méthode statique « Create » qui prend un filename en paramètre. Cette dernière retourne donc un objet instancié de type « TagLib.File » qui contient toutes les informations sur le fichier donné.

2 types d’informations importantes se distinguent et qui seront utilisées :

* La propriété « Tag » qui contient des sous-propriétés comme « Title », « Year » etc. Elles sont les tags à proprement parlé.
* La propriété « Properties » qui contient des sous-propriétés comme « Duration », « BitRate » etc. Ce sont les informations concernant le fichier en lui-même.

TODO : la librairie lit les tag MusicBrainz

### Indexation

Indexation est la partie qui consiste à enregistrer les informations des musiques de la bibliothèque dans la base de données. Pour se faire, quelques règles sont établies :

* Il ne peut y avoir qu’une occurrence par fichier. C’est-à-dire, pas de doublon. Plusieurs musiques peuvent avoir les mêmes informations dans les tags mais c’est le nom de fichier qui fait foi.
* Chaque genre musical à son enregistrement dans la table « tgender ». Si lors de l’ajout d’une musique/pub à la base de données, son genre n’est pas encore dans cette table, il est ajouté à cette table et son identifiant est récupéré. Dans l’autre cas, si le genre est déjà présent dans la table, son identifiant est juste trouvé dans la table.

Voici ces 2 règles représentées sous forme de code :

if(this.AudioFileExist(file.Filename))

return ERROR;

int genderId = this.GetGenderId(file.Gender);

//If return error, gender doesn't exist in DB, so add it

if (genderId == ERROR)

//Get the new id

genderId = AddGender(file.Gender);

Ensuite, l’ajout s’effectue avec un nouvel enregistrement dans la table « tmusic » puis l’identifiant de ce nouvel enregistrement est retourné à la fin de la méthode.

### Suppression

Pour la suppression, le même principe que l’importation par rapport à la différenciation entre le bouton  « Delete selected » de la section « Music » et « Ad » avec la propriété « Tag ».

L’utilisateur peut supprimer plusieurs occurrences d’un seul coup, pour cela il faut parcourir la propriété « SelectedRows » du composant « DataGridView » en question. Voici la boucle :

foreach(DataGridViewRow row in ((type == AudioType.Music)?dgvMusics.SelectedRows:dgvAds.SelectedRows))

{

if (!this.Controller.DeleteAudioFile(int.Parse(row.Cells[0].Value.ToString()), row.Cells[row.Cells.Count - 1].Value.ToString()))

state = false;

}

La présence d’un test ternaire dans le partie « in » du foreach permet de sélectionner le bon composant DataGridView en fonction de la section (Music ou Ad) dans laquelle le bouton de suppression a été pressé.

Pour chaque occurrence, la musique/pub va être supprimée de la base de données ainsi que du modèle et des playlist la concernant. Pour se faire, toutes les playlist de toutes les webradios du programme vont être bouclées et dans la liste de filename de chacune, les occurrences du filename de la musique/pub supprimée seront enlevées.  
Ensuite, la suppression s’effectue dans la bibliothèque du modèle puis dans la base de données.

### Vérification des données

TODO : ajouter vfonctionnalité : vérifié l’intégritée des données (si les fichier référencés dans la bdd existe toujours) poour chaque section via un bouton « check files »

TODO : ajout a analyse fonctionnelle

### Recherche

La recherche consiste à afficher seulement les lignes aillant au moins une correspondance (n’importe quel champ de la ligne) avec la chaîne de recherche entrée par l’utilisateur. Cette chaîne est en premier temps mise en minuscule afin de ne pas prendre la casse en compte.

Une boucle parcourt les lignes du DataGridView et pour chacune, une autre boucle parcourt les cellules. Ensuite, pour chaque cellule, la valeur de cette dernière est prise, un ToString suivit d’un ToLower y est appliqué (pas de casse) et enfin la méthode « Contains » va retourner un booléen si elle trouve une occurrence dans la valeur de la cellule. Si c’est le cas, cette ligne est valide et pourra être affichée. La propriété « Visible » de la ligne est donc modifiée à true mais dans le cas contraire elle sera false. Et ainsi de suite pour chaque ligne.

searchString = (sender as TextBox).Text.ToLower();

foreach(DataGridViewRow row in ((type == AudioType.Music)?dgvMusics.Rows:dgvAds.Rows))

{

foreach(DataGridViewCell cell in row.Cells)

{

if (cell.Value.ToString().ToLower().Contains(searchString))

{

valid = true;

break;

}

}

row.Visible = (valid) ? true : false;

valid = false;

}

## Listes de lecture

Les listes de lecture sont utilisées par le calendrier, qui lui, est utilisé par un transcoder.

### Classes utilisées



Figure 30 - Classes Playlist

### Génération de configuration

Une liste de lecture comprend un fichier de configuration dont la forme est une simple liste des fichiers à utiliser. Elle est stockée dans un fichier texte avec l’extension « .lst ». Voici un exemple de fichier :

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz-Those\_Were\_The\_Days\_Original\_Instrumental\_Mix-ToffMusic.mp3

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz-Everlasting\_Outro-ToffMusic.mp3

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz\_and\_Toneshifterz-Together\_Extended\_Version-ToffMusic.mp3

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz-Falling\_Extended\_Version-ToffMusic.mp3

C:\Users\MENETREYS\_INFO\Music\test\Wasted Penguinz - Wistfulness\Wasted\_Penguinz-Endless\_Extended\_Version-ToffMusic.mp3

Chaque ligne du fichier correspond au « filename » (chemin de fichier) vers la musique/pub de la playlist. C’est grâce à cela que le transcoder pourra aller chercher les musiques/pubs pour les jouer et les diffusées.

Concernant la génération, c’est le champ « AudioFileList » de la classe « Playlist » qui contient les filenames des musiques/pubs de la playlist, qui est parcouru afin de générer le fichier.

### Création d’une playlist

Pour la création d’une nouvelle playlist, une règle importante a été établie :

* Le nom d’une playlist doit être unique (au sein de la même webradio et avec le même type (Music ou Ad)).
* Par exemple : Il peut y avoir 2 playlist qui se nomment « Test » au sein de la même webradio si chacune d’entre elles a un type différent.
* Le nom ne doit pas contenir de caractères Windows invalides. C’est-à-dire, des caractères interdits pour des noms de fichiers Windows.

Concernant la création à proprement parlé, elle se décompose en plusieurs étapes qui se déroulent dans la méthode « CreatePlaylist » du modèle :

1. Création du futur filename du fichier de configuration de la playlist :  
   DEFAULT\_WEBRADIOS\_FOLDER + webradioName + "/" + DEFAULT\_PLAYLISTS\_FOLDER + name + ".lst";
2. Instanciation d’une classes PlaylistMusic ou PlaylistAd en fonction du type de playlist créée.
3. Insertion dans la base de données. La méthode d’ajout à la BDD retourne l’identifiant dans la BDD de la nouvelle playlist. Si cet id correspond « ERROR » (constante définie), une erreur est survenue lors de l’ajout. Dans ce cas, la méthode de création retourne directement « false ». Si un identifiant valide a été retourné, il est configuré à l’objet de type playlist instancié précédemment.
4. La méthode « [GenerateConfigFile](#_Génération_de_configuration)» est appelé sur l’objet Playlist afin de créer son fichier de configuration (bien que vide pour le moment).
5. L’objet est ajouté au modèle (dans la propriété « Playlists » de la webradio à laquelle la playlist est ajoutée).
6. « UpdateObservers » est appelé afin de mettre à jour toutes les fenêtres concernées.

### Génération automatique

Le principe de la génération automatique est de créer une playlist d’une durée donnée en la remplissant (aléatoirement) de musiques/pubs du genre donné. Dans le cas de pubs, le genre n’est pas pris en compte.

Il faut savoir que plus la durée demandée grande, plus il sera possible de s’en rapprocher le plus possible en utilisant les morceaux disponibles dans la bibliothèque.

Concernant l’algorithme, il consiste à parcourir la bibliothèque du modèle de façon aléatoire (avec l’objet de type Random) dans une boucle de type while. La condition de cette dernière est que la durée de la playlist doit rester plus petite que la durée demandée. Le but étant de se rapprocher le plus possible de la durée demandé sans la dépasser.  
Il est déterminé que l’algorithme essai un nombre de fois (consécutives), défini par la constante « MAX\_TRY\_GENERATE », de remplir la liste de lecture. Si ce nombre de fois est dépassé, il est considéré qu’il n’est plus possible de remplir sans dépasser la limite de durée et la boucle est donc arrêtée.   
A chaque tour de boucle, une musique/pub est « piochée » et il est testé si le temps actuel de la playlist + le temps de la musique sélectionnée dépasse la durée demandée. Si c’est le cas, le nombre d’essais s’incrémente de 1 et la boucle refait un tour avec l’instruction « continue ». Si ce n’est pas le cas, le compteur d’essais est remis à zéro et la boucle continue son traitement sans interruption.  
Ensuite, un test vérifie si le type est Ad OU si le type est Music ET du genre demandé, c’est en passant ce test que la musique/pub est ajouté à la playlist ainsi que sa durée qui est additionnée à la durée actuelle de la playlist. Une liste d’entier est aussi utilisée pour stocker les identifiants des morceaux ajoutés à la playlist afin de pouvoir les utiliser lors de l’ajout à la base de données.

Voici un schéma récapitulatif :



Figure 31 - Algorithme génération playlist

Au final, après la boucle, un test vérifie qu’il y a bien des musiques/pubs présentent dans la playlist. Si n’est pas le cas (l’algorithme n’a pas réussi à combler la durée malgré les essais), la méthode retourne false. Dans le cas contraire, la playlist est ajoutée à la base de données et au modèle. Pour finir, le fichier de configuration de la playlist est généré et la méthode « UpdateObservers » est appelée.

### Ajout à une playlist

L’ajout de musique/pub s’effectue depuis l’onglet « Library ».

Les 2 boutons d’ajout (celui dans la section Music ou la section Ad) appel la même méthode d’événement. Pour différencier quel bouton appel la méthode, la valeur « Music » ou « Ad » est inscrite dans la propriété « Tag » des boutons. De ce fait, il est facile de détecter si quel type de morceau l’utilisateur veut ajouter à quel type de playlist.

En premier temps, la vue va générer une liste de type Dictionnary<int,string>. La clé (int) correspond à l’identifiant de la musique sélectionnée et la valeur (string) correspond à son filename. Ce dictionnaire va permettre au modèle d’ajout les différents éléments dans le modèle et la base de données. Il est généré à l’aide de la propriété « SelectedRows » du composant dataGridView. Cette propriété donne la liste des lignes sélectionnées par l’utilisateur.

Par la suite, le dictionnaire est envoyé au modèle via le contrôleur. Il est parcouru par la méthode « AddToPlaylist » qui prend en paramètre un objet Playlist (correspondant à la playlist sélectionnée) et le dictionnaire. Pour chacun des éléments, il est ajouté à la base de données via la méthode de la classe Bdd « AddToPlaylist » qui prend en paramètre : la clé de l’événement (la valeur int qui correspond à l’id du morceau) et l’id de l’objet Playlist. Si cette méthode retourne une erreur, la boucle est arrêter et le modèle retourne false. Sinon, la boucle se finie et la playlist génère sa configuration (GenerateConfigFile sur l’objet playlist).

### Retirer d’une playlist

La suppression d’éléments d’un playlist se déroule exactement comme l’ajout (dictionnaire avec les morceaux sélectionnés par l’utilisateur qui est parcouru par le modèle) à l’exception que les éléments seront supprimée de la base de données (suppression du lien dans la table « tplaylist\_has\_music ») et du modèle. A la fin, la nouvelle configuration est générée.

### Affichage du contenu

Lorsque l’utilisateur choisi un élément dans un des ListBox (section Music ou Ad), l’événement « SelectedIndexChanged » est appelé. Comme pour les autres éléments, chacun des ListBox à son type dans sa propriété Tag afin de différencier dans la méthode de l’événement en question.

La méthode privée de la vue « GetPlaylistContent » qui prend un objet de type Playlist en paramètre, permet de vider le DataGridView servant d’affichage au contenu des playlists, puis, de le remplir avec le contenu de la playlist voulue. Pour se faire, elle récupère une liste d’objet de type AudioFile en provenance du modèle via son contrôleur. Le modèle va simplement parcourir sa bibliothèque et prendre les AudioFile dont le nom de fichier (filename) correspond à un des noms de fichier présent dans la playlist :

List<AudioFile> audioFiles = new List<AudioFile>();

foreach(string filename in playlist.AudioFileList)

{

foreach(AudioFile af in this.Library)

{

if (af.Filename == filename)

audioFiles.Add(af);

}

}

return audioFiles;

Ensuite, la vue va parcourir cette liste afin de remplir le DataGridView d’affichage.

Les classes Playlist disposent d’une méthode « GetInfosArray » qui retourne un tableau avec les informations de leurs champs. Cela permet de donner un tableau à la méthode d’ajout de ligne au dataGridView : dgvPlaylistContent.Rows.Add(af.GetInfosArray());  
Par la même occasion, la durée de chaque morceau ajouté est additionné à une variable de type TimeSpan afin de calculer le durée totale de la playlist et de l’affichée.

### Suppression d’une playlist

La suppression de playlist va supprimer l’enregistrement correspondant dans la base de données, la variable dans le modèle ainsi que le fichier de configuration enregistré sur le disque. Les vues sont ensuite mise à jour via UpdateObservers.

## Grille horaire

### Outil utilisé

Pour l’affichage et la gestion de la timetable[[9]](#footnote-9), un composant tiers est utilisé : <http://calendar.codeplex.com/>   
Il s’agit de Day View Calendar. Ce composant a été choisi car il propose une vue sous forme de jour et il est entièrement personnalisable (nombre de jours affichés, découpage des heures etc.). Voici un exemple d’utilisation :



Figure 32 - Day View Calendar

### Gestion du calendrier par ShoutCAST

C’est le transcoder fourni par ShoutCAST qui se charge de prendre en compte le calendrier et jouer les playlist aux bons moments. Ce calendrier est sous forme XML et il est composé d’événements (event). Chaque événement est composé des propriétés suivantes (celles présentée sont celles utilisées dans l’application. Pour une description complète, rendez-vous si le site officiel : <http://wiki.winamp.com/wiki/SHOUTcast_Calendar_Event_XML_File_Specification>) :

* Un type (playlist ou DJ) : Dans le cas de l’application, le type playlist sera toujours utilisé
* Une playlist : La lecture de la playlist dispose de plusieurs paramètres :
  + « loopatend » est un booléen qui définit si la playlist doit être rejouée quand tous les morceaux qui s’y trouvent ont été joué. Dans le cas de l’application, cette valeur est toujours « true ».
  + « shuffle » est un booléen qui définit si les morceaux de la playlist doivent être joués de façon aléatoire.
  + « priority » est une valeur entière non-signée qui définit la hauteur de la priorité de l’événement. C’est-à-dire, si 2 événements sont prévus au même moment, celui avec la plus haute priorité est joué.
  + La nom de la playlist jouée (nom du fichier .lst)
* Un horaire : L’horaire définit les paramètres suivants :
  + « starttime » est l’heure (format : hh:mm:ss) de début de l’événement.
  + « duration » est la durée (format : hh:mm:ss) de lecture de la playlist.
  + « repeat » est une valeur entière non-signée servant au transcoder pour savoir quel jour doit être lancé cet événement aux horaires données. Chaque jour de la semaine a une valeur et l’addition des valeurs des jours séléctionnés donne la valeur de « repeat ». Voici les valeurs définies par ShoutCAST :  
    1 - Every Sunday

2 - Every Monday

4 - Every Tuesday

8 - Every Wednesday

16 - Every Thursday

32 - Every Friday

64 - Every Saturday

128 - Time periodic : Cette dernière n’est pas encore implémentée. Pour plus d’information, rendez-vous au lien ci-dessus (lien vers le site officiel).

### Classes utilisées

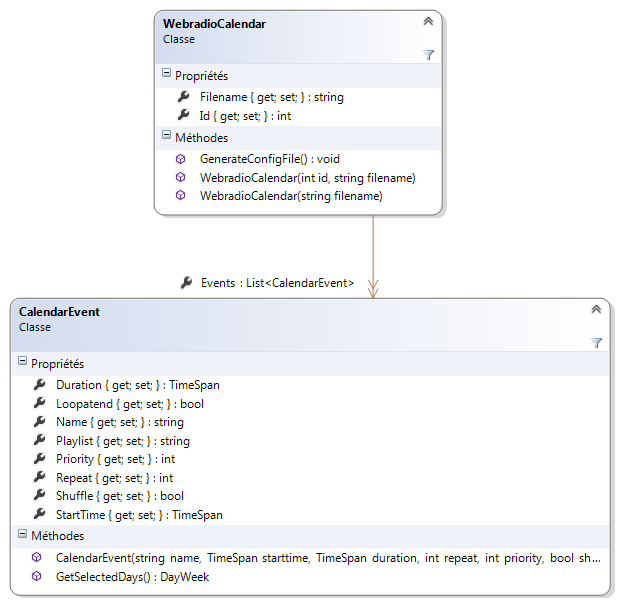


Figure 33 - Classes grille horaire

### Génération de configuration

ShoutCAST (transcoder) utilise un fichier XML pour la gestion de son calendrier. La génération d’un fichier XML sous C# se fait simplement avec un objet XmlDocument et des XmlElement. Voici un exemple de création de fichier de configuration XML pour un calendrier :

XmlDocument document = new XmlDocument();

XmlElement root = document.CreateElement("eventlist");

foreach(CalendarEvent ev in this.Events)

{

XmlElement eventelement = document.CreateElement("event");

eventelement.SetAttribute("type", "playlist");

XmlElement playlist = document.CreateElement("playlist");

playlist.SetAttribute("loopatend", (ev.Loopatend)?"1":"0");

playlist.SetAttribute("shuffle", (ev.Shuffle) ? "1" : "0");

playlist.SetAttribute("priority", ev.Priority.ToString());

playlist.InnerText = ev.Playlist;

eventelement.AppendChild(playlist);

XmlElement calendar = document.CreateElement("calendar");

calendar.SetAttribute("starttime", ev.StartTime.ToString("hh:mm:ss"));

calendar.SetAttribute("duration", ev.Duration.ToString("hh:mm:ss"));

calendar.SetAttribute("repeat", ev.Repeat.ToString());

eventelement.AppendChild(calendar);

root.AppendChild(eventelement);

}

document.AppendChild(root);

document.Save(this.Filename);

Voici le XML une fois généré :



Figure 34 - Exemple XML calendrier

### Affichage du calendrier

L’affichage du calendrier se fait avec le composant décrit dans le chapitre concernant [l’outil utilisé](#_Outil_utilisé). Il a été configuré pour afficher 7 jours (une semaine) ainsi qu’un quadrillage basé sur les demi-heures.

Les événements sont affichés selon la convention suivante :

* Evénement de type « Music » :
  + Couleur de bordure et de fond : Bleue
  + Contenu :   
    « *NomDeLevenement* (*HauteurPriorité*)  
    *NomDeLaPlaylist*  
    Shuffle : *TrueOuFalse* »
* Evénement de type « Ad » :
  + Couleur de bordure et de fond : Rouge
  + Contenu :   
    « *NomDeLevenement* (*HauteurPriorité*)  
    *NomDeLaPlaylist*  
    Shuffle : *TrueOuFalse* »

C’est lors de l’appel de la méthode « UpdateView » que le calendrier sera rempli. Ce composant a besoin de disposer d’une variable de type List<Appointment> qu’il utilisera afin de se remplir. Cette liste doit donc être remplie lors d’UpdateView et elle est globale à la vue AdminView. C’est en réalité l’événement « ResolveAppointments » du composant qui va par la suite lire cette variable et remplir ce dernier :

List<Appointment> m\_Apps = new List<Appointment>();

foreach (Appointment m\_App in this.EventsCalendar)

m\_Apps.Add(m\_App);

args.Appointments = m\_Apps;

L’événement est appelé lorsque le composant doit se rafraîchir. Il permet de personnaliser la façon dont on veut remplir le composant.

Dans le cadre du projet, une classe nommée « **EventAppointment** » a été créée. Elle hérite de la classe « Appointment » proposée par le composant mais elle propose une propriété de type Playlist afin de stocker l’objet Playlist lié à l’événement ainsi qu’une autre propriété de type CalendarEvent qui stocke l’objet CalendarEvent de l’événement. Cela est utile pour la suppression et modification. Cette classe reprend donc à l’identique la forme de sa classe parente mais seul 2 propriétés sont ajoutées.

Comme décrit précédemment, ShoutCAST utilise une valeur nommée « repeat » pour stocker quels sont les jours où l’événement doit se jouer. Calculer cette valeur se fait à l’aide des valeurs données, dans la documentation, pour chaque jour. Mais lors de l’affichage dans le calendrier et la création des EventAppointment, il faut décoder cette valeur afin d’en ressortir les jours sélectionnés. Pour se faire, une méthode utilisant des masques binaires est utilisée. La classe CalendarEvent propose une méthode « GetSelectedDays » qui retourne une structure « DayWeek » qui contient tous les jours de la semaine sous forme de booléen. Voici un exemple d’utilisation d’un masque :

DayWeek dow = new DayWeek();

dow.Monday = Convert.ToBoolean(this.Repeat & MONDAY\_MASK);

La structure ainsi remplie va être récupéré lors d’UpdateView et utilisé de cette manière :

DayWeek dw = ev.GetSelectedDays();

bool[] days = dw.ToArray();

for(int i = 0; i < 7;i++)

{

if(days[i])

{

//Création d’un EventAppointment et ajout à la liste utilisée par le composant Calendar

}

Le principe est le suivant : Une boucle est programmée pour tourner 7 fois (pour chaque jour de la semaine). Pour chaque tour (chaque jour), le programme test si le jour actuel a une valeur booléenne vraie dans le tableau retournée par la structure DayWeek rempli à l’aide de la classe de l’événement courant.  
Si la condition est vraie, un nouvel EventAppointment peut être créé, rempli et ajouté à la variable globale contenant les événements du calendrier.

### Sélection depuis le calendrier

Le composant permet de faire une sélection (d’une heure à une autre heure) avec la souris. L’événement « SelectionChanged » est appelé si tel est le cas. Les informations du formulaire de création d’événement sont automatiquement mise à jour en fonction de la sélection de l’utilisateur.

### Création d’un événement

Le formulaire de création est composé de 2 MaskedTextBox pour les champs « start time » et « duration ». Celle permet de garantir que le format d’entrée de ces 2 propriétés sera juste afin de créer des TimeSpan. Le masque utilisé est le suivant « 00:00:00 ». Afin de récupérer ces valeurs, un autre test doit être effectué : vérifier que le format de l’heure est réaliste (par exemple, pas de cette façon : 89:20:67. C’est une heure qui n’est pas possible). La méthode statique « TryParse » de la classe TimeSpan permet cette vérification :

TimeSpan start = new TimeSpan();

if (!TimeSpan.TryParse(mtbStartTime.Text, out start))

{

MessageBox.Show("Start time format is not correct.", "Error");

return;

}

Si le test passe, la méthode remplira la variable « start » avec les valeurs du MaskedTextBox.

Une durée minimum de 1 minute est exigée pour un événement. Aussi, le nom de l’événement doit être unique.

La génération de la valeur « repeat » s’effectue en fonction des cases cochées (jours de la semaine) par l’utilisateur. La méthode « GetRepeatValue » présente dans la vue va calculer et retourner la valeur. L’enum DayValue fourni les valeurs de chaque jours (définies par ShoutCAST et défini [précédemment](#_Génération_de_configuration_1)) :

int repeat = 0;

repeat += (ckbMonday.Checked) ? (int)DayValue.Monday : 0;

Et ainsi de suite pour chaque jour. Si la valeur retournée est 0, c’est que l’utilisateur n’a sélectionné aucun jour. Un message d’erreur apparaît. La valeur « loopatend » est toujours vraie.

Ensuite, l’ajout à la base de données et au modèle s’effectue de la même façon que pour les autre éléments du logiciel (ajout à la bdd, récupération du nouvel identifiant, ajout à l’objet puis ajout au modèle. Pour finir, UpdateObservers pour mettre à jour les vues. La méthode GenerateConfigFile est aussi appelée pour mettre à jour le fichier XML du calendrier avec les nouvelles valeurs.

### Modification d’un événement

Le composant Calendar permet de manipuler ses éléments avec la souris afin de les déplacer ou modifier leur longueur. Pour le moment, seule la modification de la longueur, le commencement et les jours d’un événement sont modifiables (voir le [chapitre fonctionnel](#_Modification_d’un_événement)). Les règles qui y sont décrites ont été faite pour correspondre le mieux possible avec le système de calendrier de ShoutCAST.

Lorsque l’utilisateur déplace un élément de l’événement et le change de jour, l’événement « MouseUp » du composant est appelé. En premier temps, il est testé s’il y a bien un élément (EventAppointment) sélectionné dans le composant. Puis, le traitement suivant est appliqué :



Figure 35 - Modification d'un événement

La méthode « CheckMovePossible » vérifie qu’il n’y a pas déjà un EventAppointment du même événement dans le jour où l’élément a été déplacé. Pour se faire, tous les éléments (EventAppointment) du composant sont vérifié un par un. Si un d’entre eux à la même date de début (propriété DayOfWeek de celle-ci) et le même identifiant d’événement (stocké dans l’objet CalendarEvent de l’élément), le mouvement sera alors impossible.

Concernant la mise à jour de la valeur « repeat », elle est recalculée par rapport à l’emplacement des différents éléments d’un événement dans le composant. Pour se faire, plusieurs étapes sont exécutées :

1. Récupération de tous les éléments (EventAppointment) du composant qui sont concernée par l’événement en question (l’événement déplacé). C’est la méthode « GetAllRelatedApppointment » de la vue qui va rechercher dans la liste d’EventAppointment utilisée par le composant.
2. Ensuite, la méthode « GetRepeatValueFromAppointment va calculter et retourner la valeur de repeat en fonction des EventAppointment qui lui sont donnés en paramètre. Pour chaque EventAppointment, il est vérifié (pour chaque jour de la semaine) si le « DayOfWeek » de sa propriété « StartDate » est égale au jour en question. De cette façon :  
   repeat += (ev.StartDate.DayOfWeek==DayOfWeek.Monday)?(int)DayValue.Monday : 0;

Pour finir, les configurations sont régénérées.

### Suppression d’un événement

Un clic droit sur un élément appel l’événement « MouseClick » du composant. Il est vérifié que le bouton cliqué est bien le droit et si il élément de type Appointment est bien sélectionné sur le composant. Une boite de dialogue demande confirmation à l’utilisateur puis la méthode « DeleteEvent » du contrôleur est appelé. L’objet CalendarEvent stocké dans l’EventAppointment est donné en paramètre. Cette méthode ira appeler la même méthode dans le modèle qui s’occupera, lui, de supprimer l’événement dans la BDD et ses propriétés. UpdateObservers est ensuite appelé ainsi que la régénération de configuration.

### Génération automatique

TODO

## Transcoder

### Classes utilisées

TODO : schema : tous les transcoder utilise le même executable (pourquoi ? car il faut juste mnettre a jour UN executable) mais juste des fichier de config différent. Ils utilisent aussi tous les meme calendrier

TODO explication du system de fichier xml pour gerer le transcoder etc ( détails dans des nouveau chapitre sur les playlist etc ) et qu’il gère tout seul els calendrier etc

TODO : expliquer le principe de fonctionnement général d’un transcoder shoutcast. Le fauit qu’il y ai un mot de passe etc

### Définition des bitrates, taux d’échantillonnage et type d’encoder

TODO : expliquer que pour les type c’est un enorme, les bitrate c’est un tableau fixe et static dans la classe WebradioTranscoder et de même pour les taux

### Fichier de configuration et log

A chaque generation, suppression du fichier déjà présent

TODO IMPORTANT : Il faut lister les playlist de la webradio dedans sous la forme :

playlistfilename\_1=HardLol

playlistfilepath\_1=C:\Users\MENETREYS\_INFO\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug\webradios\Gorilo\playlists\HardLol.lst

pour se faire, un paramètre de type List<Playlist> pour generer

TODO : explication des différents elements du fichier de conf ainsi que les element modifiable par l’utilisateur

Chaque transcoder = un fichier de config dans le dossier Transcoders de la webradio. Le nom du fichier de config = identifiantBdd.config pareil pour le log mais avec .log

### Licence MP3

TODO si mp3, il faut mettre sa licence dans le fichier de config (a la main et demandé a l’utilisateur)

### Création d’un transcoder

TODO : vérifier que l’ip ets juste avec TryParse de AdresseIp

TODO : nom de flux unique

TODO : ajout d’abbord le a la bdd sans filename puis update le filename avec l’id récupéré

TODO : schéma des étapes :

1. Création d’un objet transcoder avec des configfilename e tlogfilename non valide
2. Insertion dans bdd et récupérationdel’id
3. Mise à jour de l’enregistrement précédent (champ configfilename e tlogfilename) avec le bon filename (grace a l’id récup)
4. Retour dans le modèle, mise à jour de l’objet Transcoder avec l’id retournée par la classe BDD (comme dhabb)
5. Si id =/= error, c’est ok il n’y a pas de doublon
6. Mise a jour des meme champs dans l’objet transcoder
7. Generéré configfile et updateobservers

### Affichage d’un transcoder

TODO expliquer PrepareIpAddress

### Modification d’un transcoder

TODO : dredémarre le processsu si il est lancé

TODO : modifie les infos dans l’objet transcoder séléctionné dans listbox et l’envoie au modèle pour modifier la database

### Suppression d’un transcoder

TODO ; supprimer fichier config et log

### Exécution et fermeture

TODO : si exectuon ou fermture ok : updateobservers

TODO : tout fermer (processus) quand le programme se ferme

TODO : vérifier que éles fichier sont présent

TODO : prépare des StartInfo pour le lancement (permet de lancer sans fenetre apparante)

regenrationdu calendrier après le stoppage

TODO : expliquer detection de crash processus (lancement se plante a cause d’un problème de prot déjà utlisé par exemple)

TODO : erreur possible : sc\_trans déjà lancé ou non présent « sc\_trans.exe »

### Gestion des processus

TODO : avoir une liste de processus où les transcoders sont lancé afin de pouvoir récupéré leur log et statut facilement

TODO : lancement de transcoder, etc

TODO :

### Statut et logs

TODO : se référé au processus ou alors requete ajax dessus pour avoir toutes les informations

<http://wiki.winamp.com/wiki/SHOUTcast_Transcoder_AJAX_api_Specification#LogData>

Ou afficher depuis fichier de log (blinder fichier au listbox)

TODO : expliquer IsRunning

TODO : expliquer que redirection et ouverture de fichier pas possible donc affichage de la fenetre dos comme log

TODO : mode debug

### Administration web (weblet)

TODO : expliquer les commande executé, le port d’adsministration

### Historique

TODO : historique dans bdd etc

Historique dans transcoder car coter serveur = pas possible (infomaniak par exemple)

Demander au server via requete ?

## Serveur de diffusion interne

### Classe utilisée

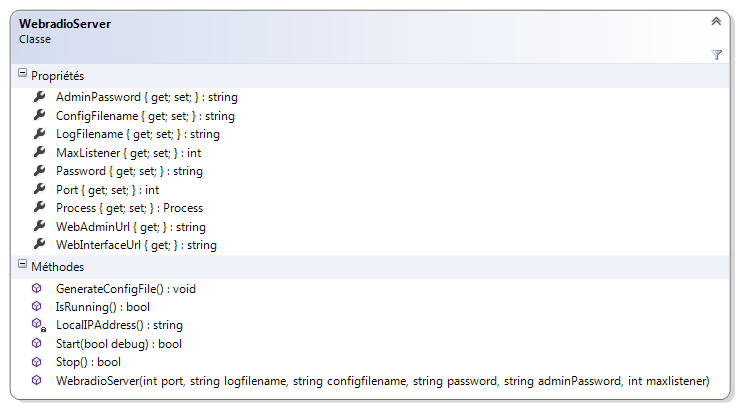


Figure 36 - Classe WebradioServer

### Outil utilisé

Shoutcast est avant tout un serveur de diffusion de flux audio ou vidéo (Shoutcast DNAS server 2). L’outil propose un serveur en ligne de commande qui sera utilisé dans ce projet. Le serveur fonctionne avec un fichier de configuration qui lui est donné en paramètre lorsque l’on l’exécute :

sc\_serv.exe myconfig.config

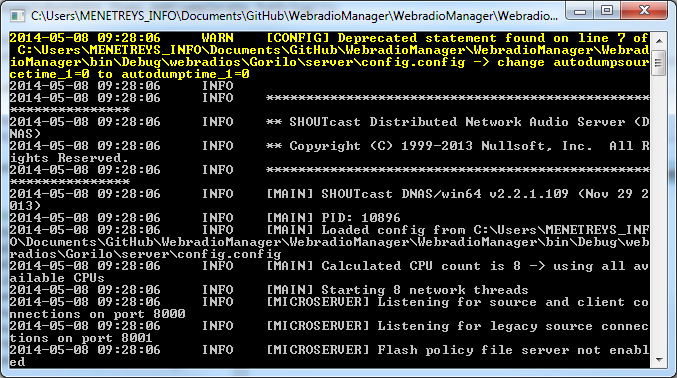


Figure 37 - ShoutCAST serveur

La documentation concernant la configuration du serveur via le fichier est expliquée ici :   
<http://wiki.winamp.com/wiki/SHOUTcast_DNAS_Server_2> . Dans le cadre de ce projet, seules quelques options seront utilisées et configurées. Une partie est configurable par l’utilisateur via l’interface de l’onglet «[server](#_Gestion_du_serveur)» et l’autre partie est configurée par défaut par l’application :

* L’emplacement du fichier de log
* L’emplacement du fichier de configuration lui même

Pour plus de détails sur les emplacements des différents fichiers, rendez-vous au chapitre concernant [la structure des dossiers/fichiers](#_Structures_des_dossiers/fichiers).

### Configuration

Voici la liste des paramètres configuré dans le fichier de configuration d’un serveur dans WebradioManager :

* Logfile : Le chemin vers le fichier de log
* Portbase : Le numéro du port du serveur (sur lequel le transcoder se connecte)
* Password : Le mot de passe de connexion pour une source (un transcoder)
* Adminpassword : Le mot de passe d’administration de l’interface web. Attention, il doit être différent du mot de passe de connexion pour les sources.
* Publicserver : Cette valeur sert à définir sur la webradio sera indexée dans le site de référencement de ShoutCAST. Il n’est pas d’actualité donc la valeur par défaut « always » est définie.
* Maxuser : Le nombre maximum de clients/auditeurs connectés
* Autodumpsourcetime : Nombre de seconde à attendre avant de déconnecter une source (transcoder) si son flux est vide. Dans le cas du projet, la valeur 0 est entrée par défaut (cela correspond à désactiver cette fonction) car il n’est pas voulu que le serveur ferme les connexions vides.

Exemple de fichier de configuration :

logfile=C:\Users\MENETREYS\_INFO\Documents\GitHub\WebradioManager\WebradioManager\WebradioManager\bin\Debug\webradios\Gorilo\server\log.txt

portbase=8000

password=lol

adminpassword=admin

publicserver=always

maxuser=32

autodumpsourcetime=0

### Mise à jour de la configuration

Lors d’une mise à jour de la configuration du serveur, il est d’abord arrêté puis modifié et ensuite redémarré (seulement dans le cas où il était démarré lors de la sauvegarde de la nouvelle configuration).

Les informations du formulaire sont passées au modèle via le contrôleur de la vue. Il est testé si le serveur est allumé, si c’est le cas il est éteint et une variable booléenne est enregistrée à « true » afin de savoir, à la fin du traitement, si le serveur doit être redémarré.

Les informations sont modifiées dans la base de données, puis si la modification a réussie, elles sont modifiées dans le modèle. En fin de traitement, la méthode « UpdateObservers » est appelée.

### Exécution et fermeture

L’exécutable du serveur ShoutCAST est lancé depuis la classe « WebradioServer » et sa méthode « Start ». Cette dernière utilise la classe « Process » afin de créer un processus où l’exécutable sera lancé. Une classe nommée « ProcessStartInfo » permet de donner des paramètres d’exécution du processus :

ProcessStartInfo StartInfo = new ProcessStartInfo(Directory.GetCurrentDirectory() + SC\_SERVER\_FILENAME)

{

CreateNoWindow = true,

WindowStyle = (debug)?ProcessWindowStyle.Minimized:ProcessWindowStyle.Hidden,

Arguments = Directory.GetCurrentDirectory() + "\\" + this.ConfigFilename.Replace('/', '\\')

};

Il y a 2 modes d’exécution pour le serveur :

* Avec debug : Ouverture de la fenêtre console de l’application serveur ShoutCAST
* Sans debug : Pas d’ouverture de l’application console

Ce mode est choisi par l’utilisateur grâce à la case à cocher qui se trouve dans l’onglet « server ».  
Dans le code ci-dessus, la variable booléenne « debug » va définir si la propriété « WindowStyle » doit être « minimized » (ouverte mais minimizée) ou « hidden » (pas de fenêtre apparente). Dans le premier cas, le debug est activé.

TODO : si exectuon ou fermture ok : updateobservers

TODO : erreur possible : pas de sc\_server.exe ou sc\_serv déjà lancé

### Affichage des interfaces web

TODO : pèropriété spécial dans la classe webradioserver

TODO : pour admin, le nom d’utilisateur est toujours admin

### Log

TODO : expliquer que redirection et ouverture de fichier pas possible donc affichage de la fenetre dos comme log

TODO : mdoe debug

# Gestion des processus

TODO : expliquer l’utilisation de la classe Process

TODO : explique que dans le programme il y a un tas de processsu mais qu’il n’est pas possible de savoir directement si l’un d’eux aplanter ou a été fermé manuellement. Pour cela, implémentation dusystzème suivanrt :

* List<Process> dans la modèle. Les différent processus sont « copié » (ajouté la référence) dans cette liste lors des start et stop du programme
* Une timer va vérifier a intervalle régulier (spécifier) si chacun des process repond, si il ne repoind pa,s on l’enlève de la liste et on fait un updateobservers

TODO : schema

TODO : fermture a la fermture de l’application

# Tests

# Plannings

## Prévu

## Final

# Apports personnels

# Conclusion

# Améliorations possibles

* Modification des informations de la bibliothèque (puis tag directement sur le fichier lié)
* Modification de la « timetable » avec des drag and drop etc.
* Multi-serveur de diffusion par transcoder
* Evènements périodiques (+ top horaire)
* Capture live
* DJ
* Gestion de semaine « type » (gestion multi semaine)

# Références

* <https://cacoo.com> : Création de diagrammes en ligne
* <http://balsamiq.com/> : Création de « mokup »
* <http://calendar.codeplex.com/> : Composant C# pour l’affichage du calendrier sur une semaine
* <http://sqlitestudio.pl/> : Logiciel de gestion de base de données SQLite

# Annexes

1. SUISA est la coopérative des auteurs et éditeurs de musique <http://www.suisa.ch/fr/> [↑](#footnote-ref-1)
2. Fournisseur de services internet [↑](#footnote-ref-2)
3. <http://www.infomaniak.com/> [↑](#footnote-ref-3)
4. Débit binaire : une mesure de la quantité de données numériques transmises par unité de temps. [↑](#footnote-ref-4)
5. Taux d’échantillonnage [↑](#footnote-ref-5)
6. Dynamic Link Library [↑](#footnote-ref-6)
7. Lecteur multimédia développé par Nullsoft [↑](#footnote-ref-7)
8. eXtensible Markup Language : Langage de balisage extensible [↑](#footnote-ref-8)
9. Table du temps ou grille horaire en anglais. [↑](#footnote-ref-9)