Rapport final

Adonis Najimi,
Valentin Stern,
Vincent Albert,
Théo Gerriet

31 mars 2014

1 Client C++

1.1 Description technique

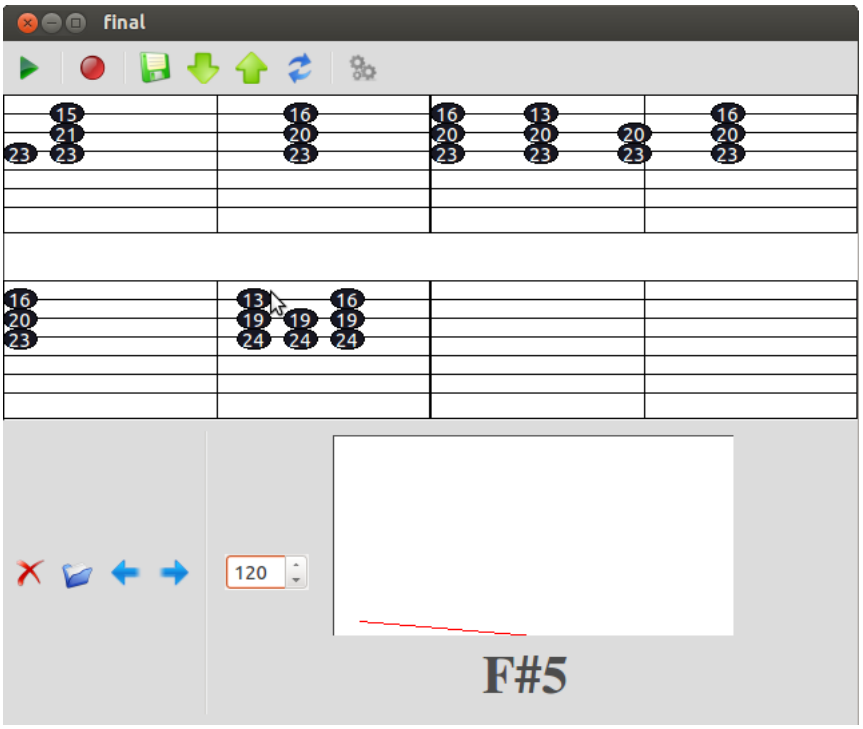

FIGURE 1 – Rendu final de l'application

Afin de lier l'application cliente et l'application web, nous utilisons un format de sauvegarde partagé par les deux langages (C++ et JS), le JSON. Nous avons donc conçu un simple parser JSON afin de pouvoir sauvegarder une partition soit sur son PC local, soit dans la base de donnée distante pour la partager avec le monde. Nous avons choisi ce format d'enregistrement car il est aisément lisible sur le serveur qui peut alors traiter rapidement les informations pour les afficher au utilisateurs, et car c'est un format de plus en plus répandu. Nous aurions pu prendre du XML, mais il aurait été moins bien traité côté serveur. De plus, le format JSON consomme moins de place en mémoire que le format XML, car il est légèrement moins verbeux. L'application possède donc des fonctionnalités de sauvegardes sous forme de fichier et de sauvegarde sur le c̈loudr̈epresenté par le site web. Il est possible de réouvrir une partition sauvegardée en local.

La bibliothèque FMOD Ex nous permet de récupérer le son entrant de l'application. Elle a été choisi car elle est simple d'utilisation tout en étant efficace. Cela nous a permis de nous concentrer sur le traitement de ce son, qui constitue une des parties les plus importante du projet. Nous avons suivi l'algorithme du rapport de conception, et y avons apporté quelques modifications pour l'implémentation réelle. Cet algorithme permet de récupérer les notes ayant le volume

le plus élevé capté par le micro. Nous pouvons également définir une plage de fréquences afin d'enlever tout parasite possible. Par exemple, dans le cas de la guitare, nous pouvons filtrer toutes les notes qui ne peuvent pas être jouée avec une guitare.

La bibliothèque Qt ne crée pas de nouveau thread pour sa partie graphique comme Swing par exemple, nous avons donc du créer deux nouveaux thread pour l'enregistrement de partitions et l'accordeur. Il a été aisé d'implementer ces thread car Qt nous permet d'en faire un héritant d'une de ses classes. Sans ces threads, l'applications serait par exemple bloquée si l'on appuie sur le bouton d'enregistrement, car celui ne serait plus accessible une fois l'enregistrement lancé.

La quantité de fréquences a traiter étant importante (8192 tous les pas de temps), nous avons décidé d'utiliser le tri rapide pour un maximum de performance. Nous avons due y ajouter une surcouche sauvegardant les places de chaque fréquences car FMOD assigne chaque volume de fréquence à sa place dans le tableau. Si nous avions uniquement trié le tableau de fréquences en fonction du volume, nous n'aurions plus la possibilité de savoir quelle fréquence a le volume le plus haut. Nous avons choisi ce nombre de fréquences à traiter car c'est le maximum proposé par FMOD, et ce nombre nous permet donc d'avoir le maximum de précision possible pour le traitement des fréquences.

Nous avons choisi de fournir deux classes sous le design pattern du singleton car cela simplifiait beaucoup le développement, et car plusieurs instances de ces deux classes serait inutiles. La classe Notes fournit donc toutes les notes possiblement enregistrable dans la musique, et la classe Guitar fournit la possibilité de convertir les notes jouées en frettes pour permettre l'affichage de la tablature. Nous nous sommes donc plus concentré sur les tablatures pour guitare et non les partitions en général, mais il serait possible de l'adapter pour tous les instrument. Il est également possible de changer l'accordage de la guitare, mais uniquement en changeant le code, faute d'un manque de temps pour développer cette fonctionnalité.

Le pattern MVC n'existant pas en C++, nous avons créé une classe NoSkin correspondant à l'affichage graphique, une classe Partition représentant le modèle de l'application et une classe Controller afin de rester au plus proche de cette méthode de conception.
Ceci nous a permis de nous répartir les tâches de manière optimale et de lier nos parties respectives en évitant la majorité des complications. La liaison entre le modèle, le controlleur et la vue se fait principalement dans la classe Controller. En effet, celle ci comporte des attributs appartenant aux autres parties. A l'exception dans la classe TuneThread qui comporte des attributs graphiques, car cette classe doit mettre à jour en temps réel l'accordeur sous forme graphique. La gestion des évènements permet de plus de faire une liaison aisée entre modèle et vue. Les slots sont appelés grâce aux élements graphique de la vue, ce qui entraîne des appels de méthodes du modèle, tout en mettant à jour la vue par la suite.

En ce qui concerne la classe graphique principale NoSkin, on y a implanté les éléments nécessaires à l'interface graphique et on lui a assigné un layout de base, tout en considérant le fait que par la suite on pourrait hériter de cette classe pour permettre des interfaces plus personnalisées.

Pour faciliter sa connexion, l'utilisateur peut enregistrer ses données de connexion à partir de l'application en les enregistrant dans le fichier üser.conf. Il peut ainsi se connecter automatiquement lorsqu'il souhaite uploader ou télécharger une de ses partitions. La fenêtre de configuration a été prévue pour y ajouter d'autres fonctionnalités de configuration de l'application, comme par exemple la personnalisation de l'aspect graphique ou plus simplement le tempo initial d'une partition à sa création ou a l'ouverture du logiciel.

FIGURE 2 – Barre de menu haut

Voici la description des boutons dans leur ordre d'apparition de droite à gauche :
- Bouton play
Permet de jouer une partition pour voir le rendu de la composition effectuée
- Bouton record
Permet de lancer l'enregistrement de la partition. Il se change en un bouton stop au premier appuie. Si on appuie à nouveau dessus, l'enregistrement s'arrêtera.
- Bouton save
Permet d'ouvrir le selecteur de fichier pour sauvegarder la partition. Si elle a déjà été sauvegardé, elle sera directement sauvegarder sans ouvrir à nouveau le selecteur de fichier.
- Bouton download
Ouvre le selecteur de partitions stockées sur le site web afin de les télécharger et de les avoir en local.
- Bouton upload
Upload sur le site web la partition courante. Si cette partition n'a pas de nom, alors l'application ouvrira une boite de dialogue pour permettre l'entrée de l'utilisateur.
- Bouton connection
Permet de se connecter au site web avec les informations fournis dans la configuration. La connexion est automatique lorsqu'on download ou lorqu'on upload une partition.
- Bouton configuration
Ouvre la fenêtre de configuration permettant de rentrer ses informations de connexion ou ses préférences pour l'application.

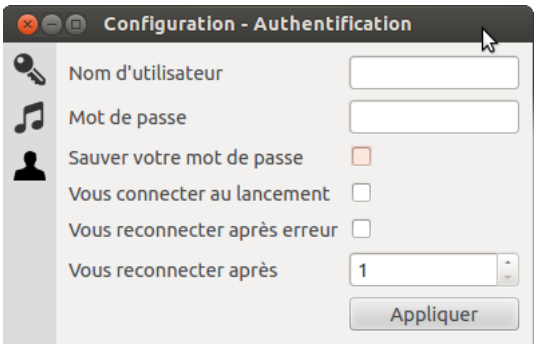

FIGURE 3 – Fenêtre de configuration

La fenêtre de configuration permet comme indiqué ci dessus de rentrer ses identifiants pour se connecter à la base de données et ainsi intéragir avec le serveur de stockage a distance des ses partitions.

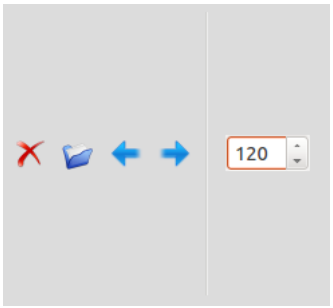

FIGURE 4 – Barre de menu basse

Voici la description des boutons dans leur ordre d'apparition de droite à gauche :
- Bouton delete
Permet de supprimer la partition en cours. Si elle n'a pas été sauvegardé, une boite de dialogue s'ouvre pour demander si l'utilisateur veut la sauvegarder ou non. Ce bouton efface donc toutes les notes de la partition.
- Bouton open
Permet d'ouvrir une partition qui a été sauvegardée sur la machine local. Ce bouton ouvre un selecteur de fichier pour choisir le fichier voulu. Si une erreur de produit lors de la lecture, rien de ne passera.
- Bouton left
Permet de déplacer l'affichage vers le début de la partition.
- Bouton right
Permet de déplacer l'affichage vers la fin de la partition.
- Selecteur tempo
Permet de modifier le tempo de la partition. Ce tempo est effectif dans la sauvegarde de la partition, son enregistrement ainsi que sa lecture.

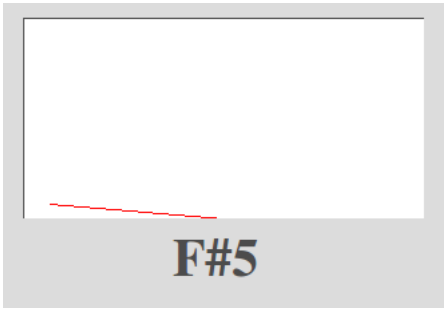

FIGURE 5 – Accordeur intégré

L'accordeur intégré permet d'accorder à tout moment son instrument pour être sur de jouer juste. Une guitare parfaitement accordée permettra donc de faire un enregistrement de partition plus précis. L'accordeur affiche la note captée la plus proche et indique dans quel sens l'utilisateur doit accorder la corde jouée.

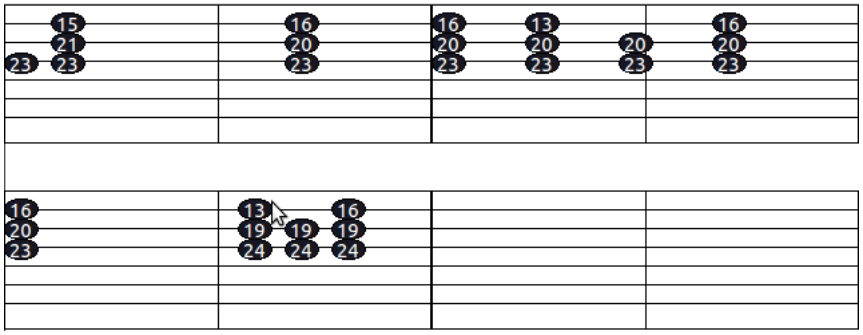

FIGURE 6 – Affichage d'une partition

Voici la partie principale de l'application, l'affichage de la partition. L'affichage fonctionne sur le même principe que les tablatures ou les partitions et la note de la corde la plus grave se trouve le plus en bas. Le chiffre affiché est le numéro de la frette à jouer sur la guitare pour produire la même note. On peut distinctement voir les accords, composés de plusieurs notes alignées verticalement.

1.2 Ambitions

La lecture d'une partition faisait partie d'une des fonctionnalités à priorité haute de l'application. Après de nombreuses recherches nous n'avons pas pu récupérer des fichiers MIDI correspondant aux sons de chaques notes de musique possible pour permettre la lecture de l'application, et c'est pour cela que la fonctionnalité n'a pas été réalisée.

Nous souhaitions faire apparaitre les notes dans la partition en temps réel, mais cela a été impossible à cause du fait que l'enregistrement de la partition se fait dans un Thread, et la création de nouveaux objets est interdite par Qt.

Nous avions donc une erreur, et une améliorations possible serait de permettre cette fonctionnalité.

Nous nous sommes surtout concentré sur l'enregistrement de partition à partir du micro, nous n'avons donc pas fini de développer complètement la modification des partitions après cet enregistrement.

La répartition des taches pour le client a un peu été changée à la fin du projet. Nous avions pensé que le dévelopemment du modèle serait aussi important que celui de la partie graphique, or en fin de projet, nous avons été plus nombreux à développer la partie graphique. Cela est surtout du au fait qu'il fallait lier la partie graphique et la partie des données, et nous avons donc plus pu travailler en collaboration à ce moment là. L'intégration n'a pas engendré trop de difficultés, car nous avions conçu dès le début les deux parties de sorte que tous les éléments soient là pour nous faciliter la jointure.

Nous savions que notre projet était ambitieux et que cela nous prendrait du temps à le réaliser, nous n'avons malheuresement pas réussi à implementer toutes les fonctionnalités que nous souhaitions à cause de cette contrainte mais nous avons pu tout de même fournir un prototype qui fonctionne.

1.3 Choix

Le langage C++ a été un choix apprécié pour l'élaboration du client. Le langage est puissant et rapide ce qui permet un traitement des fréquences très rapide, malgré une quantité d'informations importante. L'étude d'un nouveau langage au sein de ce projet a donc été une chose bénéfique car le langage a directement été appliqué à la pratique.

De plus, l'utilisation du framework Qt s'est avéré être un choix judicieux nous permettant non seulement de développer l'interface graphique du client, mais nous fournissant aussi des outils pour manipuler la base de données du site web à distance à partir de l'application. C'est grâce à cela que l'utilisateur peut donc mettre une partition à jour depuis le client. La connexion de l'application à la base de données se fait par le biais du driver adapté à la base de données utilisée. Il serait donc également aisé de changer de système de gestion de base de données si on le souhaiterait.

Pour l'analyse des sons entrant, le choix de la bibliothèque FMOD nous a été très utile durant la période de développement. Une fois installée, elle permet la récupération de toutes les fréquences entrant par le micro, et cela simplement. Cependant elle n'est pas exempte de défaut et nous n'avons pas réussi à l'utiliser convenablement sur tous les systèmes d'exploitation, ce qui empêche de rendre notre application multiplate-forme comme on l'espérait initialement. L'installation de cette librairie est difficile et très peu de guide nous informe sur les problèmes rencontrés, surtout qu'il faut de plus configurer son utilisation à Qt, différent de son utilisation en ligne de commande.

1.4 Améliorations possibles

Au niveau de la prise de sons, il y a la possibilité d'améliorer celle ci et obtenir plus de précision en implémentant un algorithme permettant de détecter quels sont les seuils à prendre en compte pour la prise de son et obtenir une analyse beaucoup plus fine du son entrant et ainsi des partitions correspondant mieux à la réalité.

Au niveau de la sauvegarde des partitions, il serait possible de compresser un peu plus chaque fichier car il y a beaucoup d'espaces. Cela consisterait à améliorer le parser JSON des partitions. En effet, sur un petit fichier de par exemple 300 caractères, représentant ainsi une partition de 2 accords comportant en tout 7 notes, le gain possible en enlevant les espaces est de 40%. Ainsi, le remplissage de la base de données serait moins rapide, et les performances seraient également plus grandes.

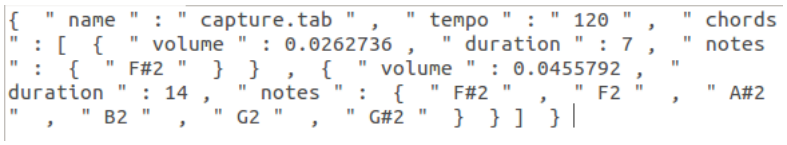

FIGURE 7 – Fichier representant une partition

Une grande amélioration serait également de rajouter la possibilité d'enregistrer plusieurs pistes. Cela permettrait donc de composer des musiques plus complètes avec par exemple une guitare rythmique et une guitare solo. Si l'on y ajoute également la possibilité de changer le son de l'instrument joué, cela permettrait de faire de ce projet une application de composition musicale vraiment très complète. En effet, cela permettrait d'ajouter exactement ce que l'on veut et ainsi avoir le rendu possible de la chanson finale.

Les données de connexion à la base de données sont enregistrées en clair dans un fichier de configuration, il faudrait changer ce fichier et le crypter pour que les utilisateurs ne puissent pas accéder à cette donnée confidentielle. Il faudrait également implémenter le même algorithme de hashage dans l'application que dans le serveur afin de sécuriser la connection de l'utilisateur par le biais de l'application.

Une autre amélioration possible serait l'ajout de la possibilité pour l'utilisateur de consulter les partitions qu'il possède sur son compte sur le site web et de les télécharger. Cette implémentation été déjà en court mais n'a pas pu être finalisée dans les délais. De plus, on pourrait ajouter la possibilité de télécharger en une fois l'ensemble des partitions qu'il possède afin de lui faciliter cette tâche fastidieuse dans le cas où il veut importer toutes ses oeuvres sur un nouveau PC.

Afin de faciliter le déplacement lors de la lecture et de l'enregistrement de la partition, nous avions dans l'intention d'ajouter deux curseurs graphiques (l'un correspondant à la lecture et l'autre à l'enregistrement) afin de pouvoir suivre

l'enregistrement ainsi que la lecture de la partition. Nous pourrions ainsi mettre sur pause à un endroit voulu et le modifier, faire revenir le curseur en arrière et voir le résultat obtenu. Ces deux curseurs faciliteraient donc amplement les taches de naviguation à travers les différentes parties de la partition. Si on joint ceci à la fonctionnalité d'avoir plusieurs pistes instrumentales, on pourrait placer le curseur de lecture à un endroit, et le curseur d'enregistrement au même endroit, et ainsi jouer en même temps que ce que l'on a déjà enregistrer mais sur une autre piste.

On a préparé la possibilité d'implémenter l'utilisation de skins pour modifier l'apparence de l'application afin que le client puisse personnaliser son logiciel comme il le souhaite. Des emplacements de personnalisation sont déjà présent dans la fenêtre de configuration.
De plus, on aurait pu réserver une zone du site web pour y recenser tous les skins existants et permettre à l'utilisateur de télécharger celui qui lui plait. Il aurait pour cela fallu faire en sorte de pouvoir enregistrer le skin sous forme de fichier afin de le partager.

2 Application Web

2.1 Description technique

Le site web permet de partager et de sauvegarder ses partitions. Ainsi, les utilisateurs peuvent accèder à leurs partitions depuis le site. Le site suit le modèle MVC, ce modèle est complété par la mise en place de routes.
Les routes de définir ce à quoi correspond chaque action (les actions sont représentées par les url). Ces routes permettent donc d'effectuer le lien entre l'URL fournie et les classes disponibles dans le dossier controllers.
Le dossier controllers contient les différentes classes qui font le lien entre la vue (le code HTML) et le modèle. Ces classes contiennent des méthodes qui sont appellées par les routes.
Enfin la vue est générée par le serveur quand l'un des controlleurs en a besoin. La bibliothèque Handlebars est utilisée pour la vue, c'est une bibliothèque qui fournit un langage de templates qui vient compléter le code HTML. Ainsi, ce language permet de rajouter de l'élégance dans le code et éviter des répétitions.
De plus, nous avons utilisé la bibliothèque Sequelize.js pour le modèle. Ce framework permet de manipuler les données en utilisant le pattern Active Record. Ainsi, les données sont facilement manipulables.
En effet, les tables sont représentées par des objets. On peut donc créer un nouvel objet et ensuite appliquer la méthode save sur ce dernier pour l'enregistrer dans la base de données.

Nous avons essayé d'utiliser plusieurs technologies webs demandées par les recruteurs ces derniers temps.
Par exemple, nous avons utilisé NodeJS. Ainsi, nous avons codé toute l'application web en javascript. Le javascript a permis d'apprendre certains aspects de la programmation non enseignée à l'IUT (programmation fonctionnelle ou encore les callbacks et l'asynchronysme).
Les websockets ont été utilisées pour intéragir avec l'utilisateur sans aucun rechargement de pages. Le navigateur web peut donc envoyer des données au serveur et le serveur peut lui répondre sans aucun raffraichissement de la page.
Ce projet a donc eu un réel impact pédagogique. De plus, NodeJS et les technologies associées étant particulièrements récentes, nous devions travailler de manière autonome et résoudre des problèmes avec des solutions parfois inexistantes sur internet.

Puis, nous avons du revoir certaines parties de notre MCD. En effet, nous avons remarqué que certaines tables n'étaient pas optimisées.

Enfin, l'application web est structurée de manière à respecter au mieux certaines conventions établies par la communauté NodeJS.
Nous avons donc un dossier config qui contient toutes les informations de configuration de l'application. Ce dossier contient plus particulièrement les fichiers routes.js et sockets.js qui permettent de rediriger vers le bon controller en fonction de l'action a effectuer.
De plus, nous avons un dossier controllers qui contient tous les controllers de l'application. Ces controllers contiennent les méthodes à effectuer pour chaque

actions. Ils utilisent les objets du modèle qui sont définies dans le dossier modèles.
Puis le dossier public contient l'application côté client. Donc les différentes sources Handlebars qui seront traduites en HTML, le code CSS et le code Javascript de l'application.
Chaque dossier contient plusieurs fichiers qui représentent les composants importants de l'application. Par exemple, le dossier modèles contient des fichiers comme user.js ou encore tab.js. De même, le controller qui intéragit avec la partie user se situe dans le dossier controllers et s'intitule user.js (Ceci est une convention de nommage et de structure utilisée par la communauté NodeJS).
Enfin, à la racine de l'application il y a le fichier principal pour lancer le serveur : app.js ainsi que différents fichiers de configurations pour des applications telles que Grunt (Gruntfile), Compass (config.rb) ou NodeJS (package.json).

2.2 Choix

La technologie NodeJS est une technologie très intéressante car elle possède une forte communauté et les frameworks qui en découlent sont très puissants.
Nous avons utilisé Grunt qui est un gestionnaire de tache, ce programme permettait de lancer le serveur à l'aide d'une simple ligne de commande qui se résumait à un mot : grunt. De plus, il permettait de notifier le client à chaque modification du serveur, ainsi à chaque modification du serveur, le client se raffraichit automatiquement.
Nous avons utilisé le framework Compass pour le CSS, ce framework possède des mixins qui permettent d'améliorer la vitesse de production. De plus, cette surcouche au CSS ajoute la possibilité d'utiliser les variables et de structurer le code plus simplement.
De plus, ce framework a été combiné à celui de Bootstrap pour permettre la gestion du responsive design.
Du côté du modèle, Sequelize est un ORM jeune mais c'est le seul ORM convenable sur NodeJS, il permet donc d'utiliser le pattern ActiveRecord sans trop de problème. (Il crée parfois des colonnes inatendues dans les tables MySQL).
Pour la base de données en elle-même, nous avons utilisé MySQL, nous avons vu cette technologie à l'IUT et c'est pourquoi nous n'avons pas rencontré de problème avec celle-ci.
Pour générer la vue (le code HTML), nous avons utilisé HandleBarsJS qui est un moteur de templates très puissant et customisable. Ainsi nous avons ajouté des fonctions à ce dernier pour qu'il convienne mieux au projet.
Enfin, nous avons utilisé les websockets pour permettre la mise a jour en instantannée de certaines informations. Cette technologie a été difficile à mettre en place à cause de la gestion de variables de sessions. Il fallait donc envoyer en plus des informations de base, le cookie correspondant à la session.

2.3 Amélioration possible

Nous voulions utiliser le framework Ember ou Angular pour coder l'application web. Cependant, nous avons fait le choix d'utiliser jQuery et un moteur de template pour éviter d'apprendre encore une nouvelle technologie que nous ne métrisions pas.

Ainsi, les échanges entre le serveur et le client pourraient être améliorés grâce à l'utilisation d'un framework plus complexe.
Étant donné le temps passé à l'apprentissage de NodeJS, nous n'avons pas eu le temps de coder toutes les fonctionnalitées voulues au départ.
Par exemple, l'envoie de messages entre utilisateur et les notifications en instantannée ne sont pas implantées. Par ailleurs, les notifications en temps réel (à l'aide des websockets) nécessiteraient un serveur suffisamment puissant pour qu'elles puissent fonctionner sans problème.
Puis, l'affichage et l'écoute des partitions pourraient être améliorés. Nous voulons utiliser un framework déjà existant pour afficher les partitions et ensuite coder notre propre framework qui s'occupera de parser les informations des partitions disponibles sous le format JSON.

Au niveau de la sécurité, certains points sont à améliorer. Il faut augmenter les vérifications lors des envoies de partitions ou d'images de profil. De plus on pourrait aussi rajouter un jeton particulier à chaque formulaire de l'application, ce jeton permettrait de vérifier si le formulaire utilisé n'est pas un formulaire détourné.

La possibilité d'ajouter en favoris une partition est aussi une amélioration possible (mais non essentielle).

Table des matières