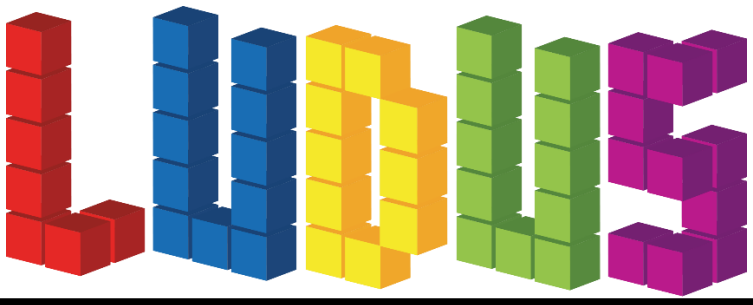


ACADÉMIE

Federation for EDucation in Europe
Fédération Européenne Des Ecoles

# Inventory in the pocket

par **Marceli Hawrysz**

**Mémoire présenté**
**en vue de l'obtention d'un Bachelor Européen**
**Jeux Vidéo et Serious Games**
2019/2020

N° de candidat : [Tapez ici votre n° de candidat de votre convocation]

# Table des matières

## Remerciements

Je tiens à remercier Jérôme Hatton, directeur de l'école Ludus Académie et du Pixel Museum, de m'avoir proposé le projet d'une application de scan servant à la gestion du stock et de l'inventaire du Musée, de m'avoir fait confiance pour ce projet et m'avoir aidé dans la réalisation de ce-dernier.

# Introduction

Les musées, centres culturels du présent et du passé, sont de nos jours confrontés à une multitudes de problèmes, dépendants de divers facteurs, tels que la nature même du musée, les sujets, les expositions temporaires etc. Un des facteurs importants à prendre en compte est la gestion du patrimoine matériel. Dans le cas de produits uniques, tels que des sculptures, le problème est moindre, mais dans le cas de pièces de monnaies ou de jeux vidéo, un répertoriage est indispensable.
Tout produit de consommation produit à plusieurs exemplaires sont équipés d'un code -barre. Cela permet une identification unique de chaque produit, en suivant les normes de certains pays.

Le Pixel Museum, collectionnant tout ce qui touche de près ou de loin les jeux vidéo, a eu une requête à la suite de tout cela : la création d'un outil qui aiderait au référencement de son inventaire. Pour cela, j'ai créé une application qui scanne le code barre d'un jeu et informe sur l'état du jeu : s'il se trouve dans les stocks, s'il est exposé ou si le musée ne le possède pas du tout.

# Le Pixel Museum

## Historique

Le Pixel Museum est le plus grand musée du jeu vidéo en France et en Europe. Sa collection presque complète de tous les jeux existants sur toutes les consoles du monde rend ce lieu agréable à visiter et unique dans son genre.

De la Magnavox© Odyssey, la première console personnelle, jusqu'à la récente PlayStation© 4, en passant par les célèbres Atari© 2600, Nintendo© Entrainement System ou une représentation gigantesque d'une Game&Watch©, le Pixel Museum présente, depuis le 25 février 2017, plus de 50 ans d'histoire des jeux vidéo.

Le Pixel Museum est situé à Schiltigheim, une commune française située au nord de Strasbourg

## Produits et services

Le Pixel Museum offre à ses visiteurs un patrimoine culturel conséquent. En plus de pouvoir tester certaines anciennes consoles exposées de manière permanente, le musée permet de retracer tout l'historique non seulement des jeux, non seulement des consoles, mais également beaucoup de licences plus ou moins connues, comme Tomb Raider ou Pac-Man. Tout fan peut y trouver son bonheur.

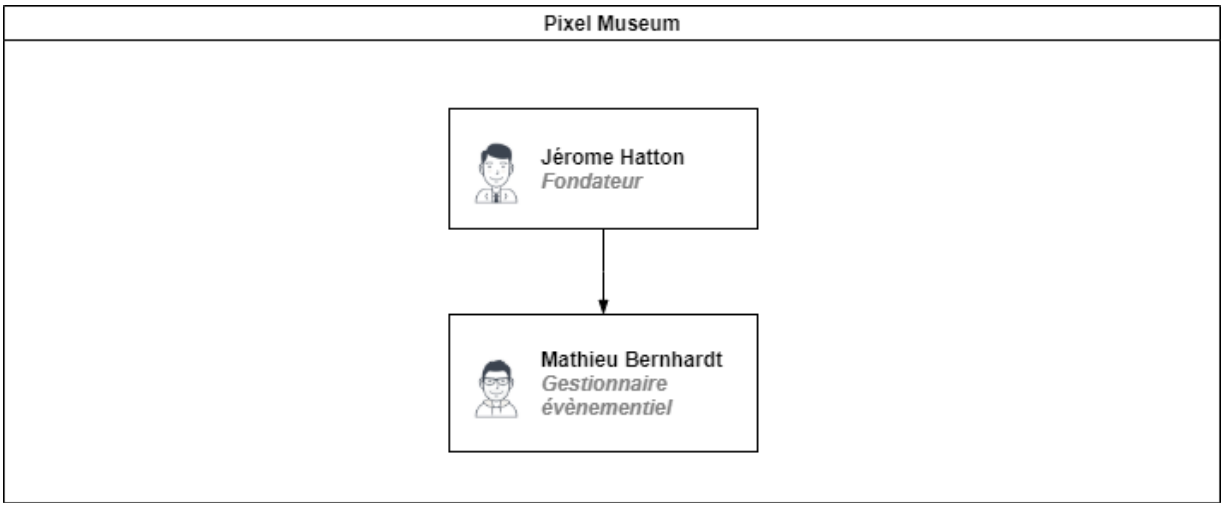
Lors des expositions temporaires sur divers thèmes non seulement du milieu vidéoludique, mais également cinématographique ou de l'animation, comme Dragonball ou Star Wars, les visiteurs peuvent découvrir toutes les choses possibles et imaginables des licences exposées : Mugs, porte-clés, céréales, figurines, statues, T-Shirts, coupes, guitares, imprimantes… Il n'est possible de trouver des collections aussi compètes ailleurs.

De plus, le Pixel Museum organise divers évènements, pour tous les âges : Qu'il s'agisse d'un enfant souhaitant participer à un atelier de création de jeu vidéo ou d'une entreprise souhaitant organiser un team building, le Pixel Museum est un endroit qui d'adapte à son époque.

En partant du Pixel Museum, il est possible de s'acheter un souvenir dans la boutique de ce-dernier.

Il y a également une boisson à l'hibiscus, mais il faut aimer les boissons amères.

## Organigramme




## Développement et perspectives

Le jeu vidéo est en constante évolution et est une industrie saine et rentable. Le fait que beaucoup de personnes eurent cru le jeu vidéo mort après le krach de 1983 ne l'a pas empêché de se reconstruire et de prospérer aujourd'hui, et même s'il y a eu des spéculations[1] quant à un nouveau krach, le jeu vidéo reste et restera une activité culturelle qui a marqué les dernières décennies. Le potentiel et la quantité de jeux vidéo ne cesse de croitre, et avec, l'avenir du musée.

[1] https://www.lemonde.fr/pixels/article/2015/09/04/la-crise-du-jeu-video-de-1983-va-t-elle-se-reproduire-avec-un-indiepocalypse-de-2016_4745640_4408996.html

# Présentation du projet

## Intention

Comme dit en introduction, chaque musée doit pouvoir gérer efficacement sont inventaire, et surtout le Pixel Museum. Pour donner un exemple, la console GameCube de Nintendo compte, à elle seule, en excluant les versions collector, 657 jeux sortis, ou, plus impressionnant, la PlayStation 2 de Sony compte au moins 4200 jeux dans sa bibliothèque. Ces deux nombres démontres, à eux seuls, que le répertoriage et le suivi des jeux possédés pour le Pixel Museum est un enjeu majeur.

Afin d'affiner le répertoriage de cet inventaire, le Pixel Museum possède une base de données regroupant tous ces jeux avec des informations détaillées, telles que la version, la jaquette, le numéro de série ou encore le code EAN 13.

Le code EAN (European Article Numbering) à 13 caractères est le système standard de codes-barres européen, et est présent sur tous les jeux vidéo présents en Europe. Ce code étant unique, il est utile pour repérer individuellement un jeu.

L'intention du projet était donc le suivant : créer un système de vérification simplifié et mobile permettant de connaitre, avec un simple scan de code barre, si un jeu donné est en possession ou non du Pixel Museum.

## Cahier des charges

| | |
|---|---|
| Octobre 2019 | Début du projet |
| Novembre 2019 | Choix de l'outil et début du projet |
| Février 2020 | Fin du projet et déploiement, phase de test |
| Mars 2020 | Fin de la phase de test, réajustements en fonction des résultats |
| Mai 2020 | Fin du projet, déploiement définitif |

# Contraintes

## La mobilité

La première et principale contrainte du projet était la mobilité. En effet, le directeur en charge de la gestion de l'inventaire, scrute les magasins à la recherche de jeux qui ne sont encore présents dans son musée. La procédure concernant la vérification de la possession d'un jeu avant le projet était la suivante :

1. Ouverture du navigateur internet
2. Arrivée sur la base de données du musée
3. Connexion à la base de données
4. Entrée manuelle du code barre
5. Analyse de résultats

Cette procédure engendre au moins deux problèmes :

- Les possibilités d'erreurs humaines et logicielles : Pour un humain, il est facile de confondre deux chiffres dans un code barre qui en contient 13. Les résultats lors d'une recherche erronée peuvent nécessiter un recommencement de la procédure à partir du point 3. Ils peuvent également donner un faux résultat, donnant ainsi une mauvaise analyse de données qui pourrait empêcher l'achat d'un jeu manquant à la collection. De plus, le disfonctionnement d'un outil dans la chaine (navigateur, base de données, page de connexion) peut également engendrer des erreurs logicielles.

- Le temps : La copie d'un code barre, chiffre par chiffre, sur un smartphone, est chronophage. Le temps moyen de l'introduction d'un code barre sur téléphone par un humain est de 15,7823[2] secondes. Cela veut dire que si l'on voudrait introduire tous les codes-barres des jeux de la PlayStation 2, cela demanderait 141 725,054 secondes, ce qui équivaut à 2362,0842 minutes ou 39,3681 heures. Ces mesures ne prennent pas en compte toutes les manipulations annexes, telles que le changement des jeux ou la possibilité qu'un jeu tombe par terre.

La contrainte de la mobilité est donc de rendre cette procédure plus simple et plus adaptée à une utilisation extérieure, en magasin plus particulièrement.

## La rapidité

Comme évoqué dans la partie précédente, le temps est un problème majeur. L'utilisation d'un outil plus adapté aux codes-barres serait plus judicieuse. C'est pour cela que l'utilisation d'un scanner de codes-barres a été la solution envisagée afin de résoudre les deux problèmes. Avec cet outil, la procédure est devenue la suivante :

1. Ouverture de l'application de scan
2. Scan du code-barre
3. Analyse du résultat

[2] Ces données ont été réalisées sur un échantillon non-représentatif composé d'une personne ayant introduit 3 codes-barres différentes sur un téléphone Honor 8 X dans des délais de 12.78, 17.72 et 18.32 secondes.

Cette procédure est beaucoup plus rapide, plus simple et plus sûre que la procédure précédente. Elle libère de la contrainte des erreurs en excluant l'intermédiaire humain dans l'introduction des chiffres, et de la contrainte du temps en réduisant plus de 10 fois le temps d'introduction du code barre. Avec ces deux contraintes supprimées, l'analyse du résultat a une chance presque nulle d'être erronée.

## La plateforme

Une autre contrainte importante était de pouvoir produire pour un téléphone mobile fonctionnant sur iOS. Cela engendre le fait que tous les outils de développement et d'exportation ne sont compatibles avec cette plateforme. Il faut un outil adapté à tout cela. C'est une contrainte à prendre en compte lors du choix de l'outil.

# Réalisation du projet

## Choix des outils

Une fois que le projet a été défini, il a fallu choisir un outil pour le développement de l'application.

### XCode et Swift



(De gauche à droite : Le logo de XCode, le logo du langage Swift)

Il y eu beaucoup de choix possibles. Parmi les choix évidents, il y a eu XCode comme Environnement de Développement Intégré (EDI) et Swift ou Objective-C comme langage de programmation. Les avantages de ce choix auraient pu être le manque de nécessité de d'optimisation, puisqu'elle se ferait au fur et à mesure du codage. Cependant, les désavantages étaient plus importants, à compter :

- Le fait qu'à Ludus Académie nous n'apprenons pas la gestion de réseau, il aurait fallu que j'apprenne toute cette partie-là. Sans compter la gestion de la connexion qui serait nécessaire pour la base de données, cela engendrerait des failles de sécurité, peu importe la qualité du système de connexion, la cyber sécurité étant un métier spécifique.
- Le codage en dur d'une telle application nécessiterait le design d'une interface qui prendrait en compte plein de paramètres comme la taille des écrans, la taille de la caméra etc. Cette compétence nécessite un minimum d'esprit graphique, ce qui n'est pas mon fort. Ce serait une compétence à laquelle je devrais m'entrainer, alors que le métier de graphiste est également un métier spécifique.
- L'utilisation de XCode nécessiterait également l'utilisation d'un ordinateur tournant sous MacOs, et cela engendre la nécessité de possession d'un ordinateur Apple, ce que je n'avais pas au moment de commencer le codage.
- Autant le Swift que l'Objective-C étaient des langages de programmation que je ne connaissais pas. Le choix engendrerait la nécessité d'apprentissage de ces langages.

Toutes ces problématiques ainsi que d'autres mineures ont fait en sorte qu'il serait impossible pour moi de développer une telle application en temps donné.

## Unity

(Le logo de Unity)

Le choix de Unity aurait pu être intéressant, premièrement, puisqu'il s'agit d'un outil que je maitrise bien, sur lequel je suis capable de faire beaucoup de choses. De plus, cet outil propose une exportation directement sur iOS. Également, il y a plein de plug-ins qui faciliteraient la gestion de la lecture des code-barres. Cependant, cela pose deux problèmes majeurs :

- Premièrement, Unity est un moteur de jeu, et pas un EDI. Cela veut dire que la plupart des composants préinstallés et nécessaires au fonctionnement de l'application, comme des mécaniques de jeu comme la gravité ou les collisions, les positions en 2 dimensions etc. feront en sorte que le projet sera lourd et pas facilement optimisable, ce qui ne répondrait pas à la contrainte de la rapidité.
- Deuxièmement, les développeurs de Unity sont en train de développer DOTS (Data Oriented Tech Stack) qui est une technologie d'optimisation par stockage d'entités séparées. Cette technologie est encore en développement et n'est pas dans une version stable. De l'autre côté, je n'ai pas les compétences pour optimiser de manière efficace une application selon l'ancien système. J'ai jugé inutile l'apprentissage d'une méthode qui sera dépréciée dans peu de temps, et trop risqué de travailler sur le nouveau système qui n'est pas encore prêt.

Ces deux problèmes majeurs m'ont fait exclure Unity comme outil pour la création de l'application.

## Le choix final : Delphi


(Logo de Delphi Community)

Un outil que je n'envisageais pas au premier abord était l'EDI Delphi par Embarcadero Technologies. Cependant, après quelques recherches, Delphi était la meilleure solution parce que :

- Delphi est un outil puissant qui est développé depuis 1995. Les bases sont donc solides et sont stables
- Delphi intègre la possibilité d'exporter vers iOS, même si cela nécessite le passage par un Mac pour l'exportation, il n'est pas obligatoire de l'avoir constamment sur soi.
- Delphi a été pendant très longtemps payant, mais à partir de juillet 2018[3], Embarcadero a abandonné l'édition d'essai Starter au profit d'une version Community, qui est gratuite et permet aux hobbyistes de profiter pleinement du logiciel.
- Depuis 2011, Delphi intègre la bibliothèque Firemonkey qui contient des composants permettant la création simple et rapide d'interface qui est user-friendly. Les composants de cette bibliothèque sont également cross-plateforme, et donc sont adaptés à iOS.
- La bibliothèque ZXing, l'une des rares bibliothèques gratuites permettant le scan de codes-barres, est directement disponible pour Delphi, contrairement à Swift qui n'a pas encore d'adaptation ou Unity où la plupart des plugins sont payants.
- Delphi est proche de Lazarus, sur lequel j'ai déjà travaillé et donc je connais ne dois pas apprendre le tout comme dans le cas du Swift.

Avec tant d'avantages, tant personnels qu'objectifs, j'ai décidé d'utiliser Delphi pour mon projet.

[3] https://fr.wikipedia.org/wiki/Delphi_(langage)

## Outils supplémentaires : les technologies web


(De gauche à droite, de haut en bas : Logos des langages HTML, JavaScript, CSS, PHP, SQL)

Malgré tous ses avantages, Delphi (autant que tous les autres) avait un souci majeur : La connexion à la base de données du musée n'était et ne pouvait pas être sécurisée sans engendrer de frais.
Pour contourner ce problème, il a fallu créer un système de vue qui donnerait les informations de la base de données et qui ne poserait pas de problèmes de sécurité. Sachant que Mr Hatton est le propriétaire de la base de données, il a été décidé avec lui que la meilleure solution serait un webservice qui recevrait en paramètre un code-barre, qu'il effectuerait les recherches dans la base de données et qu'il afficherait une information si le jeu est ou pas physiquement présent dans les stocks du musée.
Pour développer ce webservice, il a fallu utiliser le langage de programmation PHP pour l'exécution de la recherche, le langage d'exploitation SQL pour la recherche dans la base de données, le langage de structuration HTML pour mettre en place les données récupérées et le langage de style CSS pour la mise en page de la totalité.

## Gestion de projet

Avec tout le choix des outils, la procédure technique de réalisation se décompose en points suivants :

- Crée une application pour iOS
- Intégrer un système de lecture de codes-barres
- Envoyer ce code barre au webservice
- Récupérer l'information
- Afficher l'information

Ces étapes étant simple et peu profondes, j'ai jugé inutile l'utilisation d'un outil de gestion de projets

## Réalisation

En réutilisant les points mentionnés dans la gestion de projet, le projet a été réalisé en 5 étapes :

### Création d'une application iOS

La première étape nécessaire au bon fonctionnement est la mise en place de l'environnement de travail. Pour ce faire, j'ai essayé d'exporter sur un iPod 5 un projet vide créé avec Delphi. La mise en place s'est faite avec PA Server (Platform Assistant Server), un programme aidant à l'exportation pour Delphi.

### Fonctionnement de PA Server

Pour pouvoir exporter une application sur iOS, il faut que Delphi puisse compiler l'application de son côté, sur Windows donc. Or, Apple ne met pas à disposition de SDK pour pouvoir compiler directement une application sur Windows. C'est ainsi que j'ai utilisé PA Server qui télécharge la SDK sur un MacBook via l'intermédiaire de XCode, et transfère cette-dernière à Delphi. Ensuite, le programme compile l'application et une fois cela terminé, il la renvoie à PA Server, qui se charge ensuite, via l'intermédiaire de XCode, de déployer l'application sur un Emulateur iPhone ou un appareil utilisant iOS.

### Intégration du système de code-barres

La lecture du code-barres se fait par l'analyse de la taille des pixels scannés et de leur distance entre eux.  La création d'un tel programme nécessiterait l'analyse, l'intégration et l'exploitation des normes EAN 13, sans omettre le système de clés de contrôles, de sécurisation et de normalisation des codes-barres etc.

En raison de la difficulté de cette tâche, et du fait que la réutilisation de ce qui a déjà été fait est le principe même de la programmation, je me suis tourné vers ZXing.

### ZXing

ZXing est une librairie Open Source (C'est-à-dire que son code est disponible librement à la réutilisation, la modification et l'exploitation) écrite initialement en Java, qui intègre la lecture de codes-barres, selon la plupart des normes mondiales existantes (Comme le code 128 utilisé aux USA, le QR Code ou encore le EAN 13). La librairie est assez complète et j'ai décidé d'utiliser cette-dernière.

La librairie est tellement populaire qu'elle a été déclinée en version QT, C++, JavaScript, PHP, Python et notamment Delphi.

Pour l'intégration du système de codes-barres, j'ai donc utilisé cette bibliothèque pour mon projet.

### Envoi du code-barres au webservice

Pour pouvoir envoyer le code détecté à un service web, il faut créer ce webservice.

La première chose à prendre en compte est la sécurité du côté du serveur et de l'hébergeur qui aurait probablement empêché un chargement direct de la base de données. Il fallait donc trouver un moyen de se connecter à la base de données sans devoir intégrer des failles volontaires.

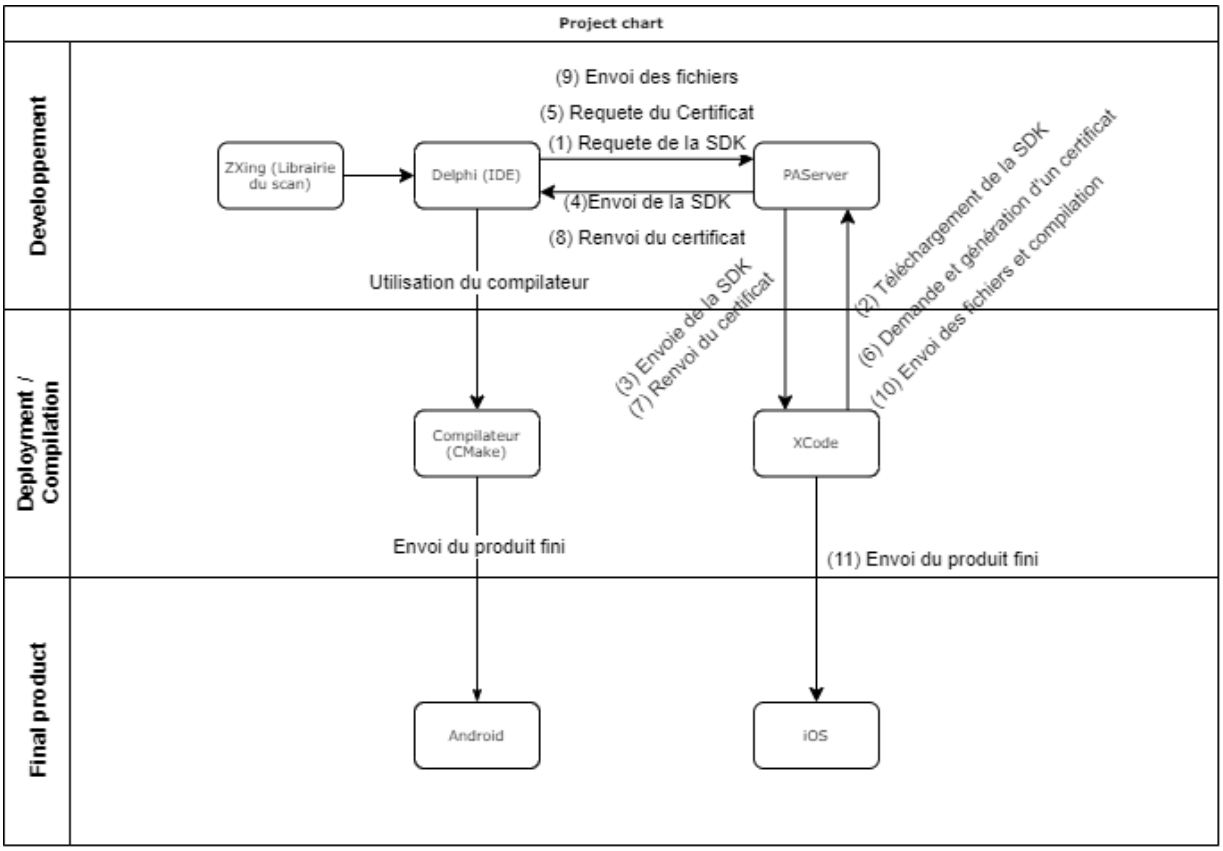
La solution a donc été de créer une page web qui ferait tout le travail de réception et d'analyse, et qui serait placée au même endroit.

### Réception et affichage du résultat

La solution de la page web est une solution tout-en-un qui permet de réceptionner et afficher les résultats. La réception est faite sur la page elle-même, ce qui fait un problème en moins.

En revanche, il faut afficher le résultat. J'ai eu deux solutions à ce problème. Le premier était d'interpréter un code html du côté du client, càd du côté de Delphi. Cependant, la deuxième solution, qui consiste en le fait d'afficher le résultat directement sur la page et d'afficher dans l'application la page elle-même, permet également d'accéder à la page depuis un ordinateur depuis un navigateur web. Cela permet de faire d'une pierre deux coup : un système de vérification pour le mobile et pour une station de travail

## Schéma du fonctionnement



# Problématiques

Au sein de tout le projet, il y a eu diverses problématiques, exposées ci-après.

## L'exportation vers iOS et la connexion à PA Server

Lors de mes premiers essais de connexion et d'exportation vers PA Server furent un échec. La stabilité des programmes, qu'il s'agisse de Delphi ou de PA Server, laisse à désirer. La connexion vers PA Server nécessite la connaissance de l'adresse IP locale du Mac. Cette dernière est assez compliquée dans un environnement scolaire (puisque le réseau comporte une cinquantaine d'ordinateurs), mais nécessite de préciser que les IP changent à chaque fois qu'un ordinateur redémarre. Ne connaissant pas cette subtilité, j'ai dû effectuer beaucoup de recherches pour trouver le problème.

Un autre problème furent les tests sur le simulateur iOS. En effet, une fois que l'application terminée, j'ai essayé d'exporter l'application vers le simulateur iOS. Cependant, cette manipulation n'a pas marché, pour une raison qui m'échappe jusqu'à maintenant. L'application était disponible et visible sur le simulateur, on pouvait la démarrer et elle se lançait, mais elle se fermait aussitôt. Les suppositions qui ont été émises par diverses personnes connaissant l'exportation vers iOS depuis Delphi parlaient du fait que l'application nécessite une autorisation de l'appareil photo, et que le simulateur, ne possédant pas d'appareil photo, est incapable de donner cette autorisation, et donc fait crasher l'application.

Le dernier problème concerne l'exportation directement sur un téléphone iOS. En effet, pour autoriser une exportation vers un appareil Apple, il faut posséder un Certificat Développeur Apple. Celui-ci étant au prix de 100$ par an et n'ayant ni le certificat, ni la somme disponible, et le simulateur iOS ne fonctionnant pas, j'ai dû trouver une alternative fiable pour faire les tests. Je me suis donc tourné vers Android, pour pouvoir exporter sur un téléphone et tester en conditions réelles.

## L'exportation vers Android durant la phase de tests

Dans le cadre des tests en temps réel, j'ai donc dû modifier le projet en version Android, et exporter le projet sur mon téléphone. Si durant les premiers tests, le système de scan fonctionnait efficacement, il y a eu des problèmes de sécurité concernant le chargement la page d'affichage. En effet, depuis Android Pie (càd Android 9), une sécurité, empêchant le trafic ClearText sans avoir de certificat HTTPS, a été mise en place. Cette dernière empêche l'affichage de la page, considérant cette dernière comme non-sécurisée.

Après quelques recherches, j'ai trouvé la solution qui consistait à allouer manuellement la connexion non-sécurisée. Il fallait ajouter l'autorisation explicite « android :usesCleartextTraffic=true ». Avec cette dernière, l'affichage de la page fonctionnait correctement.

## Le choix de la librairie d'analyse de codes-barres

Lors de ma recherche sur le fonctionnement des codes-barres et des librairies qui facilitent la lecture et l'analyse des codes-barres, j'ai trouvé de nombreuses informations assez disparates. La plupart des librairies sont soit payantes (comme dans le cas de nombreux plug-ins payants de l'asset Store de Unity), soit très flous (la documentation du scanner générique de Delphi inclut dans Firemonkey est très peu présente, voire inexistante).

Finalement, le fait qu'il existait un projet d'exemple comportant ZXing et fait sous Delphi a été prévalent dans le choix de l'outil.

## L'exportation finale

Dans le but de réduire au maximum les frais, et du au fait qu'Apple met gratuitement à disposition des certificats uniquement pour le développement et uniquement pour les applications faites avec XCode. Cette restriction permet une plus grande sécurité propre à Apple, néanmoins, elle ne permet pas de faire divers autres tests, tels qu'ils sont censés être faits avec Delphi.

Pour pouvoir utiliser ces certificats dans Delphi, il a fallu trouver un moyen de contourner cette sécurité. Pour cela, il a été nécessaire de créer un projet de test et générer un certificat de développement « vide », et ensuite utiliser ce certificat en tant que passerelle pour pouvoir exporter l'application.

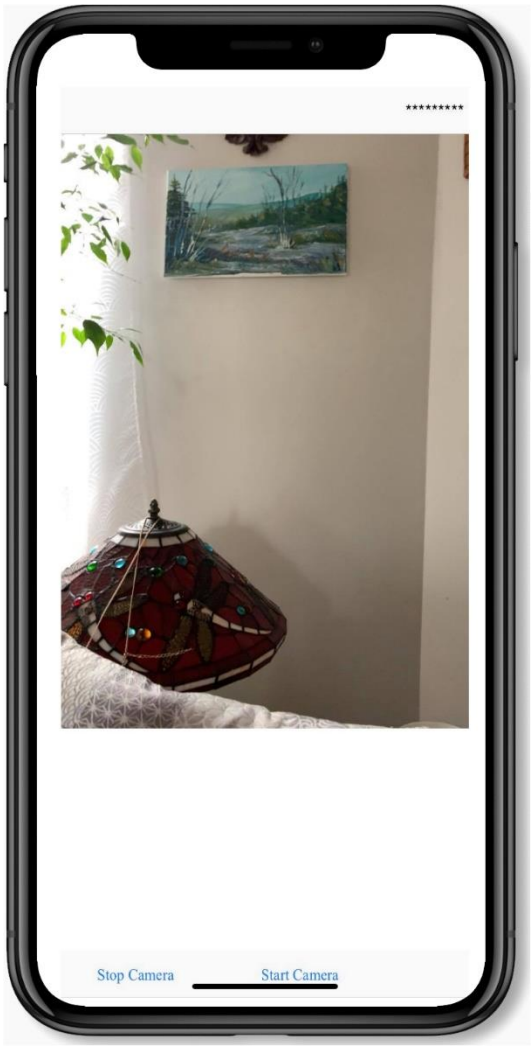
Le problème final que j'ai rencontré était le fait que le certificat ne correspondait aux données de l'application. En effet, lors de la création de l'application sur XCode, le certificat est généré avec des données spécifiques, tels qu'un UUID, un code d'identification de l'équipe de production, et un nom de l'application. Delphi utilise ensuite le certificat généré pour créer son application. Cependant, il faut manuellement adapter ces infos dans le certificat CFEBundle. Cette info est assez compliquée à trouver car la documentation ne mentionne pas cette information de manière assez détaillée.

# Conclusion

## Résultat obtenu

Pour finir, le déploiement a fonctionné, et s'est même avéré très productif. L'optimisation obtenue avec la restriction de l'analyse uniquement du code EAN13 a été conséquente, et couplée à la puissance d'un IPhone, a permis de scanner des codes à une vitesse presque industrielle. L'application fonctionne d'une manière optimisée et efficace, et permet de scanner plusieurs codes-barres de jeux à la suite.

Les interfaces obtenues sont présentées dans les images suivantes :




Capture d'écran de la caméra qui scanne


Capture d'écran du résultat du scan

## Analyse critique

Avec le recul, il est important de préciser que conceptuellement parlant, le projet est compliqué, mais lorsqu'on le divise en projet en deux parties distinctes, c'est très simple, d'un côté, il s'agit d'une simple requête SQL, et de l'autre, une analyse d'une photo. Les requêtes SQL ont été une matière apprise lors de cours, et l'analyse de photo se fait par des plug-ins externes. La partie la plus compliquée était le développement d'un système de communication entre les deux systèmes.

Également, la création d'une application mobile était, au début de l'année, et est encore pour moi, un défi assez compliqué, dû au fait que je ne sais pas utiliser les programmes natifs prévus pour les différents OS mobiles. L'utilisation de programmes permettant plus de flexibilité et des exportations multiplateformes a également facilité le projet. Je ne connaissais pas vraiment

Delphi, mais son concurrent Lazarus, qui cependant n'était pas capable d'exporter nativement vers Android, alors je n'ai même pas cherché pour iOS.

## Perspectives

Ce projet a l'avantage d'être assez simple, mais en même temps très commun. À chaque fois que l'on va dans un magasin, on scanne des produits pour que ça aille plus vite. Dans le secteur touristique, on utilise des QR Codes pour analyser ou afficher des informations diverses, ou rediriger vers des pages. Les applications qui scannent et facilitent l'identification sont omniprésentes et le seront de plus en plus.

Également, une telle application, développée une fois, reste universelle et réadaptable. Je pourrai la réutiliser comme base pour n'importe quelle autre demande impliquant des scans de codes-barres qu'ils soient en une ou deux dimensions.

## Apports personnels

Ce que le projet m'a le plus apporté est le fait d'avoir vaincu un grand blocage que j'avais depuis un certain temps, qui était la peur de la fermeté de l'écosystème Apple. En effet, j'avais peur de devoir débourser une fortune pour le matériel et les licences, ensuite de devoir apprendre le développement pour quelque chose de très spécifique et qui ne me servirait que dans un seul projet. Grace à cette demande, j'ai pu et dû trouver des alternatives qui se sont avérées multiplateformes et réutilisables, et cela sans devoir réapprendre tout un système, et sans devoir acheter quelconque matériel (j'ai pu exporter sur l'IPhone de ma copine, et j'ai acheté un MacBook en début par curiosité).

J'ai également beaucoup appris sur le fonctionnement du multi-plateforme, grâce aux recherches, mais aussi grâce différents essais d'exportations. Les restrictions ainsi que les permissions spécifiques aux différentes plateformes, notamment l'Universal Windows Platform qui gagne en popularité, me permettent de me préparer pour l'avenir avec plus de sureté que si je n'avais pas eu ce projet.

# Bibliographie

ZXing pour Delphi – répertoire du projet - https://github.com/Spelt/ZXing.Delphi
Portage du projet sur iOS - http://docwiki.embarcadero.com/RADStudio/Rio/en/IOS_Mobile_Application_Development

## Résumé

Le projet *Inventory in the Pocket* vient directement d'une nécessité de gestion de stock du Pixel Museum. Sachant que le musée collectionne tous les jeux vidéo sortis sur console depuis la Magnavox Odyssey et que chaque génération de console compte un nombre exponentiel de jeux, la question de la gestion des stocks doit être au point. Ainsi, un problème s'est posé : comment savoir rapidement en magasin ou lors d'un rangement si un jeu est répertorié ou présent dans le stock ?

C'est ainsi qu'est venue l'idée de créer une application qui scanne et identifie les jeux en fonction de leurs statuts. Ainsi, ce document répertorie et décrit en détails les étapes de la création de l'application qui a servi à rendre toutes tâche aussi simple qu'un claquement de doigts.



**Mots-clés** : Delphi, iPhone, iOS, projet, multi-plateformes, scan, code-barre, jeux vidéo, Pixel Museum, ZXing



## Abstract

The project *Inventory in the Pocket* comes directly from a necessity of the management of the inventory from the Pixel Museum. Knowing that the museum collects all the video games who came out on a console since the Magnavox Odyssey and that each console generation counts an exponential number of games, the question of the management of the inventory must be working great. So, a problem has arisen: how to know quickly in the shop or during a storage if a game has been listed or is present in the inventory?

That is where comes the idea of creating an app that scans and identifies the games according to their status. In this way, this document lists and details the steps of the creation of the app that helped making all the tasks as easy as a snap of fingers.



**Keywords**: Delphi, iPhone, iOS, project, multi-platforms, scan, barcode, video games, Pixel Museum, ZXing