

# Rapport de conception

Adonis NAJIMI,  
Valentin STERN,  
Vincent ALBERT,  
Théo GERRIET

7 janvier 2014

# 1 Présentation du projet

## 1.1 Cahier des charges fonctionnel

Priorité haute

- Analyse et enregistrement des notes
- Envoi des partitions
- Accordeur
- Jouer une partition

Priorité moyenne

- Apprentissage de la composition

Priorité basse

- Analyse automatique d'une partition

## 1.2 Repartition des charges

Adonis NAJIMI : Toute l'application web

Valentin STERN : Model du client(50%)

Vincent ALBERT : Controleur et vue du client

Théo GERRIET : Model du client(50%)

# 2 Client C++

## 2.1 Présentation du client

La fonctionnalité majeure du client est de permettre à l'utilisateur de composer et enregistrer ses propres musiques en jouant avec sa guitare. Il n'aura qu'à jouer ce qu'il veut et l'application traitera les notes jouées et les enregistrera dans une partition. Il pourra par la suite modifier les notes de la partition en cas d'erreur.

Le client comporte d'autres fonctionnalités annexes non essentielles mais intéressantes cependant. L'utilisateur pourra, avant de commencer à jouer, accorder sa guitare afin de jouer le plus juste possible. Le client a pour but de permettre à l'utilisateur de composer ses chansons, et il y aura donc une partie permettant l'apprentissage des bases de la composition à la guitare de manière interactive. Il pourra apprendre à créer des accords, jouer sur différentes gammes ou encore apprendre le rythme.

Une fonctionnalité permettant d'envoyer directement sa partition au serveur web sera également présente afin de simplifier le partage de ses créations.

Enfin, l'application permettra aussi d'analyser une partition afin de trouver quelle est la gamme principale de celle-ci.

## 2.2 Description du contenu du modele

Cette partie de l'application est principalement basée sur l'analyse et l'enregistrement des notes jouées par l'utilisateur. Le principe est d'analyser en temps

réel les fréquences à la base du son de la guitare, et de faire en sort de constituer un accord avec les notes ayant le volume le plus fort et l'enregistrer. De plus, le modèle comporte également toutes les informations sur les préférences de l'utilisateur pour l'application, ainsi que des fonctions pour enregistrer les données. Nous avons choisi de sauvegarder les fichiers au format JSON.

Afin de jouer une partition, nous utiliserons des sons de guitare en midi d'une durée courte, que nous jouerons plusieurs fois afin de correspondre à la durée de la note sur la partition, il y aura également la possibilité de changer la note jouée pour une note qui correspond plus à sa propre guitare pour une meilleure expérience utilisateur.

Nous avons décidé d'utiliser la librairie FMOD Ex pour le modèle, car cette librairie est complète et fournit toute les fonctionnalités dont nous avons besoin. En effet, elle nous permet de lire et d'enregistrer des sons, et également de récupérer les fréquences d'un son grâce à la courbe de ce son. Elle utilise pour cela une tranformée de Fourier rapide.

Fréquences des hauteurs (en hertz) dans la gamme tempérée								
Note\octave	0	1	2	3	4	5	6	7
<b>Do</b>	32,70	65,41	130,81	261,63	523,25	1046,50	2093,00	4186,01
<b>Do# ou Réb</b>	34,65	69,30	138,59	277,18	554,37	1108,73	2217,46	4434,92
<b>Ré</b>	36,71	73,42	146,83	293,66	587,33	1174,66	2349,32	4698,64
<b>Ré# ou Mib</b>	38,89	77,78	155,56	311,13	622,25	1244,51	2489,02	4978,03
<b>Mi</b>	41,20	82,41	164,81	329,63	659,26	1318,51	2637,02	5274,04
<b>Fa</b>	43,65	87,31	174,61	349,23	698,46	1396,91	2793,83	5587,65
<b>Fa# ou Solb</b>	46,25	92,50	185,00	369,99	739,99	1479,98	2959,96	5919,91
<b>Sol</b>	49,00	98,00	196,00	392,00	783,99	1567,98	3135,96	6271,93
<b>Sol# ou Lab</b>	51,91	103,83	207,65	415,30	830,61	1661,22	3322,44	6644,88
<b>La</b>	55,00	110,00	220,00	440,00	880,00	1760,00	3520,00	7040,00
<b>La# ou Sib</b>	58,27	116,54	233,08	466,16	932,33	1864,66	3729,31	7458,62
<b>Si</b>	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07	7902,13

FIGURE 1 – Toutes les frequences possibles

## 2.3 UML du modele de l'application

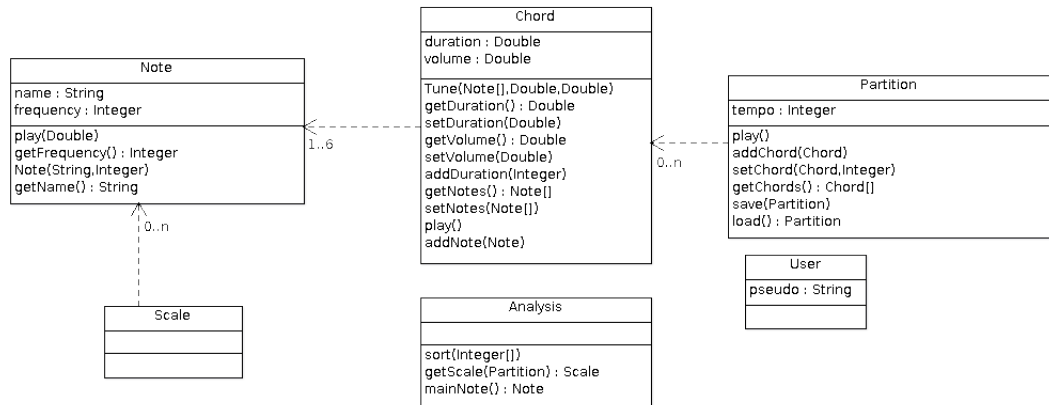


FIGURE 2 – UML du modele

La classe Note représente une Note qui n'est pas jouée, c'est-à-dire qu'elle contient la définition d'une note seulement.

La classe Chord permet donc de représenter de 1 à 6 note jouée simultanément tel un accord à la guitare.

La classe Partition représente donc une partition comme l'utilisateur va la voir.

La classe Analysis propose différentes fonctions essentielles à l'application. Cette classe propose également des fonctionnalités pour l'apprentissage de la composition de musiques.

La classe Scale représente une gamme. Une gamme est une suite de notes, et la classe est donc composée d'un certain nombre de note. La classe User est présente pour enregistrer le pseudonyme de l'utilisateur pour pouvoir envoyer des partitions sur le site. Lorsque l'utilisateur va jouer une note, le logiciel va récupérer toutes les fréquences entrant par le micro et les trier en fonction de leur volume. Il est donc aisé par la suite de trouver quelles sont les notes qui sont le plus jouées.

## 2.4 Algorithmes

Structures de donnees utilisees dans les algorithmes :

Note < frequence : entier, nom : chaine >

EntreeMicro : Type correspondant à l'entrée micro de l'ordinateur. Il permet de récupérer les fréquences à un certain instant

```

fonction recupererNote(frequence : entier, notes : Note[0..n], n : entier) : Note
debut
min <- 0
max <- n
  
```

```

trouve <- faux
Tant que min <= max et non trouve faire
indice <- (max + min) /2
frequenceNote <- notes[indice].frequence
Si frequence = frequenceNote
alors
retour <- notes[indice]
trouve <- vrai
sinon
Si frequence < frequenceNote
alors
max <- indice
sinon
min <- indice
fsi
fsi
ftantque
Si non trouve
alors
retour <- notes[min]
fsi
retourne retour
fin

```

lexique  
 frequence : entier : Fréquences de la note à récupérer  
 notes : Note[0..n] : Toutes les notes possibles  
 n : entier : Nombre de notes possibles  
 max : entier : indice maximal dans le tableau  
 min : entier : indice minimal dans le tableau  
 trouvé : booléen : Booléen à vrai si on a trouvé la note (avec exactement la même fréquence)  
 indice : entier : Indice en cours dans le tableau  
 frequenceNote : entier : Frequence de la note en cours d'analyse  
 retour : Note : Note correspondant à la fréquence

fonction trier(frequences : tableau entier[0..n], nentier)

```

fonction montrerNote(mic : entreeMicro)
debut
accorder <- vrai
Tant que
freqs <- recupererFrequences(mic)
freqs <- trier(freqs)
ecrire recupererNote(freqs[0])
ftantque
fin

```

lexique  
mic : entreeMicro : Entrée micro utilisée  
accorder : booleen : Booleen à vrai si on continue à accorder. Il peut être changer grâce à un clic dans l'application  
freqs : freqs : tableau entier[0..n] : Frequences envoyées par mic

```

fonction enregistrer(mic : entreeMicro, seuilHaut : entier, seuilBas : entier, tempo : entier) : Partition
debut
enregistrer <- vrai
j <- 0
Tant que enregistrer faire
frequences <- recupererFrequences(mic)
frequences <- trier(frequences)
complet <- faux
fini <- faux
i <- 0
Tant que non complet et non fini faire
frequence <- frequences[i] si frequence > seuilHaut
alors
ajouter(freqs, accord)
j <- j + 1
sinon
si frequence > seuilBas
alors
ajouterTemps(retour[j], tempo / 16)
sinon
fini <- vrai
fsi ftantque
i <- i + 1
attendre(tempo / 16)
ftantque
retourne retour
fin

```

Nous attendons pendant tempo / 16 car c'est la plus petite unité de temps représentable dans une partition. L'utilisateur doit rentrer le tempo dans lequel il compte jouer.

## 2.5 UML de l'interface graphique

L'interface graphique sera codée en C++ et respectera le modèle MVC. Ainsi seul le contrôleur modifiera le modèle, tandis que la classe NoSkin, représentant l'interface, ne fera qu'afficher les données. Ces deux classes sont donc représentées sur l'UML et sont complémentées par des classes complémentaires telles que la classe FormatException qui permet de gérer plus finement la conversion JSON/Objet. Les classes TabWidget, Options et Chord seront utilisées par NoSkin pour créer

les fenêtres de l'accordeur, des paramètres et le widget particulier de la tablature.

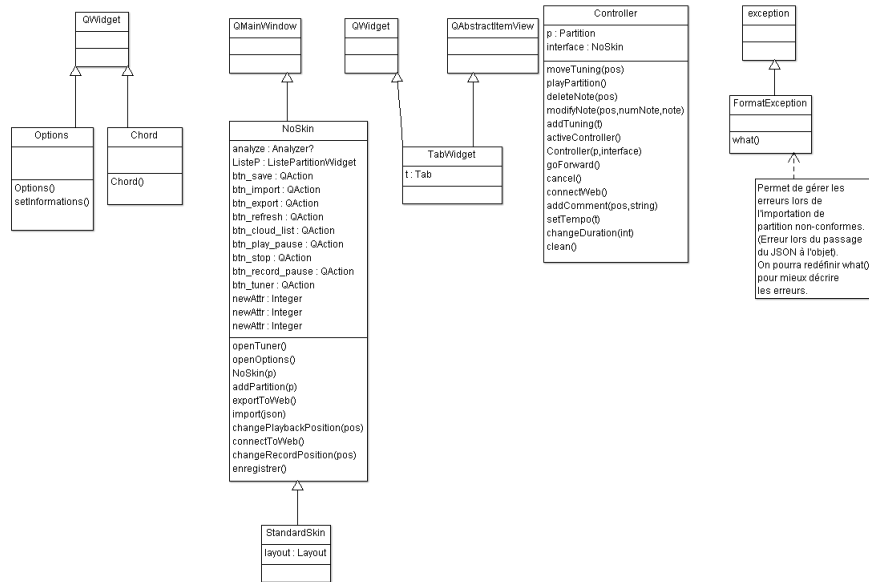


FIGURE 3 – UML de l'interface graphique

## 2.6 Maquettes du client

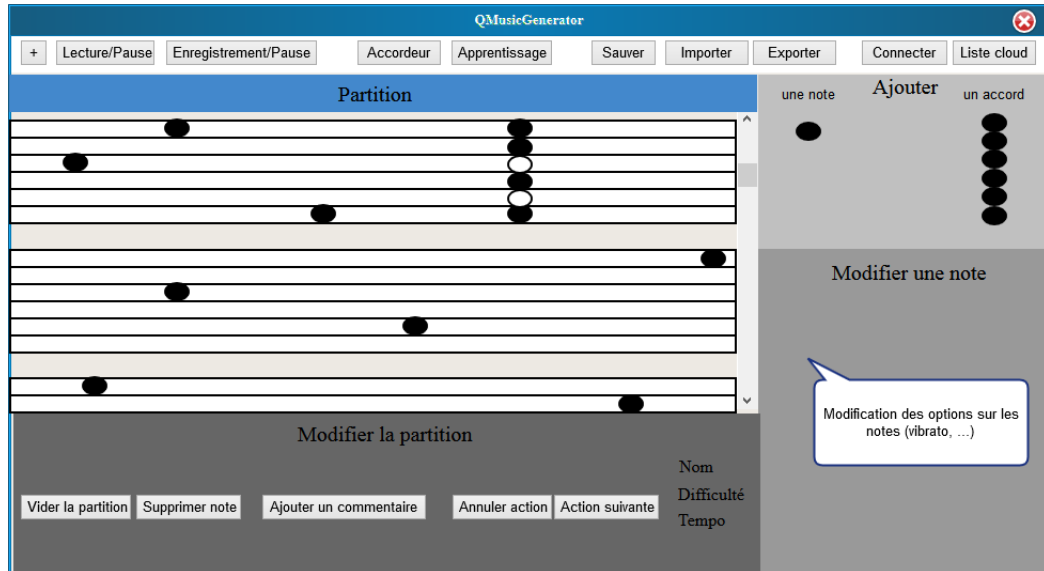


FIGURE 4 – Page principale du client

Cette fenêtre est la fenêtre principale de notre client. A partir de celui-ci, il sera possible de créer des partitions vierges, d'en ouvrir des préexistantes, d'en lire et d'enregistrer des notes à partir de l'entrée audio ainsi que de les sauvegarder.

Mais il sera aussi possible d'ouvrir un accordeur, de télécharger des partitions hébergées sur le site web, et d'en uploader.

Il sera en effet possible de connecter directement le client avec le site à l'aide des identifiants de l'utilisateur.

De plus, il y aura aussi évidemment toutes les options permettant de modifier la partition en cours, tels que son nom, sa difficulté, son tempo, ...

Enfin, on pourra modifier la partition manuellement en déplaçant les notes ou en en ajoutant.



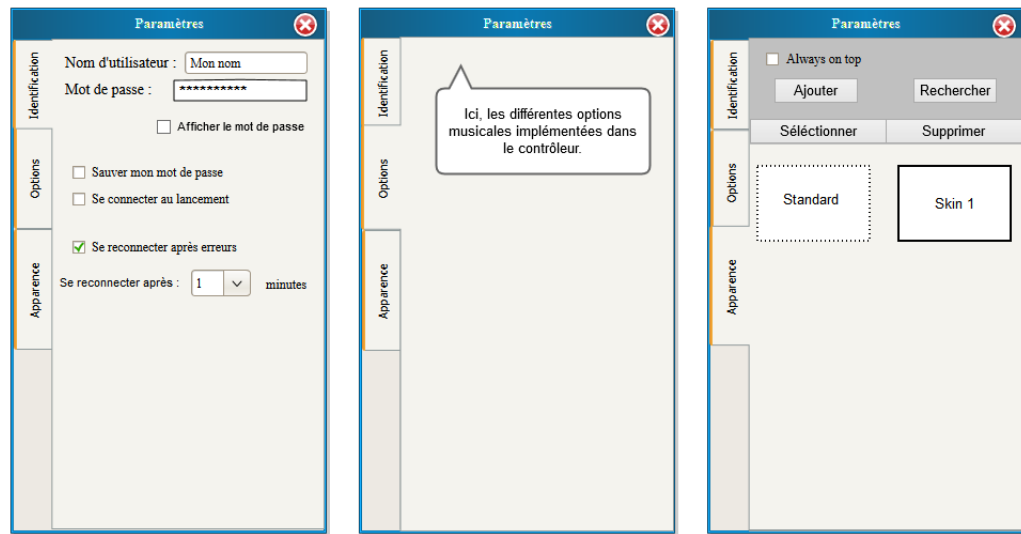


FIGURE 5 – Fenêtre des paramètres du logiciel

Cette fenêtre permettra de modifier les paramètres du logiciel. Cela recouvre les paramètres d'authentification, ceux spécifiques à la synthèse de musique en elle-même, mais aussi l'apparence du client via l'utilisation de skins.

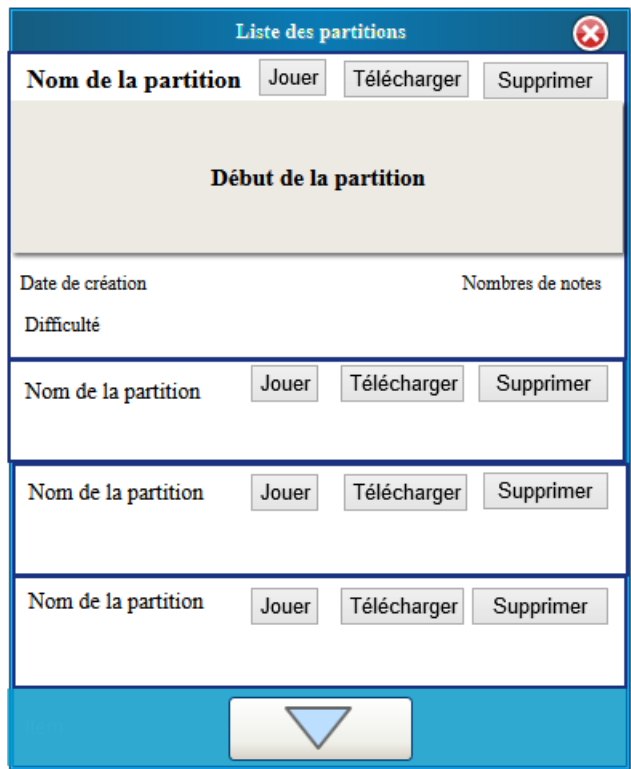


FIGURE 6 – Fenêtre des listes de partitions hébergées sur le site web  
 Cette fenêtre listera toutes les partitions que l'utilisateur a uploadé sur le site internet si celui-ci est connecté.  
 A partir de cette fenêtre, il pourra soit les supprimer du site web, soit les télécharger, ou encore les jouer.

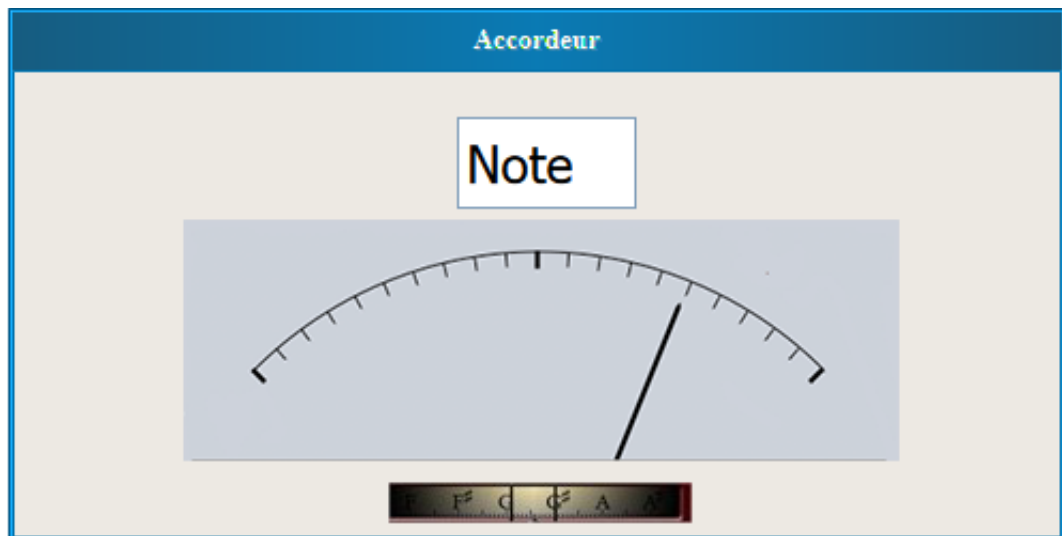


FIGURE 7 – Accordeur

Cette fenêtre affichera un accordeur qui analysera l'entrée sonore pour permettre à l'utilisateur de réaccorder son instrument directement depuis le logiciel.

## 2.7 Liaison avec l'application web

Afin de communiquer avec l'application web, le client sera doté d'un module réseau qui enverra et recevra des requêtes http. Les partitions seront envoyées et reçues en format JSON afin de faciliter ces échanges.

Le JSON aura format semblable à ceci :

```
{
  "Name" : "string",
  "Tempo" : 0,
  "private" : true/false
  "tunes" : [{
    "volume" : 0-1,
    "nb_notes" : 0,
    "temps" : "croche",
    "notes" : [{
      "name" : "do-la-...",
      "frequence" : 0
    }]
  }]
}
```

# 3 Application Web

## 3.1 Maquettes d'écran

Les différentes maquettes du site permettent d'améliorer la vision des besoins de l'application. Ces visuels ne sont pas définitifs et sont simplement là pour

nous aider à mieux structurer l'application et organiser les objectifs nécessaires à la réalisation de celle-ci.

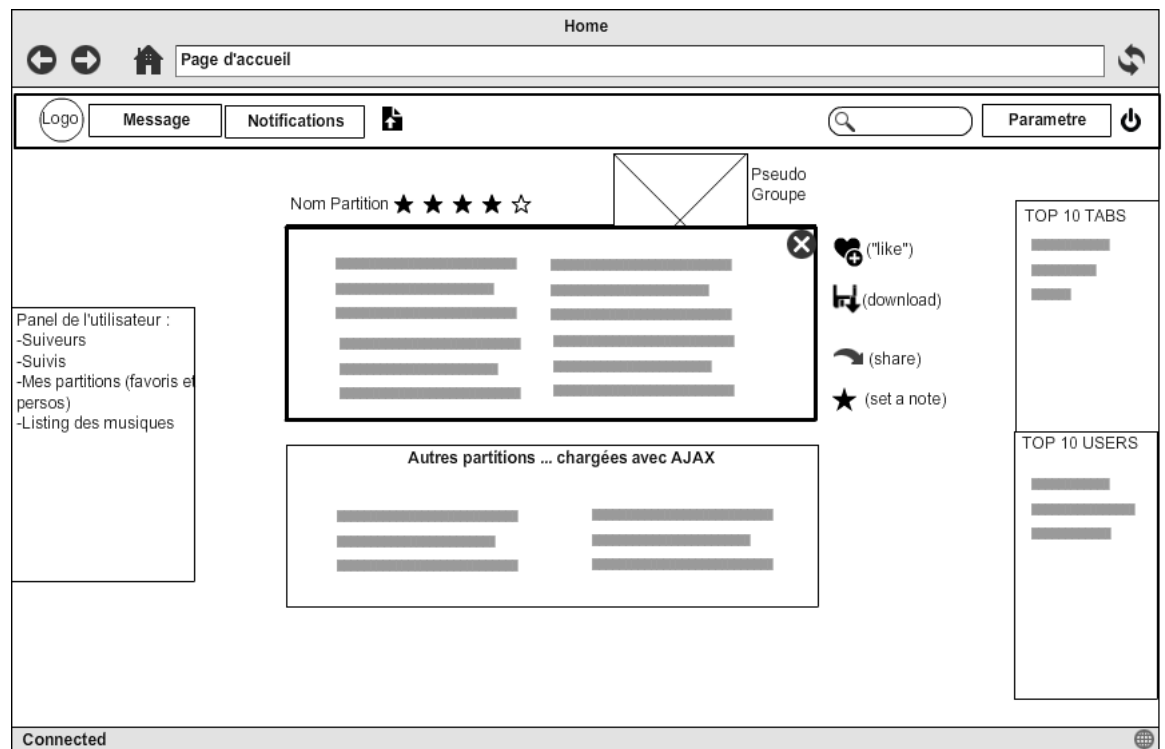


FIGURE 8 – Maquette de la page principale du site

La barre du haut est une barre de menu qui sera disponible sur toute les pages du site, elle n'est pas représentée sur toutes les maquettes par soucis de clarté. En effet, le principal sujet des maquettes suivantes n'est pas la présence de la barre de menu mais le contenu de celles-ci

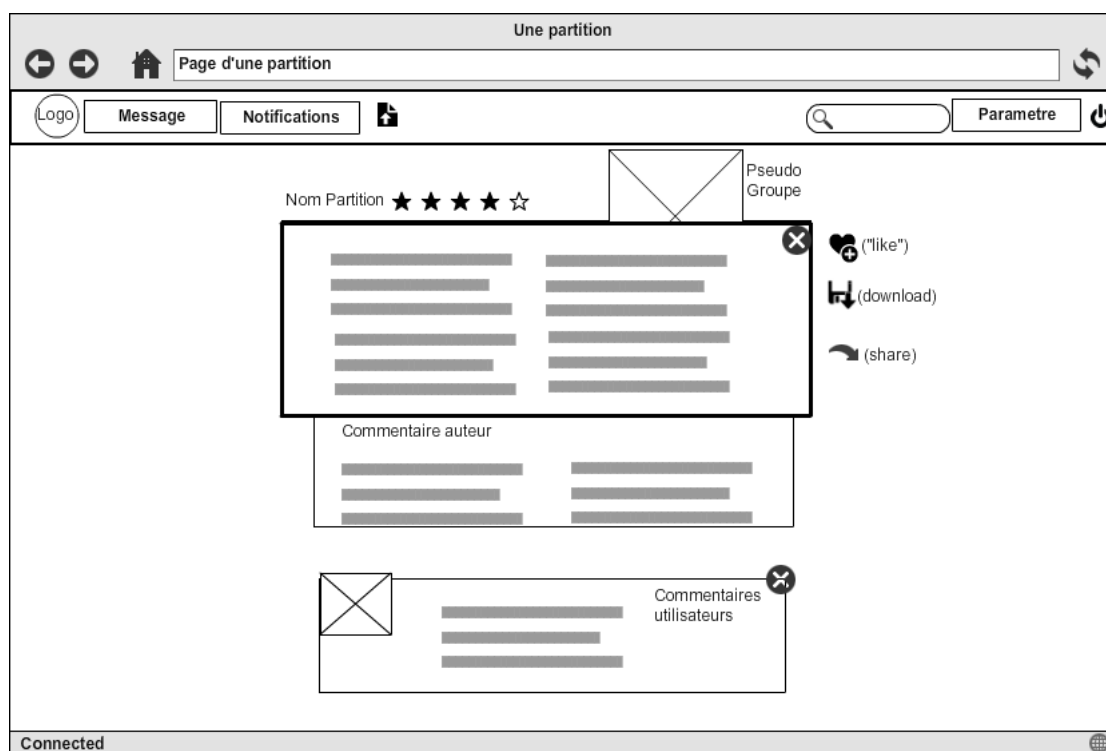


FIGURE 9 – Page principale d’une partition de musique

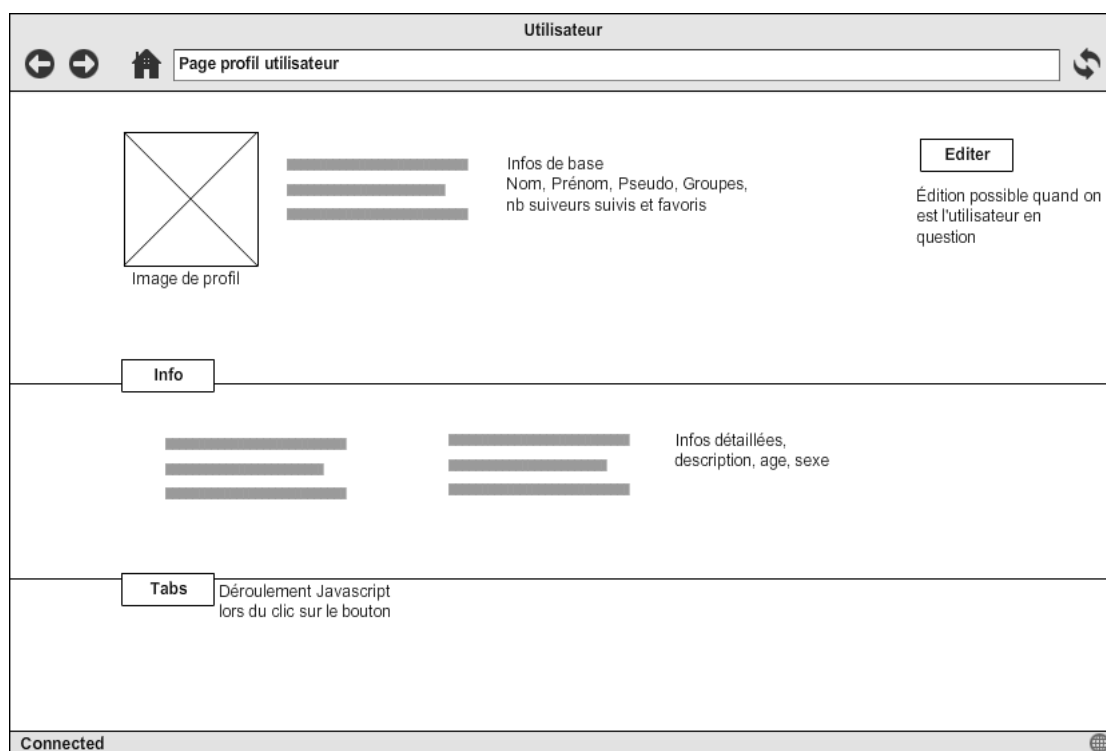


FIGURE 10 – Page de profil utilisateur

Cette page permettra à l'utilisateur concerné de modifier ses informations s'il le souhaite.

Les autres utilisateurs verront sur sa page de profil les informations rendues visibles par l'utilisateur en question.

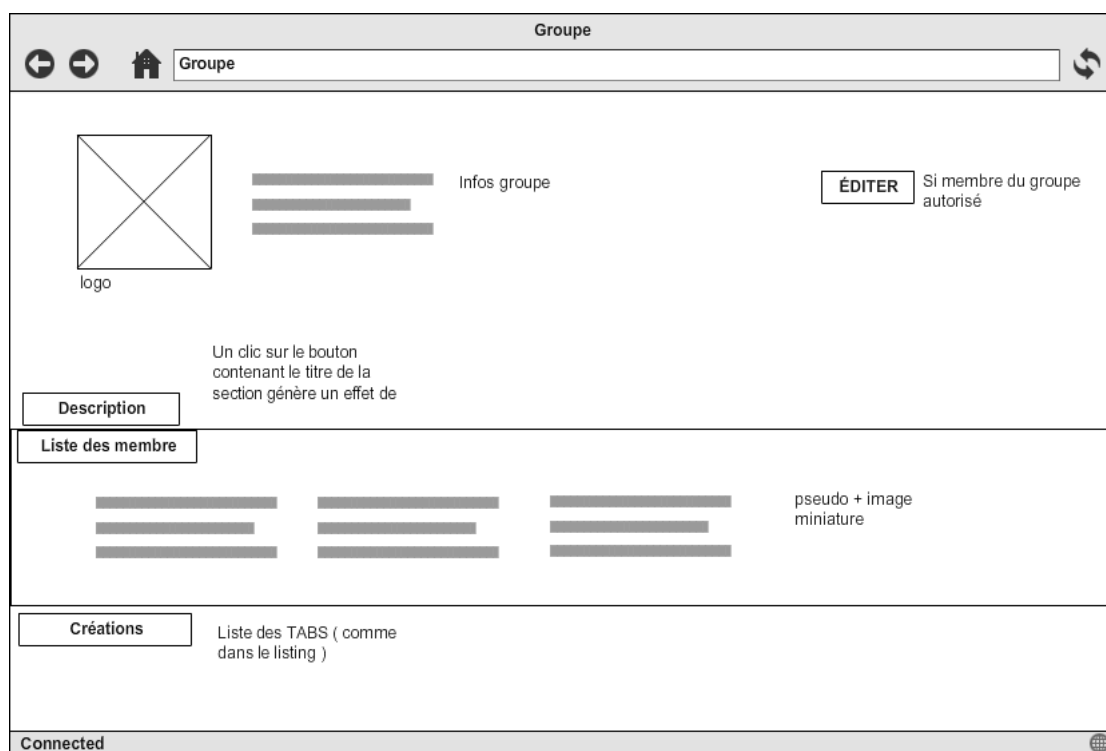


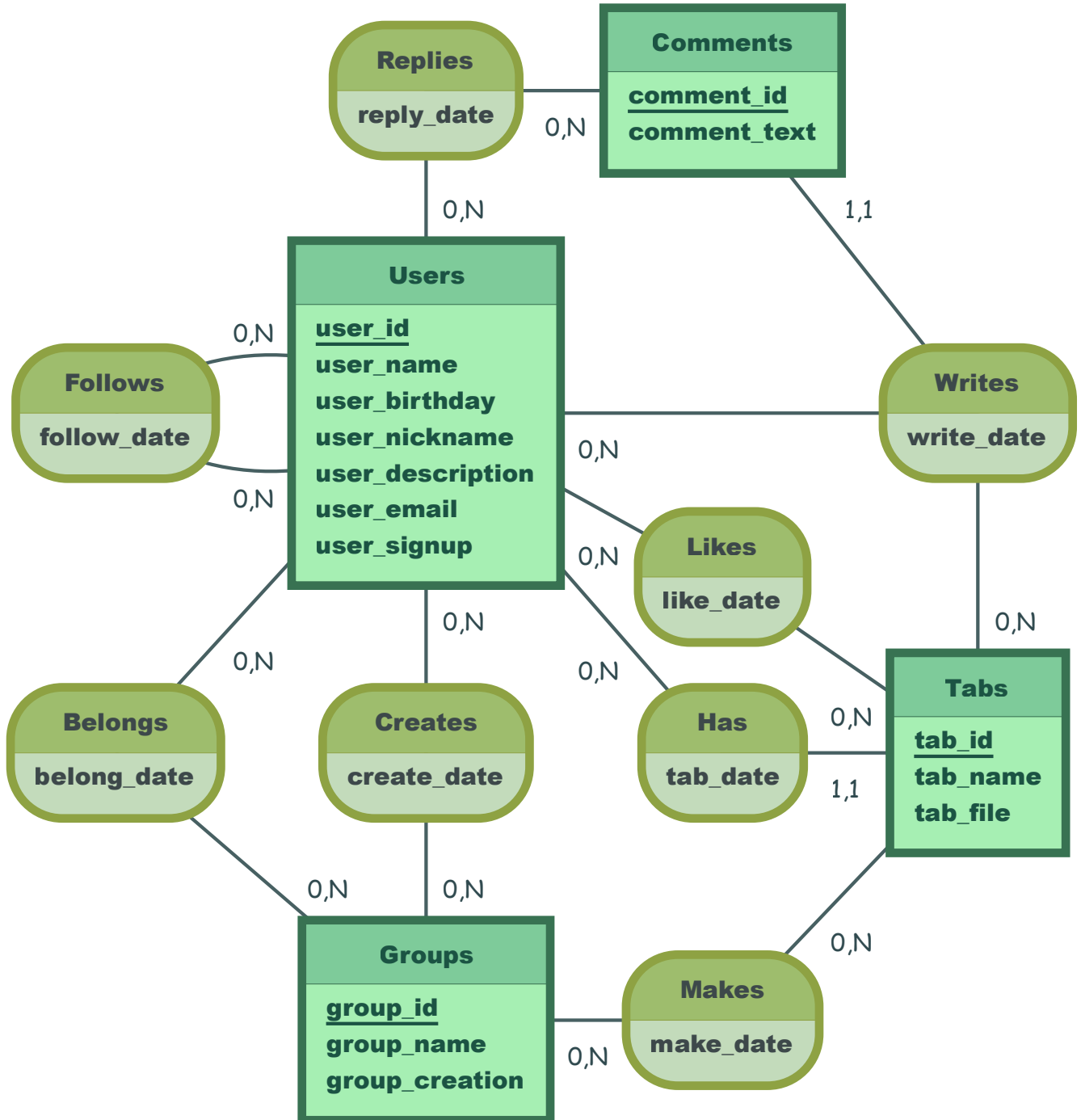
FIGURE 11 – Page d'un groupe de musique  
 Cette page permettra de modifier le groupe si l'utilisateur en possède les droits.  
 Cette page contiendra la liste des membres du groupe ainsi que les différentes  
 partitions liées au groupe concerné.



FIGURE 12 – Liste de toutes les partitions  
Le listing des partitions permettra une recherche plus approfondie qu'avec la recherche disponible dans la barre de menu (la barre du haut).



### 3.2 Modèle Merise MCD



### 3.3 Modèle Merise MLD

Replies (#user\_id, #comment\_id, reply\_date)  
Comments (comment\_id, comment\_text, #tab\_id, #user\_id, write\_date)  
Follows (#user\_id, #user\_id, follow\_date)  
Users (user\_id, user\_name, user\_birthday, user\_nickname, user\_description, user\_email, user\_signup)  
Likes (#tab\_id, #user\_id, like\_date)  
Belongs (#user\_id, #group\_id, belong\_date)  
Creates (#group\_id, #user\_id, create\_date)  
Tabs (tab\_id, tab\_name, tab\_file, #user\_id, tab\_date)  
Groups (group\_id, group\_name, group\_creation)  
Makes (#group\_id, #tab\_id, make\_date)

### 3.4 Structure de l'application

La structure du site suivra le modèle MVC.

Des tests unitaires seront utilisés pour s'assurer du bon fonctionnement de l'application.

Le site sera à l'aide d'un framework MVC Javascript (Express) pour gérer la partie Serveur et un autre framework complémentaire qui sera utilisé pour les vues et les controllers en particulier (AngularJS).

Utiliser Javascript pour coder toute l'application web aussi bien du côté client que du côté serveur nous permettra d'avoir une première approche avec cette technologie qui devient de plus en plus demandée aujourd'hui. Ce qui rend donc ce projet très enrichissant pour notre avenir professionnel.

Le site sera donc divisé en trois principales parties :

- Views
- Controllers
- Models

Plus une partie qui contiendra les routes (ie les chemins d'accès aux différents controllers)

Views :

- home => représentera l'accueil du site -compte => Profil utilisateur
- groupe => Profil du groupe
- listing => Listing des différents partitions avec option de recherche
- topbar => barre du haut qui sera incluse sur toutes les pages
- tabpanel => page principale d'une partition

Controllers :

Les controllers seront utilisés pour faire le lien entre le modèle et les différentes vues. De plus ils permettront de gérer les fonctionnalités spécifiques au site.

- home.js => controller général
- profil.js => controller du compte qui gère le profil utilisateur
- groupe.js => controller qui gère les groupes utilisateurs
- listing.js => controller qui gère la liste des partitions ainsi que la fonction recherche dans celles-ci

- topbare.js => permet de gérer la barre de menu présente sur le haut du site
- partition.js => permet de gérer les fonctionnalités d'une partition en particulier
- partition-page.js => controller qui gère la page d'une partition

Les modèles suivent l'architecture Active Record.  
Ils seront donc liés à la base de données (cf MCD et MLD).  
Une table équivaut donc à un module JavaScript.  
Ces modules contiendront toutes les méthodes nécessaires au traitement et à la modification du modèle (findById,insert,delete,update ...) et suivront donc l'approche vue en cours.

Routes :

Chaque lien est une route qui peut contenir une action et qui actionne une méthode du controller lié à la route.  
Une route est donc un chemin représenté par un URL.  
Ces routes sont donc le coeur du fonctionnement de l'application Web.

## Table des matières