Institut **S**upérieur
d'**I**nformatique, de
Modélisation et de
leurs **A**pplications

Campus des Cézeaux
1, Rue de la Chebarde
TSA 60125
CS 60026
63173 Aubière Cedex

Rapport d'Ingénieur
Projet de troisième année
Filière Informatique des Systèmes Embarqués

Bornes de diffusion Radio Campus

Présenté par : William BEDU, Mustapha Hajaoui

Tuteurs : Jean-Marie Favreau et Aurélie Grenard
Responsable ISIMA : Romuald Aufrere

Soutenance : 09 mars 2016
120 heures

Remerciements

Nous souhaitons remercier les personnes qui nous ont permis de réaliser ce projet enrichissant. Nos premiers remerciements vont à M. Jean-Marie Favreau, notre tuteur à Radio Campus Clermont-Ferrand pour sa disponibilité, son accompagnement , son écoute et son travail sur les services web. Ensuite, nous tenons à remercier Mme Aurélie Grenard la directrice de la publication pour son investissement, ses remarques pertinentes et son écoute. Enfin, nos derniers remerciements vont à notre tuteur ISIMA M. Romuald Aufrere pour avoir accepté d'encadrer ce projet ainsi qu'à l'association « Acolab » pour nous avoir proposé une aide technique et matérielle pour la réalisation du prototype.

Table des illustrations

Résumé

Radio Campus Clermont-Ferrand fait parti du réseau **Radio Campus** France composé de 30 radios étudiantes. Radio Campus Clermont-Ferrand (RCCF) est une radio associative qui offre une alternative aux radios commerciales. Elle se veut une radio proche de ses auditeurs, et un médiateur direct entre les artistes en tout genres et les auditeurs. Or, depuis quelques années, RCCF perd régulièrement des auditeurs par manque de visibilité. En réponse à ce phénomène, RCCF a initié un projet innovant : celui de développer et déployer de nouveaux dispositifs permettant une écoute collective de Radio Campus et prenant la forme de bornes multimédia.

La première étape de sélection du matériel, nous a guidée vers l'utilisation d'un ordinateur **Raspberry Pi** combiné à un petit écran tactile en lieu et place de l'utilisation d'une tablette tactile. En effet, nous souhaitions être capable de nous connecter à la borne directement avec le protocole SSH pour son administration. Nous avons également décidé de développer l'application embarquée en utilisant le **framework Qt** et le langage de programmation **C++**. L'interfaçage de notre application avec les services de RCCF a été réalisé au travers de **services web** (cf lexique) qui retournent les données demandées au format JSON (cf lexique).

Les fonctionnalités que nous sommes parvenus à implémenter sont les fonctionnalités liées à la lecture de flux audio direct, la lecture d'émissions en podcasts, l'enregistrement de message vocaux ainsi que la mise en valeur des émissions de Radio Campus. Néanmoins, nous ne sommes en revanche pas parvenus, par manque de temps à implémenter les fonctionnalités liées à l'administration à distance des bornes. De même, la réalisation physique de la borne n'a pas pu être effectuée par manque d'outillage, de matière première et d'expérience de notre part dans ce domaine.

Pour conclure, nous sommes parvenus à fournir à RCCF un premier prototype de borne qui implémente les fonctionnalités les plus importantes. La prochaine étape pour ce projet est d'ajouter la fonctionnalité d'administration à distance ainsi que de designer et construire le corps de la borne.

Mots clefs : Radio Campus, framework Qt, C++, Raspberry Pi, services web

Abstract

Radio Campus Clermont-Ferrand is part of the **Radio Campus** France network composed by 30 student radios. Radio Campus Clermont-Ferrand (RCCF) is an associative radio which offers an alternative to commercial radios. The aim of RCCF is to maintain proximity with its audience and to be the direct mediator between any kind of artists and its auditors. Besides, for a few years, RCCF has been regularly losing auditors because of a lack of visibility. In response to this phenomenon, RCCF has initiated an innovative project: developing and deploying new devices that allow a collective listening of Radio Campus by using multimedia physical terminals.

The first step, which was the selection of the hardware guided us to the use of a **Raspberry Pi** computer combined with a little touchscreen instead of using a touch-pad. In fact, we wanted to be able to connect to the terminal directly with the SSH protocol. We decided to develop the embedded application with the **Qt framework** and the **C++** language. We interfaced our application with RCCF services through **web-services** which return data in JSON format.

The features we managed to implement are the ones that are linked to direct audio stream listening, podcasts listening, vocal messages recording and the Radio Campus playlists enhancement. Nevertheless, we didn't manage to implement the features in relation with the remote administration of the terminals because of the lack of time. Likewise, the production of the physical terminal couldn't be done because of a lack of tools, raw material and experience in this field.

To conclude, we managed to give to RCCF a first prototype of the terminal which implements the most important features. The next step for this project is to add the remote administration feature to the software and design and build the physical terminal's body.

Keywords: Radio Campus, Qt framework, C++, Raspberry Pi, web-services

Table des matières

Introduction

Nous avons réalisé ce projet dans le cadre de notre troisième année à l'Institut Supérieur d'Informatique, de Modélisation et de leurs Applications (ISIMA). Il a été proposé par Radio Campus Clermont-Ferrand dans le cadre d'un projet innovant permettant à la radio de renforcer son audience en utilisant les nouvelles technologies pour se rapprocher de ses auditeurs.

Radio Campus Clermont-Ferrand est une radio associative faisant parti du réseau Radio Campus France. A ce titre, Radio Campus Clermont-Ferrand (RCCF), constitue une alternative aux radios commerciales. Son objectif est de promouvoir de nouveaux artistes, développer un espace d'expression, d'échange et favoriser la diffusion d'une information citoyenne. Depuis toujours, RCCF cherche la proximité avec ses auditeurs, en particulier avec les étudiants. Or des études récentes ont démontré que le nombre d'auditeurs étudiants était en régression. Pour lutter contre ce phénomène, RCCF a imaginé un projet novateur et unique pour lui offrir une plus grande visibilité : la conception de bornes interactives permettant la diffusion et l'écoute collective de Radio Campus dans des espaces publics de détente du campus de Clermont-Ferrand.

Notre rôle dans ce projet était de fournir le premier prototype de borne respectant le cahier des charges établi par le travail d'étudiants de l'UBP nous ayant précédé sur le projet. Nous avions à charge la livraison d'une application déployée sur un matériel pouvant facilement être intégré dans une borne physique.

La question qui servira de fil directeur de ce rapport sera : Quelles sont les étapes de réalisation d'une borne de diffusion radiophonique ? Pour tenter d'y répondre, nous aborderons tout d'abord une présentation du contexte du projet, puis nous évoquerons les outils et le matériel retenu pour en venir à parler de l'architecture imaginée. Nous nous attarderons ensuite sur la réalisation des différents modules de l'application pour terminer sur un bilan du projet.

1. Le Cadre du projet

1.1. Présentation du réseau Radio Campus France

Radio Campus France est un réseau national regroupant 29 radios étudiantes sur tout le territoire Français. Les radios du réseau Radio Campus diffusent leur contenu 24h/24h et 7j/7J aussi bien sur la bande FM que via le web et touchent ainsi environ 22 millions de personnes en France dont plus de 1 200 000 étudiants.

Toutes les radios appartenant au réseau partagent des valeurs et objectifs communs. Parmi ces objectifs et ces valeurs, on retrouve la volonté de créer une communication à double sens entre le campus et ses étudiants et la population urbaine en diffusant des informations liées à la vie étudiante, à la vie culturelle ainsi que des informations nationales. Radio Campus France se donne également pour objectif de permettre aux jeunes de s'investir dans la vie associative en devenant bénévoles tout en pouvant ainsi développer et mettre à profit leurs talents artistiques ou journalistiques. Enfin, le réseau Radio Campus se veut un relais direct entre les différentes formes d'expression artistiques et les auditeurs en favorisant la promotion de nouveaux talents musicaux ou artistiques en général, en participant ou en organisant des événements (concerts, rencontres...). Les émissions de Radio Campus présentent, analysent et décryptent l'art contemporain, le théâtre, la création sonore, la danse, la musique, la littérature, le cinéma et la création multimédia.

Radio Campus France ne vise pas un public de niche mais concerne de nombreux potentiels auditeurs : les étudiants, les jeunes, les consommateurs de culture, les acteurs culturels ainsi que les acteurs spécialisés mais aussi tous les publics urbains, citoyens, mobiles et curieux. Pour conserver et fidéliser ce public, les radios du réseau ont un véritable soucis de la proximité et du lien direct avec leur auditoire. C'est d'ailleurs dans ce soucis de développer la proximité de Radio Campus avec ses auditeurs que s'inscrit notre projet.

1.2. Présentation de Radio Campus Clermont-Ferrand

Radio Campus Clermont-ferrand (RCCF) est une association fondée en 1995 et portée par 3 salariés, 2 volontaires civiques et pas loin de 200 bénévoles qui proposent près de 60 émissions sur de nombreux domaines : musiques actuelles (musiques amplifiées, musiques du monde, jazz, chanson), culture, politique, société, science,environnement, emploi, humour. Elle diffuse sa programmation (40% de programmes musicaux, 30% d'émissions thématiques et associatives, 30% de chroniques) en FM sur le 93.3 grâce à une antenne située sur le Campus des Cézeaux et ayant une portée de 30 km autour de Clermont-ferrand. Elle diffuse également sa programmation via internet sur son site : http://www.campus-clermont.net/.

1.3. Constat et objectifs du projet

Le projet de bornes de diffusion Radio Campus est parti d'une enquête réalisée en 2013 par des étudiants de l'UBP dans le cadre du projet collectif "Musiques actuelles, programmation et diffusion". Cette enquête était portée sur les habitudes d'écoutes radiophoniques et musicales des étudiants Clermontois. En parallèle à cette étude, une

seconde étude menée par des étudiants de l' École Supérieure de Commerce sur l'influence de Radio Campus auprès d'étudiants entre 18 et 25 ans. La conclusion de ces deux analyses sont sensiblement les mêmes : l'audience étudiante de RCCF est en régression. Cette régression a été en partie expliquée par le contenu des émissions non adapté à l'attente de cette audience ainsi qu'à un manque de visibilité de Radio Campus malgré tous ses efforts. L'association propose alors une nouvelle stratégie pour augmenter sensiblement sa visibilité : placer un dispositif de diffusion nouveau dans les lieux universitaires.

En 2014, des étudiants de l'UBP en master professionnel « Action culturelle et gestion de projets arts du spectacle » sont chargés de réaliser une étude préliminaire de faisabilité d'un tel projet. Le but de cette étude était de poser les bases, définir les besoins, commencer à prendre des contacts et réaliser un premier cahier des charges. Les étudiants ont ainsi pu démontrer le caractère novateur du projet (aucun projet similaire n'existe en France selon leur analyse). Les étudiants de l'UBP formalisent alors le nouveau dispositif de diffusion sous la forme d'une borne ou d'une mallette placée dans les lieux de vie universitaires. Cette borne a pour fonction première d'implanter durablement Radio Campus sur les lieux de vie des étudiants qui sont les principaux destinataires de ses programmes. Cette borne pourrait alors permettre aux étudiants de découvrir Radio Campus dès leurs premiers jours de vie universitaire contrairement à maintenant où la majorité des étudiants n'en n'ont jamais ou tardivement eu connaissance.

Outre le fait de marquer physiquement la présence de Radio Campus dans l'université, ces bornes ont un caractère novateur dans le sens où elles permettent une nouvelle forme d'écoute de contenu radiophonique . Alors que l'écoute de la radio est traditionnellement une écoute individuelle, ce projet permet, un peu à la manière des juke-box, une écoute collective.

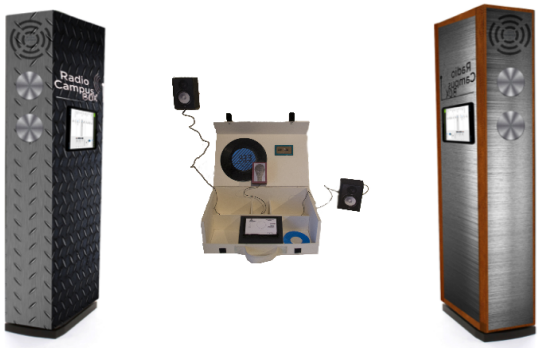

Illustration 1: Différents designs de borne imaginés par les étudiants de l'UBP nous ayant précédés sur le projet

1.4. Les besoins liés au projet

Notre projet à l'ISIMA s'inscrit dans la concrétisation du travail de ces étudiants. Notre rôle était de fournir à RCCF un prototype de borne respectant les besoins représentés sur le diagramme des cas d'utilisations suivant :

Illustration 2: Diagramme des cas d'utilisation des bornes de Radio Campus

Ce diagramme représente l'analyse que nous avons fait des principaux besoins de Radio Campus concernant les bornes. Ainsi, nous pouvons observer que les principaux besoins concernent la lecture du flux radio direct, la possibilité pour les utilisateurs de laisser un message vocal pouvant être récupéré et écouté à Radio Campus (pour donner des idées d'amélioration, des suggestions, des critiques etc...) ainsi que l'écoute des émissions déjà diffusées sous forme de podcasts (cf lexique).

Il est à noter que le souhait de RCCF concernant la lecture de podcasts est que celle-ci puisse se faire aussi bien en mode connecté que déconnecté. En résumé, la borne doit continuer à pouvoir proposer des écoutes de podcasts même si celle-ci se retrouve déconnectée du réseau. Cette fonctionnalité sous-entend donc une nécessité de synchronisation du contenu en ligne avec le contenu présent en local sur la borne.

Les autres souhaits de Radio Campus sont de pouvoir mettre en avant leurs émissions spéciales appelées « 100 % » et de faciliter leur accès par les utilisateurs. De même, la borne devra pouvoir gérer son volume sonore automatiquement en fonction de l'heure courante. En effet, les bornes étant situées dans des lieux de vie universitaire, il est nécessaire que celles-ci ne puissent pas diffuser du contenu audio avec un volume trop élevé pour ne pas déranger les étudiants en plein travail. Il doit donc être prévu une fonctionnalité permettant à la borne de couper automatiquement la diffusion sur certains créneaux horaires.

Enfin, la dernière fonctionnalité souhaitée est la possibilité d'administration de la borne à distance. Cette administration doit permettre à des administrateurs d'écouter et gérer les messages vocaux reçus, de voir l'état de la borne et de la redémarrer si besoin. A ce cas d'utilisation s'ajoute la possibilité pour l'UBP de pouvoir diffuser des messages textuels informatifs. Ce cas d'utilisation doit permettre à l'UBP de profiter du déploiement de la borne pour fournir un moyen de communication rapide et efficace auprès de ses étudiants. Cette fonctionnalité a également été imaginée pour faciliter le déploiement de la borne dans les locaux appartenant à l'UBP en suscitant l'intérêt de l'université pour le projet.

1.5. Le découpage du projet en tâches

Dès les premiers temps de la réalisation du projet, nous avons découpé celui-ci en tâches planifiées pour nous permettre d'avoir une indication sur notre retard ou avance au fil de la réalisation. Le diagramme de GANTT prévu est représenté sur la figure suivante :

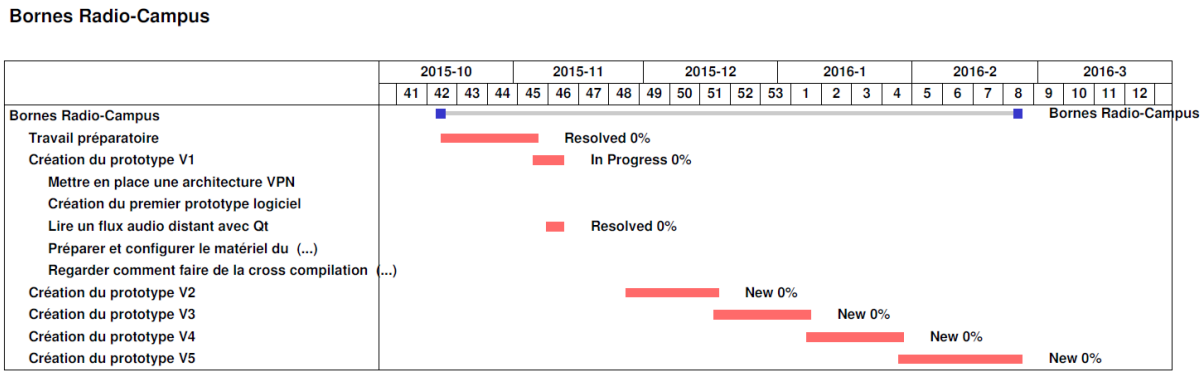

Illustration 3: Diagramme de GANTT prévisionnel du projet

Nous souhaitions initialement gérer le projet de manière itérative : Nous avions prévu de réaliser plusieurs itérations de l'application en démarrant avec les fonctionnalités de base pour aller petit à petit vers toutes les fonctionnalités souhaitées par RCCF. Les différentes itérations correspondent aux différentes versions du prototype mentionnées sur le diagramme de GANTT prévisionnel avec le découpage suivant :

- Prototype V1 : Application de lecture de flux audio distant déployée sur le Raspberry Pi
- Prototype V2 : Implémentation de la gestion des podcasts
- Prototype V3 : Implémentation de la gestion des messages provenant de l'UBP
- Prototype V4 : Implémentation de la possibilité de laisser un message vocal à Radio Campus
- Prototype V5 : Système d'administration des bornes à distance

En réalité, nous ne sommes pas parvenus à suivre ce planning ni même au mode de gestion de projet itératif initialement souhaitée. Ainsi, la réalisation du premier prototype a été beaucoup plus longue que prévu. Il a fallu imaginer et implémenter une première architecture logicielle, un premier embryon d'interface et travailler sur des problèmes de lecture du flux radio depuis Radio Campus. De même, l'implémentation des fonctionnalités liées à la gestion des podcasts a été clairement sous évaluée : nous n'imaginions pas tous les services liés qu'il était nécessaire d'implémenter. De même, nous remarquons que la fonctionnalité qui concerne la mise en valeur des émissions 100 % n'apparaît pas dans ce premier diagramme de GANTT. En effet, cette fonctionnalité n'a été demandée que tardivement dans le projet. Enfin, nous avons eu quelques soucis à régler pour le déploiement de l'application sur la Raspberry Pi. Concernant la fonctionnalité de diffusion de messages textuels par l'UBP, celle-ci a été effectuée en parallèle des autres tâches. Enfin, le temps de réalisation du rapport et de correction de bugs n'ayant pas été pris en compte sur

le diagramme initial, nous avons dû renoncer aux fonctionnalités d'administration à distance des bornes trop ambitieuses pour le temps restant. Le planning qui a donc réellement été suivi est représenté sur la figure suivante :

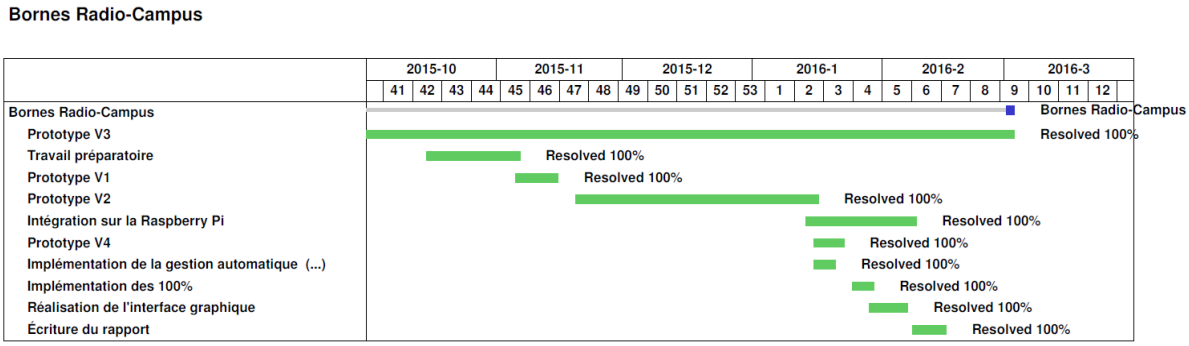

Illustration 4: Diagramme de GANTT réel

1.6. L'architecture du réseau de diffusion de RCCF

Pour commencer à imaginer une solution technique appropriée, il a été nécessaire de bien comprendre l'architecture de l'infrastructure de Radio Campus déjà existante. Cette infrastructure peut être résumée par le schéma suivant :

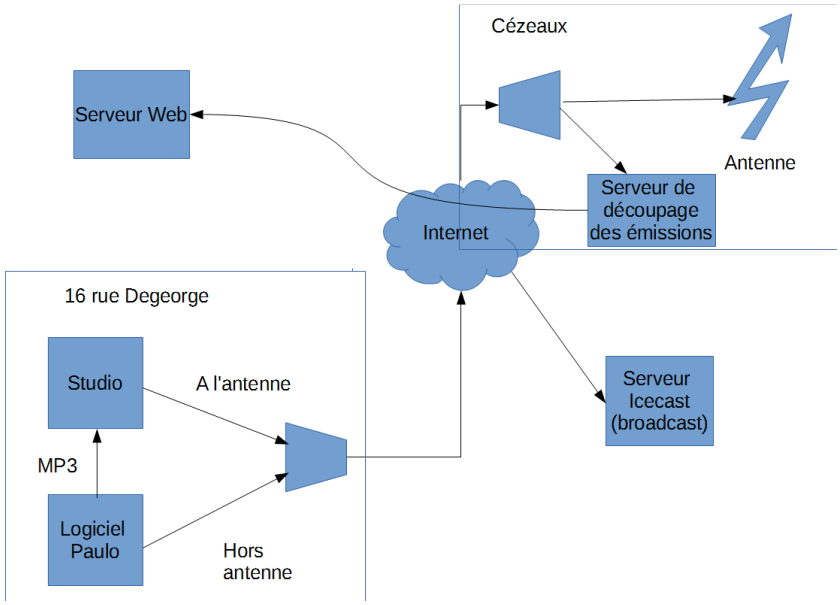

Illustration 5: Schéma résumant l'infrastructure de RCCF

Le contenu de la programmation de RCCF est produit dans leurs studios situés au 16 rue Degeorge à Clermont-Ferrand. Le logiciel Paulo est un logiciel permettant de diffuser automatiquement des titres musicaux ou des contenus enregistrés pris dans une base de données de titres disponibles en fonction de règles pré-établies. Il fournit à tout moment un

flux MP3 correspondant au titre joué qu'il envoie au studio ainsi qu'à un commutateur à la sortie des studios. En parallèle, le studio produit lui aussi un flux audio qui est aussi envoyé à ce commutateur. Le studio peut alors passer en direct, dans ce cas là, le flux audio que laissera passer le commutateur sera le flux du studio ou en hors direct. Dans ce cas là, seul le flux produit par Paulo sortira du commutateur. Le flux sortant des studios est envoyé via internet sur le campus des Cézeaux au pied de l'antenne relais ainsi qu'à un serveur de broadcast (cf lexique) Icecast2 (cf lexique) permettant d'écouter le flux radio en direct via internet. Un répartiteur se charge alors de diffuser le flux à l'antenne pour l'émission FM ainsi qu'à un serveur dont le rôle va être de générer les podcasts. Ce serveur va découper le direct en enregistrements d'une heure. A chaque nouvelle heure, le fichier MP3 généré est envoyé via internet au serveur web pour qu'il y soit stocké. C'est ce serveur web qui héberge le site internet de RCCF, qui contient les bases de données des émissions ainsi que diverses informations telles que la description des émissions, leur jaquette etc. C'est également sur ce serveur que l'on peut choisir quel podcast sera disponible à l'écoute ou non. De nombreux web services (cf lexique) sont hébergés sur cette machine. Ils vont permettre d'accéder aux informations importantes sur les émissions ou les podcasts.

2. Outils et matériel

Dans cette section, nous allons aborder les outils utilisés pour gérer ce projet, ainsi que le matériel retenu pour la création de la borne de diffusion.

2.1. Les outils

2.1.1. Un gestionnaire de code source : SVN

Notre principale idée pour organiser notre travail a été l'utilisation d'un gestionnaire de code source ou logiciel de gestion de versions. En effet, utiliser un tel logiciel est vital dans un projet collaboratif. Il permet d'enregistrer, de suivre et de garder une traçabilité des modifications effectuées sur le code source. L'idée est alors de pouvoir, à tout moment revenir à une version ultérieure, analyser l'historique des modifications pour identifier un bug lors d'un passage d'une version à une autre ainsi que d'obtenir des informations telles que le nom de la personne ayant effectué la modification, la date et éventuellement un commentaire expliquant le contenu de la modification. Outre cette traçabilité, l'utilisation du gestionnaire de code source nous permet de conserver une copie intègre du code sur un serveur distant. Ainsi, celui-ci est disponible à tout moment sur toutes les machines connectées à internet. Plus besoin de se partager une clef USB contenant des sources qui ne sont pas à jour. Le code de l'application étant écrit sur nos machines de bureau et pas directement sur la borne, cet aspect nous a également permis de le déployer rapidement et simplement sur la borne pour pouvoir le compiler et le tester.

Le gestionnaire de version que nous avons utilisé est Apache Subversion. C'est un système de gestion de version simple et très utilisé, distribué sous licence Apache et BSD. De plus Subversion (SVN) dispose d'une grande maturité car sa première version a été publiée en 2000. Pour l'utiliser simplement, nous avons également installé « TortoiseSVN », un utilitaire graphique permettant de lancer des commandes SVN sous Windows.

2.1.2. Un outil de gestion de projet : La forge de l'université

L'outil de gestion de versions SVN nous a permis d'organiser et de gérer efficacement

notre code source. Cet outil faisait partie de la forge de l'université d'Auvergne. La forge est un outil appartenant à l'université et permettant de gérer efficacement des projets. On y retrouve des outils de partage de documents, de gestion de tâches (assignation, attribution, planification etc.), de gestion de planning (GANTT), de création de Wiki ainsi que de partage d'informations. C'est sur cet outil que l'on retrouve le dépôt SVN(endroit où sont stockées les sources gérées par SVN) de notre projet. Nous nous sommes principalement intéressés à cet outil pour le travail préparatoire (découpage en tâches et création du planning prévisionnel) ainsi que pour l'extraction d' informations visuelles sur les modifications effectuées sur le code source via le dépôt GIT.

2.1.3. Un framework de développement puissant : Qt

Le principal outil que nous avons utilisé est le framework (cf lexique) de développement Qt. Qt est un ensemble de bibliothèques développées en C++ (et donc orientées objet) et sous licence LGPL ou propriétaire dont le rôle principal est de permettre le développement d'applications graphiques. Ainsi, Qt implémente un grand nombre de widgets (cf lexique) tels que des boutons, des fenêtres, des labels, des cases à coche etc... et de systèmes permettant leurs interactions sous forme d'événements via le système de signaux et slots : un composant peut émettre un signal qui peut être traité par le slot d'un autre composant.

Cependant Qt ne se limite pas à la création d'interfaces graphiques et offre également de nombreuses fonctionnalités concernant l'accès aux données, la gestion du réseau, la gestion des flux d'exécution (threads etc.), l'analyse XML et JSON (cf lexique), la gestion de contenus multimédia tels que les fichiers audio et vidéo et bien d'autres encore. Ainsi, notre projet s'appuie en grande partie sur la classe « QMediaPlayer » fournie par Qt et qui permet la lecture de fichiers audio et vidéo, ce qui nous a évité de devoir redévelopper les fonctions de bas niveau pour la lecture audio.

Enfin, l'une des caractéristiques importantes de Qt est sa portabilité. Les sources écrites grâce au framework Qt peuvent être compilées aussi bien sur les systèmes Windows, Linux, Mac et Android qui disposent d'un environnement de compilation Qt. Lors de la compilation, l'environnement Qt va traduire le code écrit pour le système d'exploitation sur lequel il est compilé. Nous avons beaucoup utilisé cette particularité qui nous a permis de développer le logiciel d'exploitation des bornes tournant sous linux sur nos ordinateurs personnels sous Windows car nous savions que le code écrit fonctionnerait également sur la borne. Cette technique nous a permis de profiter d'un environnement de développement plus performant (rapidité de compilation et d'exécution sur nos machines) et plus confortable. Il suffisait de tester de temps en temps le code sur la borne pour s'assurer que tout fonctionnait comme prévu. Enfin, cette caractéristique permet également de dissocier l'implémentation du logiciel du système utilisé : si le matériel de la borne et le système venaient à changer, le logiciel écrit resterait compatible avec la nouvelle borne permettant ainsi de ne pas avoir à le réécrire.

2.1.4. Un serveur de test VPS

Pour les tests, nous avons utilisé un serveur central hébergé chez OVH. Le serveur choisi est un VPS (Virtual Private Seerver). C'est un serveur logiciel lancé sur une machine physique hébergeant des dizaines d'autres serveurs virtuels. Le choix d'un VPS nous a offert la possibilité de profiter d'un serveur peu cher avec la puissance adaptée à notre besoin.

2.2. Le choix du matériel pour la borne

Le choix du matériel a constitué une étape importante pour notre projet. En effet, développer une borne de diffusion de Radio Campus ne se limite pas à la création d'un logiciel capable de lire un flux audio, il faut aussi sélectionner avec soin le matériel sur lequel va être déployé ce logiciel.

2.2.1. L'utilisation d'une tablette : le choix le plus approprié ?

Les études préliminaires menées par les étudiants de l'UBP les années passées ont conclu à l'utilisation d'une tablette durcie fonctionnant sous Android. L'avantage de l'utilisation d'une tablette est de permettre de disposer dans un seul et même dispositif de tout le matériel nécessaire à la réalisation de la borne : un écran tactile, une connexion Wi-fi, un micro, une caméra etc. Cependant ces tablettes disposent aussi d'inconvénients. Le premier est le prix : la tablette sélectionnée par les étudiants travaillant sur le projet avant nous était une tablette industrielle coûtant plus de 900 euros. L'avantage de cette tablette était de proposer un écran durci résistant aux chocs et rayures. Le second est le manque de liberté de développement proposé par le système Android : sur un système Android standard, le développeur ne peut pas disposer de l'accès administrateur au système, ce qui peut brider l'implémentation de certaines fonctionnalités. Il n'est par exemple pas possible d'installer un serveur web, un serveur FTP ou un serveur SSH pour la connexion à distance.

2.2.2. Un nano ordinateur à faible coût et aux performances intéressantes : le Raspberry Pi

Après réflexion, nous avons décidé de développer la borne autour d'un Raspberry Pi 2. Le Raspberry Pi 2 est un nano-ordinateur monocarte à processeur ARM ayant la taille d'une carte de crédit et permettant l'exécution de plusieurs variantes du système d'exploitation linux et plus récemment de Windows 10. Il dispose de performances intéressantes grâce à son processeur quad-core ARM Cortex-A7 CPU cadencé à 900MHz, son chipset graphique permettant l'encodage de vidéos full HD ainsi que ses 1GB de RAM. De plus, il dispose de connectiques utiles telles que 4 ports USB, 40 broches GPIO, un port HDMI, un port Ethernet, une sortie audio Jack, une interface permettant le branchement d'une caméra ainsi qu'un écran et un slot pour carte micro SD pour le stockage. Pour terminer, le Raspberry Pi chauffe très peu et consomme peu d'énergie. Ce qui a motivé notre choix de l'achat de ce type de matériel est : son faible coût (35 euros), ses performances intéressantes, la possibilité de travailler sur un système linux et donc de disposer d'une importante liberté de développement ainsi que sa popularité : de nombreux périphériques compatibles directement avec la Raspberry existent.

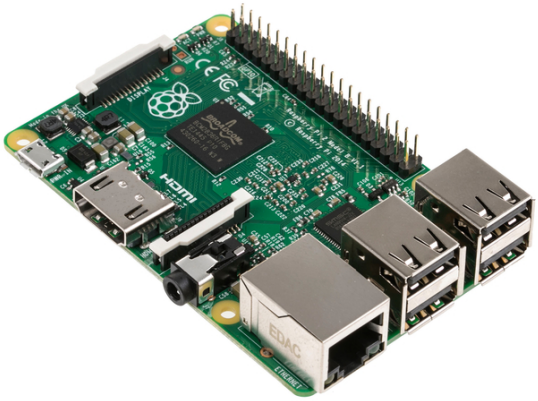

Illustration 6: Raspberry Pi 2

2.2.3. L'affichage de l'interface graphique

La Raspberry Pi fournissait la puissance de calcul mais il restait à gérer l'affichage de l'interface de l'application. Il se trouve que la fondation Raspberry Pi propose à la vente un écran tactile bas coût (55 euros) compatible directement avec la Raspberry Pi. Il s'agit d'un écran 7 " proposant une résolution de 800x480. L'installation de cet écran se fait en quelques étapes simples. Il suffit de connecter la nappe vidéo au connecteur dédié sur le Raspberry Pi ainsi que de connecter l'I2C (cf lexique) à l'écran pour la gestion du tactile. Le Raspberry Pi vient alors se fixer simplement derrière l'écran grâce à des vis. Il nous restait alors à traiter le problème de robustesse de l'écran. La borne étant déployée dans des lieux publics, il existe des risques de chocs et de rayures sur l'écran. Pour lutter contre ces dégradations, il existe déjà de nombreux films protecteurs anti-rayures et anti-chocs pour tous types d'écrans tactiles. Ces films protecteurs peuvent fournir une protection efficace à moindre coût. Nous nous sommes notamment intéressés au film « TANK Protection » qui semble très prometteur mais qui n'est pour le moment disponible que pour iPhone.

Illustration 8: Écran tactile officiel

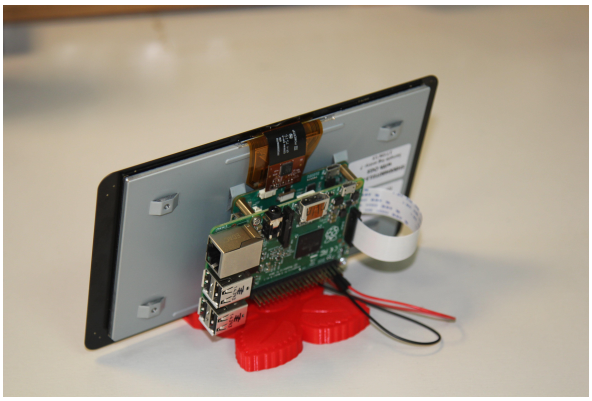

Illustration 7: Raspberry Pi monté sur l'écran tactile

2.2.4. La connexion au Wi-fi

La Raspberry Pi dispose déjà d'un port Ethernet pour permettre une connexion filaire. Cependant, nous souhaitions pouvoir connecter la borne en Wi-fi dans l'hypothèse où une

prise Ethernet ne soit pas disponible dans la zone de déploiement. Pour cela, nous avons commandé un dungle Wi-fi qui se branche sur un port USB et qui fournit une interface de connexion en Wi-fi.

Illustration 9: Dungle Wifi

2.2.5. Le stockage de données et l'installation du système

Pour le stockage des données nous avons sélectionné une carte micro-SD de classe 10 (pour une plus grande vitesse d'écriture) d'une capacité de 8GB. Cette capacité de stockage est suffisante pour permettre l'installation du système d'exploitation « Raspbian », un système basé sur une distribution « Debian » optimisée pour la Raspberry Pi et proposant de nombreux paquets prêts à l'installation tels que Qt dans sa dernière version (5). L'installation du système sur la carte SD est très simple, il suffit de télécharger l'image du système Raspbian et de la copier à la racine de la carte SD. Une fois la carte SD insérée dans la Raspberry Pi, il suffit de la mettre sous tension pour que le système démarre. A terme, la capacité de stockage de 8GB peut être insuffisante au vue du nombre de données à stocker (les podcasts à télécharger en local). Il pourrait être intéressant de dédier cette carte SD pour le système et d'adjoindre à la Raspberry un disque dur externe pour le stockage des podcasts.

2.2.6. Une carte son USB pour une meilleure qualité audio

La Raspberry Pi dispose d'une sortie audio Jack. Cependant, cette sortie audio est d'une qualité très médiocre et insuffisante pour la qualité souhaitée. Il était alors nécessaire d'ajouter une carte son externe pour une plus grande qualité audio. Il existe quelques cartes son compatibles avec la Raspberry Pi telles que la carte Wolfson, la Cirrus Logic ou la HiFiBerry. Cependant leur intégration n'est pas toujours évidente et certaines cartes requièrent la modification du noyau du système Raspbian. Nous nous sommes donc intéressés aux cartes son USB. Les cartes son USB ont un coût inférieur à 10 euros et permettent de brancher une sortie audio ainsi qu'une entrée micro directement reliées à un port USB de la Raspberry. Nous avons réalisé nos tests avec une carte son USB low cost et la qualité audio était plus que convenable. De plus l'intégration de la carte est très simple : celle-ci est directement reconnue par la Raspberry Pi, il suffit alors de modifier sa configuration pour positionner la carte son USB en tant que carte son par défaut.

Illustration 10: Carte son USB utilisée pour les tests

2.2.7. L'alimentation de la Raspberry Pi

L'alimentation de la Raspberry Pi s'effectue via un chargeur micro USB classique de 5V et de 2A. Une alimentation par batterie externe reste également possible.

Illustration 11: Alimentation officielle pour Raspberry Pi (5v et 2A)

2.2.8. Le matériel audio : les enceintes et le micro

Pour les tests de la borne, nous avons utilisé nos enceintes et micro de bureau. Nous n'avons pas sélectionné de matériel audio pour la borne car nous n'avons que peu de compétences en matière d'audio-visuel. La borne est cependant capable de fonctionner avec un large panel de matériel et RCCF saura probablement prendre une meilleure décision technique que nous sur ce point en fonction de la qualité audio souhaitée et du budget disponible.

3. Architecture mise en place

3.1. Une borne connectée

La principale caractéristique de la borne que nous devions réaliser est son caractère connecté. En effet, pour différentes raisons pratiques (facilité de mise en place, coût du matériel, couverture de l'antenne radio, diversité de l'information envoyée etc...), la borne devait être connectée sur le réseau internet de son lieu de déploiement. Ainsi, la borne devait avoir la capacité de récupérer le flux radio directement depuis l'émission du flux direct en se branchant sur l'application de broadcast (icecast) de radio campus. De plus, cette connexion au réseau nous permettait de répondre à un certain nombre de points du cahier des charges. Nous pouvons citer les besoins :

- De pilotage des bornes à distance (configuration, gestion de profil, redémarrage)
- De diffusion de messages textuels informatifs sur les bornes
- De permettre aux auditeurs de laisser un message vocal à la radio

Dès lors, nous nous sommes penchés sur trois architectures réseaux possibles pour connecter ces bornes. Nous allons en présenter les avantages et inconvénients.

3.1.1. Une borne complètement autonome connectée à un réseau GSM

L'une des pistes étudiées était de relier toutes les bornes à internet en 4G via GSM (cf lexique). Le gros avantage de cette stratégie est d'être indépendant du réseau de la zone de déploiement. En effet, le GSM peut être capté partout dans les zones urbaines. Les bornes auraient ainsi pu être déployées aussi bien en intérieur qu'en extérieur sans avoir besoin de faire des démarches administratives et techniques pour les intégrer dans les réseaux privés de l'université ou de ses écoles. De plus, grâce à cette technique, on peut s'affranchir des limitations de sécurité imposées sur ces réseaux. En effet, de nombreux services (ports) sont verrouillés sur les réseaux de l'ISIMA et de l'Université pour des raisons de sécurité et le service de diffusion du direct de radio campus (port 8000) fait partie de cette longue liste. Les bornes auraient alors été connectées à un serveur central appartenant à Radio Campus pour permettre leur administration et leur configuration. Chaque borne aurait alors pu récupérer directement le flux radio ainsi que les podcasts depuis les services de Radio Campus selon le schéma suivant :

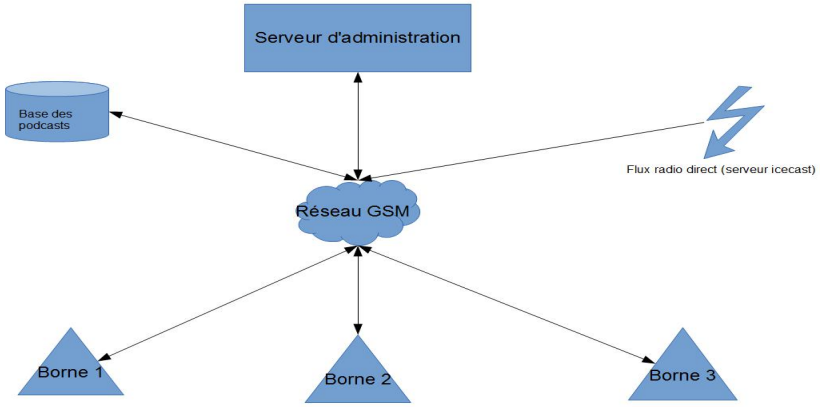

Illustration 12: Schéma illustrant l'architecture réseau via le GSM

Grâce à cette architecture, si le serveur d'administration, le flux direct ou la base des podcasts venaient à tomber en panne, la borne continuerait son fonctionnement en mode dégradé (podcasts et administration disponibles mais pas le direct en cas de panne de icecast par exemple).

Pour terminer, pour exploiter pleinement cette solution, il aurait fallu doter les bornes de batteries ainsi que d'un boîtier résistant aux intempéries. L'inconvénient majeur qui nous a dissuadé d'utiliser cette solution est le coût prohibitif d'un tel dispositif au vue des quantités importantes de données qui transitent. En effet, il aurait fallu équiper toutes les bornes de cartes SIM disposant d'un accès au réseau GSM. Bien qu'il existe des abonnements à moins de 10 euros, la quantité de données proposée par ces types d'abonnement est très insuffisante. Pour profiter d'un trafic convenable il faut débourser au minimum 20 euros par mois, ce qui reste envisageable pour une borne, beaucoup moins pour 10.

3.1.2. Une borne sur le réseau privé de l'université accessible depuis l'extérieur

Une autre architecture envisagée était de connecter les bornes au réseau de l'université. Chaque borne aurait alors été autonome. Pour régler le problème d'accès aux services de radio campus (bloqué sur les réseaux de déploiement), nous souhaitions utiliser des tunnels SSH. Le principe de ces tunnels SSH est de se connecter à un serveur distant situé en dehors du réseau de l'université via une connexion cryptée et passant par un port non verrouillé (le 9000 par exemple). Le principe est alors de demander au serveur central situé en dehors du réseau privé (et donc des limitations instaurées) de réaliser la requête à notre place sur les services de Radio Campus et de nous retourner le résultat via ce canal crypté. Toutes les requêtes passent alors par un serveur central qui sert de relais selon le schéma suivant :

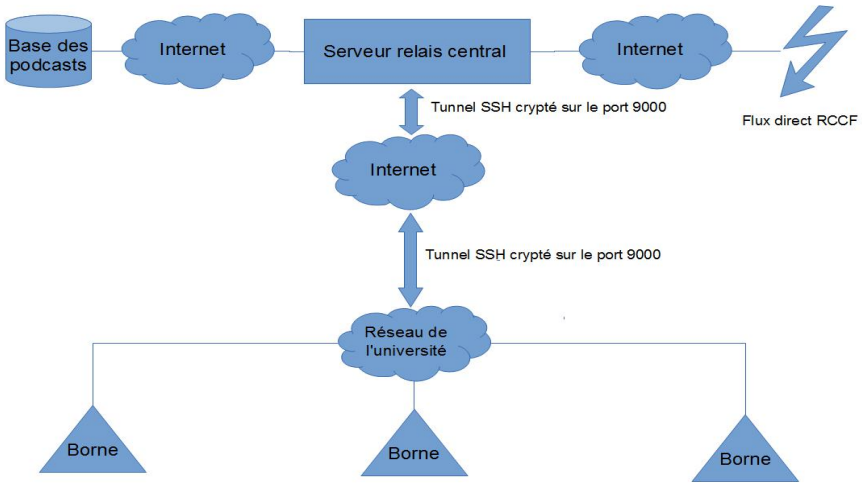

Illustration 13: Schéma présentant l'architecture réseau des bornes organisé autour d'un tunnel SSH

En ce qui concerne l'administration, notre idée était d'exploiter ce tunnel SSH dans le sens contraire et utiliser une technique appelée le « Reverse SSH Tunneling » pour pouvoir se connecter aux bornes depuis l'extérieur. En effet, sur le réseau local de l'université, les bornes n'auraient que des adresses IP locales et ne pourraient donc pas être accessibles depuis l'extérieur. Le principe du reverse SSH est le suivant : la borne se connecte au serveur central situé sur internet via une connexion cryptée sur le port 9000 et lui demande de rediriger toutes les requêtes qu'il reçoit sur un certain port via le tunnel SSH qui a été créé lors de la connexion de la borne.

Cette technique a l'avantage d'être simple à mettre en place et reste peu coûteuse. De plus, permettre de se connecter aux bornes directement depuis internet en SSH offre les meilleures possibilités en terme d'administration en rendant possible un contrôle total des bornes sur lesquelles on est connecté.

Les inconvénients sont les suivants : Si le serveur central tombe en panne, plus aucun service ne fonctionne car toutes les requêtes y transitent. De plus, le reverse SSH pose des problèmes légaux et peut constituer une faille de sécurité : on permet à des personnes extérieures au réseau de l'université de se connecter depuis internet à des machines situées dans le réseau. Un pirate pourrait alors facilement tirer parti de certaines failles sur les bornes pour s'introduire dans le réseau sensé être protégé.

3.1.3. Une borne sur le réseau privé de l'université non accessible depuis l'extérieur

La solution retenue est similaire à la solution proposée précédemment sans l'utilisation de reverse SSH. L'architecture réseau est alors strictement la même : des bornes se connectent via un tunnel SSH à un serveur central sur internet tenu par Radio Campus. C'est ce serveur central qui fait les requêtes nécessaires aux services de Radio Campus et qui retourne le résultat aux bornes via le lien SSH. La seule différence réside dans la méthode d'administration. Dans la solution précédente, on permettait aux utilisateurs de se connecter aux bornes depuis l'extérieur. Dans cette solution, nous pensions développer une

interface d'administration web déployée sur le serveur central (et donc accessible depuis internet) permettant de fournir les principales fonctionnalités d'administration des bornes. L'utilisateur peut ainsi entrer les configurations nécessaires via l'application web. Ces configurations sont alors stockées dans une base de données et les bornes n'ont plus qu'à se connecter à ces bases pour les récupérer.

L'intérêt de cette méthode est de permettre un plus grand niveau de sécurité en n'ouvrant pas de porte des réseaux privés vers l'extérieur.

Le principal inconvénient réside dans le manque de liberté dans les actions d'administration (le contrôle n'est pas total comme dans la version précédente). De plus, l'inconvénient de continuité de service persiste : si le serveur central tombe, aucun service ne fonctionne.

3.1.4. Conclusion sur l'intégration du réseau dans les bornes

Pour conclure sur l'intégration du réseau dans les bornes, il a été décidé dans un premier temps de les connecter sur les réseaux privés des écoles des lieux de déploiement. Il est également prévu que la connexion puisse se faire via un lien filaire Ethernet ainsi qu'un lie Wi-fi. La Raspberry possédant déjà un port Ethernet, il suffit de rajouter un dungle Wi-fi pour permettre facilement sa connexion. Pour les tests, un VPS (cf lexique) a été loué chez OVH (cf lexique) pour servir de serveur central. A terme, Radio Campus devra investir dans la location d'une telle machine ou permettre à un de leur serveur d'assurer ce rôle. Enfin, il convient de préciser qu'il reste des questions administratives à résoudre car la solution du tunnel SSH est temporaire, Radio Campus devra obligatoirement entamer des démarches auprès de l'université ou des écoles pour faire ouvrir les services nécessaires au fonctionnement des bornes.

3.2. La conception de l'architecture logicielle

3.2.1. Utilisation de l'architecture MVC

Pour des raisons de simplicité de développement et de maintenabilité, il a été décidé d'organiser les modules de l'application autour du modèle d'architecture MVC (Modèle Vue Contrôleur). Résumons le principe de cette architecture. Les modules de l'application sont découpés en trois composants essentiels : le modèle, la vue et le contrôleur. Le rôle de la vue est de réaliser la présentation des données à l'utilisateur. Celui du modèle est de détenir les données de l'application et de les mettre à jour ou les servir. Enfin le contrôleur est le composant qui sert d'intermédiaire entre la vue et le modèle. C'est lui qui fait le lien entre l'action utilisateur demandée dans la vue et les opérations à réaliser sur le modèle. Il permet donc de sécuriser les demandes de modification du modèle et porte la logique de l'application en réagissant aux entrées utilisateur et en les traduisant en un comportement à adopter par l'application. Ce fonctionnement est résumé sur le schéma suivant :

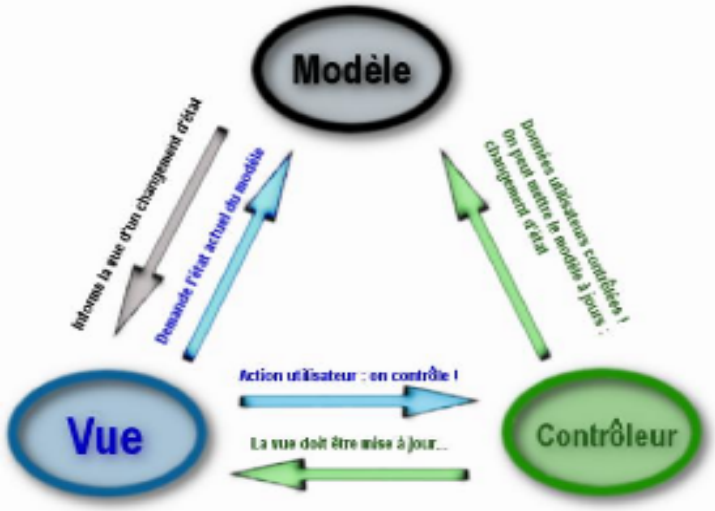

Illustration 14: Schéma de l'architecture MVC

3.2.2. Architecture globale sous la forme de modules

L'ensemble de l'application a été conçue comme un noyau capable de recevoir des modules implémentant une fonctionnalité bien définie. Les modules doivent respecter l'architecture MVC pour pouvoir être intégrés. Ainsi, pour chaque module, on devra implémenter une vue (interface utilisateur), un contrôleur et un modèle contenant les données. Le noyau de l'application peut être représenté par le diagramme UML suivant :

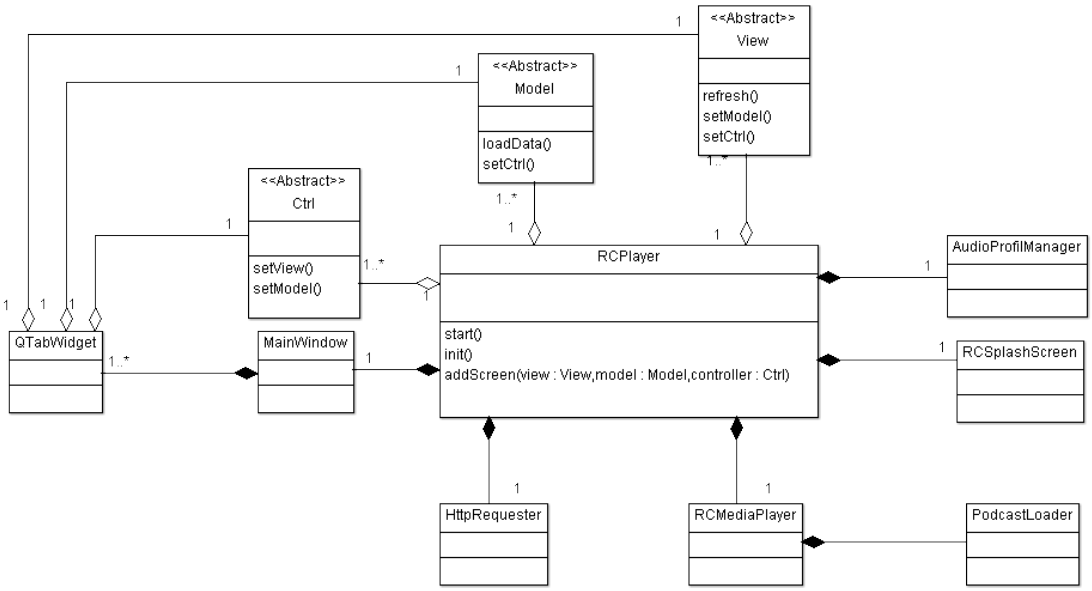

Illustration 15: Diagramme de classe de l'architecture du noyau de l'application

L'application peut alors être résumée par ce diagramme. Le composant principal est « RCPlayer » qui correspond à l'application même. Cette classe dispose des méthodes init() et start() qui permettent respectivement d'initialiser et de démarrer l'application. Elle est implémentée sous forme d'un singleton (cf lexique) de manière à être accessible de n'importe où dans le programme. Elle possède une instance de la classe « MainWindow » qui correspond à la fenêtre principale du programme. La « MainWindow » contient une série de « QTabWidget » qui sont des onglets.

Chaque onglet est destiné à accueillir un module de l'application et contient donc un modèle, une vue et un contrôleur (Model, View, Ctrl). L'ajout d'un nouvel onglet se fait dans la méthode init() de « RCPlayer » via la méthode privée « addScreen ()». Enfin, « RCPlayer » contient également 4 classes que l'on pourrait qualifier de services. « HttpRequester » qui permet de lancer des requêtes HTTP (cf lexique) (pour récupérer des données sur internet), « RCMediaPlayer » qui permet de gérer le lecteur audio (play, pause, stop, contrôle du volume etc...), « RCSplashScreen » permettant d'afficher un écran fixe en cas d'indisponibilité de la borne et enfin « AudioProfilManager » qui permet de gérer automatiquement le profil audio courant (disponible / indisponible, volume selon l'heure).

Ces services sont accessibles partout dans l'application car « RCPlayer » est un singleton et fournit des méthodes publics pour y accéder. On pourra ainsi commander la lecture via « RCMediaPlayer » depuis n'importe quel module de l'application par exemple. Il existe un autre service relié à « RCMediaPlayer ». Il s'agit de « PodcastLoader » dont le rôle est de récupérer et charger les podcasts de Radio Campus dans le lecteur audio. Les classes abstraites « Model » « View » et « Ctrl » servent de classe mère aux modèles, vues et contrôleurs des modules en leur fournissant des méthodes de base telles que « refresh() » qui permet de rafraîchir la vue et « loadData() » pour le modèle qui permet de charger les données du modèle.

A ce stade, voici à quoi ressemble l'application sans module contenant trois onglets « Ecoute », « Playlists » et «Enregistrer un message ».

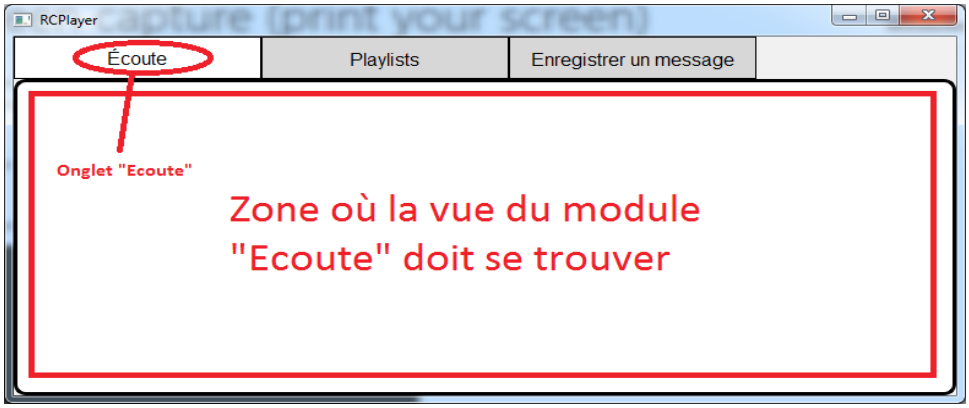

Illustration 16: Fenêtre vide illustrant la répartition des modules en onglets

4. La réalisation des modules de l'application

4.1. L'écoute du direct

4.1.1. Présentation des fonctionnalités

La principale fonctionnalité que devait supporter l'application était la lecture en direct du flux audio de radio campus ainsi que des podcasts que la radio propose (émissions enregistrées). Cette fonctionnalité a comme les autres été implémentée sous forme de module à ajouter à la fenêtre principale.Ainsi, pour ce module on retrouve les classes « PlayerModel », « PlayerView » et « PlayerCtrl ». Voici à quoi ressemble l'interface liée au module de lecture de contenu audio :

Cette vue se découpe en trois parties bien distinctes. La première est le bandeau d'en-tête contenant la date et l'heure courante. Elle fournit un repère temporel à l'utilisateur. La seconde correspond à la partie centrale de l'interface qui regroupe les informations et les contrôles sur la lecture en cours. Ainsi, sur la partie gauche on retrouve les informations suivantes : La date de lecture courante, le mode de lecture (qui peut être « En direct », « Lecture de podcast » ou encore « Arrêté »), le titre en cours de lecture, l'auteur et la position temporelle courante. Sur la partie droite la jaquette associée au podcast en cours de lecture (ou une image par défaut comme ici en cas d'écoute du direct) est affichée. Enfin, toujours dans ce panneau central, dans la partie basse on a les contrôles associés à la lecture : « Play », « Pause » et « Stop » ainsi qu'une barre de défilement permettant de se déplacer temporellement dans la lecture en cours dans le cas d'une émission. Enfin, dans la partie basse, on retrouve une ligne temporelle qui représente pour chaque jour et chaque heure les émissions disponibles (affichées en noir). Des flèches sur l'interface permettent de naviguer vers un jour précédent ou un jour suivant.

Cette interface est très inspirée de celle existante sur le lecteur en ligne de Radio

Campus. L'objectif est de fournir à l'utilisateur un accès chronologique au contenu diffusé par radio campus : a tout moment celui-ci sait quelle est la date courante, la date et l'heure de lecture et la date à laquelle il se trouve sur la ligne temporelle. Il est possible à tout moment de revenir au direct par simple clique sur le bouton « Revenir au direct » dans la partie centrale de l'interface. Pour lancer un podcast, il suffit à l'utilisateur de cliquer sur un des liens en noir sur la ligne temporelle. Il peut alors visualiser la jaquette associée ainsi que lancer la lecture, mettre en pause, arrêter ou avancer dans la lecture.

4.1.2. Architecture du module

Cette vue plutôt complexe par la diversité de missions, a été découpée en trois widgets (cf lexique) empilés de manière verticale dans un layout (cf lexique). Chaque widget va gérer l'affichage d'un élément de la vue. Ainsi, on aura « HeaderWidget » qui va gérer l'affichage du bandeau d'en-tête avec la date et l'heure courante, « PlayerWidget » qui va afficher les informations sur la lecture en cours et «Timeline » qui va permettre la génération de la ligne temporelle. « Timeline » va contenir 19 widgets « RCLink » qui correspondent aux liens cliquables dans la ligne temporelle qui peuvent s'activer ou se désactiver en fonction de la disponibilité du podcast pour l'horaire concerné. Le schéma UML représentatif de ce module est alors le suivant :

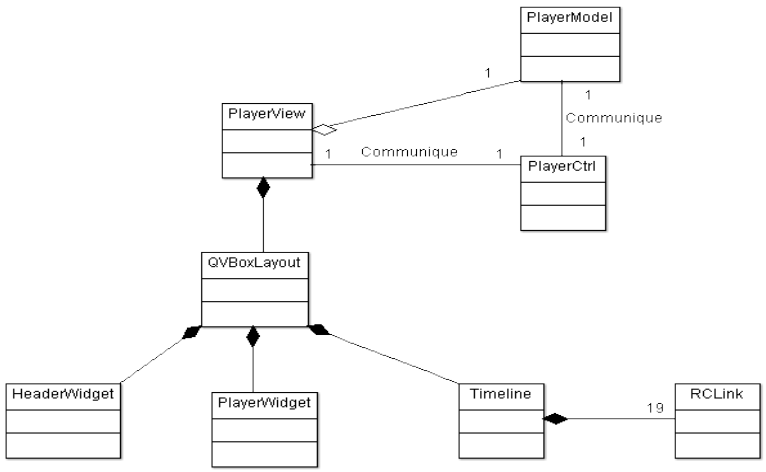

Illustration 18: Diagramme de classe de l'organisation du module d'écoute

4.1.3. Détails techniques

Après cette présentation du fonctionnement du module, nous allons rentrer dans les détails techniques et expliquer comment nous parvenons à récupérer le direct. Comme nous l'avons vu dans le chapitre 1.6, Radio Campus possède un serveur sur lequel est installé le serveur de broadcast (cf lexique) Icecast2 (cf lexique). Il suffit de se connecter au serveur pour récupérer le flux audio directement. Ainsi, ce flux peut être récupéré en se connectant à

l'URL suivante : http://campus.abeille.com:8000/campus. C'est ce flux qui est utilisé par le lecteur en ligne de Radio Campus pour lire le direct. Sur Qt, il existe la classe « QMediaPlayer » capable de gérer la lecture de fichiers multimédias (audio et vidéo) en local mais aussi depuis un flux multimédia en ligne. Pour demander à cette classe de lancer le flux audio de Radio Campus, il suffit alors d'écrire le code suivant :

```
QMediaPlayer player;
player.setMedia(QUrl(« http://campus.abeille.com:8000/campus »));
player.setVolume(50) ;
player.play() ;
```

Comme mentionné dans le chapitre intitulé « Architecture globale du programme », c'est la classe « RCMediaPlayer » qui va se charger de cette mission en héritant de la classe Qt « QMediaPlayer ». Le cheminement de la communication entre les objets pour lancer le direct depuis l'interface présentée précédemment est donc le suivant : L'utilisateur clique sur le bouton « Revenir au direct » de l'interface. L'appui sur ce bouton va alors déclencher la méthode playerDirect() de la classe « PlayerCtrl ». Le contrôleur via cette méthode va alors mettre à jour la date de lecture courante sur l'interface et faire un appel à la méthode playDirect() de « RCMediaPlayer » de la manière suivante :

```
RCPlayer::getInstance()->getPlayer().playDirect();
```

Cette méthode va avoir pour effet de passer le lecteur en mode de lecture « Direct », de charger le flux audio et de lancer la lecture. L'interface sera mise à jour automatiquement toutes les secondes par « PlayerCtrl » grâce à l'appel de la méthode refresh() de « PlayerView ».

Outre le fait de pouvoir lancer la lecture du direct, l'application aura besoin d'un certain nombre d'informations sur celui-ci. Il se trouve que Radio Campus dispose du service web : http://www.campus-clermont.net/ws/?req=onair qui permet de récupérer des informations sur le titre diffusé en direct. Voici un exemple de retour de ce service web au format JSON (cf lexique) :

```
{

    "type":"emission",
    "titre":"Live in room",
    "auteur":"",
    "podcastable":true,
    "begin":"2016-02-11 19:00:00",
    "url":"\/node\/50"

}
```

Ainsi, la classe « RCMediaPlayer » dispose de la méthode getDirectMusic() qui permet d'extraire les informations utiles de ce service web. Pour cela, getDirectMusic() va utiliser le service « HttpRequester » pour lancer une requête HTTP sur le service web

http://www.campus-clermont.net/ws/?req=onair. Une fois la réponse récupérée au format JSON (cf lexique) depuis le serveur, celle-ci est analysée et un objet « Music » contenant les données intéressantes est créé et retourné à la vue qui peut alors mettre à jour ses informations.

4.2. La gestion des podcasts

La gestion des podcasts est la partie du projet qui nous a occupé le plus de temps car elle sous entendait de nombreuses réalisations techniques. Comme expliqué dans le chapitre 1.6, un podcast pour RCCF est un fichier audio au format MP3 d'une durée de 1h déposé sur le serveur web de RCCF. Ces fichiers sont automatiquement créés toutes les heures. Les administrateurs de Radio Campus peuvent alors choisir dans la liste de ces fichiers ceux qui constituent des podcasts utiles à diffuser.

Concrètement, les fichiers sont stockés dans deux dossiers distincts : « OK » pour les fichiers considérés comme pouvant être diffusés en tant que podcast et « KO » pour les autres. Après ce premier classement, les fichiers sont stockés dans des dossiers contenant la date de diffusion et chaque fichier contient également dans son nom la date et l'heure de diffusion. En général, les fichiers audio considérés comme des podcasts sont ceux correspondant à des plages horaires d'émissions de Radio Campus.

Avec la conception de la borne, l'idée était que les utilisateurs puissent avoir accès à l'ensemble du contenu des émissions de Radio Campus. Or ces contenus représentent des volumes considérables de données et changent constamment (nouveaux ajouts, retraits d'émissions, mises à jour d'émissions etc...). Il fallait alors trouver un moyen de les gérer efficacement sur les bornes. La contrainte que nous avions était de rendre ces podcasts disponibles aux auditeurs même lorsque la borne était hors ligne, ce qui supposait une récupération en local des podcasts ainsi qu'une synchronisation du contenu local aux bornes avec les émissions disponibles en ligne. Les problématiques à résoudre étaient donc :

- Comment stocker en local les podcasts et accéder rapidement à leur contenu ?
- Comment récupérer et stocker les informations liées aux podcasts ?
- Comment gérer les jaquettes illustrant les podcasts ?
- Comment synchroniser ces podcasts (récupération de nouveaux podcasts, mises à jour ou suppression de podcasts existants etc...).
- Quelle politique de synchronisation adopter (juste à temps, synchronisation complète etc...).

Pour répondre à ces problématiques, nous avons découpé notre solution technique en plusieurs composantes.

4.2.1. Un service de requêtes HTTP

Nous l'avons déjà mentionné, les podcasts ainsi que les jaquettes et les informations sur ces podcasts sont stockés sur un serveur distant appartenant à RCCF. Il nous fallait donc un moyen de communiquer avec ce serveur pour récupérer les informations utiles au bon moment. RCCF nous a fourni les outils pour réaliser cette communication. Ces outils prennent la forme de web-services qu'il suffit d'interroger via le protocole HTTP (cf lexique) pour récupérer les informations souhaitées. Ainsi, nous disposions du web-service suivant : http://www.campus-clermont.net/onair/podcast/player/ws/?date= + date au format yyyy-MM-

dd (par exemple 2016-02-10). Nous verrons le contenu de ce service plus tard. Son rôle est de retourner la liste de tous les podcasts diffusés pour chaque heure à une date donnée ainsi que des informations les concernant. Enfin, des transferts de fichiers sont possibles via le protocole HTTP.

Nous avons donc implémenté un service dans notre application sous forme de la classe « HttpRequester ». Cette classe ne dispose que d'une seule méthode qui est sendHttpRequest() et qui prend en paramètre la requête à effectuer (une URL de web-service dans notre cas) ainsi qu'une booléen pour savoir si un système de cache doit être utilisé ou non. Cette méthode retourne un booléen pour savoir si la requête a abouti ou non (serveur distant indisponible, borne déconnectée d'internet etc...) ainsi que le résultat sous forme d'un tableau d'octets. Cette classe a donc un rôle bas niveau servant d'abstraction au protocole HTTP. Elle retourne des données brutes sans se soucier de leur contenu.

Le système de cache implémenté dans cette classe permet de répondre à des questions de performance. En effet, le protocole HTTP est un protocole lent, or notre application devra faire de très nombreux appels à ce protocole pour communiquer avec les services de RCCF. Le système de cache est alors une simple liste enregistrant les requêtes effectuées par le passé, leur résultat et leur date. Si l'utilisateur demande l'utilisation du cache, lorsqu'une requête HTTP est réalisée, la méthode va dans un premier temps vérifier que le résultat de la requête n'est pas déjà présent dans la liste du cache. Si elle est présente, l'application retourne immédiatement cette réponse. Si elle ne l'est pas, la requête HTTP est lancée. Ce cache est vidé de manière automatique à une fréquence réglable en tenant compte de la date de péremption de chaque requête s'y trouvant.

4.2.2. Un service de synchronisation

Il était donc nécessaire de stocker en local sur les bornes les podcasts disponibles. Nous avons choisi de stocker les podcasts dans des dossiers nommés avec leur date de diffusion, par exemple le dossier 2016-02-10 contiendra tous les podcasts diffusés le 10/02/2016. Dans ces dossiers se trouvent les fichiers MP3 qui correspondent aux podcasts ainsi qu'un fichier au format JSON (cf lexique) appelé « content.json » stockant toutes les informations sur tous les podcasts présents dans le dossier (titre ; auteur, nom du fichier de l'image de la jaquette etc...). Les images correspondant aux jaquettes illustrant les podcasts sont quant à elles stockées dans une dossier à part. Le service de synchronisation va donc avoir pour rôle de maintenir à jour tous ces dossiers ainsi que les fichiers JSON.

Pour invoquer le service de synchronisation (classe « PodcastSynchronizer »), il faut faire appel à la méthode syncDirectory() en spécifiant en paramètre le nom du dossier à synchroniser. Le dossier à traiter est alors placé dans une file d'attente. Si le service de synchronisation était inactif, il est lancé sous forme de thread (cf lexique). Lorsque le service est lancé, il va prendre les dossiers à synchroniser par ordre d'arrivée dans la file. Pour chaque dossier, il va commencer par lire leur contenu en local. Le fichier JSON présent dans le dossier listant déjà son contenu, il suffit d'en extraire les informations utiles directement. Une fois le contenu local analysé, le service va utiliser le service « HttpRequester » pour effectuer une requête sur le serveur de RCCF pour récupérer la liste des podcasts disponibles pour la date spécifiée (appel au web-service mentionné dans le chapitre précédent). Voici un exemple de ce que peut retourner le web-service pour un jour donné à une heure

définie:

```
"19":{
    "mp3":"2016-02-10-1900.mp3",
    "time":19,
    "title":"Court mais Trash !",
    "ok":true,
    "paulo_entries":null,
    "duration":1,
    "filetime": "2016-02-10 06:03:02",
    "shortTitle":null,
    "future":false,
    "url":"",
    "podcastable":true,
    "image":"\/onair\/podcast\/player\/images\/fond-vert.png",
    "ecoutes":[
        "9",
        "2"
    ]
}
```

Ceci est un extrait de code JSON décrivant le contenu disponible à la date du 10/02/2016 à 19h. Le fichier JSON retourné par le web-service contient ce type d'information pour chaque heure de la journée. Le service de synchronisation va transformer les données brutes reçues du service « HttpRequester » en fichier JSON de ce type puis l'analyser pour en extraire les données importantes comme « mp3 » correspondant au nom du fichier audio, «title» pour le titre, « podcastable » définissant si le fichier est un podcast ou non, « image » contenant une url vers la jaquette illustrant le contenu audio, « filetime » définissant la date de dernière modification du fichier ainsi que « paulo_entries » qui contiendra la liste des titres que contient le podcast dans les cas des émissions « 100 % ». Le service de synchronisation ne va garder que les fichiers audio correspondant à des podcasts (champs « podcastable » à vrai). Il va alors comparer la liste des podcasts en local avec la liste des podcasts récupérée via le web-service de RCCF et appliquer l'algorithme illustré par la figure suivante :

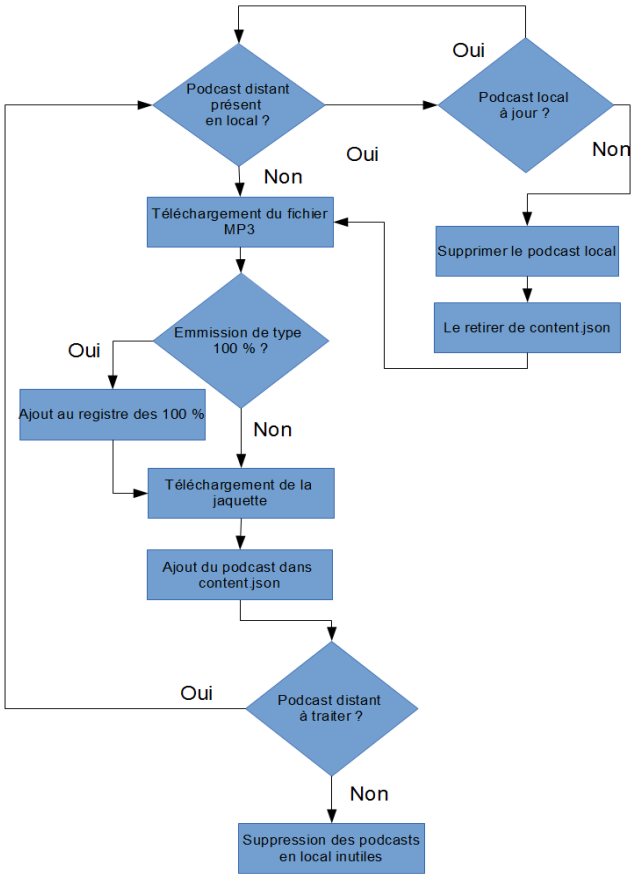

Illustration 19: Algorithme de synchronisation des podcasts

Pour résumer cet algorithme, le service de synchronisation va prendre la liste de tous les podcasts distants qu'il a récupéré en ligne et va la comparer avec la liste des podcasts en local. Si le podcast distant existe en local, et si il est déjà synchronisé (comparaison de la date de dernière modification), alors il ne se passe rien. Sinon, le podcast en local est supprimé puis re-téléchargé. Si le podcast distant n'existe pas en local, le fichier MP3 est téléchargé. Un test est ensuite effectué pour savoir s'il s'agit d'un podcast de type 100 % (type d'émission particulier correspondant à des playlists). Si c'est le cas, il est ajouté dans un fichier JSON particulier. L'utilité de ce fichier JSON sera explicité plus tard. Dans tous les cas, la jaquette du podcast est téléchargée si elle n'existe pas déjà dans le dossier des images et le podcast ainsi que ses informations sont ajoutés dans le fichier « content.json » du dossier en cours de synchronisation. Lorsque tous les podcasts récupérés en ligne ont été traités, le service de synchronisation supprime tous les podcasts locaux qui n'ont pas été mentionnés dans la liste des podcasts disponibles récupérée grâce au service web (cf lexique). Une fois toutes ces étapes effectuées, le dossier est considéré comme synchronisé et il est retiré de la liste des dossiers à synchroniser.

Ce service est appelé de manière automatique par l'application à chaque fois que celle-ci se trouve dans l'état « indisponible ». La politique de synchronisation que nous avons adopté est une politique de synchronisation souple : la borne synchronise ses podcasts quand elle en a l'occasion. Nous avons remarqué que la réalisation d'une synchronisation

entraînait des baisses de performances significatives sur la borne ce qui est susceptible de gêner les utilisateurs. Pour éviter ce phénomène la borne ne va déclencher sa synchronisation que dans les moments où elle est indisponible pour l'utilisateur. Plus concrètement, lorsque la borne passe en « indisponible », le service de synchronisation se lance et va chercher à synchroniser tous les jours des X derniers mois à partir de la date courante. Dès que la borne retourne dans un état « disponible », la synchronisation en cours se termine mais aucune autre synchronisation n'est lancée. Cette synchronisation dite souple est permise dans notre cas la synchronisation n'est utile qu'en cas de plantage réseau ce qui doit constituer un cas exceptionnel. L'utilisateur pourra donc tolérer de ne pas disposer des podcasts les plus à jours lors de ces phases de fonctionnement dégradées.

4.2.3. Un service de chargement de podcasts

Nous avons vu comment nous pouvions entretenir une liste de podcasts à jour en local sur les bornes. La problématique suivante est de pouvoir servir ces podcasts à la demande de l'utilisateur. En effet, il se peut que les podcasts demandés ne soient pas présents en local car non synchronisés. L'application doit alors quand même être capable de lancer les podcasts en ligne sans les avoir téléchargés. On pourra donc être amené à servir à l'utilisateur aussi bien des podcasts présents en local que des podcasts en ligne. Le rôle du service de chargement de podcast appelé «PodcastLoader » est de faire abstraction de la source du podcast pour les charger et les mettre à disposition de l'application.

La requête de chargement de nouveaux podcasts se fait via la ligne temporelle présentée sur l'illustration 20.

Illustration 20: Widget de la ligne temporelle extraite de l'interface du module de lecture

Sur cette illustration, la ligne temporelle se trouve sur le lundi 01 février 2016. Lors d'un clic sur les flèches de gauche ou de droite pour changer de jour courant, tous les podcasts concernant ce jour seront chargés. Plus concrètement, un appel au service « PodcastLoader » sera effectué via la méthode loadPodcasts(). Cette méthode prend une date en paramètre. Elle commence par récupérer les podcasts en local disponibles pour la date demandée. Ensuite, elle teste le réseau pour savoir si la borne est en ligne ou non. Si elle n'est pas en ligne elle va se contenter de charger et retourner les podcasts locaux. Si le réseau est accessible, le service récupère la liste des podcasts disponibles via le web-service de RCCF. Ensuite, pour chaque podcast en local, l'algorithme va tester s'il est à jour. Si il est à jour, c'est ce podcast qui sera chargé. Sinon c'est le podcast en ligne situé sur le serveur de RCCF qui est chargé.

Lorsque nous parlons de « chargement de podcast », nous parlons de création d'un objet « Podcast » à partir d'informations le décrivant, présentes dans le fichier content.json pour les podcasts en local et retournées par le web-service de RCCF pour les podcasts

distants. Voici le diagramme UML représentant la classe « Podcast » :

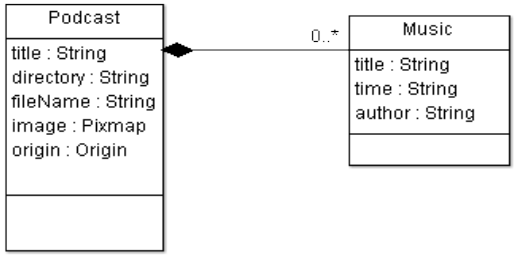

Illustration 21: Diagramme de classe présentant le modèle de podcast

Un podcast est donc un objet comportant un titre, un dossier de stockage, un nom de fichier (fichier audio), une image pour la jaquette ainsi qu'une origine. L'origine est une information très importante, c'est un flag qui permet de savoir si le podcast chargé peut être trouvé en local ou en ligne (information renseignée par « PodcastLoader »). Un podcast peut également contenir une liste de musiques dans le cas des émissions 100 % qui sont des playlists. Une musique va être caractérisée par des informations de base telles que le titre, l'auteur et l'heure de diffusion du titre dans le podcast. Tous les podcasts chargés sont stockés dans le lecteur de l'application prêts à être lancés.

A chaque fois qu'un podcast a été chargé, « PodcastLoader » va émettre un événement pour prévenir l'application. Un nouveau lien correspondant à l'heure du podcast chargé va donc s'activer sur l'interface de la ligne temporelle (cf Illustration 20) et l'utilisateur pourra cliquer dessus pour le lancer.

4.2.4. La lecture des podcasts par le lecteur

Lorsqu'un podcast est sélectionné via la ligne temporelle, un événement est envoyé au lecteur pour qu'il lance la lecture du podcast choisi via la méthode playPodcast(). Cette méthode aura pour effet de placer le lecteur en mode de lecture de podcast. Il va ensuite identifier le podcast à lancer dans la liste des podcasts chargés et vérifier son origine. Si l'origine est locale, la méthode setMedia() sera appelée avec un chemin vers le fichier en local sur la borne. Si l'origine est en ligne, la méthode setMedia() sera toujours appelée mais avec une URL pointant sur le fichier MP3 en ligne sur le serveur de RCCF. La vue de l'interface se mettra alors automatiquement à jour avec les informations extraites du podcast en cours de lecture.

4.3. La mise en valeur des 100 %

4.3.1. Présentation générale du module

Les 100 % sont des podcasts diffusés par RCCF. Ce sont cependant des podcasts un peu particuliers car ils sont constitués uniquement de musiques du même genre musical. On a par exemple les 100 % jazz, 100 % rock, 100% métal etc... Le souhait de RCCF était de mettre en avant ce type de contenu sur l'interface pour permettre d'accéder rapidement à la liste des 100 % disponibles. C'est le module « Playlists » qui va se charger de cette mission.

La première interface du module « Playlist » est la suivante :

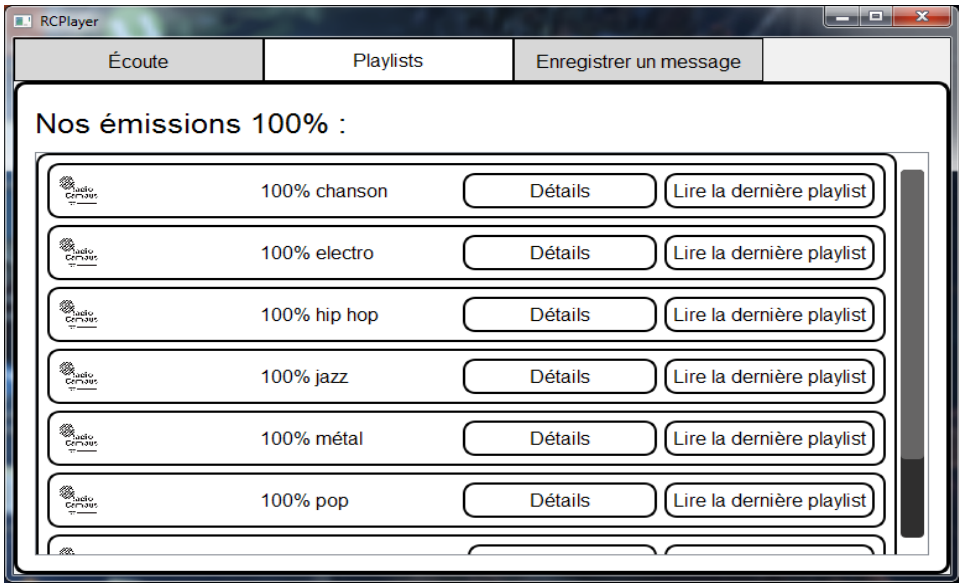

Illustration 22: Interface présentant la liste des 100% disponibles sur la borne

Cette interface permet l'affichage sous forme de liste de toutes les émissions 100 % disponibles sur la borne. Pour chaque émission 100 %, il y a deux boutons d'actions disponibles : le bouton « Lire la dernière playlist » qui va permettre à l'utilisateur de lancer automatiquement la dernière playlist de l'émission sélectionnée et le bouton « Détails » qui va amener l'utilisateur sur l'interface suivante :

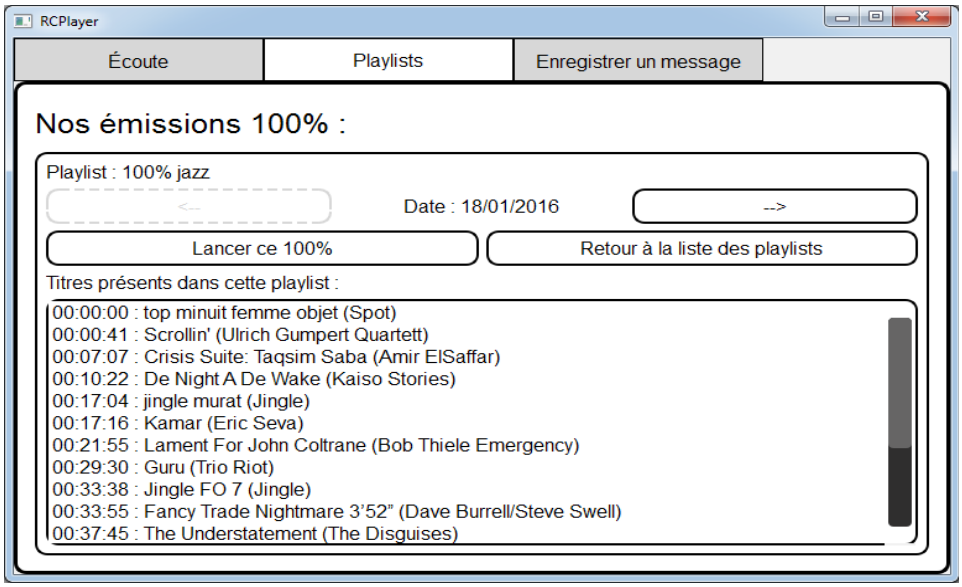

Illustration 23: Interface présentant les détails associés à un type de 100%

Le rôle de cette interface est de donner plus d'informations sur les playlists qui correspondent à la catégorie de 100 % sélectionnée. Ainsi, dans la partie supérieure, on retrouve la date de la playlist courante et des flèches orientées à gauche et à droite permettent d'avoir des détails sur la playlist correspondant à la date directement inférieure ou directement supérieure. Deux boutons d'action permettent de lancer la lecture de l'émission 100 % courante ou de retourner sur l'interface affichant la liste de tous les 100 % disponibles.

4.3.2. Considérations techniques

Nous avons vu dans l'algorithme présenté dans l'illustration 19 Que les podcasts correspondants à des 100 % étaient enregistrés dans un fichier registre. Ce fichier registre au format JSON (cf lexique) porte le nom de « 100%_registery.json ». Il permet d'enregistrer, pour chaque type d'émission 100 % qui est passée par le service de synchronisation de la borne, la liste des dates auxquelles une émission est disponible. Voici le type de contenu de fichier que l'on pourrait obtenir :

```
{
    "100% chanson": [
        "2016-01-28"
    ],
    "100% jazz": [
        "2016-01-18",
        "2016-02-01"
    ]
}
```

Lorsque l'utilisateur demande l'affichage de la liste des émissions 100 % disponibles, le

modèle du module « Playlist » va lire ce fichier de registre, l'analyser et l'enregistrer dans une structure de données en prenant soin de trier les dates de la plus récente à la plus ancienne. L'interface affichant la liste des types de 100 % disponibles va directement piocher ces types dans la structure de données. Lorsque des détails sur un type de 100 % sont demandés, l'interface affiche la première date de la structure de données, c'est à dire la date la plus récente. Grâce aux cliques sur les flèches d'action, pour passer à une date postérieure ou ultérieure, on avance ou on recule dans la structure de données via un indice de position courante.

Enfin, la dernière fonction que doit effectuer le modèle de ce module est de récupérer et charger la liste des musiques associées à la playlist pour laquelle l'utilisateur souhaite des détails. Grâce à la date de la playlist, le modèle connaît le dossier dans lequel chercher les informations qui lui sont liées car la date correspond également au nom du dossier. Il suffit alors de charger le fichier « content.json » appartenant au dossier, de le parcourir jusqu'à trouver le 100 % concerné et d'extraire la liste des musiques qui le compose directement. Une fois cette liste chargée, la vue peut la récupérer pour l'afficher.

A chaque fois qu'une demande de lecture de 100 % est émise via le module « Playlist », le contrôleur du module va demander au lecteur audio de charger tous les podcasts qui correspondent à la date souhaitée via le « PodcastLoader ». Une fois tous les podcasts chargés, le contrôleur va demander au lecteur de lire le 100 % en lui envoyant son nom. Le lecteur devra alors parcourir la liste des podcasts qu'il a chargé pour trouver celui qui a le nom souhaité et le lancer.

4.4. La gestion automatique du profil audio

4.4.1. présentation générale de la fonctionnalité

Comme expliqué dans la définition des besoins, la borne doit être capable de gérer de manière automatique son profil audio de manière à adapter son volume de diffusion en fonction des différentes heures de la journée. Par exemple, une borne située proche de salles de cours ne devra tout simplement pas émettre à certaines heures de la journée. Nous avons géré cette fonctionnalité au travers de fichiers de configuration et de services. Il existe deux fichiers de configuration : profile.ini qui va gérer le profil (activée ou désactivée) de la borne en fonction des jours de la semaine et des heures ainsi que volume.ini qui va avoir le même rôle mais pour le volume de la lecture.

Les fichiers de configuration sont des fichiers au format INI. Les fichiers INI sont des fichiers textuels. Ils sont divisés en sections et chaque section comporte un certain nombre de paramètres de configuration. Chaque section commence par un titre placé entre crochets et la valeur de chaque paramètre de configuration est indiquée par la syntaxe : paramètre = valeur.

Voici un extrait possible du fichier « profile.ini » :

```
[Lundi]
7=enabled
12=enabled
13=enabled
18=enabled
19=enabled
default=disabled
```

Ce fichier de configuration définit les heures d'activation de la borne pour les jours de la semaine qui tombent un lundi. Ainsi, on remarque que la borne sera activée à 7h, 12h, 13h, 18h et 19h. Elle sera désactivée par défaut durant les autres horaires.

L'état « désactivé » de la borne correspond à un état où celle-ci ne diffuse pas de musique (le lecteur audio est stoppé) et l'interface tactile est inaccessible. Lors de l'état « désactivé » l'écran fixe suivant est affiché sur la borne :

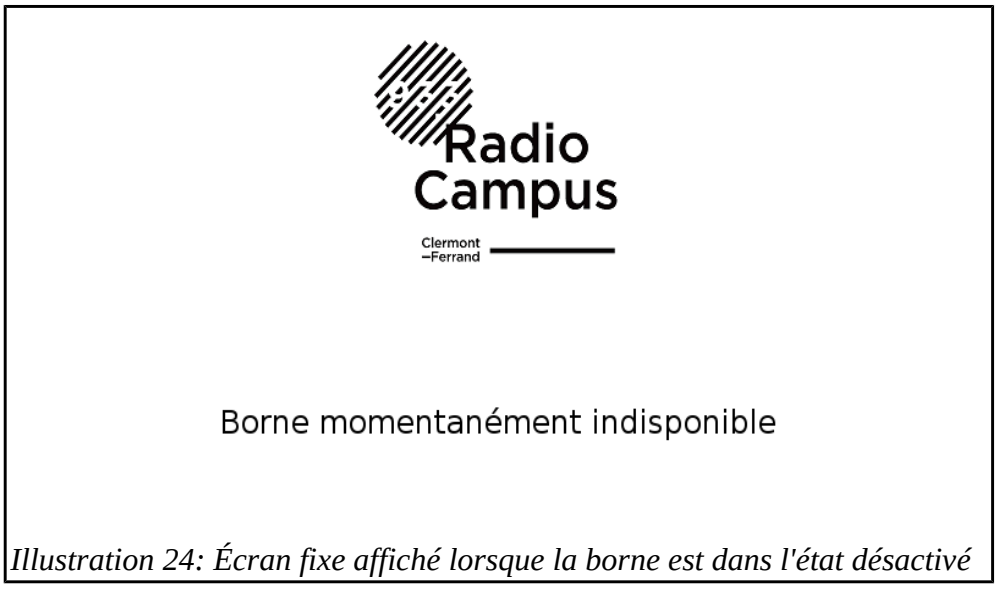

Illustration 24: Écran fixe affiché lorsque la borne est dans l'état désactivé

En ce qui concerne la gestion du volume, le fichier de configuration est similaire à celui du profil. Pour chaque jours de la semaine, il y a un volume sonore par défaut compris entre 0 et 100 et pour chaque heure de la journée, le même type de volume sonore est défini.

4.4.2. Explications techniques

La gestion du profil audio a été confiée au service « AudioProfilManager ». Ce service est connecté sur un timer et est appelé toutes les secondes. Lorsqu'une nouvelle heure de la journée commence, il va ouvrir les fichiers de configuration et récupérer directement la configuration correspondante à l'heure courante ou par défaut si celle-ci n'existe pas. Pour la gestion du volume, il suffit d'appliquer au lecteur audio de l'application le volume lu dans le fichier de configuration grâce à la méthode setVolume(). Pour la gestion de l'activation / désactivation de la borne, il suffit d'appeler des méthodes setDisabledState() de la classe « RCPlayer ». Cette méthode aura pour effet de masquer la fenêtre principale, mettre sur pause le lecteur audio et de demander à la classe « RCSplashScreen » d'afficher en plein écran l'image fixe d'indisponibilité.

4.5. Module d'enregistrement de messages audio

4.5.1. Description de la fonctionnalité

Dans le cahier des charges que nous avions, nous devions également pouvoir permettre aux utilisateurs d'enregistrer et de diffuser un message audio à RCCF. Nous avons implémenté cette fonctionnalité grâce à un module spécifique, toujours composé d'une vue, d'un modèle et d'un contrôleur. Pendant l'utilisation de ce module par l'utilisateur, la diffusion audio de la borne est coupée pour permettre l'enregistrement du message sans bruit parasite. La lecture est ensuite automatiquement ré-activée une fois l'enregistrement du message terminé.Voici l'écran de la première étape d'enregistrement d'un message audio :

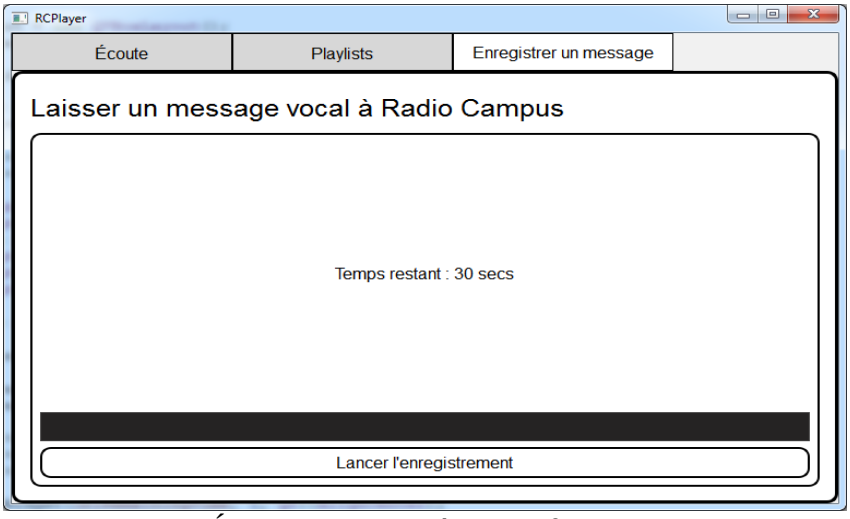

Illustration 25: Écran de la première interface de l'enregistrement de messages audio

Cet écran se compose de trois éléments graphiques différents : un label indiquant la durée possible pour l'enregistrement vocal, une barre de chargement et un bouton pour lancer l'acquisition audio via le micro. Dès que l'utilisateur appuie sur le bouton « Lancer l'enregistrement », l'interface suivante apparaît :

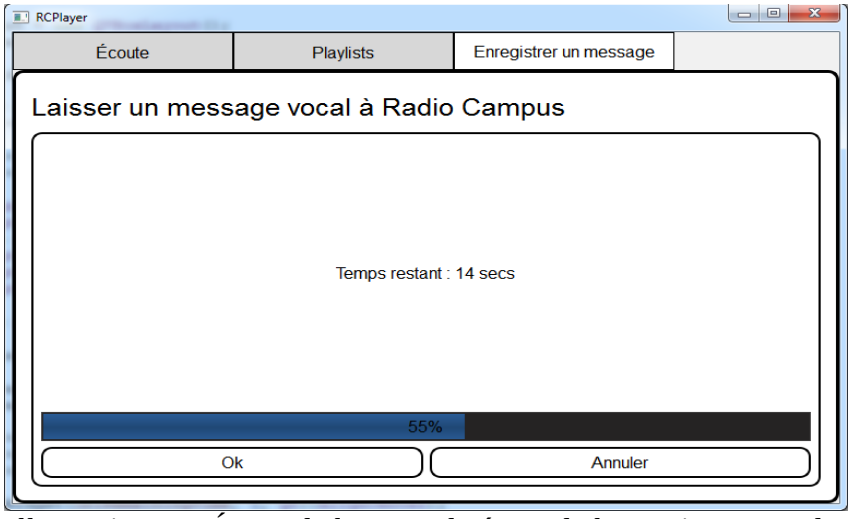

Illustration 26: Écran de la seconde étape de l'enregistrement de messages audio

Cette interface apparaît lorsque l'enregistrement est en cours. On remarque que l'utilisateur dispose de deux actions possibles : la validation de son enregistrement (« Ok ») et l'annulation de son enregistrement (« Annuler »). Le temps restant pour l'enregistrement est affiché sous forme textuelle et visuelle via une barre de chargement qui se remplit. A tout moment, l'utilisateur peut presser le bouton « Annuler » ce qui aura pour effet de supprimer l'enregistrement ou « Ok » ce qui validera l'enregistrement. S'il n'y a plus de temps restant pour l'enregistrement, celui-ci est automatiquement validé et l'interface de la troisième étape apparaît.

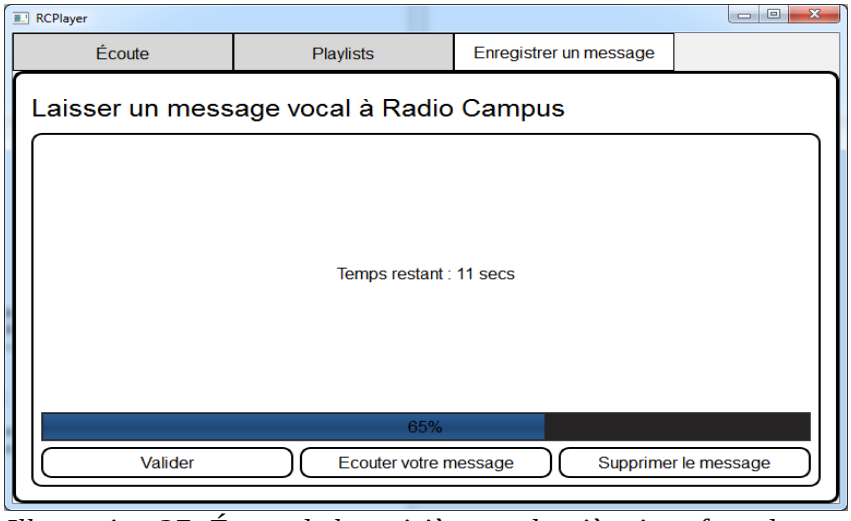

Illustration 27: Écran de la troisième et dernière interface de l'enregistrement de messages audio

Cette troisième interface est l'interface de confirmation de l'enregistrement. L'utilisateur peut le valider définitivement, écouter son message via les hauts-parleurs de la borne ou encore supprimer son message définitivement pour en faire un nouveau. Lorsque le message est définitivement validé, il est directement envoyé sur le serveur de RCCF.

4.5.2. Implémentation de la fonctionnalité

C'est la classe « MessageRecorderModel » qui a pour rôle de gérer l'enregistrement de messages audio. Dans le constructeur de cette classe, on retrouve le code permettant de configurer le micro. Une classe Qt permet de gérer facilement l'enregistrement audio avec le micro, il s'agit de « QAudioRecorder ». Voici le code utilisé pour configurer le micro :

```
QaudioRecorder audioRecorder;
QAudioEncoderSettings audioSettings; //Permet de configurer
l'AudioRecorder

//Positionnement du codec audio utilisé
audioSettings.setCodec("audio/PCM");
//Qualité d'enregistrement
audioSettings.setQuality(QMultimedia::HighQuality);

//Positionnement des paramètres du recorder
audioRecorder.setEncodingSettings(audioSettings);
//Format audio de sortie
audioRecorder.setContainerFormat("wav");
```

Grâce à ce code, l'objet « audioRecorder » va être configuré avec le micro par défaut et prêt à être lancé. Lorsqu'un enregistrement est lancé, le module va commencer par mettre le lecteur audio à l'état « Stop » pour arrêter la diffusion de contenu audio. Un fichier au format WAV va ensuite être créé et le flux audio enregistré par le micro y sera directement stocké. Ce fichier audio sera stocké temporairement, le temps que l'utilisateur le valide ou l'annule, dans un dossier nommé « records ». Le nom du fichier sera généré de la manière suivante : id_yyyy-MM-dd-hhmmss.wav. Le champs « id » correspond à l'identifiant unique de la borne sur laquelle a été enregistré le message. Cet identifiant est défini dans le fichier de configuration général de la borne. Le champs « yyyy » correspond à l'année, « MM » au mois, « dd » au jours, « hh » à l'heure, « mm » aux minutes et « ss » aux secondes. Avec cette convention de nommage de fichier, nous pouvons être certains de pouvoir identifier de manière unique deux messages distincts.

En cas d'annulation, l'interface est simplement mise à jour et le fichier WAV temporaire est supprimé. Si l'utilisateur souhaite ré-écouter son message, une méthode va faire appel au lecteur audio en lui fournissant le chemin vers le fichier WAV et lui demander de lancer la lecture. Ainsi, on utilise le même lecteur audio pour lire le flux direct, lire les podcasts et les messages laissés par les auditeurs.

Lorsque le message audio est validé, celui-ci va être déplacé dans un dossier « sync » pour être synchronisé. Une fois le fichier déplacé, l'application va lancer un script bash que nous avons écris et qui aura pour rôle de scanner le contenu du dossier de manière à envoyer tous les fichiers WAV s'y trouvant via une connexion SSH par clef RSA (cf lexique) sur le serveur distant grâce à la commande scp (Secure CoPy). Une fois les fichiers correctement envoyés, le script va se charger de nettoyer le dossier « sync » avant de s'arrêter. Les fichiers envoyés sur le serveur de RCCF pourront alors être pris en charge par une application web pour être écoutés et gérés par les administrateurs de la radio.

4.6. Le module de diffusion de messages textuels de l'UBP

Une fonctionnalité permettant à l'UBP de communiquer avec les étudiants via la borne devait également être implémentée. L'UBP devait être capable de de diffuser des messages, mais aussi les modifier ou même les supprimer en cas de besoin. Il devait donc y avoir deux interfaces. La première est celle sur la borne, permettant d'afficher les messages que les étudiants pourront alors consulter. Puisque nous avons choisi de fonctionner avec un système d'onglets nous avons décidé de dédier un onglets aux messages. La deuxième interface est celle permettant à l'UBP de créer les différents messages à afficher sur la borne. Pour faire en sorte qu'il n'y ai rien à installer et que ce soit utilisable depuis n'importe quelle machine ayant accès à internet donc moins contraignant cette interface est sous la forme d'un site web.

4.6.1. Création et modification des messages

Le site web permettant de créer les messages, alimente (ou modifie) une base de données qui est alors utilisée par la borne.

4.6.1.1. Base de données

Pour pouvoir échanger des données entre le site web et les borne une base de données mySQL a été mise en place. MySQL est un système de gestion de bases de données relationnelles. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde. Notre choix s'est porté sur ce système en partie à cause de cela, mais aussi parce que le développement web effectué pour ce projet a été réalisé sous Wamp qui fournit mySQL.

Pour l'instant une seule table est utilisée, elle contient un titre, le corps du message et ainsi que la date de création mais également un identifiant, clé primaire de la table. Ces deux derniers sont générés automatiquement au moment de la validation du message. La structure de celle-ci est la suivante:

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	id	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	title	varchar(30)	latin1_swedish_ci		Non	*Aucune*	
3	content	varchar(250)	latin1_swedish_ci		Non	*Aucune*	
4	date	date			Non	*Aucune*	

Illustration 28: Structure de la base de données de l'interface web

Depuis l'interface web, l'accès à la base de données est fait avec du code PHP, exécuté au niveau du serveur. La machine hébergeant le site web et celle la base de données étant la même, la lecture et la modification de la base de données est donc effectué en 'localhost' il n'y a donc aucun problème d'accès à celle-ci. Cependant, l'accès depuis les différentes bornes se fait depuis des machines distantes. Comme les adresses ip des bornes en place ne sont pas forcément connues, l'accès à la base de données est données à toutes les machines pouvant fournir le bon mot de passe. Pour pour donner une telle autorisation, il

faut taper la commande suivante une fois connecté au serveur mySQL:

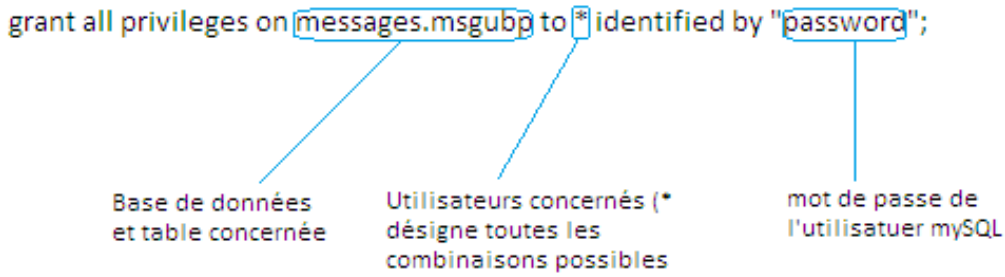

Illustration 29: Requête à effectuer pour autoriser un appareil à accéder à la base de données

4.6.1.2. Authentification

Le site web créé doit être accessible seulement par des personnes autorisées étant donné qu'il permet de modifier, supprimer et créer des nouveaux messages, messages qui sont visibles par tous à travers les différentes bornes du parc. Il fallait donc ajouter un procédé permettant d'identifier un opérateur qui tente d'accéder au site.

Une première piste envisagée consistait à passer par une page d'accueil qui demanderait un identifiant avec un mot de passe correspondant tous deux stockés dans une base de données. Une fois la vérification effectuée, l'utilisateur serait alors redirigé vers la page permettant la saisie des messages. Cela nécessitait de faire en sorte que l'on ne puisse pas accéder directement à la page utilisée pour créer les messages sans avoir fait l'authentification, mettre en place un code qui ne soit pas vulnrable à des attaques telles que l'injection SQL par exemple. Or, il existe une manière sure de garantir une authentification sur un site web, ce sont les fichiers *.htaccess*.

Les fichiers *.htaccess* sont des fichiers de configuration d'Apache, permettant de définir des règles dans un répertoire et dans tous ses sous-répertoires . On peut les utiliser pour protéger un répertoire par mot de passe, ou pour changer le nom ou l'extension de la page index, ou encore pour interdire l'accès au répertoire. Voici un exemple montrant de quoi est composé un fichier .htaccess simple (avec une seule règle):

```
AuthName "Page d'administration protégée"
AuthType Basic
AuthUserFile "/home/www/.htpasswd"

Require valid-user
```

On peut distinguer deux sections différentes au sein de ce fichier. La première est celle qui contient les différentes informations qui sont nécessaire. Cette section est représentée par les trois premières lignes. On y trouve tout d'abord le message d'information qui sera affiché dans la fenêtre qui demande l'authentification (AuthName). La ligne 'AuthType Basic' précise qu'il faut utiliser les données fournies par AuthUserFile pour l'authentification. Enfin, 'AuthUserFile' définit le chemin d'accès absolu vers le fichier de mot de passe, nommé ".htpasswd", qui contient la liste des utilisateurs autorisés ainsi que les mot de passe correspondants, qui peuvent également être cryptés plutôt que de rester en clair. Le répertoire dans lequel ce fichier est placé peut être le même que celui de .htaccess. Ce type

de fichier prend la forme suivante :

```
admin:Gy0rVVCm3j
agentubp:7aZ0Sw9hPN
```

La seconde section que l'on peut identifier au sein est celle qui contient la définition des conditions d'accès. R*equire valid-user* précise que l'on autorise uniquement les personnes identifiées. Si même au sein des personnes identifiées, il y a des autorisations différentes, il est également possible de préciser explicitement le nom des personnes autorisées à s'identifier : R*equire user {username}.*

Quand un opérateur tente d'accéder à une page web concernée par ce fichier .htaccess, la fenêtre suivante apparaît:

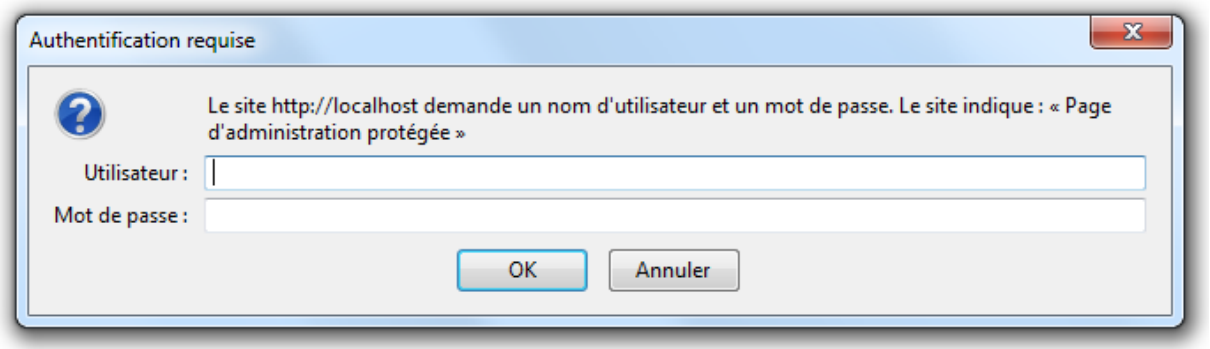

Illustration 30: Fenêtre demandant à l'utilisateur de s'authentifier pour accéder à l'interface web

4.6.1.3. Affichage et suppression des messages existants

Pour savoir quels sont les messages qui sont déjà en ligne, et pouvoir les supprimer en cas de besoin (message obsolète ou erroné), et ce, sans avoir à se connecter directement à la base de données, les messages sont affichés sur la page web, avec la possibilité de les supprimer.

L'affichage se fait simplement, en parcourant la table qui stocke tous les messages :

```php
<div id="messages">
    <!-- les messages du tchat -->
    <?php

        // on se connecte à notre base de données
        try
        {
            $bdd = new PDO('mysql:host=localhost;dbname=messages', 'root', '');
        }
        catch (Exception $e)
        {
            die('Erreur : ' . $e->getMessage());
        }

        // on récupère les 10 derniers messages postés
        $requete = $bdd->query('SELECT * FROM msgubp ORDER BY id DESC ');

        while($donnees = $requete->fetch()){
            // on affiche le message (l'id servira plus tard)
            echo "<p id=\"" . $donnees['id'] . "\"> <B> " . $donnees['title'] . " :</B>
            <br/> " . $donnees['content'] . "</p>";
        }

        $requete->closeCursor();

    ?>

</div>
```

Illustration 31: Code PHP permettant de récupérer les messages de la base de données

4.6.1.4. Création des messages

C'est grâce à un formulaire que la saisie des messages se fait. Pour ce faire, seuls 2 champs sont nécessaire, l'un pour le titre du message, et un autre, plus long, qui sera pour le corps du message, étant donné que la date de création peut être généré automatiquement. En effet, en PHP la fonction date permet d'avoir la date au format souhaité. Dans le cas du 1er janvier 2016 par exemple, date('d/m/Y') renvoie 01/01/2016.

Pour éviter de recharger la page autant que possible, et ainsi re parcourir la base de données, chose qui peut être longue quand celle-ci est volumineuse, AJAX s'est avéré utile. L'architecture informatique Ajax (pour ***Asynchronous JavaScript and XML***) permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en se servant de différentes technologies. Son utilisation durant ce projet avait pour but d'intégrer directement dans la liste regroupant la liste des messages existants tout message créé. Pour y arriver, il faut créer une fonction javascript qui est appelée à l'appui sur le bouton 'envoyer' du formulaire. Grâce aux fonctionnalités d'AJAX, on peut ajouter du contenu dans la page:

```javascript
if(title=='' || content=='') {
$('.success').fadeOut(200).hide();
$('.error').fadeOut(200).show();
} else {
$.ajax({
type: "POST",
url: "join.php",
data: dataString,
    success: function(){
        $('.success').fadeIn(200).show();
        $('.error').fadeOut(200).hide();
    }
```

Illustration 32: Code AJAX utilisé pour le formulaire

On peut afficher des messages pour informer sur la validité des données, et faire appel à un autre script php sans avoir ouvrir une nouvelle fenêtre ou recharger la page. Ici c' est join.php qui est lancé. Comme on peut le voir, on peut aussi lancer un script avec des

données 'POST', ce qui est pratique puisque l'on peut utiliser la méthode POST dans le formulaire qui permet de saisir les données. Le script auquel l'on fait alors appel ne fait que récupérer les données :

```php
<?php
    include_once("config.php");
        if($_POST)
        {
        /* VALUES */
        $title=$_POST['title'];
        $content=$_POST['content'];
        $datetime = date('d/m/Y');

        $db_selected->exec("INSERT INTO msgubp (title, content,date) VALUES ( '"
                           |.utf8_decode($title)."', '".utf8_decode($content)."', '".$datetime."')");
        } else {

            header('HTTP/1.1 500 Looks like mysql error, could not insert record!');
            exit();
        }

    ?>
```

Illustration 33: Script PHP permettant d'insérer une nouvelle entrée dans la base de données via une requête POST

4.6.2. Affichage des messages

Le système d'affichage de messages sur la borne est en fait un onglet sur l'application, qui à chaque fois que l'on clique sur le bon onglet, interroge la base de données et affiche tous les messages présents dans celle-ci. On peut effectivement fonctionner de cette manière puisque l'onglet principal, celui qui sera ouvert la majeure partie du temps sera l'un de ceux servant à écouter de la musique, l'onglet des messages de l'UBP sera ouvert quand quelqu'un voudra vérifier la présence de nouveaux messages et rebasculera alors sur un autre onglet.

Pour créer ce nouvel onglet il faut comme pour tout onglet créer 3 classes:

- MessageView: C'est la vue permettant d'afficher l'interface des messages
- MessageModel: C'est le modèle
- MessageController: C'est le controleur qui sert d'interface entre la vue et le modèle

Une fois ces trois classes créées, il suffit d'utiliser la méthode addScreen de la classe RCPlayer (avec un objet MessageView, un MessageModel et unMessageController) pour avoir un nouvel onglet dans lequel on pourra écrire le contenu de la base de données utilisée pour les messages.

Avant de pouvoir utiliser cet onglet il faut pouvoir utiliser QtSQL, et pour y parvenir il faut compiler les pilotes de ce dernier. Il faut commencer par installer le package MySQL. Ensuite il faut lancer quelques commandes sous l'invite de commande de Qt qui vont permettre de créer les fichiers '.dll'.

```
cd %QTDIR%\src\plugins\sqldrivers\mysql
qmake "INCLUDEPATH+=[Chemin MySQL]\\include"
                    "LIBS+=[Chemin MySQL]\\lib\\libmysql.lib" mysql.pro
mingw32-make -f Makefile.Debug
mingw32-make -f Makefile.Release
```

Cette étape permet de créer les fichiers '.dll' nécessaires à l'utilisation de MySQL sous Qt. Il faut alors bien placer ces fichiers dans un répertoire 'sqldrivers' lui-même placé dans un répertoire 'debug' si l'on est en mode debogage ou 'release' sinon.

Quand cela est fait, utiliser QtSQL devient alors possible. Pour utiliser la base de données utilisées pour les messages, il faut commencer par entrer les paramètres de la base de données:

- **Adresse IP :** ce sera par la suite celle du serveur MySQL utilisé, pendant le développement de cet outil le serveur MySQL était sur la machine sur la quelle le code Qt était compilé, l'adresse IP est donc localhost
- **Nom de la base de données :** il s'agit de la base de données messages
- **Utilisateur MySQL:** il s'agit de root
- **Mot de passe**

Pour chacun des champs à remplir il y a un méthode correspondante:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setUserName("root");
db.setPassword("");
db.setDatabaseName("messages");
```

Illustration 34: Code C++ permettant de configurer la base de données pour y accéder sur les bornes

Enfin, une fois connecté on peut exécuter une requête facilement. Il suffit de la prendre comme un chaine de caractère et de la donner en paramètre à la méthode exec() qui s'applique à un objet de la classe 'QsqlQuery'. Dans notre cas on ne souhaite pas une requête particulière car on ne désire pas sélectionner des messages en particulier, la requête utilisée est simplement: "SELECT * FROM msgubp"

Pour afficher le résultat au format souhaité, il faut utiliser un label. En effet, on commence par parcourir le résultat, et on stocke chacun des éléments trouvés dans un QString intermédiaire puisque l'on ne peut pas utiliser l'équivalent d'un append() sur un label. Il suffira de mettre le contenu du QString intermédiaire dans le label.

```
while(query.next())
{
    std::cout << "    Nouvelle entrée" << std::endl;
    for(int x=0; x < 2; ++x)
    {
        temp.append(query.value(x).toString());
    }
}
```

5. Bilan et perspectives

Après avoir évoqué les outils utilisés, le contexte du projet et présenté les différentes réalisations, il convient de dresser un bilan sur le projet.

5.1. Les Réussites

Tout d'abord, la première chose à souligner est que nous sommes parvenus à fournir une première version fonctionnelle du prototype de la borne qui peut constituer une version de démonstration, un premier ouvrage à déployer pour tester le concept. Notre travail nous a conduit à des choix technologiques ainsi qu'au déploiement d'une application portable sur un premier prototype de borne. Nous sommes parvenus à implémenter un certain nombre de fonctionnalités telles que : la lecture de flux radio direct, la gestion des podcasts, l'enregistrement de message audio par les auditeurs ainsi que la mise en valeur des émissions 100 %. Notre étude a permis de résoudre certains problèmes techniques tels que l'interfaçage d'une nouvelle application aux services de RCCF, le déploiement des bornes sur le réseau de l'université et la synchronisation des podcasts. Nous avons également réalisé une première version du site permettant la diffusion de messages textuels de l'UBP sur les bornes et une première réflexion quant à son intégration future sur les bornes. De plus, ce projet nous a permis de développer notre connaissance du framework Qt et de ses particularités ainsi que d'apercevoir les coulisses d'une radio associative. Il a été très enrichissant de comprendre le fonctionnement de la radio sur le plan technique.

5.2. Les limites

Des limites à notre projet existent cependant. La fonctionnalité d'administration à distance des bornes a été réfléchie mais pas implémentée par manque de temps. De plus la fonctionnalité de diffusion de messages textuels de l'UBP reste incomplète. De même, quelques problèmes persistent sur certaines fonctionnalités de la borne en particulier concernant le direct : le flux radio direct de Radio Campus fonctionne mal avec l'application contrairement aux flux d'autres radios. Le problème provient très probablement de la configuration du serveur de broadcast de RCCF mais après de nombreuses recherches nous ne sommes pas encore parvenus à expliquer ce problème. De même, il reste des soucis liés à l'installation du serveur proxy (cf lexique) destiné à recevoir et transmettre toutes les requêtes des bornes : pour des raisons inconnues tous les appels réseau passent par le proxy excepté le flux radio direct. Enfin, nous regrettons de ne pas avoir pu participer à la réalisation du prototype entier de la borne, à savoir son corps mécanique. A ce sujet nous avons rencontré l'association Acolab qui a proposé de nous aider dans le futur dès lors que leur outillage serait opérationnel et que nous disposerions de plans précis pour sa réalisation.

5.3. Les tâches à réaliser pour le futur

Il reste donc un certain nombre de tâches à accomplir pour le futur pour que les bornes soient pleinement opérationnelles. La première de ces tâches est la réalisation d'une interface d'administration web permettant : la visualisation de l'état des bornes, des actions simples telles que le redémarrage, le réglage des fichiers de configuration à distance ainsi

que le lancement de scripts pour des actions plus complexes. C'est un travail qui peut être encore long en fonction de la qualité et du type des outils demandés. De plus, il convient de poursuivre et terminer l'implémentation de la fonction permettant la diffusion de messages textuels par l'UBP, pour cela une analyse plus poussée des besoins sera nécessaire. Enfin, il faudra prévoir un temps de test des bornes grandeur nature pour corriger les différents bugs et problèmes qui peuvent persister.

Conclusion

Pour conclure ce rapport, nous pouvons dire que notre objectif initial est presque atteint. Nous avons été en mesure de sélectionner les technologies appropriées à la réalisation de la borne aussi bien logicielles que matérielles, nous avons monté le premier prototype électronique composé d'un écran tactile, d'un Raspberry Pi 2, un dungle Wifi et une carte son USB. Concernant la partie logicielle, nous avons choisi d'utiliser le framework de développement Qt 5 en C++ pour la réalisation de l'interface graphique ainsi que les fonctions liées à la lecture de flux audio.

Au sujet des fonctionnalités implémentées par le logiciel d'exploitation de la borne, nous sommes arrivés à un produit fonctionnel, permettant l'écoute en direct de RCCF, l'écoute des podcasts aussi bien en mode connecté que déconnecté, la mise en valeur des émissions spéciales « 100 % » de RCCF ainsi que la possibilité pour les auditeurs d'enregistrer un message vocal à destination de la radio. Il nous manque cependant les fonctionnalités liées à l'administration de la borne à distance ainsi que la publication de messages textuels par l'UBP qui a été en partie implémentée.

De plus, concernant la partie physique de la borne, nous déplorons de ne pas avoir pu participer à la conception du support physique qui aurait permis d'avoir un premier prototype exploitable. De même, il reste encore du travail sur ce projet, principalement concernant l'ajout d'une interface d'administration à distance ainsi que des tests et ajustements avec les utilisateurs.

Pour terminer, nous souhaitons ajouter que ce projet a été très enrichissant pour nous, il a permis de développer nos compétences en C++, d'apprendre la maîtrise du framework Qt, d'apercevoir les coulisses d'une radio associative et de mettre à profit nos compétences sur un projet d'envergure, innovant et enrichissant. Nous espérons que Radio Campus Clermont-Ferrand parviendra au bout de ce projet et nous suivrons avec attention son futur.

Lexique

Broadcast : *La notion de broadcast est employée par les techniciens en informatique et réseaux ; il s'agit à proprement parler, de transmission ou de liaison. Le principe de base est le même que la télédiffusion, étant donné que l'on diffuse des paquets de données à de nombreux clients éventuellement sans discrimination.*

Clef RSA : *Le chiffrement RSA (nommé par les initiales de ses trois inventeurs) est un algorithme de cryptographie asymétrique, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet.*

Format JSON : *JSON (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. Un document JSON ne comprend que deux types d'éléments structurels : des ensembles de paires nom - valeur et/ou des listes ordonnées de valeurs.*

Framework : *En programmation informatique, un framework ou structure logicielle est un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).*

GSM : *Global System for Mobile Communications (GSM) (historiquement « Groupe spécial mobile ») est une norme numérique de seconde génération pour la téléphonie mobile. Tel qu'il a été conçu, le réseau GSM est idéal pour les communications de type « voix » (téléphonie). La norme GSM a ensuite été étendue pour prendre en charge de plus hauts débits et le transport de données en mode « paquet ».*

HTTP : *L'HyperText Transfer Protocol, plus connu sous l'abréviation HTTP — littéralement « protocole de transfert hypertexte » — est un protocole de communication client-serveur. Le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.*

I2C : I2C (signifie : Inter-Integrated Circuit, en anglais) est un bus informatique utilisé pour faire communiquer des microcontrôleurs avec leurs périphériques à l'échelle d'un circuit. Ce bus se décompose en deux fils dont un dédié au transfert des données.

Icecast2 : *Icecast est un logiciel libre de type serveur de diffusion de flux audio et vidéo. Il permet donc à partir d'un ordinateur de diffuser de la musique et des vidéos à des logiciels « clients » (lecteurs audio ou multimédias), au travers d'Internet ou d'un réseau local.*

Layout : *C'est un composant logiciel qui permet de définir la mise en page des différents éléments graphiques qui composent une interface d'un logiciel informatique.*

Bornes de diffusion Radio Campus

OVH : *OVH est un hébergeur de sites web français. Il propose des serveurs dédiés, des serveurs privés, de l'hébergement mutualisé, du housing (ou colocation), des services de Cloud computing, de la fourniture d'accès Internet par lignes ADSL, VDSL ainsi que SDSL, l'enregistrement de noms de domaine, ainsi que de la téléphonie sur IP.*

Podcast : *Émission de radio ou de télévision que l'on peut télécharger depuis internet vers un baladeur.*

Proxy : U*n serveur proxy (serveur mandataire en français) est une fonction informatique client-serveur qui a pour fonction de relayer des requêtes entre une fonction cliente et une fonction serveur.*

Singleton : *En génie logiciel, le singleton est un patron de conception dont l'objet est de restreindre l'instanciation d'une classe à un seul objet. Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système.*

Thread : *Un thread ou fil (d'exécution) ou tâche est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle.*

VPS : *Un serveur dédié virtuel (également appelé serveur virtuel), en anglais virtual private server (VPS) est une méthode de partitionnement d'un serveur en plusieurs serveurs virtuels indépendants qui ont chacun les caractéristiques d'un serveur dédié, en utilisant des techniques de virtualisation. Chaque serveur peut fonctionner avec un système d'exploitation différent et redémarrer indépendamment.*

Service web : *Un service web (ou web service) est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.*

Widget : *En informatique, un composant d'interface graphique (aussi appelé widget en anglais ou encore control) est un élément de base d'une interface graphique avec lequel un utilisateur peut interagir (par exemple une fenêtre ou une zone de texte).*

Webographie

Documentation diverse. [En ligne]
https://fr.wikipedia.org

Ressources sur Icecast. [En ligne]
http://icecast.org/

Ressources sur Qt. [En Ligne]
http://doc.qt.io/
https://openclassrooms.com/courses/programmez-avec-le-langage-c/introduction-a-qt

Ressources sur Raspberry Pi. [En ligne]
http://raspbian-france.fr/category/mise-place-installation-raspbian-sur-raspberry-pi/
http://raspbian-france.fr/installer-raspbian-premier-demarrage-configuration/
https://www.raspberrypi.org/

Ressources sur Radio Campus. [En ligne]
http://www.campus-clermont.net/
http://www.radiocampus.fr/presentation-du-reseau

Informations techniques diverses. [En ligne]
https://openclassrooms.com/
http://stackoverflow.com/

Informations sur les tunnels SSH. [En ligne]
https://openclassrooms.com/courses/mise-en-place-d-un-tunnel-tcp-ip-via-ssh
http://blog.netapsys.fr/creer-des-tunnels-ssh/
http://www.linux-france.org/prj/edu/archinet/systeme/ch13s04.html

Informations et tutos sur les cartes son USB. [En ligne]

Bornes de diffusion Radio Campus

http://computers.tutsplus.com/articles/using-a-usb-audio-device-with-a-raspberry-pi--mac-55876

http://raspberrypi.stackexchange.com/questions/19705/usb-card-as-my-default-audio-device

Bibliographie

Sébastien Mauras, Alessandra Abruzzese et Sébastien Roudier, Rapport de projet collectif : « Conception de bornes de diffusion Radio Campus », 2015