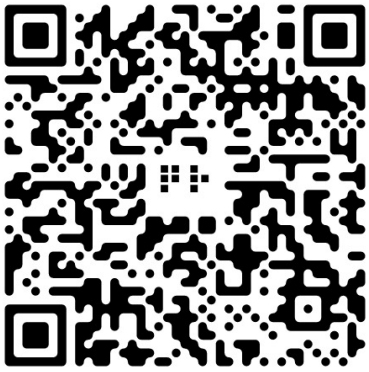

Développement d'un couple d'applications bureau et mobile

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 (LIEN FAMILLES).

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 (LIEN ENSEMBLE).

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 (LIEN GENERATION QR), ces caractères étant imprimés avec une encre spéciale gonflant à température élevée.

1. Travail Encadré de Recherche

Chapitre 2

Modélisation

2.1 Types de QRCodes

À partir des contraintes définies dans les cahier des charges (REF CAHIER CHARGES), 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 (LIEN TYPES QR) et à celle du Modèle (LIEN MODÈLE), afin d'assurer la cohérence de l'ensemble.

Notre choix s'est porté sur une structure XML [1]. Cette structure a montré ses faiblesses par la suite (voir REF CHOIX XML), mais avait déjà crée trop de dépendances pour être modifiée.

Afin de minimiser la quantité de données stockées dans le QR Code, seules les données indispensables à l'interprétation du QR Code par l'application mobile y sont stockées. Les données annexes nécessaires au chargement d'un QR Code par l'application de bureau sont stockées dans les métadonnées de l'image (REF CONTROLLEUR/METADONNÉES). On y trouve les données relatives au nom et aux couleurs du QR Code, ou encore au nom des fichiers contenus.

La structure fait apparaître clairement la dichotomie entre les données stockées dans le QR Code (noeud <donneesUtilisateur>) et les données stockées dans les metadonnées 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>).

1. Hypertext Markup Language

L'appartenance à une famille (REF TYPES QR) 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.

Un exemple complet de représentation XML d'un QR Code est visible dans la figure ci-dessous.

```
<qrcode>
  <donneesutilisateur type="atomique">
    <contenu>
      <texte>champ1</texte>
      <fichier url="URLFICHIER"></fichier>
      <texte>champ2</texte>
    </contenu>
    <famille nom="famille" ordre="4"></famille>
  </donneesutilisateur>
  <metadonnees>
    <fichiers>
      <fichier url="URLFICHIER" nom="fichier1"></fichier>
    </fichiers>
    <colorqrcode color="#085a0c"></colorqrcode>
    <textebraille texte="QR"></textebraille>
    <colorbraille color="#f11313"></colorbraille>
  </metadonnees>
</qrcode>
```

FIGURE 2.1 – Représentation d'un QR Code

Chapitre 3

Application de bureau

3.1 Conception

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 Electron entre les deux premières séances. Ce framework [1] 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 (REF STOCKAGE DRIVE).

3.1.2 Architecture

3.2 Implémentation

3.2.1 Interface graphique

3.2.2 Prévisualisation

3.2.3 Génération de QR Codes

Notre application utilise le script jquery-qrcode [2] 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 (LIEN INTRO/OBJECTIFS). Un QR Code de type atomique (REF TYPES QR) possédant des caractères centraux en braille est visible ci-dessous.

La génération des images est effectuée par la classe du Contrôleur *ImageGenerator* (LIEN ARCHITECTURE). En plus des QR Codes simples, elle permet de générer des images de sauvegarde des familles de QR Codes (LIEN TYPES QR et LIEN CHARGEMENT QR). 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.

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

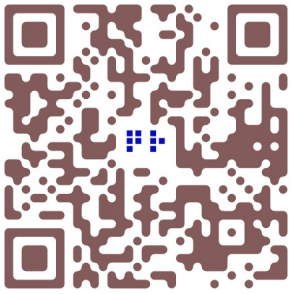

FIGURE 3.1 – QR Code simple

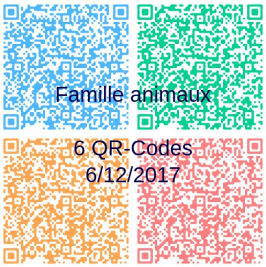

FIGURE 3.2 – Image de sauvegarde famille

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 [3] 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 [4]. 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 créé 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 (REF REPRESENTATION DONNEES). 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.

3. https ://github.com/hMatoba/piexifjs
4. Exchangeable Image File Format

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 (REF GENERATION) 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 (REF REPR. DONNEES). 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 [5] 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 le témoigne l'exemple ci-dessous.

5. 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)

```
<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 3.3 – Données stockées dans un QR Code simple sans compression

```
È1ËLe lion appartient à la famille des félidésÙanimauxH3E
```

FIGURE 3.4 – Données stockées dans un QR Code simple avec compression

3.2.7 Lecture des fichiers du drive

3.2.8 Limite de la taille des QR Codes

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

5.1 Réponse aux demandes initiales

5.2 Évolutions possibles

Annexes

.1 Organisation

.1.1 Répartition des tâches

.1.2 Calendrier

.2 Manuel Utilisateur