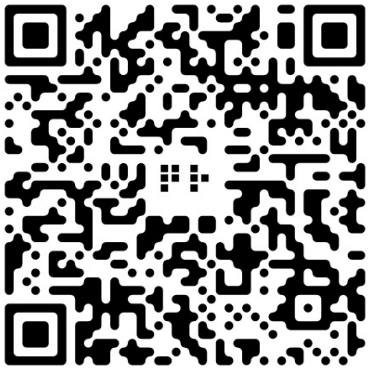

Développement d'un couple d'applications bureau et mobile

Génération et lecture de QR Codes pour l'institut Montéclair

David Dembele
Alassane Diop
Jules Leguy
Rahmatouwalet Mohamedoun

Décembre 2017

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Dans le cadre du module Concrétisation Disciplinaire de notre première année de Master 1 mention Informatique à l'Université d'Angers, nous avons développé une application de bureau et poursuivi le développement d'une application mobile.

Ces applications répondent à une demande de l'Institut Montéclair à Angers, dans le but d'apporter une dimension ludique à l'enseignement pour les mal-voyants et non-voyants inscrits à l'institut.

Ce projet a été encadré par Corentin Talarmain et Thomas Calatayud, deux étudiants en deuxième année de Master, dans le cadre de leur module Gestion de Projet.

L'objectif du projet est de fournir la possibilité aux transcripteurs de l'institut un moyen de générer des QR Codes contenant du texte et des sons et pouvant être interprétés par une application mobile. Une première version de l'application mobile avait déjà été développée par Corentin Talarmain lors de son TER [1] de fin de première année de Master. Celle-ci fournissait déjà la possibilité de lire avec une voix de synthèse du texte contenu dans des QR Codes.

1.2 Objectifs

Les transcripteurs souhaitaient néanmoins étendre les fonctionnalités de l'application mobile initiale, qui présentait quelques inconvénients limitant son utilisation. Ne présentant que la possibilité de lire des QR Codes, elle imposait le passage par des sites internet pour en générer. On ne pouvait ainsi pas contrôler la taille maximale des QR Codes générés, parmi lesquels certains étaient trop gros pour être détectables par l'application. En outre, elle ne fournissait pas la possibilité de lire des sons à partir de données contenues dans un QR Code.

Les transcripteurs de l'institut nous ont également fourni deux demandes précises autour desquelles nous avons structuré notre projet. La première demande était de pouvoir utiliser les QR Codes comme légendes de cartes de géographie pour des lycéens. Cela implique de pouvoir gérer la lecture de QR Codes adjacents dans un ordre prédéfini, peu importe l'ordre de lecture effectif.

L'autre demande était de pouvoir utiliser des QR Codes pour lire des sons dans les pages d'un livre pour enfants, sans avoir nécessairement accès à internet au moment de la lecture. Pour cela, un ou plusieurs QR Codes sur la page de couverture doivent permettre le téléchargement de tous les sons contenus dans le livre.

En outre, un espace central contenant du texte en braille devait pouvoir être inséré au centre des QR Codes, pour permettre aux non-voyants de les localiser, ces caractères étant imprimés avec une encre spéciale gonflant à température élevée.

1. Travail Encadré de Recherche

Chapitre 2

Conception

2.1 Types de QRCodes

À partir des contraintes définies dans les cahier des charges (voir 1.2), il a fallu définir précisément quels types de QR Codes allaient être crées. Une première version de notre architecture collait aux besoins de l'Institut Montéclair qui nous ont été transmis. Elle comprenait des QR Codes pour les cartes géographiques, des QR Codes pour les pages d'un livre et des QR Codes pour les pages de couverture des livres. Mais cette architecture s'est avérée trop spécifique et empêchait les transcripteurs de l'institut de créer des QR Codes pour d'autres supports.

Nous avons donc créé une seconde version qui respectait les contraintes transmises par l'institut tout en étant la plus généraliste possible, afin d'être utilisable dans tous les contextes compatibles avec les contraintes transmises. Cette architecture est composée de QR Codes contenant un ensemble de textes et de fichiers et pouvant être organisés en familles afin d'être lus dans un ordre prédéfini (les QR Codes atomiques), et de QR Codes contenant un ensemble de liens vers des sons devant être téléchargés sans être joués (les QR Codes ensembles).

2.2 Représentation des données

Les QR Codes devant stocker plus d'informations que dans la version initiale de l'application mobile, il nous a fallu définir une structure de données pour les représenter. Cette représentation s'est construite parallèlement à l'élaboration des types de QR Codes (voir 2.1) et à celle du Modèle (voir 3.1.2), afin d'assurer la cohérence de l'ensemble.

Dans le but de minimiser la quantité d'informations contenue dans le QR Code, nous nous sommes appuyés sur un modèle séparant les données nécessaires à la lecture par l'application mobile et les données nécessaires seulement au chargement à l'identique du QR Code après son enregistrement par l'application de bureau.

Pour comprendre, prenons l'exemple d'une page d'un livre pour enfant décrivant des animaux. Afin de rendre cette page plus ludique pour les non-voyants, les transcripteurs de l'institut souhaiteraient coller sur cette page un QR Code pour chaque animal décrit. Ces QR Codes contiendraient chacun un petit texte explicatif et le cri de l'animal. Ils souhaiteraient qu'ils soient lus dans un ordre prédéfini, et qu'un texte en braille représentant les lettres *QR* soit présent au centre de chacun des QR Codes. De plus, le QR Code devrait être bleu et le texte central en braille rouge afin qu'il puisse être imprimé en relief.

Parmi ces données, certaines ne sont pas indispensables à l'application mobile pour interpréter le QR Code. Elle n'a par exemple pas besoin de connaître le texte en braille ou le nom du fichier contenant le cri de l'animal. Ces données seront en revanche nécessaires à l'application de bureau pour recharger les QR Codes à l'identique le jour où les transcripteurs voudront modifier la description de l'animal. Nous allons donc séparer ces données en deux parties : les données nécessaires à l'application mobile seront encodées dans le QR Code tandis que les données qui ne le sont pas seront insérées dans le fichier image, mais pas dans le QR Code en lui-même. Nous détaillons cette séparation dans le schéma suivant.

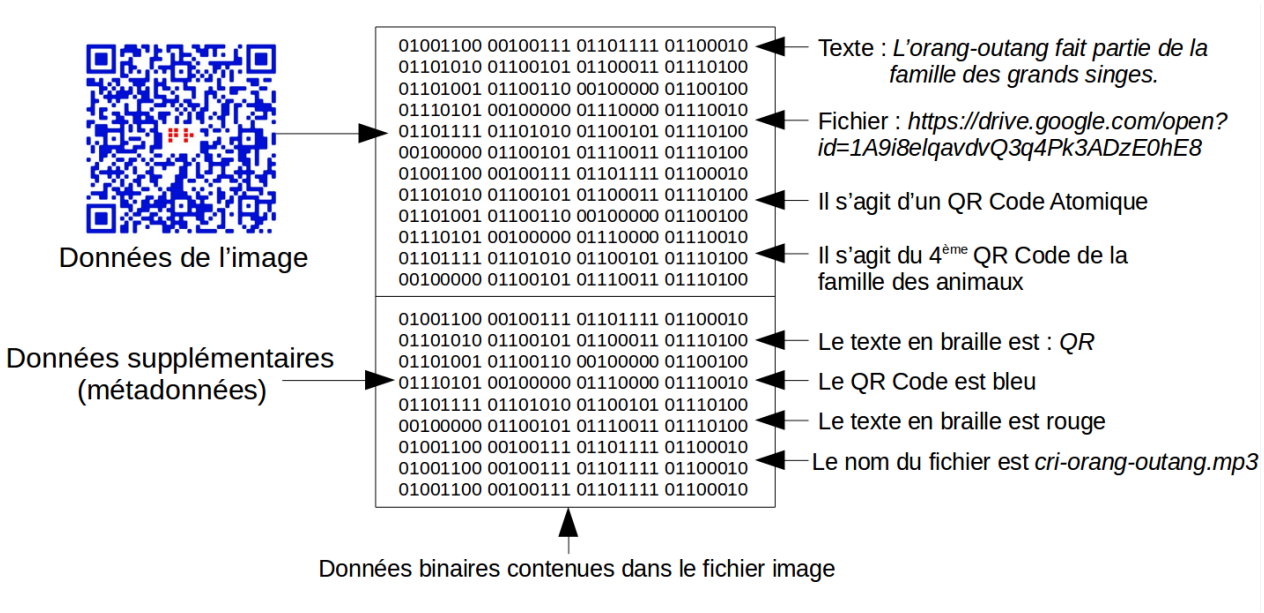

FIGURE 2.1 – Contenu d'une image d'un QR Code

Afin de stocker ces données de manière formelle, nous avons établi une structure de type XML [1] (voir 3.2.6). Elle fait apparaître clairement la dichotomie entre les données stockées dans le QR Code (noeud <donneesUtilisateur>) et les données stockées dans les métadonnées de l'image (noeud <metadonnees>).

Le type de QR Code (atomique ou ensemble) est indiqué par un attribut dans le noeud <donneesUtilisateur>. Ce noeud contient un noeud <contenu> contenant lui-même un ensemble de textes (<texte>) et de fichiers (<fichier>).

L'appartenance à une famille (voir 2.1) de QR Codes est indiquée par la présence d'un noeud <famille> ayant pour attributs le nom de la famille et la place du QR Code.

La représentation formelle de l'exemple du QR Code orang-outang est visible dans la figure ci-dessous.

```
<qrcode>
  <donneesutilisateur type="atomique">
    <contenu>
      <texte>L'organg-outang fait partie de la famille
             des grands singes.</texte>
      <fichier url="1A9i8elqavdvQ3q4Pk3ADzE0hE8"/>
    </contenu>
    <famille nom="animaux" ordre="4"/>
  </donneesutilisateur>
  <metadonnees>
    <fichiers>
      <fichier url="1A9i8elqavdvQ3q4Pk3ADzE0hE8"
               nom="cri-orang-outang.mp3"/>
    </fichiers>
    <colorqrcode color="#1606e7"/>
    <textebraille texte="QR"/>
    <colorbraille color="#ea0000"/>
  </metadonnees>
</qrcode>
```

FIGURE 2.2 – Représentation d'un QR Code

2.3 Stockage des données

L'impossibilité de stocker un fichier audio dans un QR Code nous a poussé à trouver une solution de stockage distant pour les fichiers volumineux. Les transcripteurs de l'application utilisant déjà Google Drive pour stocker des fichiers, cela nous a paru être une solution adaptée. Cette solution présente également l'avantage de ne pas nécessiter la mise en place et la location d'un serveur.

1. Hypertext Markup Language

Chapitre 3

Application de bureau

3.1 Conception logicielle

3.1.1 Langages

Le choix des langages de programmation pour l'application de bureau a été discuté lors de la première séance. Nous avons évoqué le couple Java/Swing qui avait l'avantage d'être le plus simple à programmer et qui aurait été en cohérence avec l'application Android, mais qui nécessitait un environnement Java sur les machines des utilisateurs. Nous avons également évoqué le couple C++/Qt qui était le plus portable mais potentiellement trop lourd pour l'utilisation qu'on en aurait.

Les responsables du projet ont finalement opté pour le framework [1] Electron entre les deux premières séances. Ce framework permet de développer des applications de bureau avec des langages web (Javascript/HTML/CSS), et d'utiliser les modules Node.js. Ce choix a été motivé par l'intérêt qu'avait le Javascript d'être facilement utilisable pour accéder à Google Drive (voir 2.3).

3.1.2 Architecture

L'application est structurée selon un modèle MVC (Modèle Vue Contrôleur). Les données sont représentées par les classes du modèle, sont traitées par les classes du contrôleur et affichées par la vue. Nous utilisons en outre le patron de conception [2] Façade, qui nous permet de limiter la dépendance de la vue au contrôleur à une seule classe. Nous schématisons ces interactions dans la figure suivante.

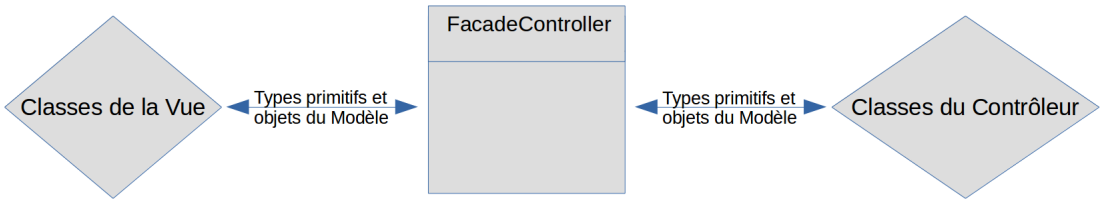

FIGURE 3.1 – Interactions entre la vue et le contrôleur à travers la classe FacadeController

Le modèle permet de représenter les QR Codes des différents types (voir 2.1). Il est composé d'une superclasse abstraite *QRCode*, étendue par deux classes *QRCodeAtomique* et *QRCodeEnsemble* représentant les spécificités des deux types de QR Codes gérés. Le modèle n'effectue pas de traitement des données mais fournit seulement des méthodes permettant d'y accéder et de les modifier, en faisant une abstraction de la façon dont elles sont représentées. Le diagramme UML des classes du modèle est visible ci-dessous.

1. Un framework désigne 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 (Wikipédia).
2. Un patron de conception (souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels. (Wikipédia)

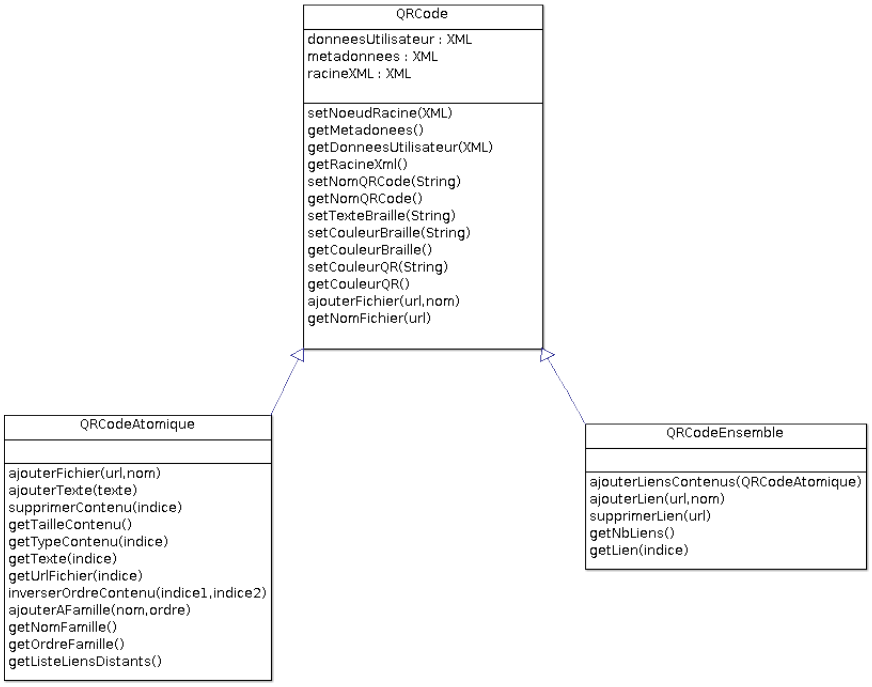

FIGURE 3.2 – Diagramme UML des classes du modèle

Le contrôleur se charge de tous les traitements permettant de générer des images de QR Codes (voir 3.2.3), ou d'instancier des objets du modèle à partir d'une image déjà générée (voir 3.2.5). L'architecture des classes du contrôleur est visible ci-dessous.

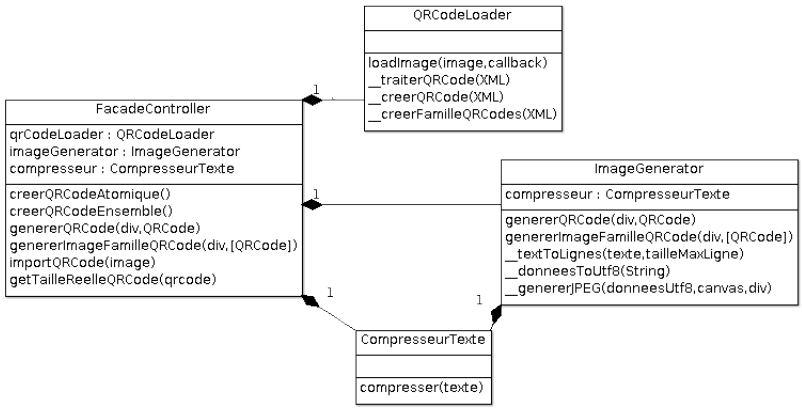

FIGURE 3.3 – Diagramme UML des classes du contrôleur

La vue est elle responsable des interactions avec l'utilisateur (voir 3.2.1). Elle offre une interface permettant de charger, de créer, d'afficher et de gérer des QR Codes.

3.2 Implémentation

3.2.1 Interface graphique

L'interface graphique offre la possibilité à l'utilisateur d'interagir avec l'application.
Elle constituée de trois blocs partant de la gauche vers la droite :

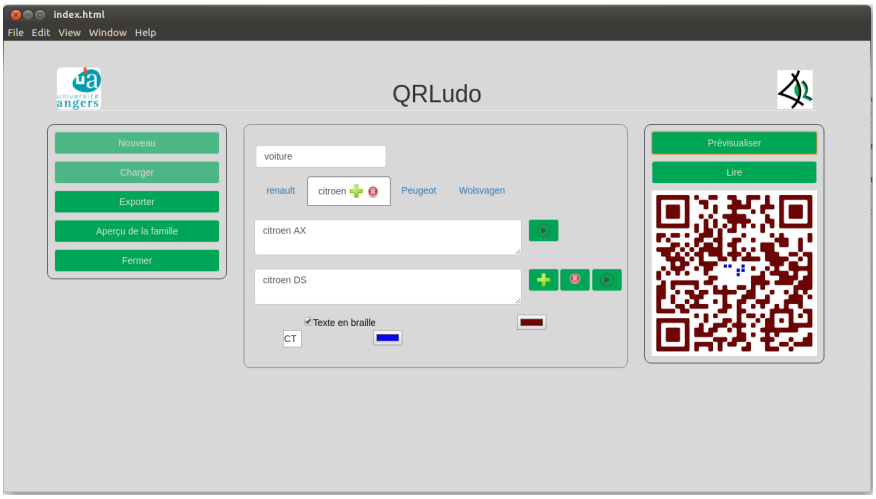

FIGURE 3.4 – Interface de création d'une famille de qrcodes

1. Bloc 1 : partant de haut en bas, il regroupe les boutons permettant les opérations suivantes :
   — la création d'un nouveau projet de QR Code unique ou d'une famille de QR Codes.
   — l'importation d'un projet de QR Code unique ou d'une famille de QR Codes.
   — l'exportation d'un projet de QR Code unique ou d'une famille de QR Codes.
   — l'aperçu de l'image d'une famille de QR Codes.

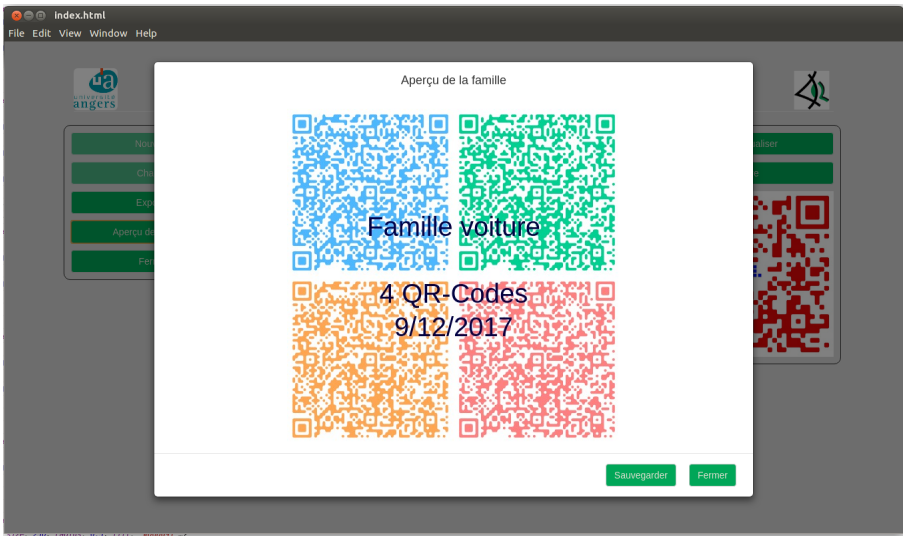

FIGURE 3.5 – Aperçu d'une image famille de QR Codes

   Les boutons Sauvegarder et Fermer permettent de sauvegarder le projet et de fermer l'aperçu.

   — la fermeture d'un projet.

2. Bloc 2 : il contient un formulaire qui regroupe des champs (musique ou texte). À coté de chaque champ se trouvent des boutons pour créer, supprimer et lire le contenu du champ par synthèse vocale.
   À la fin du formulaire, de la gauche vers la droite, se trouvent :
   — une case à cocher pour afficher ou masquer les options du texte en braille à savoir le texte et la couleur du texte via une palette de couleurs. Notons que le texte en braille ne peut pas dépasser deux caractères, afin de ne pas empêcher la lecture du QR Code.
   — une palette de couleurs pour la couleur du QR Code.
   Dans le cas d'un projet de famille de QR Codes, on note l'apparition :
   — d'une zone de texte pour le nom de la famille, en haut du formulaire.

— d'une liste d'onglets, chacun faisant référence à un formulaire. À coté de chaque onglet, figurent deux boutons pour ajouter un onglet ou supprimer l'onglet actif ; chaque onglet représente un QR Code.

3. Bloc 3 : partant du haut vers le bas, il contient :
   — un bouton pour prévisualiser un QR Code à partir des informations du formulaire (celui de l'onglet actif dans le cas d'un projet de famille de QR Codes).
   — un bouton lire pour lire par synthèse vocale tous les champs du formulaire (le formulaire de l'onglet actif dans le cas d'une famille de QR codes).

3.2.2 Prévisualisation

La prévisualisation permet de générer le QR Code avec les informations du formulaire de l'interface.
Nous faisons alors appel au contrôleur pour générer le QR Code (voir **??**. L'image générée sera celle qui sera enregistrée lors de la sauvegarde d'un projet de QR Code. Cette sauvegarde se fait de manière interactive.
Dans le cas d'un projet de famille de QR Code, l'interaction concerne la sauvegarde de l'image de la famille ; la sauvegarde des images des QR Codes de la famille se fait de manière non interactive (automatiquement).

3.2.3 Génération de QR Codes

Notre application utilise le script jquery-qrcode [3] pour générer les images des QR Codes. Il a l'avantage d'être très souple d'utilisation et de pouvoir générer des QR Codes aux formats assez complexes. Il permet notamment d'insérer une image ou un champ texte au QR Code, ainsi que de définir les couleurs du texte central et du QR Code. Nous avons tiré profit de ces caractéristiques pour laisser la possibilité aux transcripteurs de l'institut d'insérer un texte en braille central, d'une couleur pouvant être imprimée en relief (voir 1.2). Un QR Code de type atomique (voir 2.1) possédant des caractères centraux en braille est visible ci-dessous.

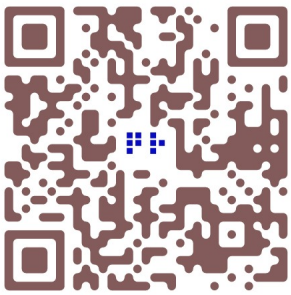

FIGURE 3.6 – QR Code simple

La génération des images est effectuée par la classe du Contrôleur *ImageGenerator* (voir 3.1.2). En plus des QR Codes simples, elle permet de générer des images de sauvegarde des familles de QR Codes (voir 2.1 et 3.2.5). Ces images contiennent le contenu des QR Codes de la famille dans les métadonnées, et des informations sur la famille imprimées sur l'image. Un exemple d'une image famille générée par l'application est visible ci-dessous.

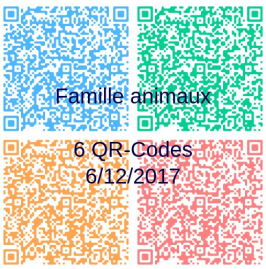

FIGURE 3.7 – Image de sauvegarde famille

3. https ://larsjung.de/jquery-qrcode/

3.2.4 Gestion des métadonnées

Dans l'objectif de minimiser la quantité de données stockées dans le QR Code, toutes les données qui ne sont pas nécessaires à l'application Android mais qui ont leur intérêt dans le chargement de QR Codes déjà enregistrés par l'application de bureau sont stockées dans les métadonnées de l'image du QR Code (REF Représentation des données). De plus, les images de type sauvegarde de famille (REF chargement QRCode) contiennent l'intégralité de la représentation XML des QR Codes la formant dans les métadonnées.

Pour insérer des métadonnées dans les images générées, nous utilisons le module Node.js piexifjs dans le processus suivant. Le QR Code est d'abord généré dans un canvas HTML 5, à partir duquel on va générer une image JPEG contenant dans les métadonnées le noeud racine de notre structure XML. Elles sont stockées dans le noeud XMLPacket des métadonnées EXIF. Nous avons choisi ce noeud pour éviter la perte d'information des caractères spécifiques au français, car il peut contenir des données binaires et pas seulement des caractères ASCII comme la plupart des noeuds EXIF. Les métadonnées EXIF n'existent pas dans les fichiers PNG, et nous n'avons pas trouvé de bibliothèque permettant d'insérer un autre type de métadonnées, c'est la raison pour laquelle nous générons des images en JPG.

3.2.5 Chargement de QR Codes

Dans le but de créer une application utilisable de façon concrète et régulière, nous avons élaboré des mécanismes permettant de modifier des QR Codes déjà enregistrés. Pour les QR Codes n'étant pas liés à d'autres (n'appartenant pas à une famille), il suffit d'enregistrer l'image représentant le QR Code en insérant dans les métadonnées le nœud racine de notre structure XML (voir 2.2). L'utilisation du nœud XML <metadonnees> est par ailleurs trompeuse car l'intégralité de la structure XML est stockée dans les métadonnées de l'image. Nous avons choisi de procéder ainsi car nous n'avons pas trouvé de librairie Javascript permettant de scanner facilement un QR Code à partir d'un fichier image, et la quantité de données insérées dans les métadonnées de l'image est négligeable par rapport à la taille des données représentant l'image.

Nous avons élaboré un mécanisme plus complexe en ce qui concerne l'enregistrement de QR Codes appartenant à une même famille. En effet, nous souhaitions toujours pouvoir enregistrer ces QR Codes séparément afin qu'ils puissent être imprimés, mais nous voulions empêcher la modification de l'un des QR Codes appartenant à une famille sans mettre à jour tous les autres. Pour cela, nous avons élaboré une façon d'enregistrer tous les QR Codes d'une famille dans un fichier unique, en plus des images représentant chacun des QR Codes. Nous avons choisi d'enregistrer ce fichier comme une image (voir 3.2.3) contenant la représentation XML de tous les QR Codes de la famille dans ses métadonnées, et affichant certaines informations concernant la famille sous forme de texte imprimé sur l'image. Les informations imprimées sont le nom de la famille, le nombre de QR Codes contenus et la date de création. Cette image possède en outre un fond contenant quatre QR Codes colorés afin de notifier son importance et d'éviter qu'elle soit supprimée par mégarde. Pour s'assurer que cette image soit utilisée, nous avons empêché le chargement de QR Codes seuls appartenant à une famille.

Le chargement d'un QR Code ou d'une famille de QR Codes s'effectue dans la classe *QRCodeLoader* du Contrôleur. Elle instancie des sous-classes de *QRCode* à partir de la racine XML des QR Codes lus dans les métadonnées de l'image, et les renvoie sous forme d'objet unique pour les QR Codes seuls ou sous forme de tableau pour les QR Codes appartenant à une famille.

3.2.6 Compression

Nous avons choisi au début du projet d'utiliser une structure XML pour représenter les données stockées dans le QR Code (voir 2.2). Ce choix était motivé principalement par la facilité de gestion du XML en Javascript.

Le XML a toutefois posé des problèmes au niveau de la concision des QR Codes générés. Nous avons tenté de remplacer la structure XML par une structure JSON plus concise lors de la première semaine de décembre, mais nous avions déjà trop de dépendances au XML. Nous avons alors tenté d'utiliser des librairies convertissant le XML déjà généré en JSON avant l'insertion des données dans le QR Code, mais nous n'en avons trouvé aucune suffisamment fiable (les attributs des noeuds XML étaient très mal gérés, et les noeuds de même nom étaient fusionnés sans respecter leur ordre).

Nous nous sommes donc orientés vers une solution de compression pour compenser l'intérêt qu'avait le JSON sur le XML. La solution que nous avons adoptée consiste à remplacer toutes les chaînes de caractères connues constituant

le XML par un caractère UTF-8 [4] prédéfini. Notre choix s'est porté sur les caractères de l'alphabet cyrillique car la probabilité qu'ils soient utilisés par les utilisateurs de l'application est très faible, et ils ne sont codés que sur deux octets (les caractères de l'UTF-8 sont représentés sur un nombre d'octets variant de un à quatre). Nous avons donc fait l'inventaire de toutes les chaînes de caractères de longueur supérieure à deux apparaissant dans notre représentation XML, et nous avons attribué à chacune un caractère de l'alphabet cyrillique. La classe *CompresseurTexte* du Contrôleur se charge de remplacer par le caractère correspondant toutes les chaînes de caractères appartenant à la représentation d'un QR Code avant qu'il ne soit généré.

Nous avons rendu cette solution la plus évolutive possible, en ajoutant deux caractères prédéfinis au début de la chaîne compressée pour notifier qu'elle a été compressée, et donc laisser la possibilité à l'application mobile avec un test simple d'interpréter les QR Codes n'ayant pas subi la compression. Le premier caractère inséré est le premier caractère de l'alphabet cyrillique, et le second est le chiffre 1, indiquant qu'il s'agit de la première version de cette méthode de compression. D'autres versions de la compression pourront ainsi être proposées dans le futur si la structure XML de représentation des données est modifiée.

Le principal inconvénient de cette solution est qu'elle ne compresse que la représentation des données et pas les données elles-mêmes. Un QR Code contenant beaucoup de texte sera toujours volumineux. Elle est toutefois très efficace pour les QR Codes contenant peu de données comme en témoigne l'exemple ci-dessous.

```
<donneesutilisateur xmlns="http://www.w3.org/1999/xhtml" type="atomique">
  <contenu>
    <texte>Le lion appartient a la famille des felides</texte>
  </contenu>
  <famille nom="animaux" ordre="3"></famille>
</donneesutilisateur>
```

FIGURE 3.8 – Données stockées dans un QR Code simple sans compression (214 octets)

```
È1ËLe lion appartient à la famille des félidésÌanimauxH3E
```

FIGURE 3.9 – Données stockées dans un QR Code simple avec compression (63 octets, soit un gain de 70%)

3.2.7 Lecture des fichiers du drive

Pour accéder aux fichiers du drive, nous utilisons l'API [5] Google Drive REST.
Le paramétrage [6] de cette API nécessite un compte Google et les modules googleapis et google-auth-library de Node.js. À l'issue du paramétrage, un dossier caché nommé .credentials est créé ; ce dossier contient un fichier contenant toutes les informations privées permettant à l'application de se connecter au drive. Ces informations sont chargées à chaque requête de connexion.

3.2.8 Limite de la taille des QR Codes

La taille du QR Code peut influer sur sa lecture. En effet si la taille du QR Code est trop grande, l'application mobile ne pourra pas le détecter. Pour pallier ce problème, nous avons mis en place une restriction sur la taille. Pour ce faire, à chaque fois qu'il y a une requête pour une prévisualisation, on vérifie si la taille du QR Code après compression (voir 3.2.6) ne dépasse pas un seuil que nous avons établi.

4. UTF-8 (abréviation de l'anglais Universal Character Set Transformation Format - 8 bits) est un codage de caractères informatiques conçu pour coder l'ensemble des caractères du « répertoire universel de caractères codés » (Wikipédia)
5. Application Programming Interface : est ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. (Wikipédia)
6. https ://developers.google.com/drive/v3/web/quickstart/nodejs

Chapitre 4

Application mobile

4.1 Application existante

4.2 Connexion à Google Drive et gestion des droits

4.3 Adaptation aux nouveaux QR Codes

Chapitre 5

Conclusion

Tout au long du développement, nous nous sommes efforcés de produire des applications répondant concrètement aux demandes de l'institut Montéclair, tout en gardant à l'esprit que les besoins de l'institut pourraient évoluer. Par conséquent, nos applications ne se veulent pas des versions définitives. En dehors du travail qui reste à effectuer pour les rendre totalement utilisables au quotidien, nous nous sommes efforcés de les rendre ouvertes à l'extension pour des versions futures avec peut-être d'autres types de QR Codes, d'autres types de contenus ou d'autres méthodes d'optimisation de la quantité de données contenues dans les QR Codes.
Cette volonté s'est traduite par la réalisation d'une architecture des classes souple et non spécifique aux besoins qui nous ont été transmis. Nous avons en outre fait en sorte que la compression des données, qui est la partie la plus susceptible d'être modifiée, soit modulable tout en restant facilement compatible avec les QR Codes que nous générons actuellement.

Il reste toutefois une dernière phase de développement avant de pouvoir livrer un couple d'applications totalement fonctionnelles et ergonomiques. Les modifications restantes concernent principalement l'interface graphique de l'application de bureau, afin qu'elle soit plus ergonomique avec un système de liste de QR Codes plutôt que d'onglets, un système de glisser-déposer de fichiers provenant de Google Drive, et l'implémentation de la création de QR Codes de type ensemble.

Elles concernent également la complétion de l'interprétation des QR Codes par l'application mobile, afin de gérer dynamiquement le téléchargement de fichiers à la lecture, et l'ajout de messages vocaux à destination de l'utilisateur.

Lorsque cette phase de finalisation sera terminée, un nouveau projet pourra débuter à partir de là où nous l'avons laissé, comme nous l'avons fait à partir de l'application mobile initiale. En plus de l'implémentation d'éventuelles nouvelles demandes de l'institut Montéclair, un travail complexe mais intéressant pourra être effectué pour compresser le contenu des QR Codes de manière plus efficace, en compressant le texte contenu dans les QR Codes à partir d'un dictionnaire fixe externe. Une solution de transformation du texte en son directement sur l'application de bureau pourrait également permettre de créer des QR Codes contenant des textes de grande taille.

Annexes

.1 Organisation

.1.1 Répartition des tâches

	David Dembele	**Alassane Diop**	**Jules Leguy**	**Rahmatou Walet Mohamedoun**
1	Conception, uml, diagramme de classe, diagramme de séquence, cas d'utilisation.	Conception, uml, diagramme de classe, diagramme de séquence, cas d'utilisation.	Conception, uml, diagramme de classe, diagramme de séquence, cas d'utilisation.	Conception, uml, diagramme de classe, diagramme de séquence, cas d'utilisation.
2	Prise en main de l'application android.	Mise en place HTML,CSS(boots trap) pour l'interface graphique.	Conception Modèle.	Mise en place HTML, CSS(bootstrap) pour l'interface graphique.
3	Classe de gestion des Qrcodes.	API google: -google_drive -text_to_speech	Implémentation Modèle Application Bureau.	Développement de l'interface graphique, pour correspondre aux attentes graphiques.
4	Fonction de parsage des Qrcodes.	Prévisualisation QR Code.	Implémentation de QR Codes.	Travail sur une version alternative de l'interface graphique.
5		Générer QR Code à partir du formulaire.	Génération de QR Codes avec texte en braille central.	
6		Charger QR Code.	Insertion et lecture des métadonnées des images.	
7		Exporter famille de QR Code.	Création d'objets du modèle à partir d'une image.	
8		Écoute individuelle d'un champ.	Compression données QR Code/Décompression.	
9		Champ en braille et palette couleur.	Gestion des droits Android.	
10			Téléchargement fichier Android.	

	Application de bureau			Application mobile
	Modèle	**Vue**	**Contrôleur**	
Semaine 2 04/10	Conception Modèle.	Mise en place HTML, CSS(bootstrap) pour l'interface graphique.		
Semaine 3 11/10	Implémentation Modèle	Développement de l'interface.		
Semaine 4 18/10				
Semaine 5 25/10		Utilisation de text_to_speech pour la lecture des données. Développement de l'interface.	Génération QR Codes avec image centrale.	Prise en main Application Android.
Semaine 6 01/11		Prévisualisation QR code. Développement de l'interface.	Écriture de métadonnées dans l'image générée.	
Semaine 7 8/11		Chargement QR code. Développement de l'interface.		Création d'une classe de gestion des QR codes.
Semaine 8 15/11		Création Champ en braille et palette couleur.		
Semaine 9 22/11		Développement de l'interface. Création d'un objet QR Code à partir d'une image enregistrée. Compression texte avec algo LZ.		Création fonction de parsage des données QR code.
Semaine 10 29/11	Texte central en braille	Génération image de famille. Compression texte avec solution ad-hoc Exportation image de famille de QR codes.		
Semaine 11 06/12		Release démonstration.		Téléchargement fichier Android. Droits Android. Compression/Décompression données. -Modification de la fonction swipe.

.1.2 Calendrier

.2 Manuel Utilisateur