Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Filière Informatique de gestion
Studiengang Wirtschaftsinformatik

# Travail de Bachelor 2015

# Analyse du développement mobile multiplateforme avec Xamarin

Étudiante : Audrey Dupont

Professeur : Jean-Pierre Rey

Déposé le : 27.07.2015

HES-SO Valais-Wallis • rue de la Plaine 2 • 3960 Sierre
+41 27 606 89 11 • info@hevs.ch • www.hevs.ch

# Sources des illustrations

http://xamarin.com/content/images/pages/mobileapi/MobileGraphic.png

# Avant-propos

En 2014, il s'est vendu plus de 1,3 milliard de smartphones dans le monde (ZDNet, 2015). Cette demande est en constante augmentation et les utilisateurs de ces mobiles sont de plus en plus exigeants. En effet, la demande continuelle de nouveauté pousse les développeurs d'applications à créer plus régulièrement des nouvelles applications. L'univers actuel du développement mobile est dominé par trois grands systèmes d'exploitation : Android, iOS et Windows Phone (IDC, 2015). Chacun de ces systèmes a des limitations particulières concernant ce développement d'applications. Le langage de développement, par exemple, diffère d'un système à un autre.

Dans le cadre d'un thème de travail de Bachelor proposé par Jean-Pierre Rey, professeur à la HES-SO Valais/Wallis, une application pour 3 systèmes d'exploitation mobile, crée via le logiciel Xamarin, a été délivrée. Cette dernière a été développée afin d'analyser le comportement d'un développement multiplateforme (Android, iPhone, Windows Phone).

Ce rapport fournit toutes les informations concernant l'analyse du marché actuel des systèmes d'exploitation mobile et l'analyse du développement multiplateforme mobile. De plus, toutes les étapes importantes de la création de l'application test y sont documentées.

# Remerciement

Je tiens à remercier cordialement toutes les personnes qui m'ont permises de réaliser ce travail de bachelor dans les meilleures conditions.

Un remerciement particulier à :

M. Jean-Pierre Rey, mon professeur responsable, pour m'avoir encadrée et suivie tout au long de ce travail. Merci pour ses conseils, sa disponibilité et son implication.

M. Christophe Hadorn et M. Saša Arsić, du centre e-learning Cyberlearn, pour le prêt d'une machine physique Mac qui m'a permis de tester l'application sur iOS.

M. Joseph Sylvester, Customer Success Specialist chez Xamarin, qui m'a permis d'avoir rapidement un accès étudiant au programme Xamarin.

Finalement, merci à toutes les personnes ayant relu ce rapport.

# Résumé managérial

Le monde actuel du développement est en pleine effervescence. Entre des applications pour Android, pour iOS et pour Windows Phone, la demande en développeurs n'a jamais été aussi grande. Ces trois grands acteurs se disputent la tête du classement. Chacun a un langage de développement qui lui est propre. Cela oblige les entreprises à avoir un panel de personnel possédant les capacités de développement des trois plateformes dans le but de créer une application sur chacune d'entre elle.

Une question se pose alors. Pourquoi ne pas faire une application smartphone qui serait écrite une seule fois mais disponible sur plusieurs plateformes ?

C'est ce que nous appelons, dans le milieu du développement, des applications multiplateformes. Il existe trois sortes d'applications multiplateformes : Application web, application native et application hybride. La première utilise les technologies du web pour être créée. La seconde est conçue spécifiquement pour un système d'exploitation dans le langage de programmation du système. La dernière se trouve être un mix entre les deux premières. Elle utilise les technologies du web pour être créée mais possède l'apparence des applications natives.

De nombreux logiciels nous permettent de créer des applications multiplateformes comme par exemple Xamarin. Ce logiciel permet d'écrire une application multiplateforme dans un unique langage : le *C#*. Plus de 90% du code écrit via ce logiciel est **écrit une seule fois** et est **utilisé par les trois principales plateformes** (Android, iOS, Windows Phone). Les applications créées via ce logiciel seront de type **natif**. De plus, il n'est en aucun cas nécessaire de connaître les langages de développement des différentes plateformes.

Ce travail analyse les capacités de Xamarin via une application. Cette application vise à tester les éléments graphiques et les accès vers les fonctionnalités natives des plateformes. Elle met aussi en valeur la gestion d'une base de données en local. Toute la partie de développement de ce projet fut mis en place de manière itérative, basée sur la méthodologie Agile.

*Mots-clés : travail de Bachelor, développement, multiplateforme, Xamarin, Android, iOS, Windows Phone*

# Table des matières

# Liste des tableaux

# Liste des figures

# Abréviation et glossaire

| | |
|---|---|
| **API** | *Application Programming Interface*. Une application fournit une interface afin de pouvoir interagir avec celle-ci. |
| **Application Layer** | Interface de programmation qui permet la conversion des données entre la couche métier et l'interface utilisateur. |
| **Back-End** | Partie non visible d'un logiciel (couche métier). |
| **Bytecode** | Code résultant d'une compilation (Rouse M. a., 2005) |
| **BLC** | Bibliothèque de classe. |
| **Blue Screen** | Ecran d'erreur fatale sur Windows. |
| **BOO** | Langage de script dérivé du Python. |
| ***C#*** | Langage de programmation orienté objet créé par Microsoft. |
| ***Compilateur*** | Permet de traduire un code écrit dans un langage de développement pour l'humain vers un langage de bas niveau, compréhensible par la machine. |
| **Couche métier** | La couche métier d'une application décrit les actions que l'application va faire selon les requêtes de l'utilisateur. |
| **CLI** | *Common Language Infrastructure.* ~ Multiplateforme. Permet à un programme écrit dans l'un des plus communs langages de développement (*C#*, VB.*Net*, J#, …) d'être exécuté sur n'importe quel système d'exploitation (Rouse M. b., 2015). |
| **CRUD** | *Create, read, update, delete*. Les quatre opérations de base pour la gestion de données. Crée, lire, mettre à jour, effacer. |
| **CSS** | *Cascading Style Sheets*. Document en informatique permettant de décrire la présentation de page internet. |
| **Framework** | Structure logicielle. Ensemble fournissant le squelette d'un programme. |
| **Front-End** | Partie visible d'un logiciel (UI). |
| **HTML** | *Hypertext Markup Language***.** Langage utilisé pour mettre en page des données afin de créer une page web. |

**IDE** *Integrated Development Environment.* Environnement de développement intégré.

**Java** Langage de programmation orienté objet crée par Sun Microsystems.

**JavaScript** Langage de programmation de script permettant de rendre des pages web dynamique.

**LINQ** *Language Integrated Query*. Requête intégrée au langage. Composant du Framework *.Net* de Microsoft qui permet d'interroger toutes sortes de données. Syntaxe proche de SQL.

**.NET** Plateforme standard proposé par Microsoft. Contient un ensemble d'outils tel que des applications, une suite d'outils et de service (bibliothèque) ou un environnement d'exécution de code basé sur la CLI. Permet de rendre des applications plus facilement transportables.

**NuGet** Gestionnaire de package .Net et C++.

**Objective-C** Langage de programmation orienté objet utilisé principalement dans les *OS* d'Apple.

**Open source** Lorsque le code d'un logiciel est disponible et utilisable par le grand public.

**OS** *Operating System*. Système d'exploitation. Les capacités d'une machine (Ordinateur, mobile…) sont pilotées par un ensemble de programmes.

**SDK** *Software Development Kit*. Kit de développement. Permet aux développeurs d'avoir à leur disposition un ensemble d'outils pour un environnement de programmation spécifique (p.ex. iOS SDK, Android *SDK*, …)

**Swift** Nouveau langage de programmation dérivé de l'Objective-C utilisé principalement sur les OS d'Apple.

**Tizen** Système d'exploitation *open source* multiplateforme dont le développement technique est supervisé par Intel et Samsung.

**UI** *User interface*. Interface utilisateur. Permet de définir les écrans, les contrôles, les images, …

**XAML** *eXtensible Application Markup Language.* XAML est un dialecte du

langage balisé XML.

**XML** *eXtensible Markup Language*. Langage balisé permettant la notation d'un fichier texte avec une structure hiérarchique.

# Convention typographique

Dans le but de faciliter la lecture de la suite de ce travail, différents styles d'écriture ont été utilisés. Ces derniers sont illustrés ci-dessous :

- Le texte normal
- `Le code source (langage de développement)`
- *Les termes apparaissant en italiques réfèrent à une entrée dans le glossaire en début de document*

Suite à la validation de M. Jean-Pierre Rey, il a été décidé de réduire l'espacement interligne de 1.5 pt à 1.2 pt.

# Introduction

Actuellement, le nombre d'utilisateurs de smartphone est en constante augmentation. En 2012, ce chiffre atteignait le milliard d'utilisateurs. Désormais, nous comptons plus de 1.75 milliard de personnes utilisant un smartphone (eMarketer Inc, 2004). Android, iOS, Windows Phone sont les acteurs phares dans le domaine des systèmes d'exploitation de smartphone (IDC, 2015).

Sur ces différents systèmes d'exploitation, les utilisateurs peuvent télécharger des applications de leur choix via les différentes boutiques en ligne mises à disposition par les fournisseurs d'appareils. Chacune de ces boutiques possède une base de données de plusieurs milliers d'applications. D'après des statistiques, 1.43 million d'applications étaient téléchargeables en 2014 dans la boutique en ligne de Google Play Store. A contrario, et ceci depuis peu, l'App Store de la firme à la pomme ne possédait que 1.21 million d'application (Ariel, 2015). Ces chiffres sont en constantes augmentation et la demande d'applications ne se fait que plus forte envers les développeurs.

De ce fait, le développement mobile gagne de l'importance d'année en année. Grâce aux différents *SDK* et à ces différentes boutiques en ligne, de nombreuses applications font leur apparition engendrant ainsi de nouveaux besoins. Ces nouveaux besoins poussent les utilisateurs à migrer plus rapidement sur un smartphone.

# Contexte

A la fin des 3 ans de formation de la filière Informatique de gestion au sein de l'HES-SO Valais-Wallis, chaque étudiant est dans l'obligation de rendre un sujet d'étude dans le but d'obtenir le Bachelor en Informatique de gestion. En moyenne, un travail de ce niveau doit être effectué dans un laps de temps de 360 heures.

Ce travail a été commencé le 27 avril 2015 et s'est terminé le 27 juillet 2015. Pendant les deux premiers mois, l'écriture de ce travail fut en parallèle de la formation scolaire. Le mois de juillet fut uniquement consacré à la rédaction de ce travail.

# Délivrables

Un travail de bachelor contient des documents à rendre dans le temps imparti. Pour ce travail, les délivrables sont les suivants :

- Le rapport final
- Le canevas de l'application
- L'application bêta

Tous ces documents sont ajoutés à la clé USB que vous trouverez en annexe de dossier.

# 1 État de l'art

Ce chapitre va résumer l'état des connaissances actuelles au niveau du développement d'applications mobiles. Il sera fait mention des principales barrières d'entrées. Par la suite, la problématique de ce sujet est expliquée.

## 1.1 Parties prenantes

Les personnes dont les intérêts sont affectés à la suite de ce projet sont principalement les développeurs d'applications, mais aussi les entreprises clientes de ses développeurs.

Lorsqu'une application doit être créée, les entreprises décident sur quelle plateforme lancer l'application en premier. Suivant le choix de ces dernières, l'engagement du développeur varie. En effet, la plateforme choisie définira les compétences du développeur à embaucher.

## 1.2 Situation actuelle

Au début de l'année 2015, Apple a annoncé un nouveau record en matière de facturation sur leur boutique en ligne, App Store. En effet, cette dernière a généré plus d'un demi-milliard de dollars de bénéfice grâce aux achats d'Apps et d'achats intégrés (Apple Inc. a, 2015).

Lors du Mobile World Congress en 2013, Microsoft a annoncé que 4 millions d'applications étaient téléchargées par jours sur le Windows Store (Wilhem, 2014). Cette boutique comprend tous les appareils sous Windows 8 et supérieurs.

Au commencement, le Google Play (anciennement Android Market) proposait un panel de 2300 applications (Lawson, 2009). Désormais, cette boutique d'applications est celle qui possède le plus grand choix avec 1.43 million d'applications (Ariel, 2015).

Les faits exposés ci-dessus ne sont qu'un bref aperçu du marché des applications mobiles. Malgré ces chiffres, la création d'applications possède de nombreuses barrières d'entrées. Bon nombre de ces barrières sont dues à la concurrence entre les différents acteurs du marché. Chaque analyse de ces acteurs dans les chapitres suivants possède la même structure. En premier lieu, il est question de la définition de l'acteur. Par la suite, l'influence de ce dernier sur un marché économique est expliquée. Et finalement, un aspect technique de cet acteur est abordé.

### 1.2.1 Android

Android est un système d'exploitation mobile racheté en 2005 par Google (Elgin, 2005). C'est un *OS* (Operating System) *open source* sous licence Apache conçu pour toute sorte d'appareils

(smartphone, tablette tactile, PDA, téléviseurs, voitures …). Cette entreprise est symbolisée par un petit robot vert illustré grâce à la Figure 1 : Logo Android. Toutes les versions d'Android ont un nom emprunté au milieu de la pâtisserie et se suivent toutes dans l'ordre alphabétique. La version actuelle s'appelle Android Lollipop et la prochaine version annoncée porte le nom de code Android M (Eason, 2015).

*Figure 1 : Logo Android*

*Source : (Android, 2015)*

Le marché mondial des OS mobiles montre une prédisposition pour Android. En effet, selon une étude de l'International Data Corporation (IDC) datant du trimestre 2014, 81.5% du marché mondial est occupé par des smartphones possédant *l'OS* Android (IDC, 2015).

Au niveau technique, le système d'exploitation Android est composé de 5 couches distinctes (Lee, 2012) comme illustré dans la

Figure 2 : Architecture d'Android.

Le noyau Linux (Kernel) permet de faire le lien entre l'aspect logiciel d'Android et l'aspect matériel. Nous pouvons y retrouver, par exemple, un pilote pour contrôler la caméra.

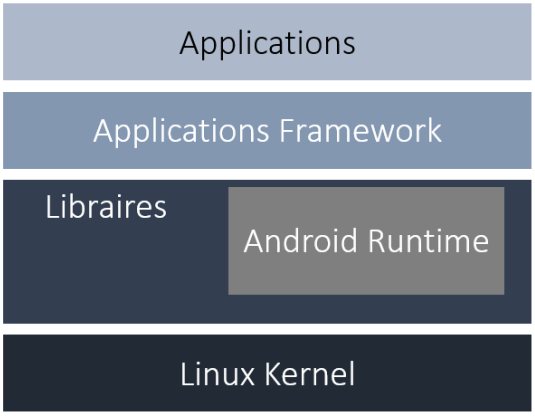
La couche librairies contient un panel de librairies natives au smartphone comme SQLite, un gestionnaire pour des bases de données.

Une de ces couches, appelées Android runtime, contient un panel de plusieurs librairies afin que les développeurs puissent écrire des applications Android en *Java*. Cette couche comporte une machine virtuelle, Dalvik, qui exécutera les applications écrites en *Java*.

Les *API (Application Programming Interface)* sont fournis par la couhe Applications *Framework*.

La dernière couche, applications, correspond aux nombreuses applications se trouvant sur le smartphone tel que les contacts, les sms ou le calendrier.

*Figure 2 : Architecture d'Android*




*Source : Adapté de (Espiau, 2015)*

### 1.2.2 IOS- Apple

*Figure 3 : Logo Apple*

*Source : (W12, 2015)*

Apple est une entreprise proposant des produits électroniques et des logiciels. Elle fut fondée en 1976 par Steve Jobs, Steve Wozniak et Ronald Wayne (Isaacson, 2011). Comme son nom l'indique, l'entreprise Apple est symbolisée par une pomme, illustrée grâce à la Figure 3 : Logo Apple. Dans sa gamme de produits, cette entreprise propose un système d'exploitation mobile connu sous le nom d'iOS disponible sous iPhone, iPad et iPod Touch.

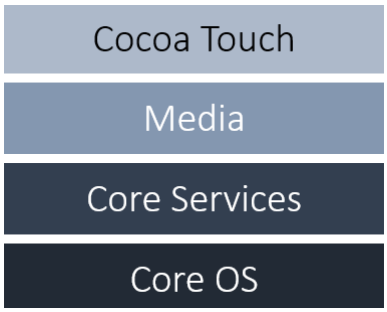
La dernière version d'iOS disponible pour les utilisateurs est la version iOS 8.4. Néanmoins, Apple a annoncé la sortie de son prochain OS, iOS 9, en automne 2015 (Merli, 2015). Selon les annonces de l'entreprise, iOS 9 apportera une amélioration notable pour les applications telles que Notes, Maps ou même Siri. De plus, il sera désormais possible de mettre sur le même écran, côte à côte, deux applications (Apple Inc. b, 2015)

Unique en son genre, le marché suisse des smartphones demeure fidèle à la frime Apple. En effet, selon un article du Temps (Seydtaghia, 2015), Apple possède entre 50 et 60 % du marché des smartphones en Suisse. Malgré le fait qu'Apple ne possède que 14,8 % de marché mondiale des OS, c'est l'entreprise qui génère 90% des profits de l'industrie mobile (Nguyen, 2015).

Comme expliqué précédemment, les appareils d'Apple tournent sous un système d'exploitation appelé iOS. Ce système contient 4 couches distinctes illustré dans la Figure 4 : Architecture de iOS. (Mercy, 2011).

La première couche, Core OS, est la couche basse du système. C'est elle qui gère par exemple le système sécurité ou bien même les certificats (Martin, 2012).

*Figure 4 : Architecture de iOS*




*Source : Adapté de (Jacobs, 2012)*

Les Cores Services peuvent supporter, par exemple, l'accès vers une base de données SQLite.

La couche Média, comme son nom l'indique, permet de gérer les médias audio ou vidéo.

La dernière, quant à elle, est *l'API* qui est constituée de plusieurs programmes pré-écrits. Elle est dédiée à l'écriture des applications (Martin, 2012).

Toutes les applications qui sont écrites sur iOS doivent être faites grâce à *l'Objective*-C, langage de développement d'Apple. Néanmoins, depuis 2014, Apple met peu à peu en place son nouveau langage de programmation, *Swift*. *Swift* est dérivé de l'*Objective-C* mais Apple nous assure que ce nouveau langage est 2.6 x plus rapide que son prédécesseur (Apple Inc. d, 2014). Par contre, lorsque nous développons pour Apple, la simulation d'applications est uniquement possible sur machine Mac via le *compilateur* xCode. En effet, seul ce dernier possède les *SDK* officiels d'Apple (Martin, 2012).

### 1.2.3 Windows- Microsoft

Windows est le principal système d'exploitation vendu par l'entreprise Microsoft fondée en 1975 par Bill Gates et Paul Allen (Microsoft c, 2013). Le logo de Microsoft représente une fenêtre. Ce dernier est illustré dans la Figure 5 : Logo Microsoft. Le 29 juillet 2015, Microsoft sortira son nouvel OS multiplateforme, Windows 10. Celui-ci succède à la version actuelle, Windows 8.1. Ce nouvel OS sera un mélange entre l'ancienne version Windows 7 et Windows 8.1 et par la même occasion, le menu démarrer fait son retour (Microsoft d, 2015).

*Figure 5 : Logo Microsoft*


*Source : (Microsoft e, 2015)*

Depuis l'année 2010, Windows phone est le système d'exploitation mobile qui succède à Windows Mobile (Hollister, 2010). Windows Phone se différencie de ses concurrents grâce à son interface utilisateur composée de tuiles dynamiques. Les applications publiées dans la boutique en ligne de Windows se distinguent de deux manières. Premièrement, il y a les applications développées avec une version antérieure de Windows Phone 8 grâce au Silverlight 7.0. Elles continuent d'être supportées sur le Windows Store grâce à la mise à jour de Silverlight en version 8.1. Deuxièmement, il y a les nouvelles applications créées depuis la version 8 de Windows. Elles utilisent le Common *XAML* (Cathala, 2014).

Le système d'exploitation Windows phone 8 dispose d'une architecture de 4 couches comme

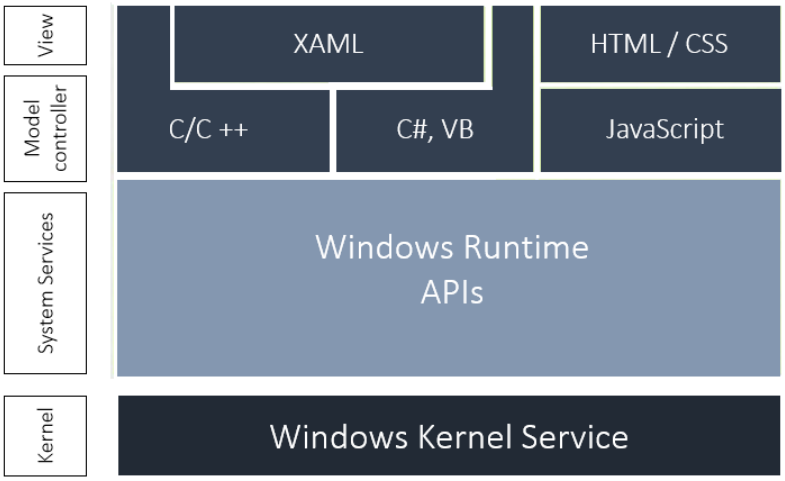
illustré dans la Figure 6 : Architecture de Windows de la page suivante.

La base est le noyau central de Windows qui permet d'accéder au matériel du smartphone via divers pilotes.

Ensuite, nous trouvons le Windows Runtime. Les *API* contenues dans ce dernier permettent d'accéder aux données, aux médias, …

Les deux dernières couches sont liées ensembles. Si une page *XAML* est lié avec une classe en *C#*, nous trouverons dans la page *XAML* la définition des éléments graphiques. Mais les actions concernant ces éléments seront décrites dans la classe liée à la page *XAML*. On peut tout de même, via une classe, définir des éléments graphiques. Pour résumer, le Model Controller (les classes) définit les actions des éléments graphiques contenus dans les vues (View).

*Figure 6 : Architecture de Windows*




*Source : Adapté de (Brynte, 2014)*

Avec le prochain Windows 10, une même application sera disponible sur tablette, mobile et pc. C'est ce que Microsoft appelle le « Universal Windows Plateform ». Les applications créées depuis Windows 10 pourront être écrites en *C#*, C++, *JavaScript* et Visual Basic (Microsoft a, 2015).

## 1.3 Développement Mobile

Dans le milieu du développement mobile, toutes les applications ne se ressemblent pas. Plusieurs différences majeures sévissent et font de ce domaine un milieu complexe. La suite de ce chapitre nous expliquera les différents types d'applications au niveau du développement. Nous ne parlerons pas d'applications de type jeux, productivité ou communication. Il sera question des différentes approches de développement d'applications smartphone.

En premier lieu, nous trouvons des applications de type **web**. Ce sont des sites web dont le design s'adapte à l'écran sur lequel il s'affiche (Responsive design). Des applications de ce genre n'accèdent nullement aux fonctionnalités de base du smartphone.

En total opposition aux applications web, nous pouvons trouver les **applications natives**. Une application native est une application développée spécifiquement pour un système d'exploitation. Elle pourra accéder aux fonctionnalités de base d'un smartphone (tel que la caméra ou le GPS) et possède un accès complet à l'interface utilisateur propre à chaque plateforme.

Dernièrement, il existe un mix entre une application web et une application native. Cette approche, appelé **hybride**, possède les caractéristiques visuelles d'une application native. Par contre, le développement est fait grâce à des langages de web et généré via les navigateurs web.

Ces approches diverses posent des barrières d'entrées lorsque du développement d'applications smartphone doit être fait.

## 1.4 Synthèse

Le marché mondial des smartphones est actuellement dirigé par trois grandes entreprise: Android, Apple et Microsoft. Ces trois entreprises distribuent des systèmes d'exploitations disponible sur smartphone, tablette ou bien même des ordinateurs.

- Android Lollipop[1] est la dernière version du système d'exploitation d'Android. Le développement d'applications sur cette plateforme nécessite de connaître le *Java* mais aussi le *XML* pour la création des éléments graphique d'une page.

- Le système d'exploitation d'Apple, iOS 8.4 [2], permet d'avoir des applications écrites en *Objective-C* mais aussi dans le nouveau langage d'Apple, le *Swift*.

- Windows phone 8.1 [3] est la version actuelle du système d'exploitation proposé par Microsoft. Les applications conçues sur cet OS peuvent être écrite en divers langage tel que le *C#*, *JavaScript* ou bien même l'*HTML*, …

Les entreprises proposant ces systèmes d'exploitation mettent à disposition un univers de développement officiel pour créer des applications. Cet univers contient tous les *SDK*, composants et bibliothèques nécessaires au bon déroulement de la phase de développement.

[1] Android prévoie une nouvelle version de son OS, Android M. La date de sortie n'a pas encore été communiquée (Eason, 2015).
[2] iOS 8.4 est sortie le 4 juillet 2015. iOS 9 est prévu pour automne 2015 (Merli, 2015).
[3] La prochaine version de l'OS de Windows, Windows 10, sera disponible dès le 29 juillet 2015 (Microsoft d, 2015).

Lorsqu'une application est créée pour une de ces plateformes, elle est disponible sur les plateformes de téléchargement légal pour application (boutique en ligne) propre au système d'exploitation.

Un résumé des points ci-dessus sous forme de tableau (tableau 1 : synthèse acteurs du marché des smartphones) est disponible à la page suivante.

*Tableau 1 : Synthèse acteurs du marché des smartphones*

| | **Android** | **iOS** | **Windows Phone** |
|---|---|---|---|
| **Langage** | *Java* | *Objective*-C / Swift | *C#* |
| **IDE** | Android Studio | XCode | Visual Studio |
| **Boutique en ligne** | Play Store | AppStore | Windows Store |
| **Version OS (au 20.07.215)** | Lollipop | iOS 8.4 | Windows 8.1 |

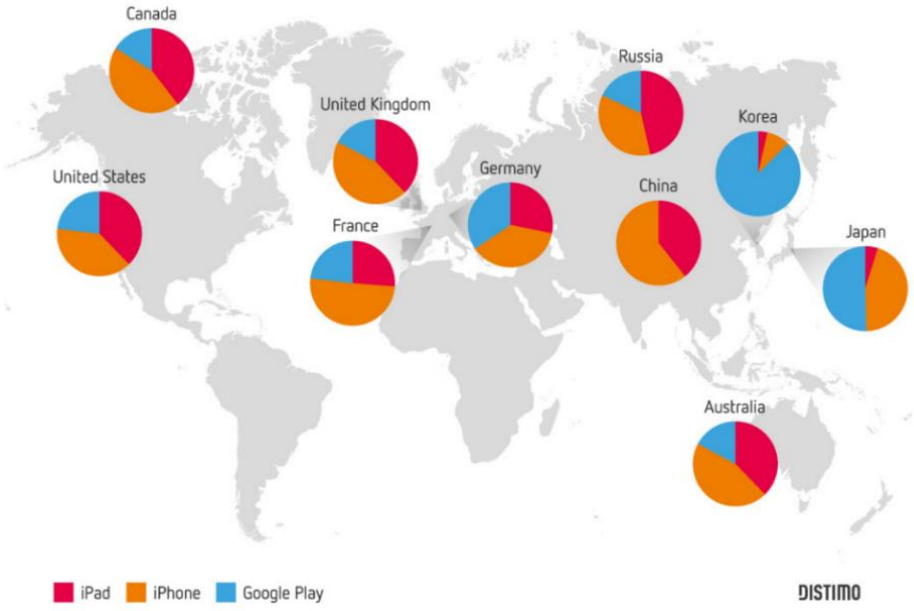
*Source : Données de l'auteur*

## 1.5 Problématique

Comme vu précédemment, l'univers du développement mobile est vaste et plusieurs barrières d'entrée sont présentes.

Premièrement, le marché sur lequel nous allons lancer notre application sera déterminant face au choix de l'*OS*. La Figure 7 : Comparatif des revenus de l'Apple App Store et du Google Play Store nous illustre cet écart des revenus généré par IOS et Android mondialement. Même si Android est le leader sur le marché mondial, il se trouve que l'Apple store génère le plus de profit. Le profit généré de notre future application dépendra donc de ce choix au niveau du marché. C'est un enjeu principal pour les entreprises qui veulent créer des applications ; celui d'être présente sur les plateformes en tête du marché mais générant un maximum de profit. (Pierrat, 2012).

*Figure 7 : Comparatif des revenus de l'Apple App Store et du Google Play Store*




*Source : (Distimo, 2013)*

De cette première barrière se découle une contrainte, le développement de l'application. Si une entreprise oriente sa stratégie dans une application mobile multiplateforme, elle se retrouve face à la difficulté de créer une application commune à différentes plateformes. Comme vu dans le chapitre « 1.2 Situation actuelle », chaque acteur principal du marché mondial des OS mobiles possède son propre langage de développement. Choisir sur quelle plateforme lancer son application n'est pas simplement porter un choix sur le marché de lancement. Il en découle le choix de la personne à engager pour le développement de l'application. Si nous avons choisi de lancer l'application sur le marché de l'iOS, nous aurons donc la nécessité d'avoir dans l'équipe un développeur connaissant l'*Objective-C* ou *Swift*. Les ressources humaines sont donc une variable importante à considérer dans la prise de décision du marché de lancement.

La dernière barrière d'entrée se trouve être les tendances actuelles et la demande des applications. Selon une étude de l'institut Nielsen, le nombre d'applications utilisées par mois se comptait en moyenne au nombre de 26 applications différentes par utilisateurs (Nielsen Institut, 2014). De plus, le nombre d'heures passées sur ces applications varie selon les catégories. Dans le Tableau 2 : Top 10 des catégories d'applications par personne et par temps, nous pouvons nous apercevoir que le temps passé sur des applications de la catégorie « Recherche et social » se monte à pratiquement 11 heures. En 2014, pour la même catégorie, un utilisateur y passait 8h30. Cela nous indique une nette hausse dans l'utilisation de cette catégorie d'applications. Des statistiques de ce genre peuvent nous aider à créer une application qui répond aux besoins actuels des consommateurs.

*Tableau 2 : Top 10 des catégories d'applications par personne et par temps (minutes)*

| CATÉGORIE | 2012 | 2013 | Croissance |
|---|---|---|---|
| UNITÉ | heures | heures | pourcent |
| **Recherche et social** | 8.33 | 10.56 | 28% |
| **Divertissement** | 6.11 | 10.34 | 71% |
| **Communication** | 3.29 | 3.48 | 9% |
| **Productivité & Outils** | 2 | 2.16 | 14% |
| **Commerce & Shopping** | 1.23 | 1.33 | 12% |
| **Information** | 1 | 1.33 | 55% |
| **Voyage** | 1.14 | 1.18 | 6% |
| **Photographie** | 0.26 | 1.01 | 131% |
| **Finance** | 0.33 | 0.35 | 10% |

*Source : Adapté de (Nielsen Institut, 2014)*

Ces trois barrières dans le développement d'application peuvent être résumées de la manière suivante :

- Le choix du marché est une question à se poser dès le départ. Android n'a pas le même impact sur le marché européen que sur le marché asiatique.

- Les Ressources Humaines à dispositions dans l'entreprise auront une importance sur le choix précédent. En effet, si nous avons décidé de nous orienter sur le marché d'iOS, nous aurons besoin d'un développeur connaissant le langage de programmation d'iOS, c'est-à-dire l'*Objective*-C ou le *Swift*.

- Les tendances actuelles auront une influence sur le type de notre application. Actuellement, une application de type social aura un plus grand succès qu'une application de type finance (Nielsen Institut, 2014).

Une question demeure. Pourquoi ne pas créer une application sur plusieurs plateformes ? En effet, cela éliminerai la première barrière du choix du marché. Malheureusement, la condition concernant les ressources humaines restera valable. De plus, si nous choisissons de recréer à chaque fois l'application pour une autre plateforme, nous risquons de vouloir l'améliorer et au final, l'application divergera avec celle de base proposée sur l'autre plateforme.

La problématique de ce rapport est donc le choix de créer une application multiplateforme tout en enlevant cette contrainte des ressources humaines à disposition dans une entreprise.

La question suivante résume cette problématique :

*Comment créer une application iOS, Android et Windows Phone, sans avoir besoin de plusieurs développeurs connaissant les langages de développement propres à chaque plateforme ?*

# 2 Développement mobile multiplateforme

Une solution à la problématique développée dans le chapitre « 1.5 Problématique» se trouve dans des applications dites « multiplateforme » développées dans un unique langage. Une application développée dans ce sens pourra fonctionner sur des plateformes telles qu'iOS, Android ou Windows Phone tout en étant rapidement et facilement développable. Ainsi les barrières d'entrée sont évincées.

Il existe de nombreux logiciels qui permettent de passer facilement d'un système d'exploitation à un autre. Pour les besoins de ce travail, trois logiciels ont été sélectionnés.

L'analyse suivante listera diverses approches quant à l'écriture d'application multiplateforme. La description du programme comprendra un historique de la solution ainsi qu'un descriptif du fonctionnement et les avantage & inconvénients de ce dernier.

## 2.1 Application : Native

Une première solution pour le développement d'application mobile multiplateforme est d'utiliser la technologie de chaque plateforme.

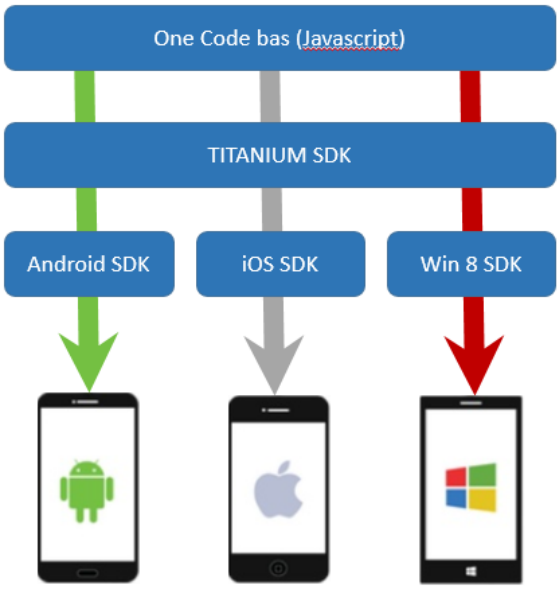
Un logiciel proposant ce genre de service disposera d'une série d'outils graphiques qui s'adaptera sur chaque plateforme. Proposer ce genre d'outils permet d'optimiser les performances de l'application sur chaque plateforme. Néanmoins cela demande une immense charge de travail et limite donc le panel d'option. De plus, il se peut qu'une adaptation manuelle soit faite par le développeur sur chaque plateforme.

### 2.1.1 Titanium Mobile Appcelerator

Appcelerator Titanium mobile est un *Framework* de développement d'application mobile qui propose de développer des applications natives pour chaque plateforme comprenant Android, iOS, Windows Phone, BlackBerry *OS* et *Tizen* (Whinnery K. b., 2010). Cette solution est distribuée par l'entreprise Appcelerator depuis décembre 2008 (Hendrickson, 2008).

Afin de créer une application native via ce logiciel, les créateurs de Titanium Mobile ont mis à disposition une série de composants native aux plateformes en *JavaScript*. C'est-à-dire qu'ils ont créé des méthodes en *JavaScript* qui appellent les méthodes officielles écrites dans le langage de programmation de la plateforme. Lors de l'émulation, le logiciel comprendra le code *JavaScript* puis le compilera en *Objective*-C pour iOS, *Java* pour Android ou *C#* pour Windows Phone. La Figure 8 : Fonctionnement de Titanium Mobile nous illustre le comportement du Titanium Mobile Appcelerator. Lorsque nous créons un élément graphique en *JavaScript*, ce dernier va appeler l'élément graphique correspondant dans le *SDK*. Ainsi, L'affichage final sera un élément graphique propre au système d'exploitation.

*Figure 8 : Fonctionnement de Titanium Mobile*

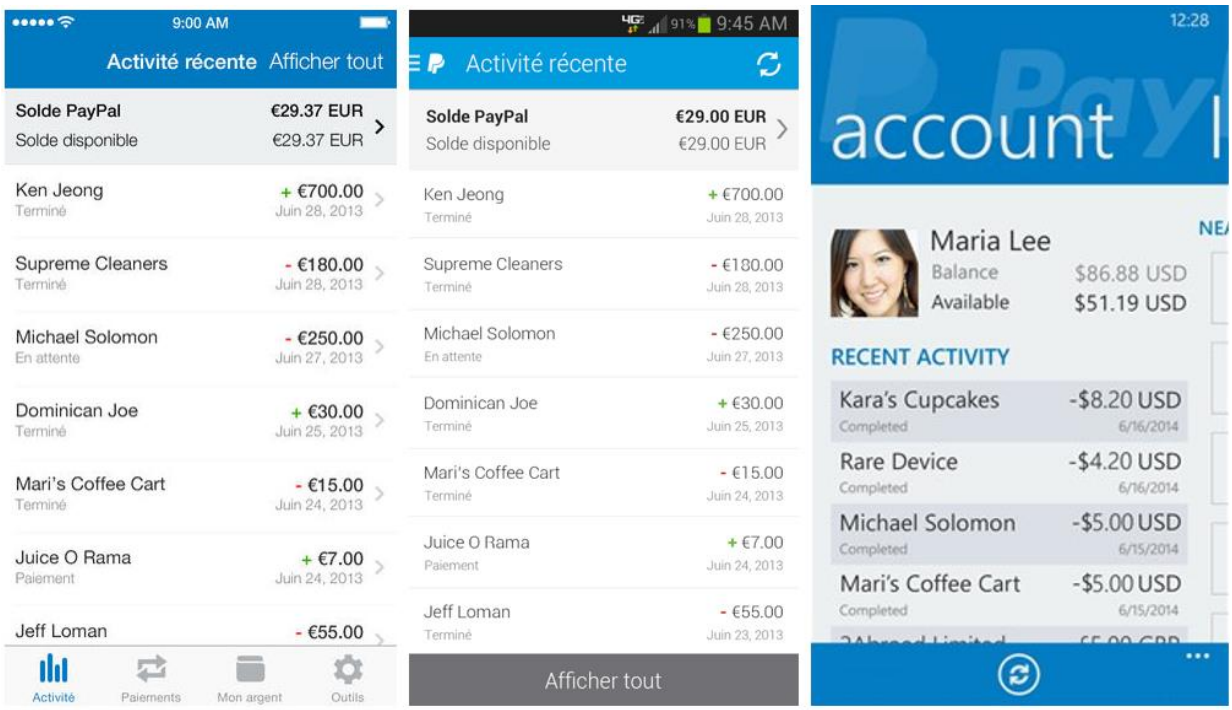



*Source : (Jayamaha, 2014)*

Pour développer sur Titanium Mobile, le développeur devra installer les *SDK*. Malheureusement, si nous voulons développer des applications iOS, nous sommes dans l'obligation d'installer Titanium Studio sur une machine Mac dans le but de posséder le *SDK* pour iOS. Dans ce cas, la simulation d'iPhone et d'Android est possible. Dans le cas contraire, si nous utilisons une machine Windows, seules les applications Windows Phone et Android pourront être testées (Lamoureux, 2012).

De nombreuses entreprises ont fait confiance à ce logiciel pour le développement de leur application. Tel est le cas de Paypal, service de paiement en ligne, dont l'application iOS, Android et Windows Phone est illustré dans la Figure 9 : Exemple d'application avec Titanium Appcelerator (Paypal) de la page suivante.

*Figure 9 : Exemple d'application avec Titanium Appcelerator (Paypal)*




*Sources : iOS :* (PayPal a , 2015)*, Android* (PayPal b, 2015)*, Windows Phone :* (Paypal c , 2015)

Selon un article apparu sur le blog d'Appcelerator, les points forts et les points faibles de Titanium sont les suivants (Whinnery K. a., 2012) :

- ✓ Interface utilisateur native, développement facile et rapide en *JavaScript*
- × Impossible de faire des applications complexes, mauvaise gestion des animations

## 2.2 Application : « dessinée »

Une seconde option dans le développement d'application multiplateforme, qui découle de la première, est de ne pas utiliser des objets préconstruits, mais de les dessiner soi-même via des librairies de bas niveau (p.E. OpenGL). Cela permet d'avoir une grande flexibilité dans la gestion de l'application sur chaque plateforme. Cette approche est principalement utilisée pour créer des applications de type jeu.

### 2.2.1 Unity3D

Unity3D est un moteur de jeu conçu pour les smartphones, le bureau, les navigateurs et les consoles de jeu vidéo. Ce moteur propose un logiciel nous permettant de créer des jeux vidéo multiplateformes en 2D et 3D. De nombreux jeux vidéo actuels ont été édités via Unity3D.

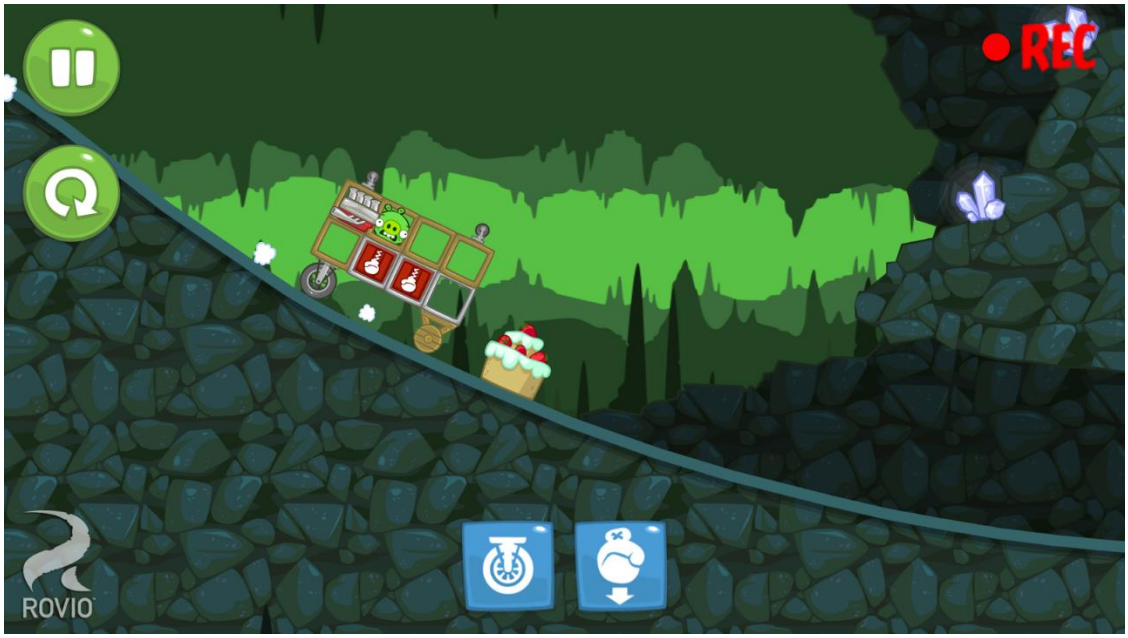
Développer des applications en utilisant Unity3D demande des scripts qui soient écrits soit en *C#*, en UnityScript ou en *BOO* (Birch, 2014). De ce fait, le moteur Unity3D sera apte à les

lancer sur le smartphone cible. La version libre propose de créer des applications sur les smartphones du marché actuel (Unity, 2015).

Ce genre d'application « dessinée » n'accède pas aux fonctionnalités de base d'un smartphone.

Unity3D a été utilisé par l'entreprise Rovio, à qui nous devons les jeux d'Angry Bird, afin de créer Bad Piggies. Ce jeu est illustré dans la Figure 10 : Exemple d'application avec'Unity3D (Bad Piggies).

*Figure 10 : Exemple d'application avec'Unity3D (Bad Piggies)*




*Source :* (Rovio Entertainment Ltd., 2015)

Selon un article apparu sur un site internet dédié au développement de jeu vidéo, les points forts et les points faibles d'Unity3D sont les suivants (Samad, 2012):

- ✓ L'utilisation des scripts rend le développement facile, peut être entièrement développé en *JavaScript*
- ✗ Dépendant entièrement d'Unity3D et impossible de rajouter des éléments supplémentaires.

## 2.3 Application : Navigateur web

L'option du navigateur web est une excellente approche en termes d'interopérabilité. En effet, chaque plateforme possède un navigateur web et ces navigateurs possèdent tous le même standard qu'est *l'HTML5*.

Néanmoins, le plus gros désavantage d'une telle application réside dans le fait qu'elle n'accède pas aux fonctionnalités du mobile tel que la caméra. De plus, elle ne sera pas considérée comme une application native de l'appareil.

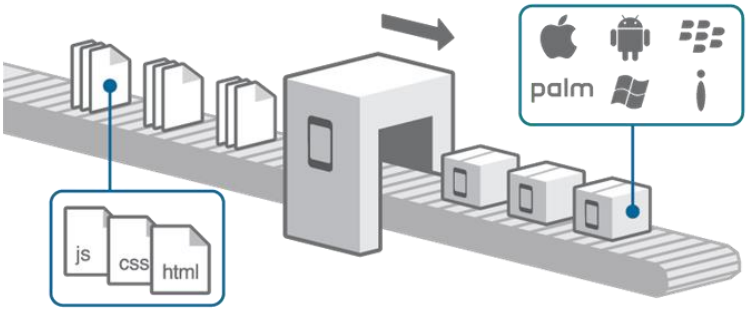
### 2.3.1 PhoneGap

Logiciel dit *open source*, PhoneGap propose un environnement pour créer des applications multiplateformes sur navigateur web. Ce logiciel fait partie de la famille des produits proposés par Adobe depuis 2011 (Feugey, 2011)

PhoneGap est un Framework dit hybride. Il permet de créer des applications sur navigateur, mais il permet aussi d'accéder aux fonctionnalités de l'appareil cible via des Frameworks *JavaScript*. Niveau développement, des technologies du web peuvent être utilisées telles que *JavaScript*, *HTML5* ou le *CSS*. Le fonctionnement, illustré grâce à la Figure 11 : illustration de PhoneGap, réside dans le concept où le développeur peut définir les éléments graphiques via des pages *HTML* et *CSS* puis lier ces éléments graphiques à des actions écrites en *JavaScript*. Le tout sera compilé pour chaque plateforme (BenjaminL, 2014).

Grâce à un tel logiciel, les problèmes de comptabilité ne se posent plus. Les applications iOS, Android et Windows Phone sont basées sur un seul et unique code, que ce soit au niveau *back-end*, mais aussi *front-end.*

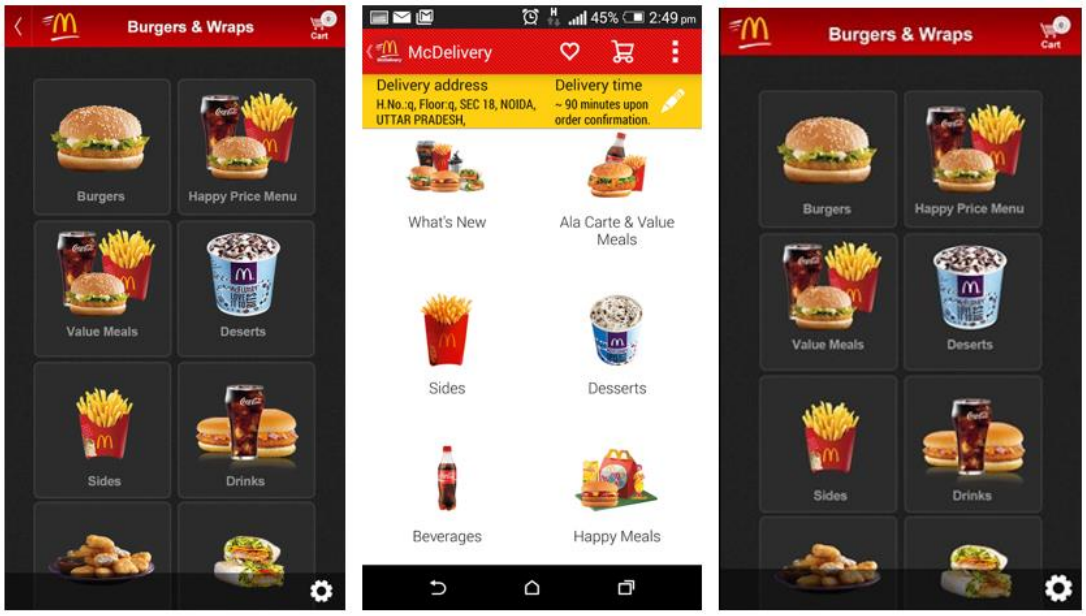
*Figure 11 : illustration de PhoneGap*




*Source :* (Cowlabstudio, 2014)

L'application McDelivery, illustré par la Figure 12 : Exemple d'application avec PhoneGap (McDelivery), est disponible sur les trois principaux OS grâce à PhoneGap. Cette application est utilisée en Inde (Nord et Est) afin de commander à domicile un menu McDonald.

*Figure 12 : Exemple d'application avec PhoneGap (McDelivery)*
(Idealake Information Technologies PVT LTD a, 2015)




*Source : iOS : Android :* (Idealake Information Technologies PVT LTD b, 2015)*, Windows Phone :* (Idealake Information Technologies PVT LTD c, 2015)

Selon un article paru sur le site internet Cygnet Infotech (Cygnet Infotech., 2015), les points forts et les points faibles de Phone Gap sont les suivants :

- ✓ Rapidité de développement grâce à l'*HTML*/*CSS*. Aucune nécessité d'installer les *SDK*.
- × Mauvaises performances dû à l'utilisation d'un navigateur web, *Framework* de trop bas niveau.

## 2.4 Xamarin

Xamarin, originellement appelé MonoTouch, est une entreprise américaine fondée en 2011 dans le but de créer rapidement et facilement des applications mobiles. Les produits de Xamarin permettent la création et la maintenance d'applications multiplateformes sur des appareils tels que les téléphones, les tablettes, mais aussi les appareils embarqués sous licence iOS, Android et Windows.

Cette entreprise possède son propre environnement de développement utilisant le langage de développement *C#* et le *.Net Framework*. Cela nous permet de créer des applications de type natives en utilisant les différents *API* et interfaces utilisateurs de chaque plateforme.

Xamarin permet de faire du développement multiplateforme grâce à sa bibliothèque Xamarin.Forms. Elle permet d'écrire en *C#* tout l'univers de l'application. Que ce soit le côté

business ou le côté interface utilisateurs, le code est **écrit une seule et unique fois**, sans nécessiter de modifications tant du développeur que de l'utilisateur final, possédant un des trois système d'exploitations.

Une application développée grâce à la technologie multiplateforme de Xamarin est Wordament. Wordament est un jeu de lettre en temps réel réalisé par Microsoft disponible sur iOS, Android et Windows Phone (Xamarin Inc. p, 2014). Cette application est illustrée dans la Figure 13 : Exemple d'application avec Xamarin (Wordament) avec le jeu sur une plateforme iOS.

*Figure 13 : Exemple d'application avec Xamarin (Wordament)*


*Source : (Wordament, 2013)*

Les points forts et les points de Xamarin, selon les données récoltées par l'auteur, sont illustrés dans le chapitre « 5.1 Synthèse générale ».

## 2.5 Comparatif

Après avoir brièvement introduit ces différents logiciels, nous pouvons nous apercevoir que chaque logiciel possède des caractéristiques similaires mais aussi différentes. C'est pourquoi, le Tableau 3 : Compratif des logiciels nous résume ces différences.

*Tableau 3 : Compratif des logiciels*

| | **Titanium** | **PhoneGap** | **Xamarin** | **Unity 3d** |
|---|---|---|---|---|
| **Plateformes supportées** | iOS, Android, Windows Phone, BlackBerry, Tizen [a] | iOS, Android, Windows Phone, BlackBerry [g] | iOS, Android, Windows Phone [j] | iOS, Android, Windows Phone, BlackBerry, Tizen [4] [d] |
| **Langage** | *JavaScript* [a] | *HTML5*, *CSS*, *JavaScript* [g] | *C#* [j] | *BOO*, *JavaScript*, *C#* [e] |
| **Interface utilisateur** | Native [a] | Web [i] | Native [j] | Dessinée |
| **Accès aux fonctionnalités natives** | Complet [b] | Partiel [h] | Complet [j] | — [d] |
| **Prix de la Licence / mois** | $ 259 [c] | $ 9.99 [i] | $ 83 [k] | $ 75 [f] |

[4] Sans compter toutes les plateformes non-mobile tel que Windows, Windows Store Apps, Mac, Linux, Steam OS, navigateur web, WebGL, PlayStation 3, PlayStation 4, Playstation Vita, Xbox One, Xbox 360, Wii U, Android TV, Samsung SMART TV, Oculus Rift, Gear VR et Microsoft Hololens (Unity Technologies a, 2015)

Sources : *Tableau de l'auteur provenant de sources multiples*

[a] Appcelerator (2015), The Appcelerator Platform

[b] Appcelerator (2015), Titanium Plateform Overview

[c] Appcelerator (2015), Pricing

[d] Unity3d (2015), Multiplatform

[e] Unity3d (2015), Editor

[f] Unity3d (2015), Get Unity Professionnal Edition

[g] PhoneGap (2015)

[h] PhoneGap (2015), Feature

[i] PhonGap (2015), Build Informations

[j] Xamarin (2015)

[k] Xamarin (2015), Store Business with Visual studio support (Windows Phone)

Afin de ressortir le prix de la licence, les logiciels devaient fournir les caractéristiques suivantes :

- Développement et simulation iOS, Android, Windows Phone
- *IDE*
- Permet le déploiement final des applications sur les boutiques en ligne
- Accès à une base de données local (p.ex. SQLite)
- Accès aux fonctionnalités (Si disponible)
- Développement de plus d'une application

Le logiciel PhoneGap est sous licence Apache. Tant que ce dernier possède cette licence, le logiciel reste gratuit et libre de droit (*open source*). Néanmoins, afin de tester les applications créées, il est nécessaire de payer une licence de compilation à Adobe d'un montant de $ 9.99.

# 3 Xamarin – Analyse détaillée

Au commencement, les ingénieurs qui ont créé la société Xamarin avaient déjà développés un logiciel du même style lors de leur engagement dans la société Novell. Cette solution, Mono, est un logiciel *open source* du *.Net* et de l'implémentation du *compilateur* de *C#*. Le projet Mono devait apporter un meilleur outil de développement aux utilisateurs de Linux, mais, il devait aussi être capable de lancer des applications écrites en Microsoft *.Net*. C'est lors du départ de Miguel de Icaza (Icaza, 2003) que Mono fut supporté par Xamarin.

Xamarin nous promet de pouvoir développer une application grâce à un seul langage ; le *C#*. Un code écrit grâce à leur logiciel met en pratique le slogan original de Sun Microsystems : « ***write once, run anywhere*** [5]» (ComputerWeekly, 2002). De plus, l'application développée via Xamarin sera considérée comme une application native sur les smartphones. De ce fait, elle pourra facilement accéder aux fonctionnalités propres de chaque plateforme.

Suite à l'utilisation de Xamarin, nous avons pu observer que les promesses d'écriture de l'application dans un seul langage ont été tenues. En effet, il n'y a pas eu besoin de connaître l'*Objective*-C / *Swift* ou le *Java* Android pour développer l'application sur chaque plateforme. L'engagement de Xamarin à proposer un code partagé à plus de 90 % est accompli.

Sauf cas contraire, tous les problèmes illustrés au chapitre « 5.4 Problèmes rencontrés » sont liés au développement et ne sont en aucun cas des problèmes liés à l'utilisateur final.

## 3.1 Points forts

Xamarin possède une multitude de points forts et ces derniers font de ce programme un élément phare sur le marché du développement multiplateforme d'applications.

En premier lieu, Xamarin propose une liaison complète pour les *SDK* des plateformes, que ça soit iOS, Android ou Windows Phone. Grâce à cela, l'utilisation, la compilation et la vérification au cours du développement se déroule facilement. Cela nous permet d'avoir moins d'erreurs d'exécutions et une application de meilleure qualité, car elle utilisera son propre *SDK* (Strakh, 2015).

Secondement, les bibliothèques des langages de développement natif des applications smartphone (cf. Tableau 1 : Synthèse acteurs du marché des smartphones en page 8) peuvent être rapidement invoqué. Le but de cette fonction est de nous donner la possibilité d'utiliser des instructions natives à chaque plateforme (Wilson, 2014).

Le choix de Xamarin quant au langage de développement – *C#*, n'est pas anodin et cela nous amène au 3e point positif de ce logiciel. En effet, le *C#* est un langage moderne et qui possède des améliorations notables sur le C++, l'*Objective*-C ou *Java*, tel le *LINQ*[6]. La gestion des évènements et des propriétés font de ce langage un excellent choix pour des applications ayant une interface utilisateur graphique (Petzold, 2015).

[5] Ecrire une fois, fonctionne n'importe où
[6] Language Integrated Query – Requête intégrée au langage

*Figure 14 : Exemple d'élément de la bibliothèque Xamarin*


*Sources : (Xamarin Inc. h, 2015)*

Quatrièmement, Xamarin possède une étonnante bibliothèque de classe de base (BCL). Cela permet à notre projet d'avoir accès à des bases de données, de la sérialisation, du support réseau et bien d'autres options. De plus, grâce au *C#*, nous avons accès aux packages inclus dans le gestionnaire *NuGet* de Microsoft. Cela nous ouvre un panel de milliers de packages applicables pour notre application. Cette bibliothèque BCL est utilisable sur toutes les plateformes supportées par Xamarin (pour rappel : Android, iOS et Windows Phones). Néanmoins, les addons ont des spécificités propres à chaque plateforme et ne pourront pas tous être utilisable partout. Pour illustrer ce propos, la Figure 14 : Exemple d'élément de la bibliothèque Xamarin de la page précédente nous montre un élément de cette bibliothèque qui propose le service pour signer des documents sur smartphone. Cependant, nous pouvons nous apercevoir, grâce aux icônes des plateformes sur la droite de l'image, qu'il est disponible uniquement sur Android et iOS.

Si nous voulons écrire une application grâce à la technologie de Xamarin, nous pouvons avoir à notre disposition un environnement de développement intégré (*IDE*) et cela est le cinquième point positif de l'utilisation de Xamarin. Sur une machine Mac, nous pourrons utiliser uniquement Xamarin Studio pour iOS. Par contre, sur PC, nous avons le choix entre Xamarin Studio, mais aussi un Visual Studio avec l'extension Xamarin. Cet *IDE* nous permettra d'écrire l'application, de la tester, mais aussi d'ajouter des bibliothèques externes et de la déployer sur un émulateur ou smartphone physique.

Dernièrement, et un point non négligeable, Xamarin nous offre un support multiplateforme pour les trois plateformes principales (iOS, Android et Windows Phone). Pour certaines applications, 90% du code peut être partagé entre les applications (Xamarin Inc. a, 2015). Ce pourcentage dépendra de l'utilisation de l'application, du choix dans le type de projet multiplateforme que nous sélectionnons et des spécificités dans l'accès aux éléments natifs du smartphone. Ces spécifications sont expliquées plus en détails au chapitre « 3.3 Projets multiplateformes » de ce présent document.

## 3.2 Fonctionnement

Les points forts de Xamarin énumérés au point précédent nous promettent un logiciel puissant. L'analyse suivante nous expliquera comment ce logiciel permet de compiler du *C#* sur de l'iOS, de l'Android ou du Windows Phone. Pour rappel, les applications Windows Phone sont

écrites en *C#* ; elles possèdent donc les capacités de comprendre le code écrit sur Xamarin.

### 3.2.1 Compilation

Xamarin se base sur la technologie Mono qui est le *Framework .Net* en version *open source*. Les applications de Xamarin s'exécutent via des *compilateurs* inclus sur chaque smartphone. Le cache de l'application et son interopérabilité de plateformes sont des fonctionnalités gérées par cette exécution (Xamarin Inc. a, 2015). Les *compilateur*s Mono/*.Net* ne génèrent pas de code spécifique à chaque plateforme. C'est-à-dire qu'il ne va pas directement transformer du *C#* en *Objective*-C, par exemple. Il va générer plutôt une instruction qui va être interpréter par la machine virtuelle *.Net*. Vu que le code va être interprété sur machine virtuelle, il sera plus long à s'exécuter à contrario d'un code écrit dans un langage natif qui sera exécuté directement par le processeur. La manière dont agi ces *compilateur*s diffère d'une plateforme à une autre.

Sur un smartphone Android, une application Xamarin utilisera un *compilateur* dit JIT (Just-in-Time). Ce genre de compilation permet d'analyser le *bytecode*, d'identifier les domaines qui pourraient être traduits en code natif (et donc être accéléré) tout au long de la compilation (Nickel, 2014). En résumé, cette compilation peut être appelée une compilation de type dynamique.

Sur un smartphone iOS, la compilation diffère. En effet, les termes de l'Apple Store refusent la génération de code dynamique. De ce fait, une application Xamarin sur iOS utilisera un *compilateur* dit Ahead-of-Time (AOT). Cela implique que cette dernière soit réalisée statiquement avant le déploiement de l'application (Mono Project, 2015).

Le système d'exploitation Windows Phone supporte déjà le *C#*. De ce fait, cette compilation ce fait directement sans passer par une machine virtuelle interne au smartphone.

## 3.3 Projets multiplateformes

Outre le fait de proposer un environnement de développement pour iOS et pour Android, Xamarin propose une licence permettant de faire du développement d'application multiplateforme.

Lorsque nous choisissons cette option de développement, l'idée de Xamarin est la suivante :

Nous écrivons une seule fois le code qui décrit, par exemple, un bouton, et sans apporter des modifications, le code se déploie sur chaque plateforme sans aucun souci. Cet exemple est très bien illustré dans la Figure 15 : Exemple multiplateformes Xamarin de la page précédente où le code illustré en arrière-plan a été écrit une seule fois et est utilisable par les trois smartphone.

*Figure 15 : Exemple multiplateformes Xamarin*[7]

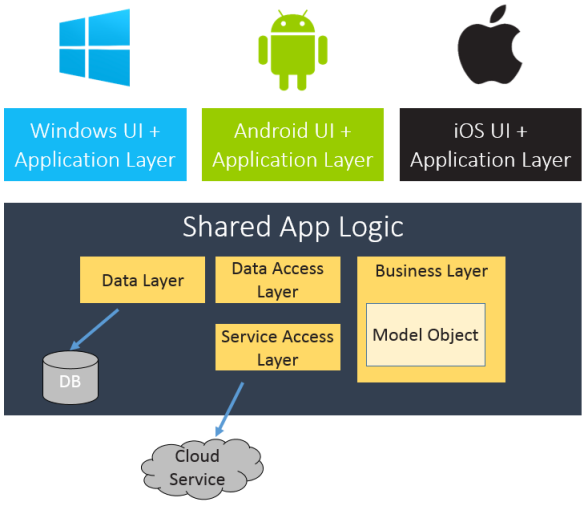

*Source : Adapté de (Xamarin Inc. a, 2015)*

Dans le cas du développement d'application multiplateforme, il existe deux solutions différentes pour y arriver. Ces deux solutions, Native Shared et Xamarin.Forms, sont analysées en détails dans les chapitres suivants.

### 3.3.1 Native Shared

La première, celle par défaut, est une application dont l'architecture nous est décrite dans la Figure 16 : Architecture d'une application Xamarin par défaut.

*Figure 16 : Architecture d'une application Xamarin par défaut*




*Sources : Adapté de* (Xamarin Inc. b, 2014)

[7] Toutes les prochaines figures voulant montrer ce contraste entre écrire une fois le code et pouvoir l'utiliser sur les trois plateformes utiliseront la mise en page de cette figure.

Le tronc commun présenté sur l'image avec le nom « Shared App Logic » est une base écrite en *C#*. Dans cette base, nous retrouvons toute la *couche métier* qui est partagée entre chaque plateforme. La *couche métier* gère tous les accès vers les bases de données (local ou sur le cloud) ainsi que leur gestion (insertion, mise à jour, effacement des données).

Toutes les caractéristiques graphiques (*UI*) et tout le côté *Application Layer* de chaque plateforme sont également écrites en *C#* dans leur propre sous-dossier. Par exemple, si nous voulons retrouver une cellule particulière d'une application iOS, nous devons écrire le code suivant dans le dossier iOS de l'application par défaut :

```
UITableViewCell cell = tableView.DequeueReusableCell(cellIdentifier)
```

Le code ci-après correspond également à la recherche d'une cellule mais écrit dans le langage natif du développement iOS ; l'*Objective-C*.

```
UITableViewCell*cell =
[tableView dequeueReusableCellWithIdentifier:cellIdentifier];
```

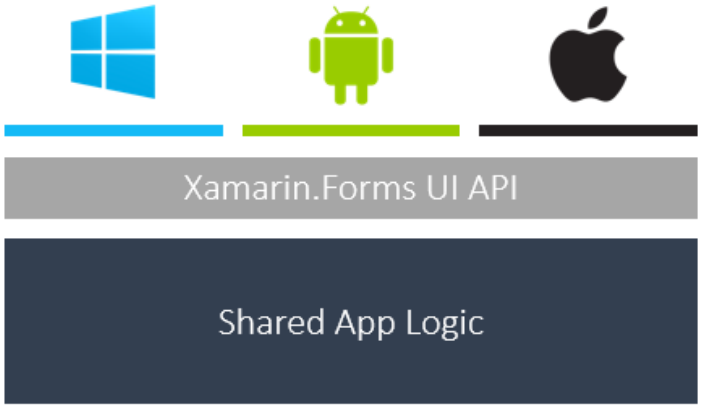
Xamarin nous permet très facilement de créer des applications iOS, Android ou Windows sans avoir besoin de connaissance dans les langages de développement propre à chacune des plateformes.

Une application suivant ce modèle est recommandée lorsque la partie graphique de cette dernière demande une attention particulière comme il est le cas pour des applications possédant de nombreux graphique pour les statistiques ou bien des applications de type jeu.

### 3.3.2 Xamarin.Forms

Le second type d'application multiplateforme que propose Xamarin est appelé Xamarin.Forms.

*Figure 17 : Architecture d'une application avec Xamarin.Forms*




*Sources : Adapté de (Xamarin Inc. b, 2014)*

Comme nous pouvons le voir dans la Figure 17 : Architecture d'une application avec Xamarin.Forms, le tronc commun gérant la *couche métier* d'une application reste le même que dans une application défaut de Xamarin (cf. Figure 16 : Architecture d'une application Xamarin par défaut). Le point essentiel de changement demeure dans la conception graphique de l'application. Dans ce cas, les trois différentes plateformes partagent un seul code dit « graphique ». Cela permet d'avoir un code potentiellement partagé à 100%.

Cette *API* est développée en *C#* et en *XAML* ce qui nous permet d'acquérir rapidement la logique de développement. De plus, un élément graphique écrit via cette *API* aura les spécificités graphiques de la plateforme sur laquelle l'application aura été déployée.

Pour illustrer mon propos, prenons la déclaration d'un bouton à deux positions (On/off). Avec cette *API* fournie par Xamarin.Forms, le code pour définir un tel bouton se fait de la sorte :

```
Switch switcher = new Switch {…}
```

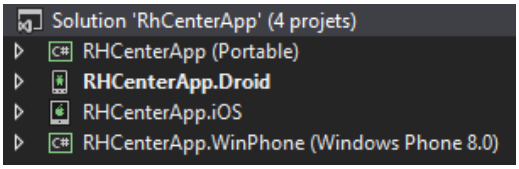
Si nous voulions déclarer un bouton de ce style, nous aurions dû l'écrire en *XML* pour Android et Windows Phone et jouer avec l'interface graphique de xCode pour iOS.

Une application suivant ce modèle est recommandée lorsque des éléments basiques doivent être affichés et pour des prototypes d'applications. De ce fait, c'est le type de développement choisi pour le test de Xamarin. Les détails de ce choix sont décrits dans le point « 4.1 Caractéristique » de ce travail.

## 3.4 Gestion du code en commun

À ce stade, nous nous retrouvons face à un deuxième choix ; celui de la gestion du code en commun (shared code). Il existe deux options possibles.

*Figure 18 : Exemple de la structure d'une solution Xamarin*




*Source : Données de l'auteur*

Chacune de ces deux options possède la même base de projet décrite dans la Figure 18 : Exemple de la structure d'une solution Xamarin.

Un dossier global, nommé solution, contient 4 projets distincts. Le projet de poupe est le projet qui partage le code de l'application. Chacun des trois autres projets (Droid, iOS, WinPhone) accède à ce projet de base via les références. Ces projets sont la définition de chaque plateforme où cette application peut être déployée.

### 3.4.1 Shared

Le premier style de projet possible est un projet dit « Shared Projet ». Ce genre de projet permet de définir rapidement et efficacement chaque accès de données dans un projet. Via des

directives de compilation, la transition de commande entre les différentes plateformes peut être décrite. Grâce à ce système, il est possible de définir qu'une partie du code ne soit exécutable uniquement sur une certaines plateforme. L'architecture de ce style de projet est définie dans la Figure 19 : Structure Shared Code Projet.

Dans la partie partagée du code, nous pouvons trouver les éléments suivants :

**Data Layer** : Cet élément définit toutes sortes de données non volatiles par exemple une base de données *SQLite*.

**Data Access Layer (DAL)** : Cela permet de gérer les données définies dans le Data Layer. Il permet de lire, écrire et effacer (*CRUD*) de manière sécurisée. C'est-à-dire que le code qui implémente cet élément ne connaitra pas les requêtes de donnée contenue dans le Data Access Layer.

**Business Layer (BLL)** : Souvent appelé Business Logic Layer, ce dernier contient les modèles de données et la logique métier. Il permet de définir les actions à effectuer, par exemple, lors d'une modification de donnée.

**Service Access Layer** : Cet élément possède les mêmes caractéristiques que le DAL, simplement qu'il est orienté pour une base de donnée en cloud (web service).

L'accès vers une base de données diffère d'une plateforme à un autre (cf. « chapitre 5.4.4 Base de données SQL »). De ce fait, cet accès doit être défini dans chaque projet de plateforme. Cet élément est appelé ***Application Layer*** et nous le trouvons dans chaque projet plateforme.

*Figure 19 : Structure Shared Code Projet*


*Sources : Adapté de (Xamarin Inc. c, 2015)*

Les points positifs de ce genre de projet sont les suivants :

- Il permet de partager le code à travers de nombreux projets.
- L'utilisation de directive pour le *compilateur* est possible. Par exemple, si un élément de code est disponible uniquement sur Android, il devra être entouré de l'instruction suivante : `#if __ANDROID__` [instruction] `#endif`
- Les projets des applications (Droid, iOS & WinPhone) peuvent utiliser des références que le projet du partage de code utilise.

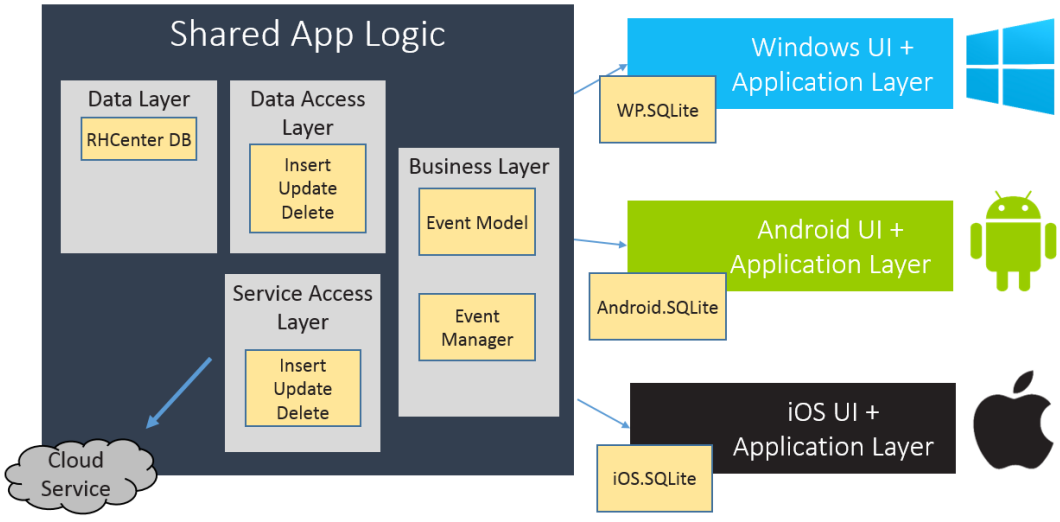
Malheureusement, Xamarin déconseille d'utiliser ce genre de projet lorsque plusieurs développeurs travaillent dessus. Expérience faite, le code devient très vite ingérable (cf. chapitre « 5.4.5 Shared App vs. Portable App »).

### 3.4.2 Portable (PCL)

Le second type de projet, celui choisi au final pour l'application test, est le Portable Class Libraries (PCL). Ce genre de projet, illustré dans la Figure 20 : Structure PCL, permet de partager les librairies.

A contrario du type shared, la structure d'un projet PCL est mieux définie. Avec ce genre de projet, nous définissons une fois le Data Layer, le DAL et le BLL puis ces derniers sont utilisés par des classes définies pour chaque plateforme.

*Figure 20 : Structure PCL*




*Sources : Adapté de (Xamarin Inc. c, 2015)*

De plus, l'utilisation de ce genre de projet nous permet d'utiliser un plus grand nombre d'addons ou de plugin proposés dans la bibliothèque Xamarin ou *NuGet*. Par exemple, XLabs, plugin pour accéder aux fonctionnalités natives de chaque plateforme et utilisé dans l'application test, n'est

disponible qu'en utilisant un projet de type PCL.

Néanmoins, ce type de projet nécessite quelques configurations de base (Xamarin Inc. e, 2015) Son utilisation n'est disponible qu'à partir de :

- Xamarin.Android 4.10.1
- Xamarin.iOS 7.0.4
- Xamarin Studio 4.2
    - PCL est disponible automatiquement sur *OS* X et Visual Studio. Si nous voulons l'utiliser sur l'*IDE* Xamarin Studio pour Windows, il est nécessaire de télécharger l'exécutable PCL sur Xamarin et de suivre les instructions d'installation contenue dans cet exécutable (Xamarin Inc. k, 2015).
- Visual Studio 2013 avec l'extension Xamarin

Les points positifs de ce type de projets sont les suivants :

- Le code partagé est centralisé au même endroit sans être interrompu via des directives pour les *compilateur*s de chaque plateforme.
- Le PCL permet le *refactoring*, c'est-à-dire permettre une modification dans un fichier et propager si nécessaire cette modification dans d'autres fichiers dans le but de maintenir l'intégrité du code (Doudoux, 2005).
- Un projet PCL peut facilement accéder et être référencer à d'autres projets dans la solution.

Par contre, vu que le projet PCL partage les librairies, il ne peut pas accéder à des bibliothèques propres à chaque plateforme (Les projets de plateformes dans la même solution le peuvent).

## 3.5 Installation

Avant de commencer la partie développement et test de ce travail, j'ai dû installer Xamarin Studio avec toutes les extensions. Cela comprend :

- Android *SDK*
- l'aide graphique afin de créer des applications sur Xamarin Studio
- le programme Xamarin Studio

- les extensions Xamarin sur Visual Studio

Xamarin Studio est un environnement de développement permettant de gérer des applications Android et iOS. Les caractéristiques de la machine sur lequel il a été installé sont listées dans le Tableau 4 : Carctéristiques de l'ordinateur d'installation de Xamarin.

*Tableau 4 : Carctéristiques de l'ordinateur d'installation de Xamarin*

| | |
|---|---|
| **Système d'exploitation** | Windows 8.1 |
| **RAM** | 12 GO |
| **Hyper-V** | activé avec capacité de faire de la virtualisation |
| **Espace disque nécessaire** | ~ 10 GO [8] |
| **Visual Studio** | Visual Studio Ultimate 2013 Update 5 |
| **Windows Phone SDK** | Windows Phone 8.1 |

*Source : Données récoltées par l'auteur*

Cet environnement de développement sur PC propose uniquement la simulation de smartphone Android. Lors de l'installation de Xamarin Studio, un package d'extension a été installé sur ma version de Visual Studio. Ce dernier permet, la simulation d'application sur Android et Windows Phone. De plus, il a la possibilité de se lier avec une machine Mac afin de permettre une simulation d'un appareil possédant un iOS (cf. chapitre « 5.4.1 Simulation iOS »).

Étant donné que mon travail consiste à tester une application multiplateforme, toute l'application a été écrite grâce à Visual Studio Ultimate 2013 update 5 qui comprenait l'extension Xamarin.

*Tableau 5 : Caractéristiques du Mac Book Pro pour la simulation*

| | |
|---|---|
| **Système d'exploitation** | *OS* X 10.9.3 |
| **iOS SDK** | iOS 8.4 |
| **xCode** | version 6.4 |

*Source : Données récoltées par l'auteur*

Pour la simulation d'un iOS, il faut posséder une machine Mac avec l'*OS* X 10.9.3, la dernière version de l'iOS *SDK* et la dernière version de l'*IDE* d'Apple, xCode (Xamarin Inc. d, 2015). C'est

[8] Ceci comprend l'installation des diverses SDK (Android, *Java*) et l'intégration dans Visual Studio

ce dernier qui nous permettra de lancer une simulation d'iOS. Ces caractéristiques sont listées dans le Tableau 5 : Caractéristiques du Mac Book Pro pour la simulation.

## 3.6 Synthèse

Xamarin est une entreprise qui permet de faire du développement mobile. Il permet de développer des applications dans un unique langage (*C#*) pour Android, pour iOS ou pour Windows Phone. Une suite d'outils est proposée par cette entreprise. Comme illustré dans le Tableau 6 : Synthèse IDE de Xamarin, chacun de ces outils ou extensions possède des limites que ce soit au niveau du développement ou au niveau de la simulation. Par exemple, sur Xamarin.iOS, il est impossible de développer et de simuler une application Windows Phone. Sur Windows, il est impossible de directement simuler un iOS, mais il est possible de lier le projet à un *OS* X et d'émuler dessus l'application (cf. chapitre « 5.4.1 Simulation iOS »).

*Tableau 6 : Synthèse IDE de Xamarin*

| | **Android** | **iOS** | **Windows Phone** |
|---|---|---|---|
| **Xamarin Studio Mac** | Développement<br>Simulation | Développement<br>Simulation | — |
| **Xamarin Studio Windows** | Développement<br>Simulation | Développement<br>Simulation * | — |
| **Extension Visual Studio** | Développement<br>Simulation | Développement<br>Simulation * | Développement<br>Simulation |

** uniquement en liaison avec une machine Mac*

*Source : Données récoltées par l'auteur*

Pour le développement multiplateforme, nous avons le choix entre plusieurs types de projets. Soit nous utilisons le type Native Shared, soit nous utilisons Xamarin.Forms. Dans le premier cas, l'importance est mise sur l'aspect graphique de l'application de chaque système d'exploitation. Le deuxième type d'application est très souvent utilisé pour des prototypes, c'est-à-dire lorsque le graphisme est moins important que la gestion des données.

Le Tableau 7 : Default App vs. Xamarin.Forms de la page suivante nous montre divers points de différence entre les deux types de projets.

*Tableau 7 : Default App vs. Xamarin.Forms*

| | Native Shared | Xamarin.Forms |
|---|---|---|
| **Code partagé** | Oui | Oui |
| **Interface utilisateur** | Définit uniquement dans chaque projet de plateformes | Définit dans le projet partagé |
| **Graphisme** | Important et différent pour chaque plateforme | Basique |
| **Type d'application** | Graphique | Prototype, Affichage de données |

*Source : (Xamarin Inc. f, 2015)*

Un second type de choix est proposé. C'est celui du partage de code. Le Tableau 8 : Shared code vs. PCL nous montre les principaux points de comparaison entre un projet de type shared et de type PCL développé dans le chapitre « 3.4.1 Shared » de ce présent travail.

*Tableau 8 : Shared code vs. PCL*

| | Shared Code | Portable Class Librairies (PCL) |
|---|---|---|
| **Directives pour *compilateurs*** | Oui | Non |
| **Utilisation des références par les projets d'applications** | Oui | Oui |
| **Structuré** | Non | Oui |
| **Bibliothèque de plugin** | Peu de plugin disponible | Grande bibliothèque |

*Sources : (Xamarin Inc. e, 2015)*

# 4 Application

Le but de ce travail est de tester un logiciel, Xamarin, permettant de créer des applications mobiles multiplateformes. Cette troisième partie consiste à développer une application en répondant à la problématique citée dans le chapitre « 1.5 Problématique ».

## 4.1 Caractéristique

L'application bêta développée dans le cadre de ce travail est une extension du site web RH-Center. Lors de l'option Business Expérience (BEX) proposée à l'HES-SO Valais Wallis, la start-up RH-Center a été fondée par Steiner François, Navarria Alessandro et Dupont Audrey. RH-Center est un site web pour la gestion des plannings d'une entreprise employant plus de 50 personnes.

Afin d'être mobile, RH-Center avait besoin d'une application disponible sur plusieurs plateformes.

L'application RH-Center est un complément à son site internet. Le site internet permet la gestion complète des ressources humaines par un responsable de ces dernières. L'application, quant à elle, permet à un employé de rapidement voir les dates auxquelles il travail mais aussi d'annoncer une indisponibilité pour ces dernières.

Un canevas de l'application a été créé afin d'illustrer les cas d'utilisation. Les images de ce canevas sont disponibles dans l'Annexe III Annexe I et les cas d'utilisation de ce dernier sont illustrés dans le Product Backlog de l'Annexe II .

## 4.2 Scénario

Pour des raisons de sécurité et de confidentialité, les informations suivantes ne correspondent en aucun cas à un cas réel. Elles sont présentes afin d'aider le lecteur de ce travail à la compréhension de l'application.

Notre cas pratique se déroule dans une entreprise du milieu de la nuit. Cette société, le Valais Excellency, possède dans son sein un panel de 10 personnes travaillant à temps partiel. Seul 3 employés, travaillant depuis 2 ans dans cette société, ont reçu les accès pour travailler avec l'application RHCenter. Selon leur feedback, Valais Excellency décidera si l'application RHCenter sera utilisée par chaque employé.

Les employés sélectionnés pour travailler avec cette application sont listés dans le Tableau 9 : Liste employées Valais Excellency pour RHCenter de la page suivante. Les informations de login sont inscrites afin de pouvoir tester l'application.

*Tableau 9 : Liste employées Valais Excellency pour RHCenter*

| Nom prénom | Login | Mot de passe |
|---|---|---|
| Aurélie Denis | ad | pwdad |
| Erwoan Denis | ed | pwded |
| François Steiner | fs | pwdfs |

*Source : Données de l'auteur*

Valais Excellency organise de nombreux évènements. Comme présenté dans le Tableau 10 : Listes des évènements pour Valais Excellency, chaque évènement possède un titre le décrivant. De plus, une date de commencement et une date de fin sont définies pour chaque évènement.

*Tableau 10 : Listes des évènements pour Valais Excellency*

| Titre | Date commencement | Date Fin |
|---|---|---|
| Mousse Party | 18.05.2015 | 19.05.2015 |
| Happy Week End | 17.07.2015 | 19.07.2015 |
| Black and White Party | 27.07.2015 | 28.07.2015 |
| 1er Aout | 31.07.2015 | 01.08.2015 |

*Source : Données de l'auteur*

De base, chaque employé travail pour des soirées selon ses disponibilités. Le Tableau 11 : Liste des disponibilités par employés de Valais Excellency nous montre les évènements de travail de chaque employé.

*Tableau 11 : Liste des disponibilités par employés de Valais Excellency*

| Employé | Evènement |
|---|---|
| Aurélie Denis | Mousse party, Happy Week End |
| Erwoan Denis | Happy Week End, Black and White Party |
| François Steiner | Black and White Party, 1er Aout |

*Source : Données de l'auteur*

Grâce à ces diverses informations, la compréhension des points suivants sera plus claire et concise.

Afin de d'illustrer l'utilisation de cette application, imaginons que Madame Aurélie Denis souhaite se connecter afin d'annoncer une indisponibilité pour l'évènement Mousse Party. Une fois l'opération effectué, elle souhaite modifier son mot de passe. L'explication des pages suivra cet exemple.

## 4.3 Développement

Comme expliqué au point « 3.5 Installation », toute la création de l'application a été faite dans l'environnement de développement de Visual Studio.

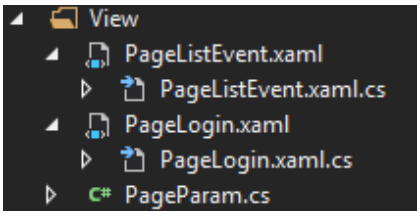
Le choix concernant le type d'application multiplateforme s'est porté sur le Xamarin.Forms. Cela permettait de tester les prétentions de Xamarin concernant le pourcentage de code partagé entre les différentes plateformes. De plus, il a été décidé par les membres de l'équipe RH-Center que cette application ne devait être qu'un prototype.

Le choix pour la gestion du code fut plus complexe. Au départ, le développement fut commencé par le code partagé (shared). Néanmoins, au fur et à mesure que des conditions de

plateforme agrémentaient le code, il devint de plus en plus difficile de le relire facilement. C'est pourquoi un changement d'orientation fut choisi. L'application fut mutée sur une gestion de code de type PCL. Ainsi une meilleure structure du code fut implémentée et la lisibilité de ce dernier, augmentée. L'explication complète de ce changement de stratégie peut être lu dans le chapitre « 5.4.5 Shared App vs. Portables App »).

Afin de tester le plus d'options proposées par Xamarin.Forms, il a été décidé de changer de technique de développement des éléments graphiques pour chaque page de l'application comme illustré dans la Figure 21 : Pages de l'application.

*Figure 21 : Pages de l'application*




*Source : Données de l'auteur*

- La vue login utilise une page *XAML*.
- La vue évènements utilise un page *XAML* et sa classe.
- La page paramètre utilise uniquement une classe.

Ce choix est une voie intéressante à prendre dans le but de tester Xamarin.Forms. De ce fait, nous pouvons nous apercevoir quelle technique est la plus performante dans la définition des éléments graphique. Le résultat global de cette expérience sera expliqué dans le point « 4.7 Synthèse ». Néanmoins, chaque chapitre expliquant les différentes vues de l'application fera le point sur la technique utilisée.

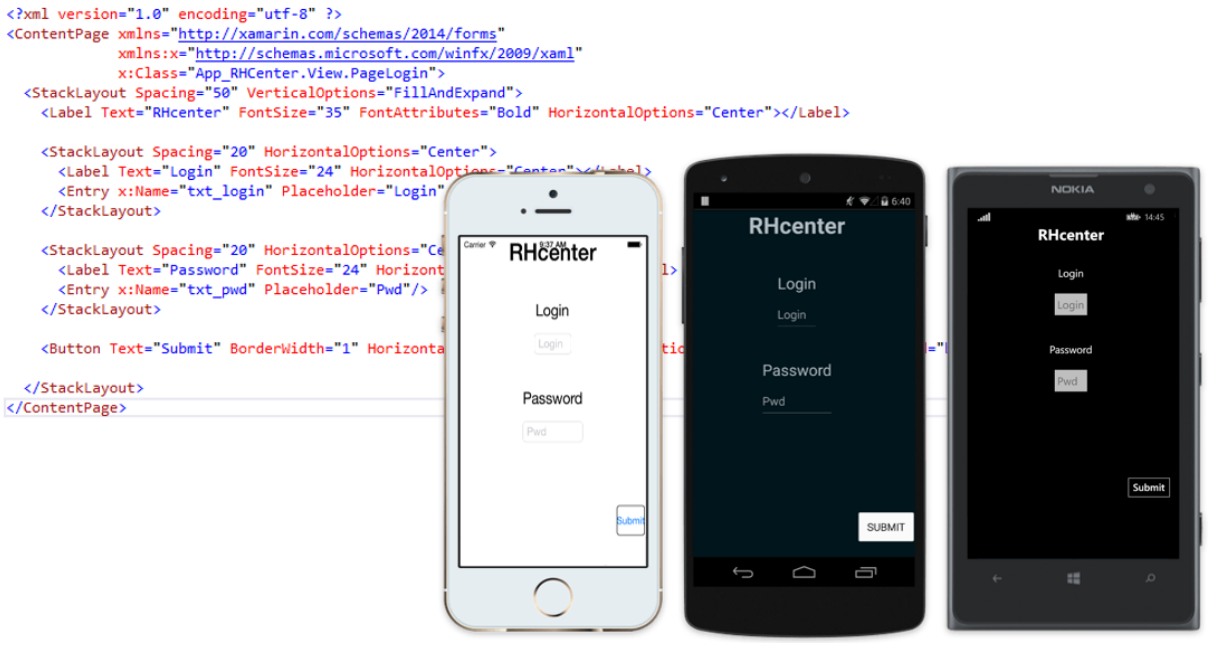
La suite de cette analyse expliquera chaque écran en développement avec l'*API* de Xamarin. Toutes les données inscrites sur les captures d'écrans ou utilisées dans les exemples sont tirées du cas pratique illustré dans le point « 4.2 Scénario » de ce présent travail.

Si dans la suite de ce travail, il est question d'instancier une variable dans un projet de plateforme, cela veut dire que cette instanciation est faite dans la page de démarrage de chaque projet de plateforme (MainActivity pour Android, AppDelegate pour iOS et MainPage pour Windows Phone).

Il est à rappeler que toutes les images qui vont suivre contenant du code en arrière-plan sont présentes pour illustrer le concept de multiplateformes : Ecrire une fois, faire fonctionner n'importe où.

### 4.3.1 Page Login

*Figure 22 : Page Login*



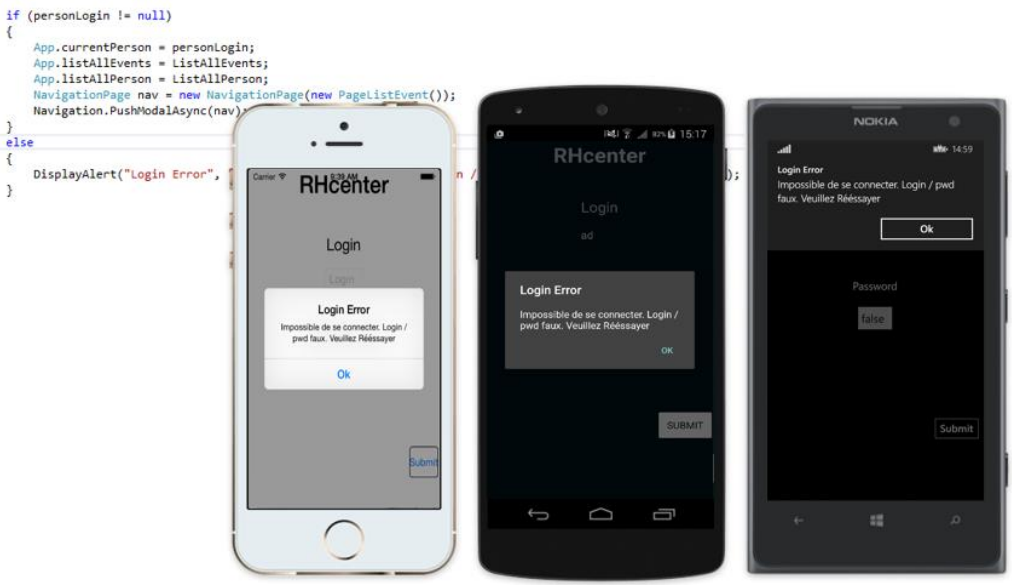

*Source : Données de l'auteur*

Cette page est la première page de l'application. Elle permet à un employé de Valais Excellency de se connecter à sa page personnelle via des informations de login. La Figure 22 : Page Login nous montre cette première page. Deux simples champs texte existent afin d'insérer les informations de connexion.

Le bouton submit permet de valider ces informations. Si les informations sont correctes, l'application nous dirige vers la page suivante. Dans le cas contraire, un message d'erreur apparait. Ce message, illustré dans la Figure 23 : Page Login - Erreur de connexion, annonce à l'utilisateur que ses informations sont erronées. L'application invite l'utilisateur à exécuter une nouvelle tentative de connexion.

Cette première vue permet à notre employé, Aurélie Denis, de se connecter. Dans le champ login, elle inscrira son identifiant (AD). Puis dans le champ Password, elle devra rentrer son mot de passe. Finalement, il faudra qu'elle valide ses informations en cliquant sur le bouton submit.

*Figure 23 : Page Login - Erreur de connexion*




*Source : Données de l'auteur*

**Développement**

Pour cette première page et comme expliqué dans l'introduction du chapitre « 4.3 Développement », il a été décidé d'écrire les éléments graphiques uniquement via une page *XAML*. Etant un dialecte du langage *XML*, le *XAML* possède une structure définie. Grâce à ce dernier, il est très facile de séparer la déclaration des éléments graphiques d'une application du code sous-jacent. Dans le cas présent, toute la partie graphique est scrupuleusement décrite dans le fichier *XAML*.

Dans la classe liée à cette page *XAML*, tous les aspects techniques sont décrits. Par exemple, l'action sur le bouton est écrite dans une méthode. Cette méthode va vérifier les informations de connexion en questionnant la base de données SQLite puis, selon la réponse, soit rediriger l'employé vers la page des évènements, soit afficher un message d'erreur.

La vérification des données se passe de la manière suivante. Le programme va récupérer les informations de connexions inscrites dans les champs textes. Ensuite, une méthode questionnant la base de données SQLite avec une requête est appelée. Grâce à cette méthode, nous vérifions pour une personne donnée si l'identifiant et le mot de passe sont correctes. Si c'est le cas, cette méthode nous retourne l'objet Personne comprenant toutes les informations de cette dernière (Nom, Prénom, Adresse, …) contenues dans la base de données. Cet objet Personne est sauvegardé dans une variable globale accessible par toutes les pages de l'application. Si dans le cas contraire les informations de connexions sont fausses, une valeur nulle est retournée. Grâce à cette valeur nulle, le programme affichera le message d'erreur de connexion.

Développer une page d'une application de cette manière est une excellente idée. Ayant déjà eu l'habitude de travailler avec des pages asp.Net, il fut très facile de faire cette distinction entre l'aspect graphique et l'aspect technique des éléments. De plus, le *XAML* nous aide à avoir une excellente structure et permet d'améliorer la lisibilité du travail.

### 4.3.2 Page Évènements

*Figure 24 : Page de la liste des soirées de travail*


*Source : Données de l'auteur*

Cette page est la page centrale de l'application. Elle permet à l'employé, dans le cas présent Aurélie Denis, de vérifier les dates de travail déjà validées par son supérieur. Ces dates, comme illustrées dans la Figure 24 : Page de la liste des soirées de travail, sont rapidement visibles grâce à une liste. Un bouton juste au-dessus de la liste nous permet de passer à la vue de tous les évènements organisés par Valais Excellency.

Si nous regardons de plus près la liste, nous pouvons voir que chaque ligne comporte plusieurs éléments. Le premier élément en blanc se trouve être le nom de la soirée organisée. Juste en dessous nous pouvons trouver, dans une autre couleur, les dates de commencement et de fin de cet évènement.

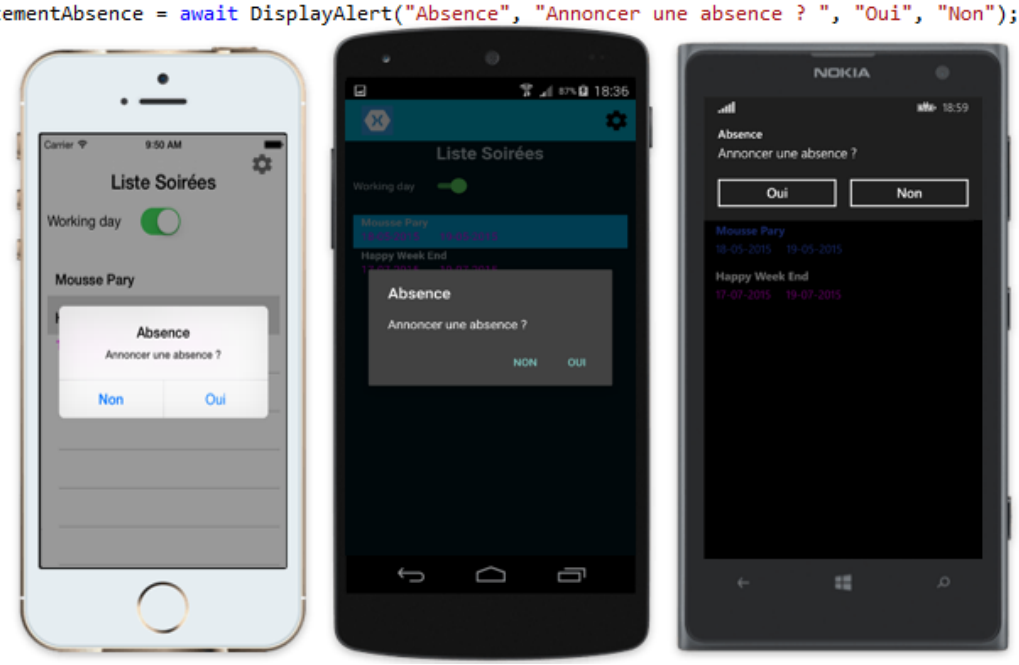
L'utilisateur a la possibilité de cliquer sur ses dates de travail. Lorsque ce dernier le fait, nous trouvons un message proposant d'annoncer une absence au responsable, illustré dans la Figure 25 : Annoncer une absence. Si la réponse est positive, un point rouge se mettra juste à côté de l'évènement sélectionné comme nous pouvons le voir dans la Figure 27 : Visualisation d'une

absence. Néanmoins, l'employé à aussi le choix de répondre par la négative et dans ce cas, rien n'est fait graphiquement. Uniquement la base de donnée est mise à jour pour indiquer que la personne peut travailler à cette soirée.

*Figure 25 : Annoncer une absence*

```
public async void onSelectedItem(object sender, SelectedItemChangedEventArgs e)
{
    Soiree s = (Soiree) e.SelectedItem;

    if (s.isWorkDay)
    {
        s.annoncementAbsence = await DisplayAlert("Absence", "Annoncer une absence ? ", "Oui", "Non");
    }
```




*Source : Données de l'auteur*

Notre collaborateur, Aurélie Denis, peut ainsi voir ses jours de travail et annoncera une indisponibilité pour l'évènement Mousse Party qui a lieu du 18.05.2015 au 19.05.2015.

**Développement**

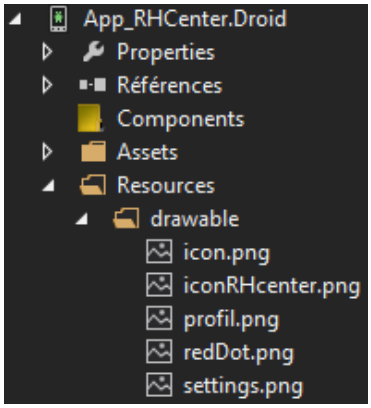
Pour cette page, il a été décidé de mélanger l'utilisation d'une page *XAML* pour le côté graphique mais aussi d'écrire ce dernier dans la classe liée à la page *XAML*. Ce choix fut pris afin de tester les différentes techniques de développement des éléments graphiques.

Dans la page *XAML*, nous pouvons retrouver la définition de la liste ainsi que du bouton à deux positions.

Le bouton position est défini dans un Stack Layout (cf. Figure 29 : Stack Layout) mais en positionnement horizontal. Cela veut dire, qu'au lieu que les éléments soient empilé du haut vers le bas, ils sont mis côte à côte de gauche à droite.

Ensuite, la listView possède de nombreuses spécifications. Une cellule est composée d'un Stack Layout empilant les éléments les uns-sur les autres. Cet élément est composé d'un label pour définir le nom de la soirée mais également d'un autre stack Layout pour définir les dates. Ce dernier présente les dates côte à côte et permet d'afficher ou non un petit point rouge en cas d'absence. Toutes les informations contenues dans la liste sont récupérées de la source de la listView. Dans le cas présent, la source de la liste est l'ensemble des évènements dans la base de données. Cette liste contient uniquement des objets Soirée (un objet Soirée est composé d'un nom de soirée et des dates de commencement et de fin). Pour récupérer automatiquement, par exemple le nom de la soirée, il a été nécessaire d'inscrire le code suivant :

*Figure 26 : Structure de la gestion des images Android*




*Source : Données de l'auteur*

```
<Label Text={Binding nameSoiree} />
```

En fonction de la liste source, le programme va récupérer automatiquement les bons éléments de la soirée. Pour comprendre le fonctionnement, la cellule d'une liste était liée à un objet de la source. Lorsque nous faisons appel au control « Binding » c'est comme si la cellule demande au programme : Donnes moi le nom de la soirée de l'objet avec lequel je suis lié.

L'élément graphique écrit dans la class *XAML*.CS se trouve être la barre de navigation. Nous pouvons le voir dans la Figure 24 : Page de la liste des soirées de travail entouré en rouge. Cet élément permet de naviguer vers une autre page. Ce dernier possède une image dont la localisation change selon les projets de plateformes. C'est pourquoi, toujours en références à la Figure 24, nous définissons l'image pour chaque plateforme via le code suivant :

```
Icon = Device.OnPlatform(« icône source pour iOS », « icône source
    pour android », « icône source pour WinodwsPhone ») ;
```

Ce morceau de code en *C#* nous permet de décrire, en une ligne la source de chaque image pour chaque plateforme. Quand nous définissons une source pour iOS et Windows Phone, l'application va chercher cette dernière dans la racine du projet plateforme.

Pour Android, le fonctionnement diffère. Dans ce cas-là, l'application va rechercher dans le dossier Ressource puis dans le dossier drawable. Nous pouvons voir cette structure dans la Figure 26 : Structure de la gestion des images Android.

Cette définition de source d'image est aussi utilisée pour un élément graphique définit dans la page *XAML*. Il s'agit du petit point rouge qui avertis l'employé afin qu'il sache qu'il a annoncé une absence comme nous le voyons dans la Figure 27 : Visualisation d'une absence de la page suivante. Dans le cas présent, nous pouvons voir que la définition en *XAML* possède la même structure que celle en *C#*. L'élément OnPlateform permet de faire cette distinction graphique

entre chaque plateforme ; comme la taille d'un objet, sa couleur ou bien encore la source de l'image.

Ce choix concernant le mix entre le fichier *XAML* et le fichier *XAML*.CS nous permet de tester les limites de Xamarin. Et dans ce cas, ce mélange fonctionne très bien. Aucune erreur de compilation n'est faite lors du déploiement de l'application.

Il existe néanmoins deux points négatifs pour le développeur.

Premièrement, le langage utilisé n'est pas le même. Si en *XAML*, nous utilisions le code contenu dans la Figure 27 : Visualisation d'une absence pour décrire la source des images dans chaque plateforme, en *C#*, nous écrivons uniquement le code suivant :

```
Icon = Device.OnPlatform(« icône source pour iOS », « icône source pour android », « icône source pour WinodwsPhone ») ;
```

Les deux manières d'écrire sont juste. L'une est plus structuré (*XAML*) et l'autre est plus rapidement écrite (*C#*).

Le deuxième problème se retrouve dans la structure du document. L'un des principaux avantages d'écrire via une page *XAML* est d'utiliser cette séparation entre le code graphique et le côté de la gestion des actions. Dans le cas présent, nous mélangeons cette frontière et l'utilisation d'une page *XAML* perd de son efficacité.

*Figure 27 : Visualisation d'une absence*


*Source : Données de l'auteur*

### 4.3.3 Page Paramètre

Cette page permet à notre employée, Aurélie Denis, de modifier ses informations personnelles telles que son image de profil, son identifiant (Login) et son mot de passe.

La dernière page de l'application, illustrée par la Figure 28 : Page des paramètres se situant à la page suivante, se trouve être celle des paramètres. Dans cette page, l'employé va pouvoir accéder à la galerie de son téléphone, afin de modifier son image de profil. Par défaut, cette dernière se trouve être une silhouette d'un homme.

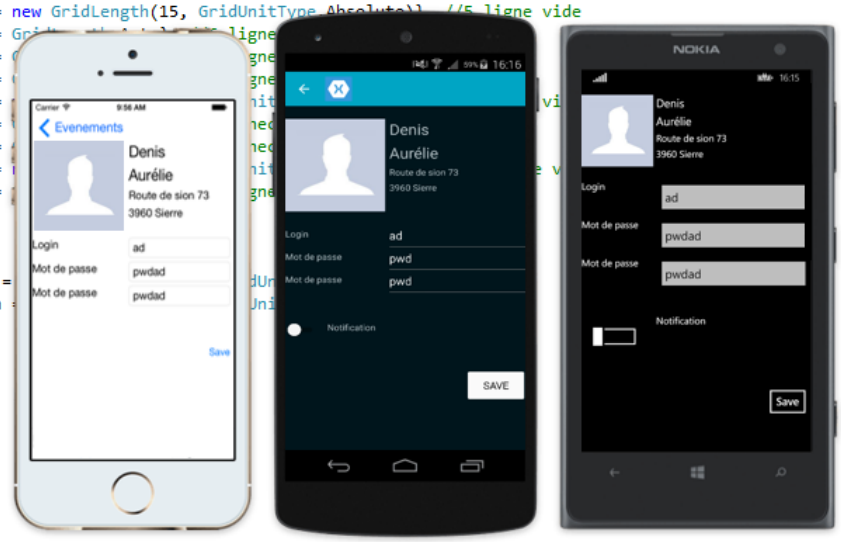
Par la suite, l'utilisateur aura la capacité de changer le login et le mot de passe de son compte. Pour se faire, il devra uniquement cliquer sur le champ texte du login ou du mot de passe. Pour le mot de passe, il est nécessaire d'insérer deux fois le même mot de passe afin qu'aucun message d'erreur n'apparaisse lors de la sauvegarde.

*Figure 28 : Page des paramètres*

```
//Grid layout
Grid grid = new Grid
{
    VerticalOptions = LayoutOptions.FillAndExpand,
    RowDefinitions = {
        new RowDefinition { Height = new GridLength(15, GridUnitType.Absolute)}, //première ligne = espacement
        new RowDefinition { Height = GridLength.Auto}, //1 ligne nom
        new RowDefinition { Height = GridLength.Auto}, //2 ligne prenom
        new RowDefinition { Height = GridLength.Auto}, //3 ligne adresse
        new RowDefinition { Height = GridLength.Auto}, //4 ligne npa
        new RowDefinition { Height = new GridLength(15, GridUnitType.Absolute)}, //5 ligne vide
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =
        new RowDefinition { Height =

    },
    ColumnDefinitions = {
        new ColumnDefinition {Width =
        new ColumnDefinition { Width =

    }
};
```




*Source : Données de l'auteur*

Le bouton save permet de sauvegarder les modifications effectuées. Cet élément est lié à 2 messages distincts qui vont s'afficher :

- **Success** : Ce message annonce que les modifications sont enregistrées dans la base de données.

- **Error** : Ce message indique qu'il y a eu une erreur dans la sauvegarde des informations et que l'utilisateur doit vérifier si les mots de passe sont égaux.

**Développement**

Toute la partie développement graphique de cette page fut écrite dans une classe. Comme nous pouvons le voir dans la Figure 28 : Page des paramètres de la page précédente, toutes les déclarations des éléments graphiques sont écrites en *C#* dans cette classe. De même, les actions sur ces éléments sont écrites dans cette classe.

Dans cette page, nous pouvons accéder à deux fonctionnalités natives.

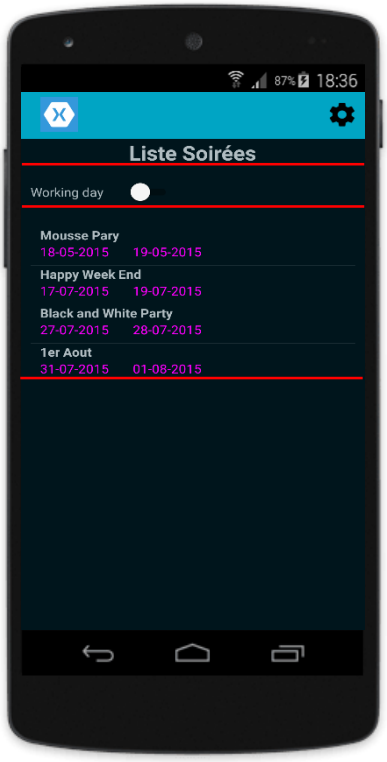
La première se trouve être l'accès à la galerie image du smartphone. Pour accéder à la galerie, le plugin XLabs a été utilisé. Ce dernier fut obtenu dans la bibliothèque *NuGet*. Ce dernier nous permet facilement d'accéder à de nombreux éléments natifs de chaque plateforme (cf. chapitre 4.5.1 XLabs). L'accès à la galerie se fait en cliquant sur l'image de profil de l'employé.

La deuxième fonctionnalité native dont l'application accède se trouve être les alertes toasts. Ces alertes sont représentées sous formes de bannières affichant un message particulier du succès ou d'erreur. Pour accéder à cette fonctionnalité, le plugin Toasts.form a été utilisé (cf. 4.5.3 Toasts.Form).

La structure de l'emplacement des éléments de cette page diffère de celle des deux précédentes pages. En effet, les éléments des pages précédentes sont empilés les uns sur les autre grâce à un Stack Layout comme nous pouvons le voir dans la Figure 29 : Stack Layout. Au contraire, la page paramètre nécessitait un autre style d'affichage des éléments. C'est pourquoi, le choix du gridLayout s'est imposé par lui-même. Comme illustré dans la Figure 30 : Grid Layout de la page suivante, le positionnement des éléments se fait dans une grille. Il est possible de fusionner des cases ensemble, comme pour la photo de profil. Dans un cas de fusionnement de cellule, il est nécessaire de définir chaque position (Gauche, droite, Haut, bas) de l'élément. Par exemple, pour fusionner ensemble les cellules où se trouve l'image nous avons dû définir les endroits de positionnement de l'image comme suit :

- à gauche = 0
- à droite = 1
- en haut = 1

*Figure 29 : Stack Layout*




*Source : Données de l'auteur*

- en bas = 5

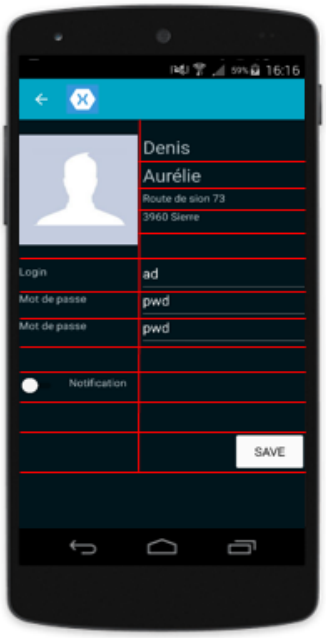
Le programme va ainsi donc comprendre que l'image sera positionnée la colonne 0 à gauche jusqu'à la colonne 1 en partant sur la droite (Colonne non comprise). Puis, il partira de la première ligne pour aller jusqu'à la cinquième ligne afin de fusionner ces cellules.

Toutes les informations inscrites sur cette page sont récupérées de la variable globale définie lors de la phase de connexion.

*Figure 30 : Grid Layout*

```
grid.Children.Add(image, 0, 1, 1, 5);
//left, right , top, bottom

//ligne 6
grid.Children.Add(new Label
{
    Text = "Login",
}, 0, 6);
//left, top
```




*Source : Données de l'auteur*

Finalement le bouton Save permet une vérification et une sauvegarde des informations. Lorsque notre employé va cliquer dessus, une méthode va vérifier plusieurs éléments.

En premier lieu, elle commence à vérifier l'ancien login contenu dans la variable globale avec le champ texte login. Si les deux valeurs sont les mêmes, cela indique que l'utilisateur n'a pas effectué de changement. Dans le cas contraire, l'application garde en mémoire de faire une mise à jour dans la base de données, modifie le login de la variable globale et continue les tests.

La suite des tests vérifie les mots de passe. Tout d'abord, l'application vérifie le premier champ texte de mot de passe avec le deuxième. S'il ne concorde pas, un message toast d'erreur est levé indiquant à l'employé que les deux champs ne concordent pas. Si au contraire, ils sont égaux, la méthode effectue un second test et vérifie qu'ils ne soient pas égaux au mot de passe contenu dans la variable global. Si ce test indique un changement, l'application garde en mémoire de faire une mise à jour et modifie le mot de passe de la variable globale.

Une fois ces tests effectués, l'application effectue, si besoin, une mise à jour vers la base de données. Pour ce faire, elle fait parvenir à la base de données la variable global. Grâce à l'identifiant contenue dans cette dernière, la base de données sait quelle personne modifier.

Les différents choix de développement pour cette page m'ont permis de tester d'autres fonctionnalités de Xamarin.Forms comme :

- L'ajout de plugin
- L'accès aux fonctionnalités natives
- La gestion de l'affichage des éléments graphiques
- L'écriture d'une page complète en *C#*

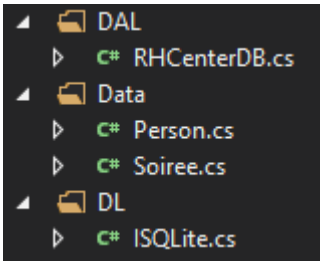
Ce fut la page que j'ai eu le plus de mal à écrire. Tout d'abord, le mélange des éléments graphiques et de leurs actions font de cette page un élément non structuré. Et enfin, le rajout de plugin, spécialement le XLabs, fut laborieux car les explications concernant son application sont toujours en cours d'écriture. Par contre, comprendre le fonctionnement du GridLayout fut très facile grâce aux différents tutoriels disponibles sur le site internet de Xamarin (Xamarin Inc. n, 2015).

## 4.4 Base de données

Pour cette application, il a été décidé d'utiliser une base de données afin de sauvegarder les informations. Au départ, deux style de base de données avaient été choisies ; une en local et une en cloud. Malheureusement de nombreux échecs et problèmes, principalement des erreurs liées aux références, ont fait que ce dernier choix fut évincé de l'application. De plus amples informations concernant ces erreurs sont disponibles dans le chapitre « 5.4.5 Cloud Azure » de ce présent travail.

Pour la base de données en local, j'ai pu utiliser le plugin SQLite-net PCL. Ce plugin permet la gestion d'une base de données en local. La Figure 31 : Structure de la gestion de SQLite nous montre la gestion de ce plugin dans le projet partagé.

*Figure 31 : Structure de la gestion de SQLite*




*Source : Données de l'auteur*

Dans le dossier *DAL*, nous apercevons la classe qui permet de retrouver, d'insérer ou modifier des données dans la base de données.

Dans le dossier Data, nous trouvons les classes modèles de tables contenues dans la base de données. Ces déclarations de classes permettent de définir les tables de la base de données. Par exemple, la classe personne ne contiendra uniquement que des déclarations de variables tels que :

```
public int id { get; set; }
public string Name { get; set; } […]
```

Finalement, l'interface contenue dans le dossier *DL* va être appelée lors du lancement de l'application. Cette interface est instanciée dans les différents projets de plateformes. L'utilisation d'une interface est nécessaire car chaque plateforme demande un accès particulier pour la base de données. Ainsi illustré dans la Figure 32 : Accès base de données selon plateforme nous voyons que chaque projet de plateforme défini cet accès. Selon la plateforme de lancement de l'application, le programme saura vers quelle instance se diriger afin de créer cet accès vers la base de données.

*Figure 32 : Accès base de données selon plateforme*


*Source : Données de l'auteur*

## 4.5 Plugin

Afin de tester le développement multiplateforme proposé par Xamarin, il a été décidé d'utiliser divers plugins. Tous les plugins utilisés ont été installés via l'interface de la gestion des packages *NuGet* proposée par Visual Studio.

### 4.5.1 XLabs

Ce plugin fut choisi car il permettait l'accès à la galerie d'un smartphone en respectant l'idée du multiplateforme. Dans notre cas pratique, il permet à Aurélie Denis de modifier son mot de passe mais aussi d'enregistrer sa demande d'absence.

Xamarin Forms Labs (XLabs) est un projet *open source* qui nous permet de gérer des fonctionnalités natives aux plateformes tout en conservant l'esprit du multiplateforme : Ecrire une fois et le faire fonctionner n'importe où. Dans ces fonctionnalités supportées, nous pouvons par exemple trouver :

- Les boutons images
- L'accès à l'accéléromètre

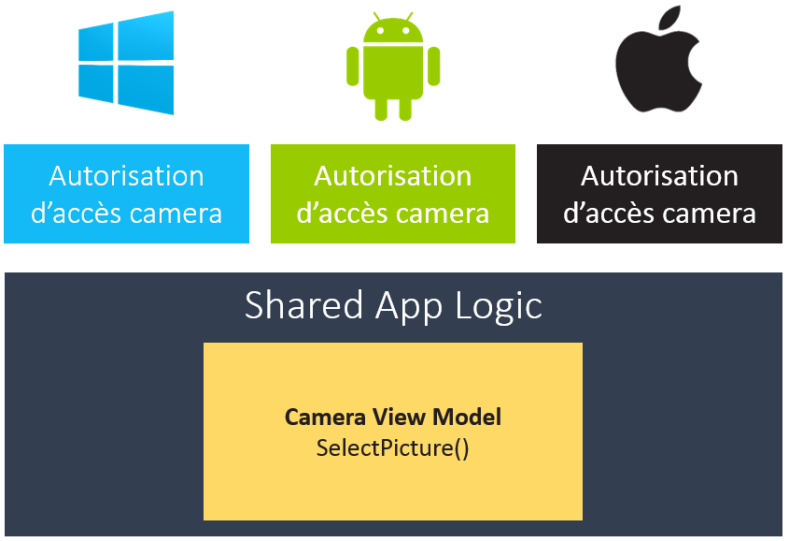
- L'accès à la caméra
- L'accès au GPS
- L'accès à la galerie

Pour la présente application, XLabs nous fut utile pour accéder à la galerie. Pour ce faire, il a été nécessaire d'installer le package XLabs.forms et ses dépendances via la gestion des packages *NuGet* de Visual Studio.

XLabs propose des exemples d'implémentation pour ces différentes fonctionnalités dans leur espace GitHub (XLabs, 2015). Le code source pour l'accès à la caméra de la présente application a été inspiré par ces exemples.

Pour utiliser XLabs dans un projet multiplateforme, il a été nécessaire de créer une classe gérant l'ouverture de la galerie et récupérant une image sélectionnée via ce plugin dans le projet partagé. Ensuite, en utilisant toujours les ressources mises à disposition par ce plugin, chaque projet plateforme devait être modifié afin d'autoriser l'accès à la galerie. Pour ce faire, il était nécessaire de modifier les permissions dans les fichiers de configuration de chaque plateforme. De plus, XLabs demande d'instancier une interface dans chaque plateforme afin que le plugin soit capable de personnaliser chaque accès à la galerie selon la plateforme. La Figure 33 : XLabs Camera nous permet de comprendre la structure d'utilisation de XLabs.

*Figure 33 : XLabs Camera*




*Source : Données de l'auteur*

### 4.5.2 SQLite

Afin de sauvegarder les données de manière locale, un plugin proposé par Xamarin (Xamarin Inc. i, 2015) a été utilisé. Ce dernier, SQLite, propose la gestion d'une base de données en local

sur les smartphones.

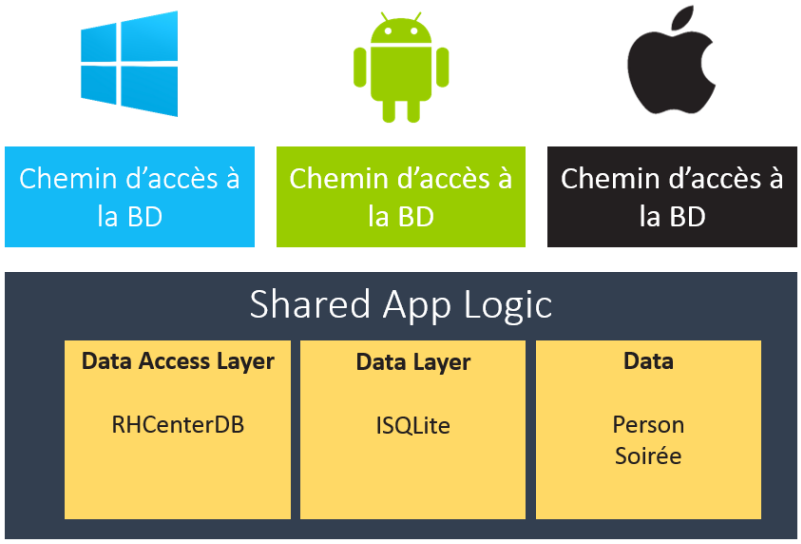
Comme expliqué dans le chapitre « 4.4 Base de données », tous les chemins d'accès vers la base de données sont définis dans les projets plateforme. Autrement, tout le reste de la gestion de la base de données se trouve dans le projet partagé. Cette gestion comprend :

- La gestion des données (*CRUD*)
- Les modèles de données (Personne et soirée)
- L'interface permettant la définition des chemins d'accès

La Figure 34 : Structure utilisation de SQLite nous illustre de manière graphique cette gestion de la base de données SQLite pour un projet multiplateforme

En plus de nous proposer ce choix de plugin, Xamarin nous met à disposition un explicatif nous montrant comment implémenter ce plugin à notre solution. Pour Android et iOS, il est simplement nécessaire de télécharger le package dans la bibliothèque *NuGet*. Malheureusement, pour Windows Phone, il est nécessaire d'installer la version compilée de SQLite sur Windows Phone. Par la suite, nous pouvons rajouter ce plugin. Il est nécessaire de faire la précédente étape, car à l'inverse d'Android ou d'iOS, Windows Phone ne possède pas le moteur de base de données SQLite. C'est pourquoi il faut le télécharger et l'inclure à notre projet au préalable.

*Figure 34 : Structure utilisation de SQLite*




*Source : Données de l'auteur*

### 4.5.3 Toasts.Form

Le dernier plugin utilisé est un plugin pour l'affichage des notifications (couramment appelé dans le milieu du développement mobile des Toasts). L'application utilise ce plugin pour afficher

des communications concernant la sauvegarde des informations personnelles de la page paramètre dans la base de données. Ce genre de notification apparait en haut de l'écran comme une bannière. Selon les cas, cette bannière peut représenter 4 stades d'information :

- Succès
- Warning
- Information
- Erreur

Au cours du développement de cette application, ce genre de notification est utilisé pour la page de paramètre lors de la sauvegarde des données. Cela permet d'informer l'employé si ses données sont sauvegardées ou non.

L'implémentation de ce plugin fut très simple. Il a été nécessaire de télécharger ledit plugin dans la bibliothèque de package *NuGet* et de l'installer dans tous les projets de la solution. Puis, l'instanciation du plugin se fait via le code suivant inscrit dans chaque projet de plateformes :

```
ToastNotificatorImplementation.Init();
```

Cela permet d'instancier selon chaque plateforme le plugin Toast.Form ainsi, les notifications auront les attributs graphiques du système d'exploitation sur lequel l'application a été installée.

## 4.6 Tests

Tout au long du développement de l'application, divers tests ont été accomplis. Afin d'effectuer ces tests, il était nécessaire de sélectionner le bon projet pour lancer la phase de débugge. Pour ce faire, il était indispensable de modifier une option dans les propriétés de la solution. Sous l'onglet projet de démarrage, il faut sélectionner « sélection actuelle ». Ainsi lorsque nous sélectionnons le projet Android, le test sera lancé sur une plateforme Android.

Les tests se déroulaient en deux étapes.

1. Lorsqu'un élément graphique ou une fonctionnalité était écrit, un test était lancé. Chaque premier test de ce style était effectué sur une plateforme Android. Étant propriétaire d'un smartphone Android, un Galaxy S5 possédant la cinquième version d'Android (Lollipop), tous les tests de l'application de cette plateforme ont pu être effectués sur ce dernier. Le deuxième test était un test sur un émulateur Windows phone 8.1 généré par Visual Studio. En général ce test retournait une réponse positive. Néanmoins, dans de rares cas, par exemple l'erreur avec SQLite, des fonctionnalités sur Windows Phone ne fonctionnaient pas (cf. chapitre 5.4.4 Base de données SQL). Étant donné que les tests pour iOS nécessitaient une machine Mac, ces tests étaient effectués une fois sur deux afin de ne pas perdre de temps.

2. Lorsqu'une page était complètement finalisée sur Android, une batterie de tests était lancée à la suite sur Windows Phone puis sur iOS.

Avec cette méthode, j'ai pu très rapidement faire face aux problèmes d'émulations. En de rares occasions, des problèmes liés à la gestion du code entre plateformes ont été soulevés (cf. chapitre «  5.4 Problèmes rencontrés »).

## 4.7 Synthèse

L'application test fut développée pour la startup RHCenter. RHCenter propose une solution pour les entreprises afin de gérer les plannings de leurs employés plus facilement et plus rapidement. Cette application est donc un complément à cette solution.

Pour des raisons de confidentialité, l'application réalisée pour ce projet utilise des données d'une entreprise fictive.

L'application comporte les caractéristiques décrites dans Le Tableau 12 : Caractéristiques de l'application ci-dessous.

*Tableau 12 : Caractéristiques de l'application*

| | |
|---|---|
| **Type d'application multiplateforme** | Xamarin.Forms |
| **Gestion du code partagé** | PCL |
| **Base de données** | SQLite |
| **Accès aux fonctionnalités** | Xlabs et Toasts.Form |
| ***IDE*** | Visual Studio Ultimate Update 5 |
| **Version de l'extension Xamarin sur Visual Studio** | Xamarin 3.11 |
| **Version de Xamarin Studio** | Xamarin 5.9.4 |

*Source : Données de l'auteur*

Cette application comporte 3 pages. Chacune de ces pages teste une façon d'écrire les éléments graphiques. Le Tableau 13 : Écriture des éléments graphiques de la page suivante nous montre ces différentes façons d'écrire.

Au final, la meilleure façon d'écrire des éléments graphiques reste de le faire uniquement sur une page *XAML*. Ce genre de page, dérivé du *XML*, possède la structure adéquate pour définir des éléments graphiques. De plus, elle permet de séparer cette définition graphique de la définition des actions.

*Tableau 13 : Écriture des éléments graphiques*

| | |
|---|---|
| **Page Login** | *XAML* |
| **Page Evènements** | *XAML* +*XAML*.CS |
| **Page Paramètre** | Classe |

*Source : Données de l'auteur*

Tout au long du développement de l'application, une multitude de tests ont dû être effectués. Le Tableau 14 : Plateformes de tests nous montre les caractéristiques de chaque plateforme de tests. Lorsque l'application fut terminée, il a été possible de déployer l'application sur un smartphone possédant *l'OS* Windows 8.1.

*Tableau 14 : Plateformes de tests*

| | |
|---|---|
| **Android** | Samsung Galaxy S5 avec Lollipop |
| **Windows Phone** | Simulateur Windows Phone 8.1<br>Nokia Lumia 930 avec Windows Phone 8.1 |
| **iOS** | Simulateur IPhone 5S avec iOS 8.4 |

*Source : Données de l'auteur*

D'un aspect général, la création de cette application m'a permis de tester quelques fonctionnalités de base de Xamarin. Ces fonctionnalités sont celles que nous trouvons dans chaque application tel que :

- Un champ texte
- Un label
- Une image
- Une liste
- Un bouton à deux positions

# Conclusion

Ce chapitre représente les observations faites sur ce présent travail, d'un point de vue technique ainsi que personnel. La gestion de projet et les améliorations à apporter sont également abordées ici.

## 5.1 Synthèse générale

Sur la première page d'accueil de leur site internet, Xamarin nous dit : « **Xamarin apps look and feel native because they are.** [9] » (Xamarin Inc. l, 2015). Il apparait que les promesses de Xamarin, c'est-à-dire de développer des applications multiplateformes en *C#* et qu'elles soient natives, sont tenues.

En effet, durant ce travail, une application multiplateforme a été entièrement développée via Xamarin pour Visual Studio en *C#*. De cette applications, plus de 95% du code est écrit une fois (partagé) pour les trois plateformes.

Les seules fois où les projets de plateformes ont dû être modifié fut :

- Lors de la création de l'accès base de données
- Les autorisations pour l'accès à la galerie
- Les autorisations pour les notifications

Ce genre de modifications n'intervient uniquement lorsque nous voulons accéder à des fonctionnalités spécifiques aux plateformes. Autrement, tout le code est contenu dans le projet partagé.

Développer uniquement en *C#* des applications Xamarin est possible. Néanmoins, il est apparu que travailler des éléments graphiques avec une page *XAML* est plus facile. L'inconvénient majeur de développer uniquement en *C#* est qu'il y a une mauvaise distinction entre la définition des éléments graphiques et la définition de leurs. C'est pourquoi il est préférable de séparer, via une page *XAML*, cette définition graphique des éléments et leurs actions grâce la à classe *XAML*.cs.

Les simulations de base proposées pour Xamarin.Studio sur PC ne contiennent que de l'émulation d'appareil Android. Pour avoir la possibilité de simuler l'application crée sur Windows Phone nous devons nous équiper du programme Visual Studio avec la *SDK* de Windows Phone 8.1. De plus, la simulation d'iOS fonctionne uniquement sur machine Mac.

[9] Les applications Xamarin ressemble à des applications natives, car elles le sont.

Donc, il est nécessaire de posséder un PC et une machine Mac pour simuler l'application sur les trois plateformes.

Les tutoriels et le forum proposé par Xamarin sont très fournis et aident les novices à apprendre à utiliser Xamarin. Une large communauté de développeurs poste régulièrement des astuces de développement et des plugins sur le forum de Xamarin. De plus, l'entreprise Xamarin, en partenariat avec Microsoft, met à disposition légalement et gratuitement le livre de Charles Petzold : Creating Mobile Apps with Xamarin.Forms Preview Edition 2. Grâce à ce livre, la compréhension de Xamarin.Forms devient plus claire.

En bref, les points forts de Xamarins sont :

- ✓ La possibilité de créer une application multiplateforme native
- ✓ ~90 % du code est partagé
- ✓ De nombreux tutoriels et une excellente communauté sur le forum
- ✓ Liaisons avec l'*IDE* Visual Studio
- ✓ Possibilité de créer toute l'application en *C#*
- ✓ Aucun besoin de connaître les langages de développement propres de chaque plateforme

Néanmoins, quelques points négatifs doivent être relevés :

- ✗ Nécessite un Mac et un Visual Studio pour la simulation de toutes les plateformes
- ✗ Pour une meilleure structure du code, nécessite d'avoir des bases en *XML* pour travailler avec une page *XAML*
- ✗ Nécessite d'avoir au minimum 10Go de place sur l'ordinateur pour l'installation complète de Xamarin.Studio (Sans compter Visual Studio)
- ✗ De nombreux *Blue Screen* sur le PC depuis l'installation de Xamarin.

Au final, Xamarin répond à la problématique exprimée au chapitre « 1.5 Problématique » de ce travail. Nous pouvons bel et bien développer une application multiplateforme en écrivant uniquement en *C#*. De plus, cette application sera native à la plateforme sur laquelle elle sera déployée. Il n'y a eu en aucun cas besoin de connaître les langages de développement propres à chaque plateforme.

### 5.1.1 Evolutions futures

Le futur de Xamarin annonce un accès dans le cloud Service. En effet depuis 9 juillet 2015, Oracle et Xamarin ont entamé un partenariat (Leemputten, 2015). Les outils analytiques de la plateforme Oracle vont s'intégrer à l'univers de développement de Xamarin. Par exemple, le nouveau système de test cloud de Xamarin se verra doté, par exemple, d'analyse de performance.

De plus, pour la prochaine version de l'iOS 9, Xamarin propose déjà, sur son espace développeur, divers introductions à la compatibilité de ce système d'exploitation (Xamarin Inc. o, 2015). Cela nous annonce donc le support iOS 9 par Xamarin.

## 5.2 Retour d'expérience

Personnellement, l'utilisation de Xamarin fut une magnifique expérience. Malgré de nombreux *Blue Screen* dès l'installation de Xamarin sur mon PC, j'ai pu apprécier de travailler avec un tel logiciel. Avant de faire du développement multiplateforme avec Xamarin, j'avais déjà testée le développement sur Android. Néanmoins, cette expérience d'écrire des applications smartphone en *C#* se révèle intéressante, car ce fut une autre vision de développement sur smartphone qui m'est apparu.

Pouvoir travailler avec l'*IDE* Visual Studio est une excellente opportunité. Connaissant la structure interne de cet *IDE*, il me fut très facile de gérer une solution multiplateforme Xamarin. Ce fut un gain de temps, car grâce à cela, l'apprentissage de l'utilisation de Xamarin Studio fut considérablement réduit.

Concernant les *Blue Screen*, il m'est arrivé 3 fois dans une semaine de reformater complètement tout mon PC à cause d'un *Blue Screen* qui me bloquait toute la partition contenant Windows. Après des tests, il m'est apparu que sur mon PC, des mises à jour Windows 8.1 rentraient en conflit avec Xamarin et faisaient dysfonctionner le PC. La solution trouvée pour me permettre de continuer à tester Xamarin sur le même PC fut d'empêcher les mises à jour automatiques de Windows Updates. Ce problème n'est pas cité dans les problèmes rencontrés au chapitre « 5.4 Problèmes rencontrés », car ce fut un problème touchant uniquement mon PC. D'autres développeurs de ma connaissance utilisant Xamarin sur PC tournant sous Windows 8.1 n'ont eu aucun problème de ce genre.

## 5.3 Recommandations

Xamarin est un logiciel puissant conçu pour des entreprises voulant créer des applications fonctionnant sur les trois systèmes d'exploitation phare du marché actuel (Android, iOS, Windows Phone).

Une étude pour l'Ile de France mené par la société Urban Linker, nous montre le salaire moyen pour des développeurs d'application en 2014 dans le Tableau 15 : Compratif des salaires annuels des développeurs mobiles 2014.

*Tableau 15 : Compratif des salaires annuels des développeurs mobiles 2014*

| | **Applications mobiles hybrides** | **Applications mobiles natives** |
|---|---|---|
| **Débutant 0 à 1 an** | 32-36 K€ | 34-40 K€ |
| **Intermédiaire 1 à 2 ans** | 36-42 K€ | 38-44 K€ |
| **Confirmé 2 à 4 ans** | 42-46 K€ | 43-50 K€ |
| **Sénior 4 à 6 ans** | 46-50 K€ | 50-60 K€ |
| **Expert / Architecte 6 ans et +** | — | 60-80 K€ |
| **Chef de Projet 4 à 8 ans** | — | 55-65 K€ |

*Source : Adapté de* (Urban Linker, 2014)

Partons du principe qu'un développeur ne sache qu'un seul langage de programmation mobile. Si une entreprise souhaite publier son application native sur les trois *OS* mobiles, il est nécessaire d'engager 3 développeurs. Dans le meilleur des cas, en recrutant 3 développeurs débutants, l'entreprise devra allouer 102'000 € [10] au service du développement mobile par année.

En utilisant Xamarin, cette entreprise pourrait engager une seule personne. Le montant alloué au service développement mobile par année (comprenant le salaire du développeur débutant + la licence annuelle business) redeviendrait à la somme de 34'909.98 €[11], soit, 67'000 € d'économie. Ainsi, elle économisera des ressources pour d'autre département.

Je recommande aux lecteurs voulant tester Xamarin d'avoir un excellant PC permettant la virtualisation. De plus, si ces derniers veulent simuler un iOS, il est impératif de posséder un Mac. L'utilisation de Visual Studio est également nécessaire. En effet, la simulation d'appareil Windows Phone n'est disponible que sur ce dernier. De plus, si l'emploi de cet *IDE* ne vous est pas étranger, il sera plus simple pour vous de créer l'application.

La licence de Xamarin est convertie en euros. Le taux de conversion a été relevé sur le site internet www.xe.ch le 24 Juillet 2015 à 18h04 UTC. Cette licence se montant à $ 999 correspond à 909.98 €.

[10] 3 développeurs x 34'000 € salaire/an
[11] 1 développeur débutant (34'000 €) + 909.98 € de licence

- Taux de conversion Dollar : $1 = 0.9109 (XE, 2015)

## 5.4 Problèmes rencontrés

Tout au long du développement de l'application, divers problèmes sont apparus. La majorité des problèmes ont été résolus.

### 5.4.1 Simulation iOS

Malgré le fait que Xamarin propose un développement multiplateforme, l'émulation pour une application iOS demeure problématique. Pour ce faire, il est nécessaire de posséder une machine Mac.

**Problème 1 – connexion avec une machine Mac**

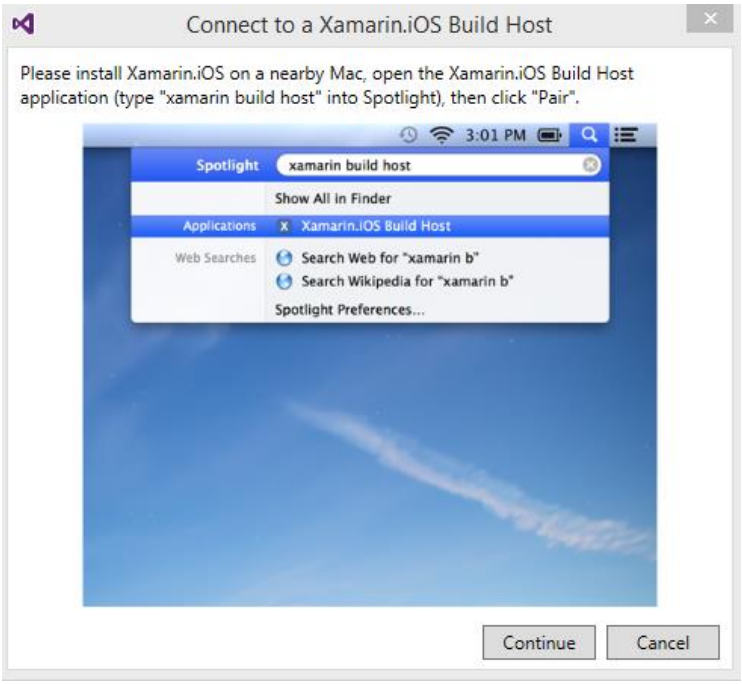
En temps normal, le développement d'une application iPhone se fait en *Objective*-C et dans certains cas avec le nouveau langage de développement d'Apple (Le *Swift*). Pour pouvoirs compiler ce code, nous avons besoin de l'*IDE* d'Apple qu'est X-Code. C'est pour cela que la programmation d'application d'iOS passe obligatoirement par une machine Mac (Juliegri, 2012).

Il existe une solution. Dans Visual Studio, l'extension de Xamarin nous propose de lier un PC vers une machine Mac comme illustré dans la Figure 35 : Connexion à Xamarin.iOS Build Host.

Pour cela, il nous faut :

- une connexion internet qui autorise les ping
- une machine Mac ayant son firewall désactivé
- une machine Mac avec l'*OS* X 10.10
- Visual Studio avec l'extension Xamarin
- une machine Mac avec Xamarin.iOS (la même version que Xamarin.iOS sur Visual Studio)

*Figure 35 : Connexion à Xamarin.iOS Build Host*




*Source : Données de l'auteur*

**Problème 2 – clé de signature**

Le second problème découle de la solution du premier problème. Lorsque que l'application a dû être testée pour la première fois sur la machine Mac l'erreur suivante m'est apparue :

*No valid iOS code signing key found in keychain.*

Cette erreur nous annonce que nous n'avons pas l'autorisation (Code signing) pour installer notre application sur un appareil Apple (Apple Inc. c, 2015).

La première solution serait de nous faire certifier développeur Apple afin de posséder un certificat délivré par Apple pour notre application. Cette certification coute 109 CHF par an. Ainsi, nous pourrions lancer notre application sur un appareil Apple

La seconde solution est de supprimer un mot dans le fichier csproj du projet iOS de notre solution Xamarin.Forms comme illustré dans la Figure 36 : Clé de signature de la page suivante. Ce mot (Entitlements.plist) est la signature du certificat sur notre application. En le supprimant, nous supprimons cette liaison et permettons à notre machine Mac d'émuler notre application sur un iPhone/iPad.

*Figure 36 : Clé de signature*

BEFORE

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|iPhoneSimulator' ">
    <CodesignEntitlements>Entitlements.plist</CodesignEntitlements>
</PropertyGroup>
```

AFTER

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|iPhoneSimulator' ">
    <CodesignEntitlements></CodesignEntitlements>
</PropertyGroup>
```

*Source : (Yardy, 2015)*

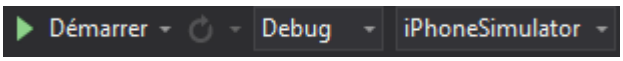
Dans le cadre de ce travail, la deuxième option fut choisie.

**Problème 3 – Simulateur**

Une fois les deux problèmes cités ci-dessous dépassés, il m'a fallu régler le problème de la simulation en elle-même.

Étant donné que le projet a été écrit sur Visual Studio, la simulation de l'iOS se fait sur une machine Mac. Il faut néanmoins faire attention à un point lors de l'exécution de cette dernière. Pour régler ce problème, il faut bien vérifier que les options de débogage illustré sur la Figure 37 : Simulateur iOS sont sélectionnées au moment de lancer cette émulation.

*Figure 37 : Simulateur iOS*




*Source : Données de l'auteur*

Dans le cas présent, nous forçons Visual Studio à lancer une instance sur le Mac qui est lié au projet. Cette instance invoque le simulateur d'iOS. Néanmoins, pour pouvoir utiliser ce simulateur, il faut posséder la dernière version de xCode sur la machine Mac.

### 5.4.2 Simulation Windows Phone

Les différents tests avec Windows Phone ont été exécutés sur un simulateur Windows Phone 8.1. Pour ce faire il faut avoir le Visual Studio possédant le *SDK* de Windows Phone 8.1. Cette option peut être disponible lors de l'installation de Visual Studio ou bien, elle peut être téléchargée en ligne sur le site développeur de Microsoft (Microsoft b, 2015). Néanmoins, deux problèmes liés uniquement à l'émulation de l'application sur Windows Phone 8.1 ont été

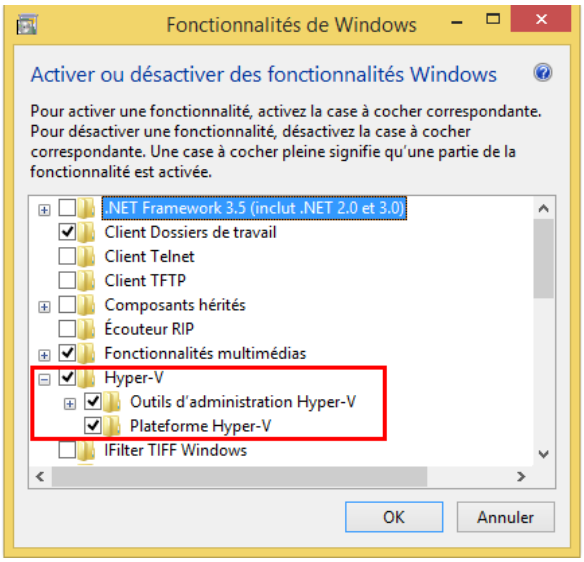
remarqués et corrigés.

**Problème 1 – Hyper V**

Afin de pouvoir lancer un simulateur de smartphone possédant le système d'exploitation Windows Phone 8.1, il faut posséder un ordinateur permettant la virtualisation. Cette virtualisation est une option connue sous le nom d'Hyper V.

Afin d'activer cette fonctionnalité de Windows, il est nécessaire d'aller dans les options de paramètre nommées : Activer ou désactiver des fonctionnalités Windows. La Figure 38 : Activer Hyper-V nous illustre la fenêtre qui sera ouverte. Si votre ordinateur peut gérer l'Hyper-V, vous y trouverez une option Hyper-V nécessitant d'être cochée.

*Figure 38 : Activer Hyper-V*




*Source : Données de l'auteur*

Une fois cette étape enregistrée, un redémarrage de l'ordinateur sera demandé. Désormais, il vous sera possible de lancer la simulation de l'application sur Windows Phone.

**Problème 2 – Accès photo de la galerie du simulateur**

L'application développée pour tester Xamarin demande un accès et l'utilisation des photos de la galerie d'images disponible sur le smartphone.

Étant donné que les tests ont été effectués sur simulateur, l'utilisation des photos était impossible, car le simulateur ne possédait aucune photo.

Afin de régler ce problème, il est nécessaire :

1. De lancer une première fois l'application. Cela va ouvrir le simulateur Windows Phone.

2. De stopper, via Visual Studio, l'application mais de laisser ouvert le simulateur.
3. Dans le simulateur, accéder à l'appareil photo et faire des photos.

En faisant ces étapes, le simulateur enregistrera des photos dans la galerie. Par la suite, il suffit de relancer l'application avec le même simulateur et, ainsi, il sera possible d'accéder et d'utiliser les photos de la galerie d'image.

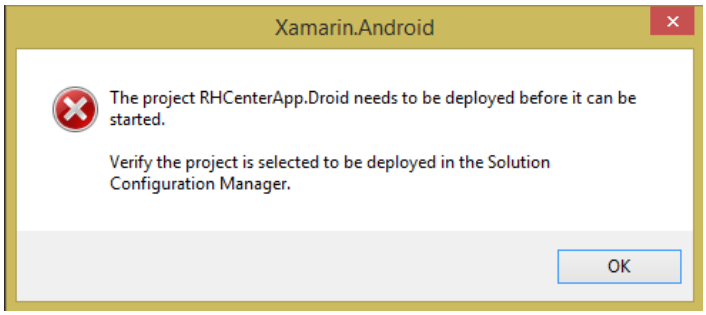
### 5.4.3 Tests sur Android

Pour ce travail, tous les tests concernant le projet Android ont été effectués sur un Samsung Galaxy S5 possédant la version 5 d'Android ; Lollipop.

**Problème 1 – déploiement application Android**

Lors de tests Android sur Visual studio, il se peut que vous tombiez sur une erreur expliquant que le projet Android doit être déployé avant d'être lancé tel illustré dans la Figure 39 : Erreur lancement application Android.

*Figure 39 : Erreur lancement application Android*




*Source : Données de l'auteur*

Cette erreur signifie que Visual Studio n'a pas pu lancer l'application Android sur l'appareil, car tout simplement cette dernière n'existe pas sur le smartphone.

La solution à ce problème se trouve dans les propriétés de la solution globale. Sous l'onglet propriété de configuration, la case Déployer doit être sélectionnée pour le projet Droid (Android). De ce fait, il nous sera possible de lancer la phase de tests sur Android.

### 5.4.4 Base de données SQL

Un problème rencontré fut l'implémentation de la base de données. Le premier test sur Android fut un grand succès, le second sur iOS aussi. Malheureusement, lors du test sur Windows Phone, l'application s'éteignait.

Après de nombreuses recherches, il s'est avéré que Windows Phone ne possède pas de moteur pour la base de données SQLite et de l'installer.

Afin de résoudre ce problème, il est nécessaire de télécharger la version compilée du programme SQLite (Xamarin Inc. m, 2015).

### 5.4.5 Shared App vs. Portable App

Étant l'unique développeuse de cette application, j'ai suivi les conseils de Xamarin et ai choisi d'utiliser un projet en Shared Code.

De fil en aiguille, j'ai remarqué que le code devenait ingérable. En effet, en écrivant une classe dans le projet partagé, il fallait très régulièrement utiliser les directives compréhensibles par le *compilateur* afin de décrire les actions possibles sur les différentes plateformes. Plus le projet avançait, plus la structure et les limites de ce dernier devenaient floues.

C'est pour ces raisons que j'ai choisi de m'orienter vers un projet de type portable. Ainsi, les actions ou directives propres à chaque plateforme étaient décrites dans les projets de ces dernières. La solution complète avait une meilleure structure et en résultait une meilleure compréhension et navigation.

### 5.4.5 Cloud Azure

Au commencement du développement de l'application, il avait été décidé d'utiliser une synchronisation entre une base de données en Local et une base de données en Cloud.

Malheureusement, l'implémentation proposée sur le site Azure ne permettait que le support d'un projet Android ou iOS. Il n'était nullement question de projet multiplateforme. Malgré cette première barrière, l'implémentation fut testée sur l'application. Bien que toutes les conditions requises pour que cela fonctionne étaient remplies, cette dernière refusait l'accès au projet multiplateforme du cloud Azure.

De plus, l'ajout du plugin d'Azure Cloud Mobile depuis Xamarin Component rendait les liens de mes précédentes références corrompus. Par exemple, l'accès à la galerie via le plugin XLabs ne fonctionnait plus une fois Azure Cloud Mobile installé.

Ce fut donc un choix technique que de ne pas tenir compte de cette synchronisation entre une base de données locale et une base de données Cloud.

## 5.5 Respect du cahier des charges

Toutes les fonctionnalités minimales du cahier des charges en Annexe I ont été implémentées. Néanmoins, certaines fonctionnalités optionnelles n'ont pas pu être réalisées :

- L'accès aux SMS / emails : Il est apparu que pour la fonctionnalité de l'application, cette option n'est pas nécessaire. En effet, l'employé doit être capable uniquement de voir les évènements de travail. La communication en cas d'absence se fait par un

signalement via la base de données lors de la sélection de la soirée en question.

- La connexion aux réseaux sociaux : Étant une application à titre privé il n'est pas nécessaire d'accéder à des réseaux sociaux. Néanmoins, il serait intéressant, dans une amélioration future, de permettre à un employé de partager un évènement sur les réseaux sociaux.

- Accès à une base de données en Cloud : L'accès à une base de données en cloud (azure) ne fut pas possible avec l'application RHCenter. De nombreux conflits liés à des références du plugin XLabs ont rendu impossible cette liaison (cf. chapitre « 5.4.5 Cloud Azure »).

Une fonctionnalité supplémentaire fut tout de même implémentée. Au départ, il était question d'enregistrer les données dans un fichier. Mais au court du développement, la sauvegarde des données dans une base de données SQLite fut rajoutée.

## 5.6 Gestion de projet

Tout au cours de la réalisation de ce projet, l'approche de gestion de projet dite « Agile » a été utilisé avec des itérations de deux semaines.

Cette approche ne se focalise pas sur les détails. Elle nous permet d'avancer sur le projet et à un moment donné de s'arrêter pour réévaluer sa direction (Lothon, 2013). Dans le cas présent, ces moments de réévaluation stratégique se trouvaient être la fin de chaque itération (Sprint). Ce projet fut constitué de 5 itérations :

L'itération de commencement (numéro 0) fut la plus longue (du 28.04.2015 au 31.05.2015). Elle m'a permis de poser le champ de recherche de ce sujet d'étude. De plus, ce fut le temps nécessaire pour que ma demande d'accès étudiants à Xamarin fut étudiée et acceptée. Ces accès possèdent les mêmes caractéristiques que la licence Business (Xamarin Inc. j, 2015).

La première itération s'est orientée sur le positionnement de Xamarin face à la concurrence. C'est grâce à ce sprint que j'ai pu comprendre les différences cruciales entre Xamarin, PhoneGap ou bien même Titanium Appcelerator. De plus, les recherches effectuées sur Xamarin durant ce moment m'ont permis de comprendre son fonctionnement et m'ont aidé à implémenter rapidement l'application.

C'est justement durant la deuxième itération que plus de 90% de l'application fut écrite. Durant ce sprint, j'ai pu mettre en pratique la méthodologie Agile. Quand un problème surgissait, je m'octroyais maximum 3 heures de temps pour le gérer tout de suite. Si je n'arrivais pas à le résoudre, je passais à l'implémentation d'une autre fonctionnalité. C'est uniquement lorsque mes objectifs journaliers étaient terminés que je me permettais de résoudre les erreurs. Ainsi, l'écriture de l'application était fluide et rapide. Le plus gros problème rencontré fut cette

gestion de SQLite avec Windows Phone (cf. chapitre « 5.4.4 Base de données SQL ».

Le troisième et le denier sprint furent un mélange entre de la recherche pour la rédaction du rapport et la finalisation de l'application.

Le descriptif complet de ces itérations est disponible à l'Annexe IV .

## 5.8 Améliorations envisageables

De nombreuses améliorations pourraient être apportées à cette application.

Premièrement, cette application a été développée dans le but de tester des fonctionnalités. Elle ne possède donc aucun charme graphique. C'est un point qui sera envisageable d'améliorer dans le futur. Avec Xamarin.Forms, il est possible d'implémenter des styles en *C#* aux éléments graphiques. Il serait intéressant de tester cette option de customizing.

Deuxièmement, une liaison entre la base de données locale et une base de données en cloud sera intéressante à avoir avec cette application. Il existe de nombreuses bases de données en cloud. Néanmoins, le choix de départ était de lier la base de données locale, SQLite, avec une base de données sur le Cloud d'Azure. L'avantage d'avoir une base de données en cloud est que nous pouvons intégrer un système de notification. Dès qu'une donnée est mise à jour sur la base de données en cloud, l'application smartphone reçoit une notification pour avertir l'utilisateur d'une nouveauté.

Dernièrement, dans le cahier des charges en Annexe I , nous pouvons lire dans les fonctionnalités supplémentaires la connexion vers des réseaux sociaux. Il serait intéressant au collaborateur de partager une date d'un évènement afin d'inviter des gens à venir à ce dernier.

Si ce travail était à refaire, j'envisagerai d'implémenter la méthodologie Scrum plutôt que celle d'Agile. Ainsi, une meilleure structure du temps de travail serait mise en place et il y aurait une possibilité d'amélioration de ce dernier. Dès le sprint 0, j'essaierai de commencer avec la version d'essai gratuit de Xamarin. Comme cette dernière possède les mêmes caractéristiques que la version Business, cela m'aurait permis de plus rapidement commencer la phase de test du logiciel.

# 6 Webographie

Adobe Systems Inc. a. (2015, Juillet 22). *PhoneGap | Home*. Récupéré sur phonegap: http://phonegap.com/

Adobe Systems Inc. b. (2015, Juillet 22). *PhoneGap | Supported Features*. Récupéré sur phonegap: http://phonegap.com/about/feature/

Adobe Systems Inc. c. (2015, Juillet 22). *Adobe PhoneGap Build*. Récupéré sur build.phonegap: https://build.phonegap.com/

Android. (2015). *Brand Guidelines*. Récupéré sur Developer Android: http://developer.android.com/distribute/tools/promote/brand.html

Appcelerator Inc. a. (2015, Juillet 22). *Appcelerator Platform - Appcelerator Docs*. Récupéré sur Appcelerator: http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_Platform_Overview

Appcelerator Inc. b. (2015, Juillet 22). *Appcelerator Platform Plans and Pricing*. Récupéré sur Appcelerator: http://www.appcelerator.com/pricing/

Appcelerator Inc. c. (2015, Juillet 22). *Mobile App Development Platform – Appcelerator*. Récupéré sur Appcelerator: http://www.appcelerator.com/

Apple Inc. a. (2015, Janvier 08). *Apple (France) - Informations presse - L'App Store aborde 2015 avec de nouveaux records.* Récupéré sur Site Web Apple: http://www.apple.com/fr/pr/library/2015/01/08App-Store-Rings-in-2015-with-New-Records.html

Apple Inc. b. (2015, Juillet 19). *iOS 9 Preview*. Récupéré sur Apple: https://www.apple.com/ios/ios9-preview/

Apple Inc. c. (2015, Juin 26). *Code Signing*. Récupéré sur developer.apple: https://developer.apple.com/support/code-signing/

Apple Inc. d. (2014). *Apple - Swift*. Récupéré sur Apple: https://www.apple.com/chfr/swift/

Ariel. (2015, Janvier 13). *App Stores Growth Accelerates in 2014*. Récupéré sur appfigures: http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/

BenjaminL. (2014, Octobre 16). *PhoneGap, création d'applis mobiles multi-plateforme*. Récupéré sur alsacréations: http://www.alsacreations.com/tuto/lire/1646-introduction-phonegap.html

Birch, J. (2014, Juin 17). *Unity Vs. Marmalade Vs. V-Play Vs. Corona Vs. Cocos2D: Five Cross-platform Game Engines Compared.* Récupéré sur Renatus: http://renatus.com/unity-marmalade-v-play-corona-cocos2d-cross-platform-game-engines

Brynte. (2014, Mars 31). *Windows Phone 8.1 for Developers-Converging the App Models*. Récupéré sur blogs.msdn:

http://blogs.msdn.com/b/thunbrynt/archive/2014/03/31/windows-phone-8-1-for-developers-converging-the-app-models.aspx

Cathala, C. (2014, Avril 04). *Les nouveautés des applications windows phone Silverlight 8.1*. Récupéré sur Soat blog: http://blog.soat.fr/2014/04/les-nouveautes-des-applications-windows-phone-silverlight-8-1/

ComputerWeekly. (2002, Mai). *Write once, run anywhere ?* Récupéré sur ComputerWeekly: http://www.computerweekly.com/feature/Write-once-run-anywhere

Cowlabstudio. (2014, Juin 30). *Come scaricare, installare ed utilzzare PhoneGap 3.5.* Récupéré sur cowlabstudio: http://www.cowlabstudio.com/phonegapitalia/come-scaricare-installare-ed-utilizzare-phonegap-3-5-guida-base/

Cygnet Infotech. (2015, Janvier 09). *PhoneGap or Titanium or Xamarin - Which Cross-Platform Framework Should You Choose?* Récupéré sur Cygnet Infotech: http://www.cygnet-infotech.com/blog/phonegap-or-titanium-or-xamarin-which-cross-platform-should-you-choose

Distimo. (2013, Septembre). *The Mobile App Ecosystem, Global trends in established and growth markets.* Récupéré sur Distimo: http://www.distimo.com/publications

Doudoux, J.-M. (2005, Juillet 05). *Le refactoring*. Récupéré sur jmdoudoux: http://www.jmdoudoux.fr/java/dejae/chap009.htm

Eason, J. (2015, Juillet 09). *M Developer Preview Gets Its First Update*. Récupéré sur Android Developers Blog: http://android-developers.blogspot.com.tr/2015/07/m-developer-preview-gets-its-first.html

Elgin, B. (2005, Août 17). *Google Buys Android for Its Mobile Arsenal.* Récupéré sur businessweek.com: http://archive.is/rfYj

eMarketer Inc. (2004, Janvier 16). *Smartphone Users Worldwide Will Total 175 Billion*. Récupéré sur eMarketer: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

Espiau, F. (2015, Juin 30). *L'architecture d'Android*. Récupéré sur OpenClassrooms: https://openclassrooms.com/courses/creez-des-applications-pour-android/l-architecture-d-android

Feugey, D. (2011, Octobre 06). *MAX 2011 : Adobe rachète Nitobi et offre PhoneGap à la fondation Apache*. Récupéré sur silicon: http://www.silicon.fr/max-2011-adobe-rachete-nitobi-et-offre-phonegap-a-la-fondation-apache-62355.html

FusePowered. (2015, Juin 24). *About.* Récupéré sur fusepowered: https://www.fusepowered.com/about/

Hendrickson, M. (2008, Décembre 09). *Appcelerator raises à 4.1 Million for Open Source RIA Platform*. Récupéré sur techcrunch: http://techcrunch.com/2008/12/09/appcelerator-raises-41-million-for-open-source-ria-platform/

Hezemans, A. (2014, Janvier). *Asia: The Leading App Market in the World.* Récupéré sur Distimo: http://www.distimo.com/publications

Hollister, S. (2010, Septembre 26). *Microsoft prepping Windows Phone 7 for an October 21st launch? (update: US on Nov. 8?)*. Récupéré sur engadget: http://www.engadget.com/2010/09/26/microsoft-prepping-windows-phone-7-for-an-october-21st-launch/

Icaza, M. d. (2003, Octobre 13). *[Mono-list] Mono early history.* Récupéré sur Ximian.com: http://lists.ximian.com/pipermail/mono-list/2003-October/016345.html

IDC. (2015, Février 24). *Android and iOS Squeeze the Competition, Swelling to 96.3% of the Smartphone Operating System Market for Both 4Q14 and CY14, According to IDC*. Récupéré sur IDC Analyse the Future: https://www.idc.com/getdoc.jsp?containerId=prUS25450615

Idealake Information Technologies PVT LTD a. (2015, Juin 16). *McDelivery India - west & south on the App Store on iTunes [Application]*. Récupéré sur itunes: https://itunes.apple.com/us/app/mcdelivery-india-west-south/id920750178?mt=8

Idealake Information Technologies PVT LTD b. (2015, Juillet 20). *McDelivery India – North & East – Applications Android sur Google Play [Application]*. Récupéré sur Google Play: https://play.google.com/store/apps/details?id=com.mcdeliveryonline.app

Idealake Information Technologies PVT LTD c. (2015, Juillet 20). *McDelivery - Microsoft Store [Application]*. Récupéré sur Windows Store: https://www.microsoft.com/en-US/store/apps/McDelivery/9WZDNCRDC2R5

Jacobs, B. (2012, Décembre 13). *Exploring the iOS SDK*. Récupéré sur code.tutsplus: http://code.tutsplus.com/tutorials/exploring-the-ios-sdk--mobile-13959

Jayamaha, D. (2014, Septembre 15). *Native Mobile Apps with Titanium Mobile.* Récupéré sur slideshare: http://fr.slideshare.net/DharshanaJayamaha/titanium-mobile-39109750

Juliegri. (2012, Novembre 27). *Initiation au développement sur iPhone*. Récupéré sur alsacréations: http://www.alsacreations.com/article/lire/744-initiation-au-dveloppement-sur-iphone.html

Lamoureux, F. (2012, Juillet 7). *Présentation d'Appcelerator Titanium : développez en JavScript des applications natives iOS et Android.* Récupéré sur florentlamoureux: http://www.florentlamoureux.fr/blog/presentation-dappcelerator-titanium-developpez-en-javascript-des-applications-natives-ios-et-android/

Lawson, S. (2009, Mars 17). *Android Market Needs More Filters, T-Mobile Says.* Récupéré sur pcWorld.com: http://www.pcworld.com/article/161410/article.html

Leemputten, P. V. (2015, Juillet 09). *Oracle et Xamarin entament une collaboration* . Récupéré sur datanews: http://datanews.levif.be/ict/actualite/oracle-et-xamarin-entament-une-collaboration/article-normal-404795.html

Lothon, F. (2013, Juin). *Introduction aux méthodes agiles et Scrum*. Récupéré sur L'Agiliste:

http://www.agiliste.fr/fiches/introduction-methodes-agiles/

Martin, M. (2012, Juillet 10). *Créez des applications pour iPone, iPad et Ipod Touch.* Récupéré sur openclassrooms.com: http://openclassrooms.com/courses/creez-des-applications-pour-iphone-ipad-et-ipod-touch

Mercy, J.-S. (2011, Juin 06). *Principes de la programmation pour iOS*. Récupéré sur Osaxis: http://www.osaxis.fr/blog/principes-de-programmation-pour-ios/

Merli, N. (2015, Juin 10). *iOS 9 : date de sortie, beta et nouvelles fonctionnalités de la denière version du système d'exploitation mobile d'Apple*. Récupéré sur Gentside: http://www.gentside.com/ios-9/ios-9-date-de-sortie-beta-et-nouvelles-fonctionnalites-de-la-deniere-version-du-systeme-d-039-exploitation-mobile-d-039-apple_art69666.html

Microsoft a. (2015). *What's a Universal Windows App ?* Récupéré sur Centre de développement Windows: https://msdn.microsoft.com/library/windows/apps/dn726767.aspx

Microsoft b. (2015). *Windows Phone SDK archives*. Récupéré sur dev.windows: https://dev.windows.com/en-us/develop/download-phone-sdk

Microsoft c. (2013, Novembre). *L'histoire de Windows*. Récupéré sur Windows Microsoft: http://windows.microsoft.com/fr-CH/windows/history#T1=era0

Microsoft d. (2015). *Fonctionnalités Windows 10 - Microsoft*. Récupéré sur Microsoft: http://www.microsoft.com/fr-ch/windows/features

Microsoft e. (2015). *Find Nonprofit Resources: Toolkit - Get Files*. Récupéré sur Microsoft: http://www.microsoft.com/about/corporatecitizenship/en-us/community-tools/nonprofittoolkit/get-files/

Mono Project. (2015, Juillet 05). *AOT*. Récupéré sur mono-project: http://www.mono-project.com/docs/advanced/aot/

Nguyen, L. (2015, Février 26). *Apple & Google dominent le marché mondial du smartphone*. Récupéré sur übergizmo: http://fr.ubergizmo.com/2015/02/26/apple-google-dominent-le-marche-mondial-du-smartphone.html

Nickel, B. (2014, Février 10). *Xamarin Ahead-of-Time (AOT compiler vs. an ordinary compiler*. Récupéré sur stackoverflow: http://stackoverflow.com/questions/21689993/xamarin-ahead-of-time-aot-compiler-vs-an-ordinary-compiler

Nielsen Institut. (2014, Janvier 07). *Smartphones: So Many Apps, So Much Time.* Récupéré sur Nielsen: http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html

Offermann, S. (2011, Octobre 03). *Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap.* Récupéré sur adobe: http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html

PayPal a . (2015, Juillet 20). *PayPal pour iPhone, iPod touch et iPad dans l'App Store sur iTunes [Application]*. Récupéré sur itunes: https://itunes.apple.com/fr/app/paypal/id283646709?mt=8

PayPal b. (2015, Juillet 20). *PayPal – Applications Android sur Google Play [Application]*. Récupéré sur Google Play: https://play.google.com/store/apps/details?id=com.paypal.android.p2pmobile

Paypal c . (2015, Juillet 20). *PayPal - Microsoft Store [Application]*. Récupéré sur Microsoft Store: https://www.microsoft.com/en-US/store/apps/PayPal/9WZDNCRFJ1VK

Pierrat, O. (2012, Février 28). *Le développement multi-plateforme : enjeux, promesses et réalité*. Récupéré sur Journal du net: http://www.journaldunet.com/developpeur/expert/51048/le-developpement-multi-plateforme---enjeux--promesses-et-realite.shtml

Rouse, M. a. (2005, Septembre). *bytecode*. Récupéré sur WhatIS: http://whatis.techtarget.com/definition/bytecode

Rouse, M. b. (2015, Juin 26). *Common Language Infrastructure (CLI)*. Récupéré sur SearchSOA: http://searchsoa.techtarget.com/definition/Common-Language-Infrastructure

Rovio Entertainment Ltd. (2015, Juillet 20). *Bad Piggies – Applications Android sur Google Play [Application]*. Récupéré sur Google Play: https://play.google.com/store/apps/details?id=com.rovio.BadPiggies

Samad, K. (2012, Février 11). *What are the advantages and disadvantages of using AndEngine to create commercial games in Android?* Récupéré sur Quora: http://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-AndEngine-to-create-commercial-games-in-Android

Seydtaghia, A. (2015, Avril 29). *La Suisse demeure une exception sur la carte d'Apple.* Récupéré sur LeTemps.ch: http://www.letemps.ch/Page/Uuid/c4b45268-edb6-11e4-8a43-4ad205b10b56/La_Suisse_demeure_une_exception_sur_la_carte_dApple

Strakh, A. (2015, Juillet 05). *10 Reasons for Choosing Xamarin Cross Plateform Mobile Development*. Récupéré sur EastBanc Technologies: http://www.eastbanctech.com/10-reasons-for-choosing-xamarin-cross-platform-mobile-development/

Unity. (2015, Juin 24). *Get unity.* Récupéré sur unity3d: https://unity3d.com/get-unity

Unity Technologies a. (2015, Juillet 22). *Unity - Multiplatform*. Récupéré sur unity3d: https://unity3d.com/unity/multiplatform

Unity Technologies b. (2015, Juillet 2015). *Unity - Unity - Editor*. Récupéré sur unity3d: https://unity3d.com/unity/editor

Unity Technologies c. (2015, Juillet 22). *Unity - Get Unity*. Récupéré sur unit3d: https://unity3d.com/get-unity

Urban Linker. (2014). *Salaire moyen 2014 : Je suis un développeur*. Récupéré sur Urban Linker :

http://www.urbanlinker.com/salaire-moyen/salaire-technique-2014/

W12. (2015). *Photos apple logo png* . Récupéré sur W12: http://www.w12.fr/apple-logo-png.html

Whinnery, K. a. (2012, Mai 12). *Comparing Titanium and PhoneGap*. Récupéré sur Appcelerator Blog: http://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/

Whinnery, K. b. (2010, Décembre 01). *Titanium Guides Project: JS Environment*. Récupéré sur appcelerator: http://www.appcelerator.com/blog/2010/12/titanium-guides-project-js-environment/

Wilhem, A. (2014, Février 27). *Now Serving 4M Downloads Daily, The Windows Store Has Grown 135% Since October*. Récupéré sur techCrunch.com: http://techcrunch.com/2014/02/27/now-serving-4m-downloads-daily-the-windows-store-has-grown-135-since-october/

Wilson, J. (2014, Mars 12). *5 reasons to use Xmaarin for cross-platform development*. Récupéré sur pluralsight: http://blog.pluralsight.com/5-reasons-xamarin

Wordament. (2013, Juin 19). *Wordament 2.5 released for Windows Phone and iOS | Wordament*. Récupéré sur Wordament: http://www.wordament.com/wordament-2-5-released-for-windows-phone-and-ios/

Xamarin Inc. a. (2015, Juillet 22). *Introduction to Mobile Development*. Récupéré sur developer.xamarin: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/

Xamarin Inc. b. (2014, Décembre 16). *Mobile Enterprise Success with Xamarin and IBM.* Récupéré sur slideshare: http://fr.slideshare.net/Xamarin

Xamarin Inc. c. (2015, Juillet 22). *Sharing Code Option*. Récupéré sur Developer Xamarin: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/

Xamarin Inc. d. (2015, Juillet 22). *Installation - Xamarin*. Récupéré sur Developper Xamarin: http://developer.xamarin.com/guides/ios/getting_started/installation/Mac/

Xamarin Inc. e. (2015, Juillet 22). *Introduction to Portable Class Librairies*. Récupéré sur Developer Xamarin: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

Xamarin Inc. f. (2015). *Build native UIs for iOS, Android and Winodws Form a single, shared C# codebase*. Récupéré sur Xamarin: http://xamarin.com/forms

Xamarin Inc. g. (2015, Juillet 22). *Mobile App Development App Creation Software - Xamarin*. Récupéré sur xamarin: http://xamarin.com/

Xamarin Inc. h. (2015, Mai 19). *Signature Pad*. Récupéré sur coponents.xamarin:

https://components.xamarin.com/view/signature-pad

Xamarin Inc. i. (2015, Juillet 19). *Working with a Local Database*. Récupéré sur developer Xamarin: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/working-with/databases/

Xamarin Inc. j. (2015, Juillet 22). *Store - Xamarin*. Récupéré sur store.xamarin: https://store.xamarin.com/

Xamarin Inc. k. (2015, Juillet 22). *Introduction to Portable Class Libraries*. Récupéré sur developer xamarin: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

Xamarin Inc. l. (2015, Juillet 22). *Mobile Application Development to Build Apps in C# - Xamarin*. Récupéré sur Xamarin: https://xamarin.com/platform

Xamarin Inc. m. (2015). *Adding SQLite database support to Windows Phone*. Récupéré sur Developer Xamarin: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/working-with/databases/#Adding_SQLite_to_WinPhone

Xamarin Inc. n. (2015). *Xamarin.Forms Layouts*. Récupéré sur Developer Xamarin: http://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/layouts/

Xamarin Inc. o. (2015, Juillet 22). *iOS 9 Compatibility*. Récupéré sur Developer Xamarin: developer.xamarin.com/guides/ios/platform_features/ios9/

Xamarin Inc. p. (2014, Janvier 23). *Xamarin Case Study Wordament.* Récupéré sur Resources Xamarin: http://resources.xamarin.com/rs/xamarin/images/Xamarin-Case-Study-Wordament.pdf

XLabs. (2015). *XLabs/Xamarin-Forms-Labs.* Récupéré sur github: https://github.com/XLabs/Xamarin-Forms-Labs

Yardy, D. (2015, Mai 06). *iPhoneSimulator build results in "No valid iOS code signing keys found in keychain."*. Récupéré sur forum.xamarin: https://forums.xamarin.com/discussion/39674/iphonesimulator-build-results-in-no-valid-ios-code-signing-keys-found-in-keychain

ZDNet. (2015, Juin 08). *Chiffres clés : les ventes de mobiles et de smartphones*. Récupéré sur zdnet: http://www.zdnet.fr/actualites/chiffres-cles-les-ventes-de-mobiles-et-de-smartphones-39789928.htm

# 7 Bibliographie

Lee, W.-M. (2012). *Beginning Android 4 Application Development.* Crosspoint Boulevard: John Wiley & Sons, Inc.

Isaacson, W. (2011). *Steve Jobs.* (Lattes, Éd., Dominique, D. Defert, & C. Delport, Trads.) Malesherbes, Etats-Unis: Lattes. Consulté le Juillet 19, 2015

Petzold, C. (2015). *Creating Mobile Apps with Xamarin.Forms* (éd. 2). Redmond, Washington: Microsoft Press. Consulté le Juillet 14, 2015

# Annexes

## Annexe I  Cahier des charges de l'application RHCenter

Caractéristique de l'application test pour ce présent travail. Réalisé par l'auteur.

**Description de l'application**

L'application doit pouvoir permettre à un employé de voir ses jours de travail et ainsi d'annoncer une absence. Elle doit être codée avec Xamarin et être disponible sur les 3 principales plateformes des smartphones actuels (Android, Windows Phone, iOS)

**Travail à effectuer**

- Minimal
  - Fonctionnalités classiques d'une application
  - Gestion du stockage local avec des fichiers
  - Système de notification
  - Accès au calendrier, la caméra, la galerie image, contact  -> Enregistrement des jours de travail en sync avec la DB, modification de la photo de profil, envoyer un message a un contact (messagerie interne de RH-Center)
- Add-on
  - Accès au SMS/mail
  - Connexion réseaux sociaux
  - Intégration de données provenant d'un cloud externe
  - Accès avec le cloud spécifique aux plates-formes (possibles, mais difficile)

# Annexe II Product Backlog

Présentation des user-stories de l'application RHCenter. Réalisé par l'auteur.

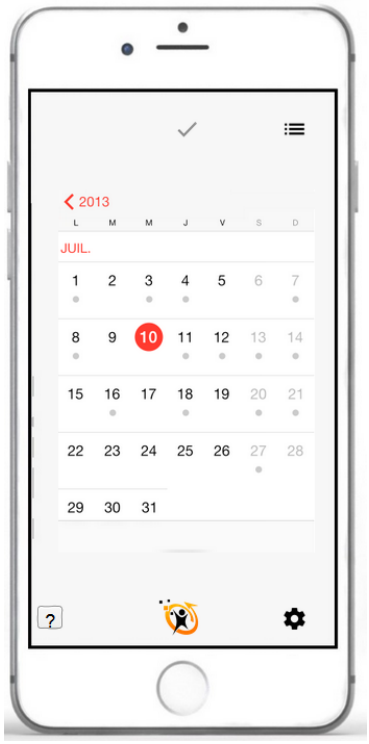
| US Nr. | Theme | User Stories | | | Priority | Status | Sprint |
|---|---|---|---|---|---|---|---|
| | | As an/a ... | I want to ... | so that ... | | | |
| 1 | Planning | Ressource humain | pouvoir afficher le planning du mois en cours | je puisse gérer mon temps de travail | 1 | | 2 |
| 2 | Planning | Ressource humain | pouvoir afficher les planning à venir | je puisse planifier mon temps de travail | 2 | | 2 |
| 3 | Planning | Ressource humain | pouvoir annoncer une indisponibilité | je puisse avertir mon supérieur d'un empêchement de dernière minute | 3 | | 2 |
| 4 | Planning | Ressource humain | pouvoir annoncer une disponibilité | je puisse planifier mon temps de travail | 3 | | 2 |
| 5 | Planning | Ressource humain | pourvoir Afficher / modifier mon profil | je puisse mettre à jour mon profil | 5 | | 2 |
| 6 | Planning | Ressource humain | Pouvoir me logger | je puisse avoir une meilleure sécurité de mes données | 7 | | 1 |
| 7 | Planning | Ressource humain | avoir les envents dans lequel je suis dispo dans mon calendrier interne du téléphone | je puisse avoir une meilleure vue de mon temps de travail | 4 | | 4 |
| 8 | Planning | Ressource humain | avoir des alertes en cas de MàJ/insert des planning | je puisse avoir les informations en temps réel | 6 | | 3 |
| 9 | Planning | Ressource humain | savoir si les changements de mes informations ont été enregistrées | je puiss être sûr de la sauvegarder | 8 | | 3 |

# Annexe III Canevas de l'application RH-Center

Mockup de l'application Test. Réalisé par l'auteur.


Page de Login

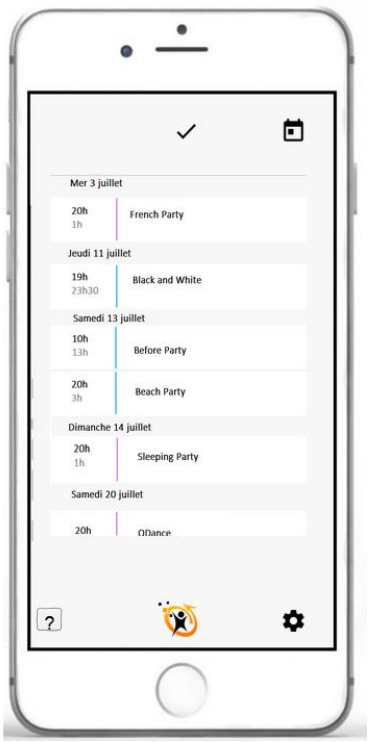

Calendrier des évènements de



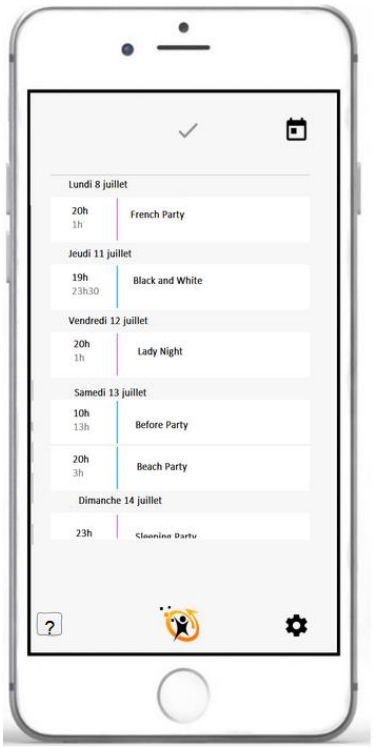

Calendrier de tous les évènements


Avertissement d'une absence


Visualisation d'une absence




Liste des évènements de




Liste de tous les évènements


Avertissement d'une absence


Visualisation d'une absence


Page des paramètres

# Annexe IV Sprint

Explication du temps de travail de ce présent rapport et de l'application test. Réalisé par l'auteur.

Les heures de travail des descriptions « rédaction du rapport » comprenne également divers recherches.

**Sprint 0 — 48h**

Du 28.04.2015 au 31.05.2015

| Date de fin | Description | Heures de travail |
|---|---|---|
| 28.04.2015 | Recherche sur le sujet des applications multiplateformes | 5 h |
| 29.04.2015 | Etablissement du cahier des charges | 5 h |
| 03.05.2015 | Rédaction de la table des matières | 3 h |
| 06.05.2015 | Recherche sur Xamarin | 3 h |
| 08.05.2015 | Discussion avec RHCenter concernant l'application | 2 h |
| 12.05.2015 | Mise à jour du cahier des charges | 2 h |
| 20.05.2015 | Création des uses cases | 3 h |
| 21.05.2015 | Rédaction du rapport | 14 h |
| 27.05.2015 | Création du canevas de l'application | 5 h |
| 28.05.2015 | Installation de Xamarin et premier pas avec | 4 h |
| 31.05.2015 | Recherche de l'implémentation de Xamarin sur iOS | 3 h |

**Sprint 1 — 39h**

Du 01.06.2015 au 14.06.2015

| Date de fin | Description | Heures de travail |
|---|---|---|
| 01.06.2015 | Finalisation des canevas et uses case | 9 h |
| 09.06.2015 | Rédaction du rapport | 15 h |
| 10.06.2015 | Recherche logiciel développement multiplateformes | 5 h |
| 11.06.2015 | Choix du type de projet Xamarin (Xamarin.Forms + shared code) | 5 h |
| 14.06.2015 | Création de l'interface login | 5 h |

**Sprint 2 — 73h**

Du 16.06.2015 au 28.06.2015

| Date de fin | Description | Heures de travail |
|---|---|---|
| 16.06.2015 | Création de l'interface Calendrier | 8 h |
| 17.06.2015 | Mise à jour de l'interface Calendrier avec une Liste | 5 h |
| 18.06.2015 | Mise en place de la base de données SQLite | 5 h |
| 20.06.2015 | Test de la base de données avec Android et iOS | 5 h |
| 21.06.2015 | Mise à jour de l'interface Calendrier | 5 h |
| 24.06.2015 | Test de la base de données sur Windows Phone + correction de l'erreur | 9 h |
| 25.06.2015 | Création de la page Paramètre | 9 h |
| 26.06.2015 | Mise en place des accès aux fonctionnalités sans plugin | 9 h |
| 27.06.2015 | Mise en place des accès aux fonctionnalités avec Xlabs | 9 h |
| 28.06.2015 | Tests et correctifs des accès aux fonctionnalités sur les plateformes | 9 h |

**Sprint 3 — 95h**

Du 29.06.2015 au 10.07.2015

| Date de fin | Description | Heures de travail |
|---|---|---|
| 29.06.2015 | Mise à jour de l'analyse de Xamarin VS concurrence | 10 h |
| 30.06.2015 | Essai de mise en place de la synchronisation Azure = échec | 10 h |
| 01.07.2015 | Mise en place du système de notification avec le plugin Toast.Form | 10 h |
| 07.07.2015 | Finalisation de l'application. Tests sur les trois plateformes | 10 h |
| 10.07.2015 | Rédaction du rapport | 55 h |

**Sprint 4 — 123h**

Du 13.07.2015 au 26.07.2015

| Date de fin | Description | Heures de travail |
|---|---|---|
| **13.07.2015** | Essaie de l'accès vers le calendrier de chaque plateforme = échec | 12 h |
| **20.07.2015** | Rédaction du rapport | 61 h |
| **22.07.2015** | Correctif et mise à jour de la partie rédaction | 20 h |
| **23.07.2015** | Mise en page des annexes | 13 h |
| **24.07.2015** | Préparation de l'application à rendre (Nettoyage) | 10 h |
| **25.07.2015** | Préparation des documents à rendre | 5 h |
| **26.07.2015** | Impression + reliure + rendu du rapport | 2 h |

**Total global**

| Numéro de Sprint | Heures de travail Total |
|---|---|
| Sprint 0 | 48 h |
| Sprint 1 | 39 h |
| Sprint 2 | 73 h |
| Sprint 3 | 95 h |
| Sprint 4 | 123 h |
| **TOTAL** | **378 h** |

# Annexe V Contenu de la clé USB

La présente clé USB en annexe de ce dossier contient les éléments suivants :

- Le rapport en format PDF
- Le projet Visual Studio de l'application
- Le projet Web interactif du canevas de l'application

Afin de tester le canevas, il est nécessaire de tenir compte des recommandations suivantes :

- Aucune information de login n'est nécessaire
- Seul le 11 juillet peut être annoncé comme absence

# Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seule, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du Responsable de Filière et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après : M. Jean-Pierre Rey.

Sierre, le 26 juillet 2015

Audrey Dupont