Rapport de conception

Adonis Najimi,
Valentin Stern,
Vincent Albert,
Théo Gerriet

5 janvier 2014

1 Présentation du projet

2 Client C++

2.1 Présentation du client

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.

2.3 UML du modele de l'application

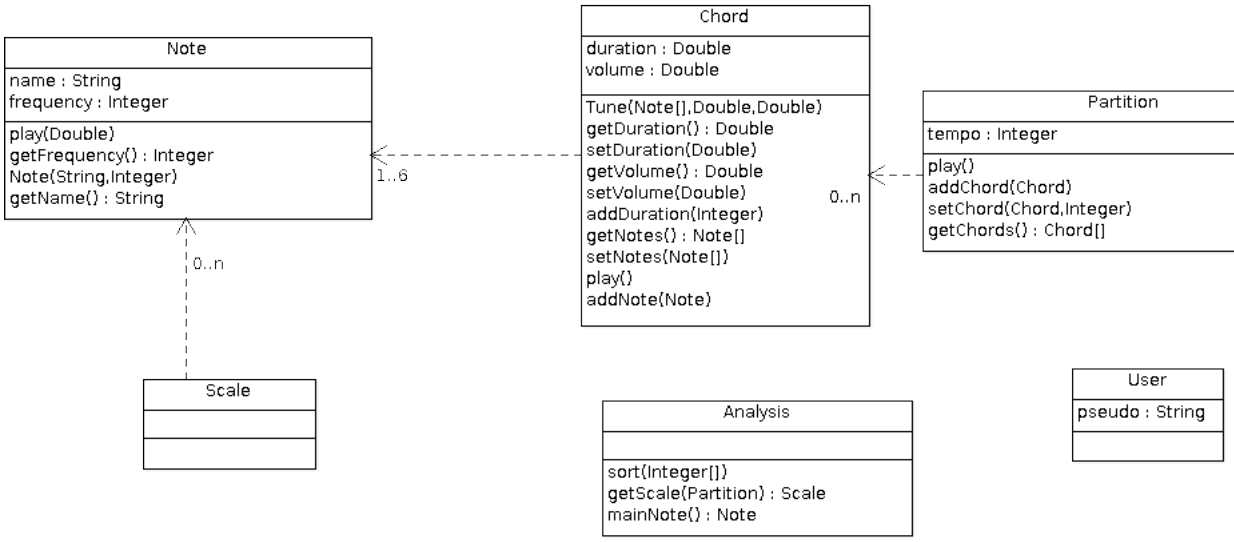

FIGURE 1 – UML du modele

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

fonction analyser(frequences : tableau entier[0..n], n : entier, seuilMin : entier) : tableau Accord
debut
frequences <- trier(frequences)
i <- 0
complet <- faux
Tant que i < n et non complet faire
freq <- frequences[i]
Si freq > seuilMin et non present(retour, freq)
alors
ajouter(retour, freq)
Si taille(retour) > 5
alors
complet <- vrai
fsi
fsi
ftantque
retourne retour
fin

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


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

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


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.

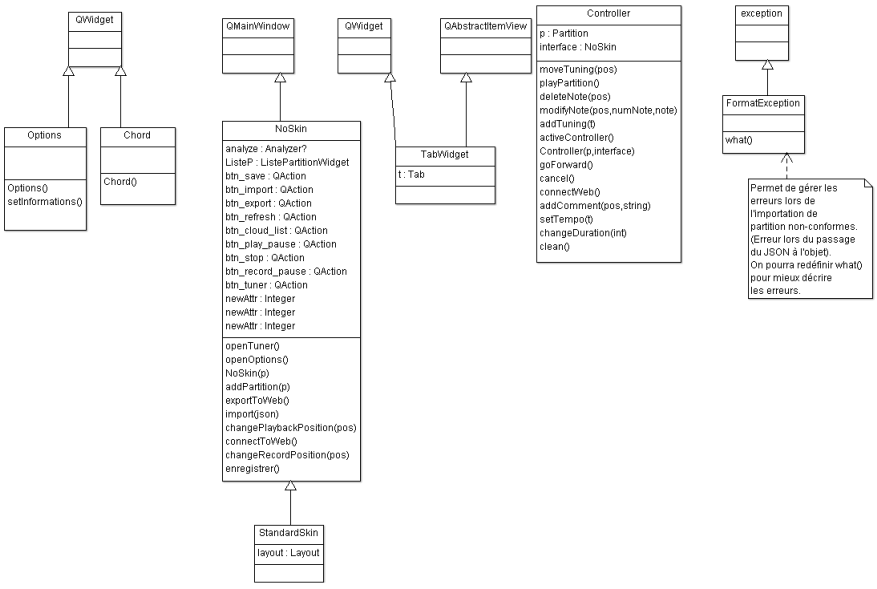

Figure 2 – UML de l'interface graphique

2.6 Maquettes du client

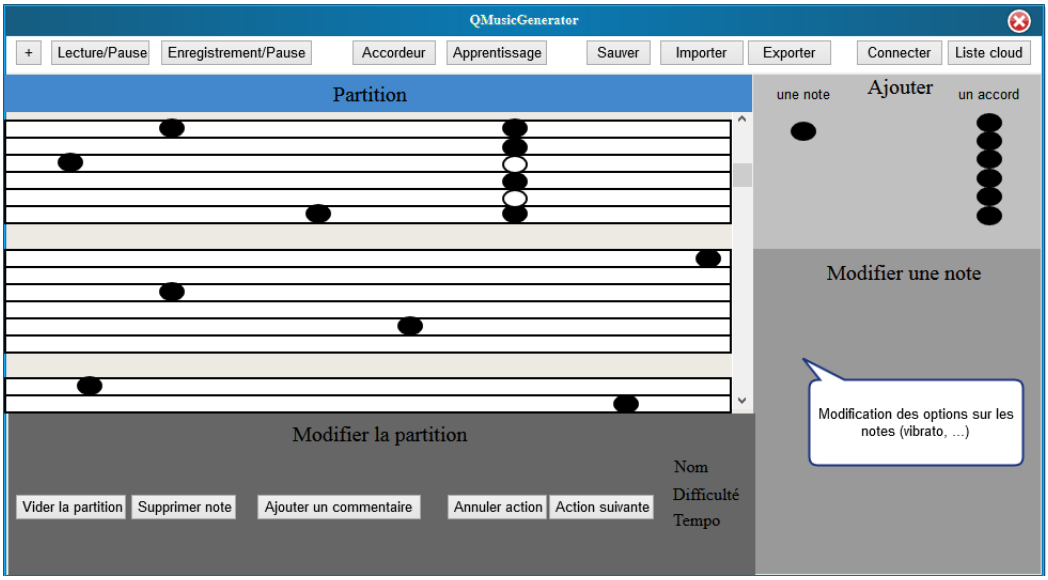

FIGURE 3 – Page principale du client

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.

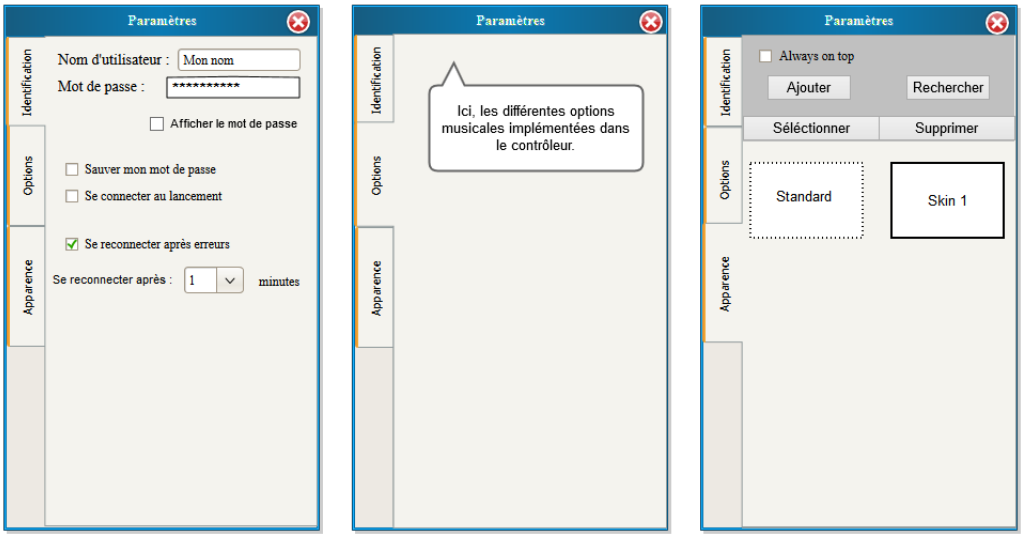

FIGURE 4 – Fenêtre des paramètres du logiciel
Cette fenêtre permettra de modifier les paramètres du logiciel.
Cela recouvre les paramètres d'authentification, ceux spécifiques à la synthèse de musique en elle-même, mais aussi l'apparence du client via l'utilisation de skins.

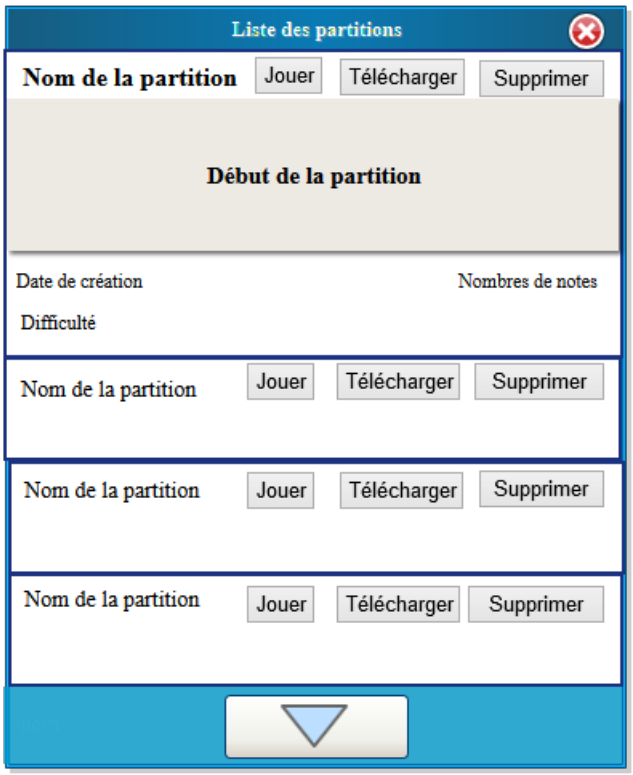

FIGURE 5 – Fenêtre des listes de partitions hébergées sur le site web
Cette fenêtre listera toutes les partitions que l'utilisateur a uploadé sur le site internet si celui-ci est connecté.
A partir de cette fenêtre, il pourra soit les supprimer du site web, soit les télcharger, ou encore les jouer.

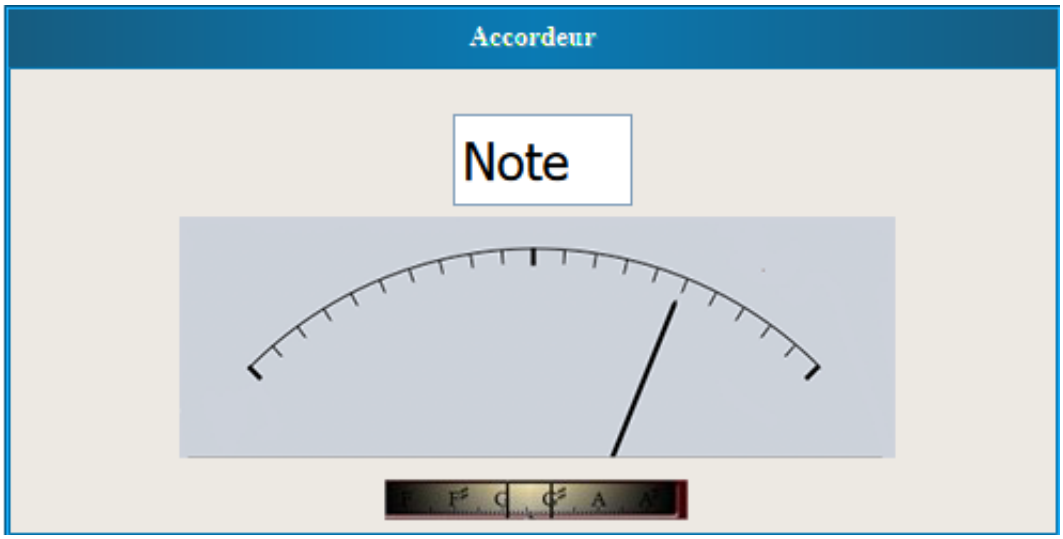

FIGURE 6 – Accordeur
Cette fenêtre affichera un accordeur qui analysera l'entrée sonore pour permettre à l'utilisateur de réaccorder son instrument directement depuis le logiciel.

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 differentes 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 7 – 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 8 – Page principale d'une partition de musique

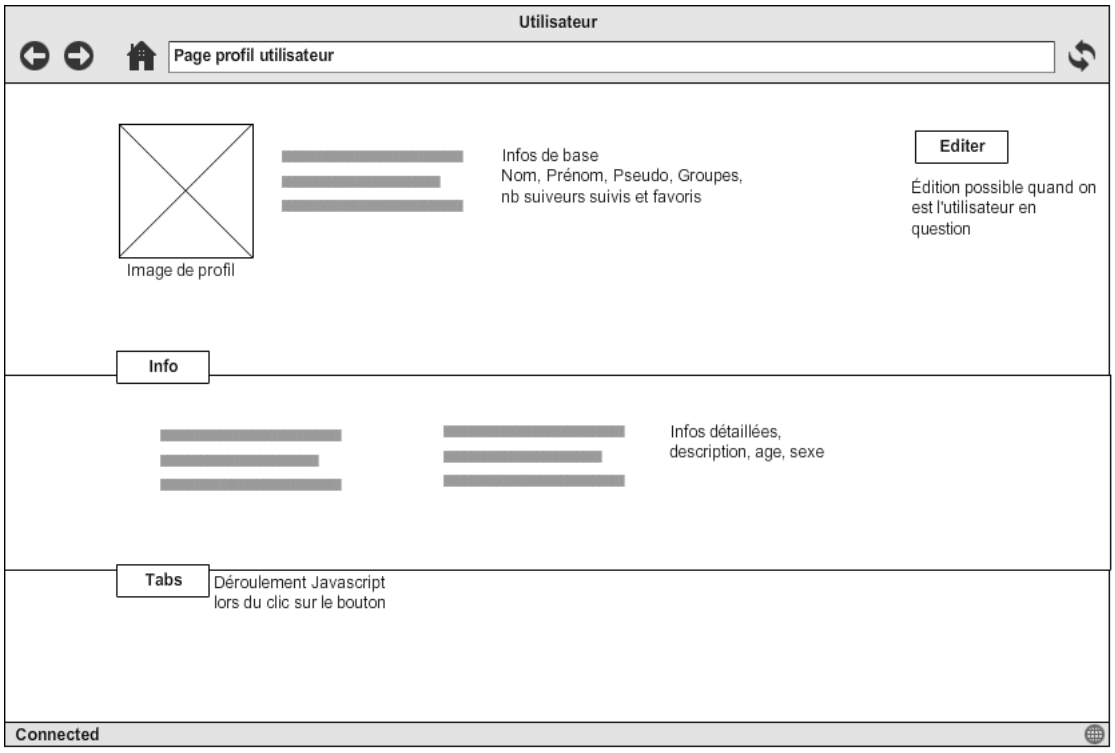

FIGURE 9 – Page de profil utilisateur
Cette page permettra à l'utilisateur concerné de modifier ses informations s'il le souhaite.
Les autres utilisateurs verront sur sa page de profil les informations rendues visibles par l'utilisateur en question.

FIGURE 10 – 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.

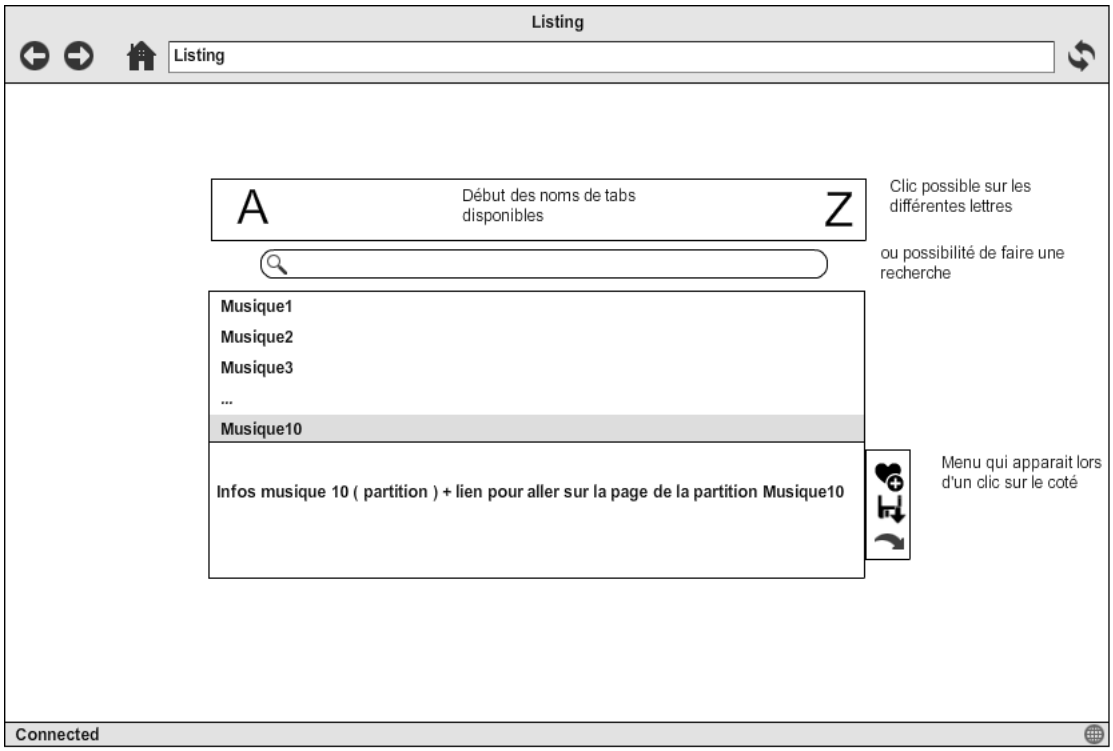

FIGURE 11 – Liste de toutes les partitions
Le listing des partitions permettra une recherche plus approfondie qu'avec la recherche disponible dans la barre de menu (la barre du haut).

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

—Structures— views : -home //invité ou non -compte // config ou non -groupe // config ou non -listing -barre en haut ? (gérer upload) -page partition -partition elle même

controllers (gèrent les animations et DATA ?) ANGULAR : home.js || controlleur général compte.js || controlleur du compte => gère les effets groupe.js || controlleur gère effet listing.js || gèrent effet bare.js partition.js partition-page.js

home.js => gestion des animations + gestion des datas à afficher

DATA => voir base de données Active record => une table = un module -methode de base (findAll,byId,insert,update,delete)+ sorted by ...

les routes express gèrent les chemins de navigations

Routes : chaque lien est une route qui peut contenir une action et qui actionne une méthode du controller lié à la route

Table des matières