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.

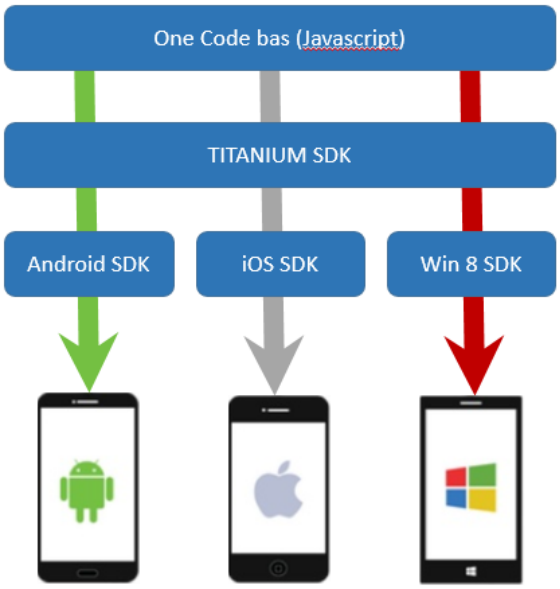
Un logiciel proposant ce genre de service disposera d'une série d'outils graphiques qui s'adaptera sur chaque plateforme. Proposer ce genre d'outils permet d'optimiser les performances de l'application sur chaque plateforme. Néanmoins cela demande une immense charge de travail et limite donc le panel 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

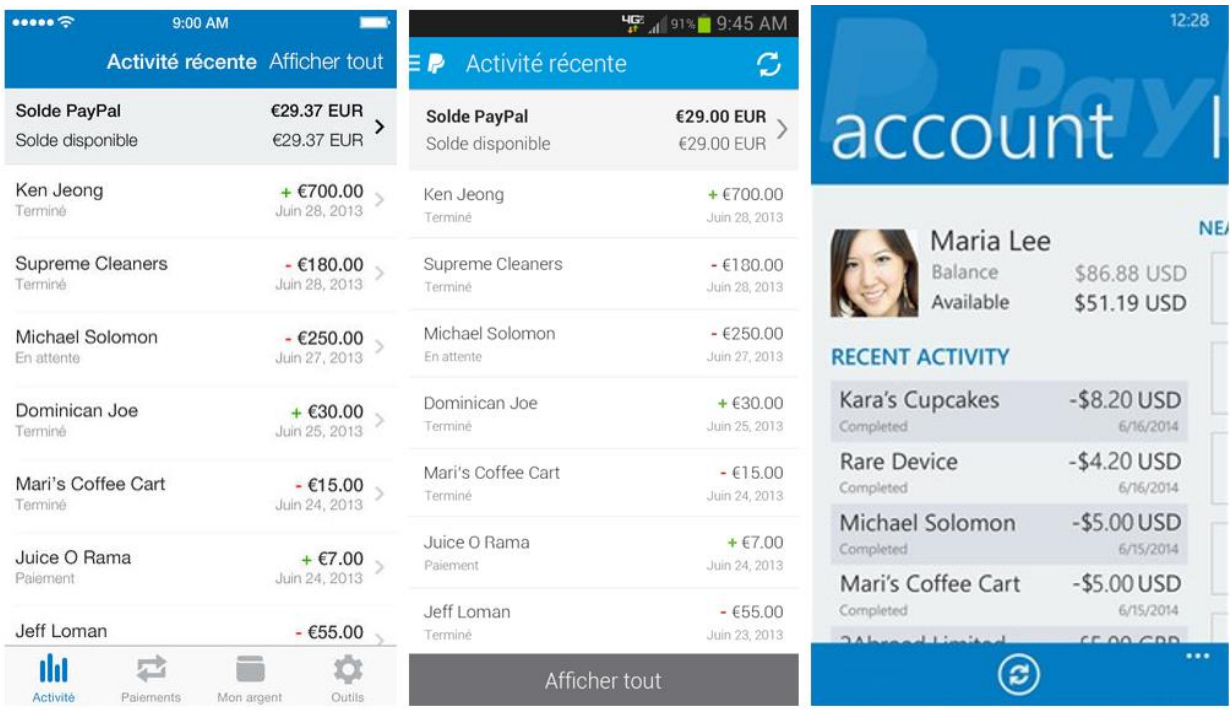


Source : (Jayamaha, 2014)

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.

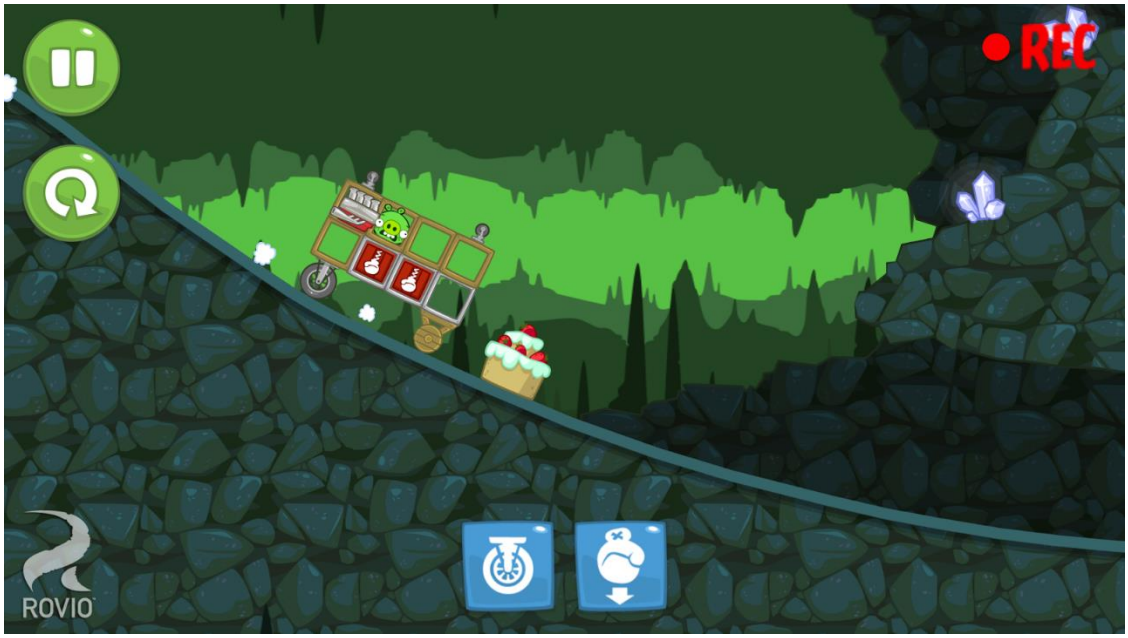
Développer des applications en utilisant Unity3D demande des scripts qui soient écrits soit en *C#*, en UnityScript ou en *BOO* (Birch, 2014). De ce fait, le moteur Unity3D sera apte à les

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.

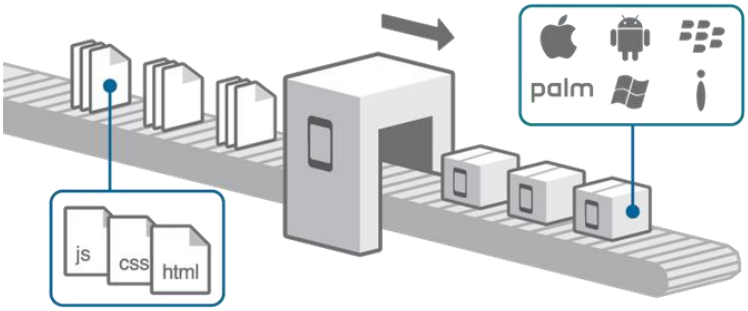
2.3.1 PhoneGap

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.*

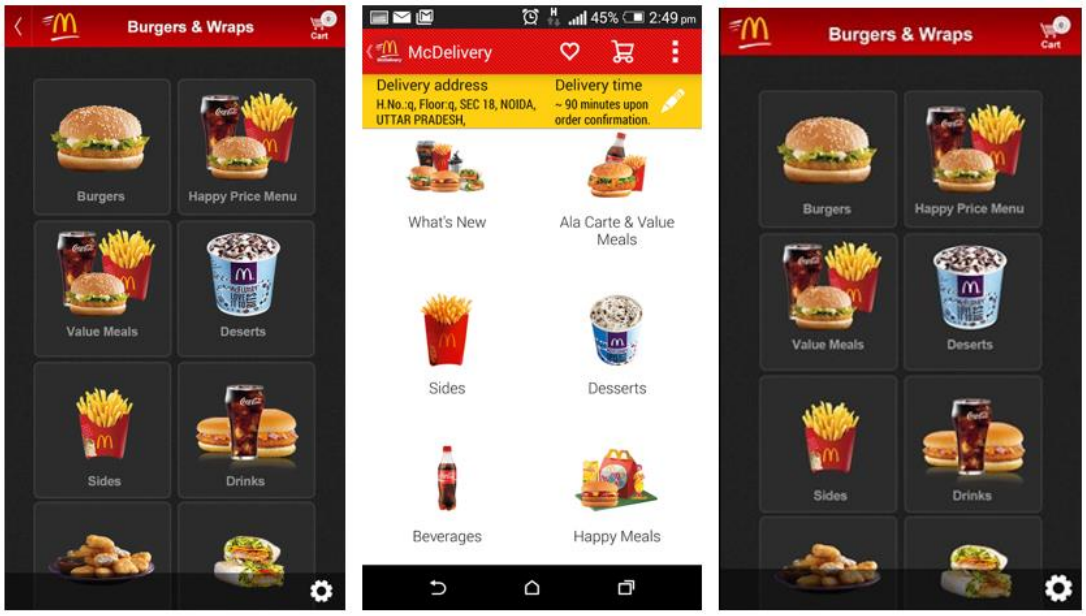
Figure 11 : illustration de PhoneGap



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]

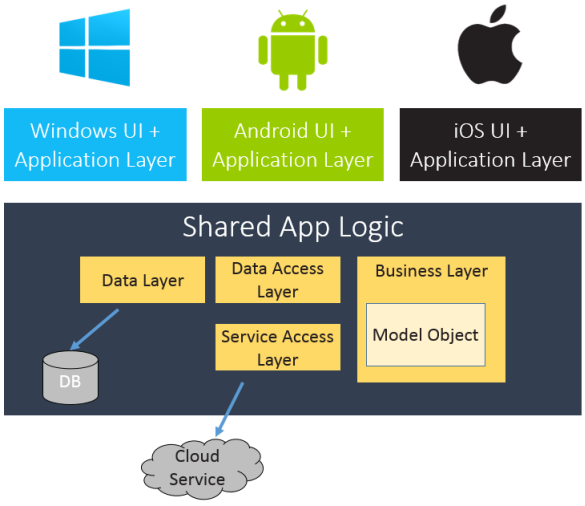
Source : Adapté de (Xamarin Inc. a, 2015)

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];
```

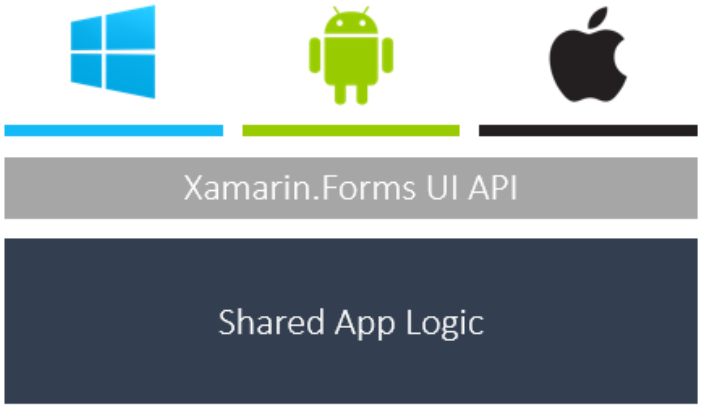
Xamarin nous permet très facilement de créer des applications iOS, Android ou Windows sans avoir besoin de connaissance dans les langages de développement propre à chacune des plateformes.

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 {…}
```

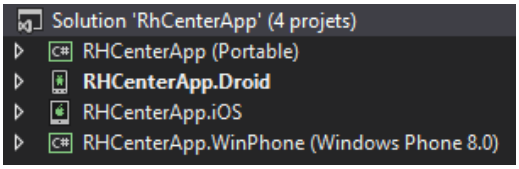
Si nous voulions déclarer un bouton de ce style, nous aurions dû l'écrire en *XML* pour Android et Windows Phone et jouer avec l'interface graphique de xCode pour iOS.

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.

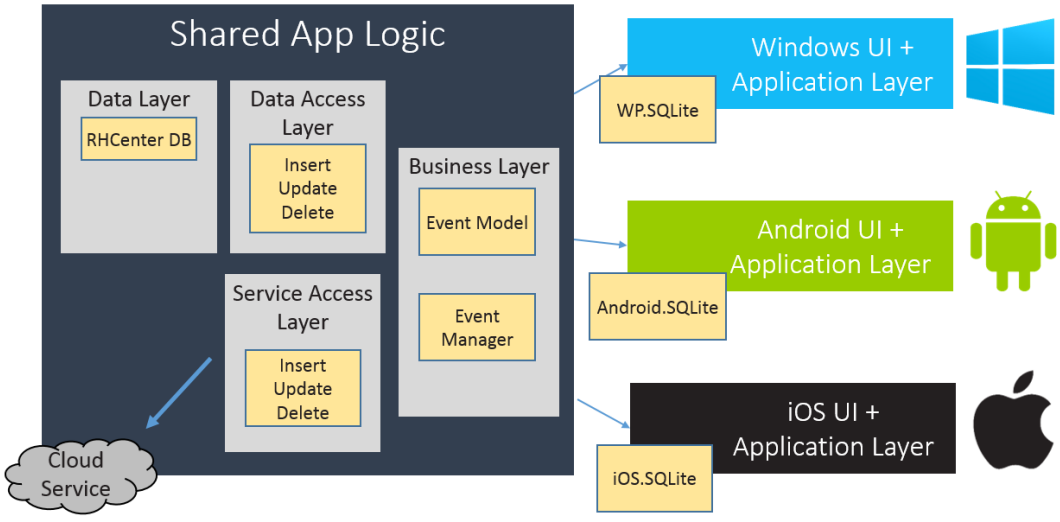
Malheureusement, Xamarin déconseille d'utiliser ce genre de projet lorsque plusieurs développeurs travaillent dessus. Expérience faite, le code devient très vite ingérable (cf. chapitre « 5.4.5 Shared App vs. Portable App »).

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 Simulation	Développement Simulation	—
Xamarin Studio Windows	Développement Simulation	Développement Simulation *	—
Extension Visual Studio	Développement Simulation	Développement Simulation *	Développement 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.