Les	Caractéristique d'pplication	Scénario
Graphisme	Important et différent pour chaque plateforme	
Type d'application		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)

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 ».

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 .

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.

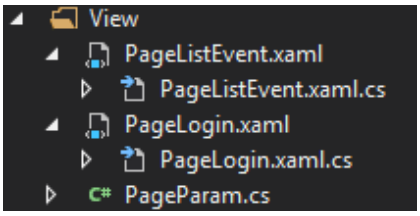
Le choix concernant le type d'application multiplateforme s'est porté sur le Xamarin.Forms. Cela permettait de tester les prétentions de Xamarin concernant le pourcentage de code partagé entre les différentes plateformes. De plus, il a été décidé par les membres de l'équipe RH-Center que cette application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

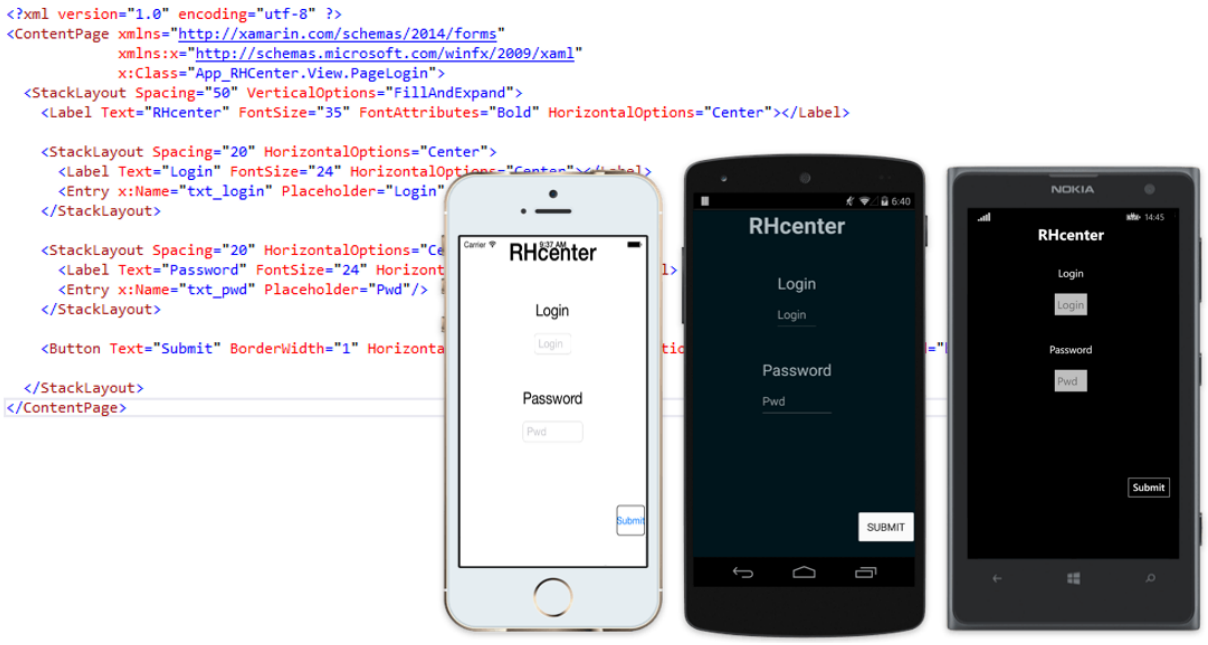
La suite de cette analyse expliquera chaque écran en développement avec l'*API* de Xamarin. Toutes les données inscrites sur les captures d'écrans ou utilisées dans les exemples sont tirées du cas pratique illustré dans le point « 4.2 Scénario » de ce présent travail.

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



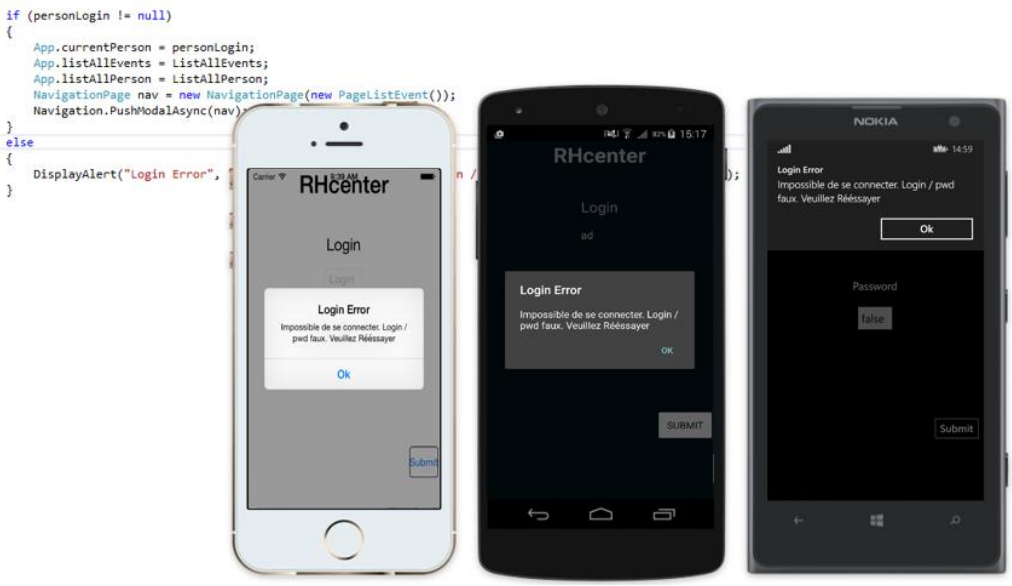
Source : Données de l'auteur

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.

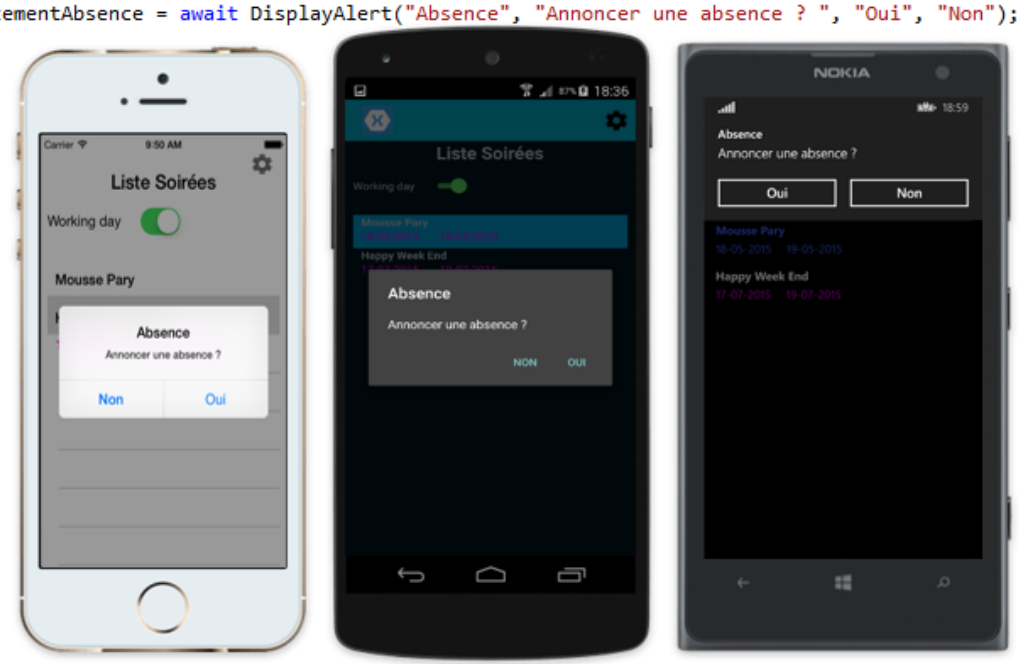
L'utilisateur a la possibilité de cliquer sur ses dates de travail. Lorsque ce dernier le fait, nous trouvons un message proposant d'annoncer une absence au responsable, illustré dans la Figure 25 : Annoncer une absence. Si la réponse est positive, un point rouge se mettra juste à côté de l'évènement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

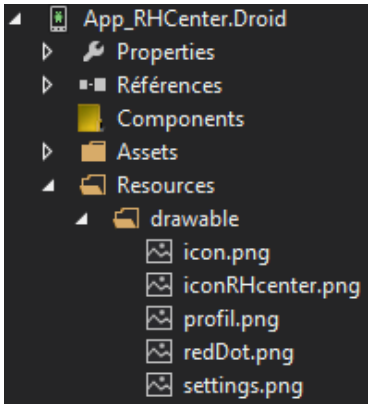
Pour cette page, il a été décidé de mélanger l'utilisation d'une page *XAML* pour le côté graphique mais aussi d'écrire ce dernier dans la classe liée à la page *XAML*. Ce choix fut pris afin de tester les différentes techniques de développement des éléments graphiques.

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.

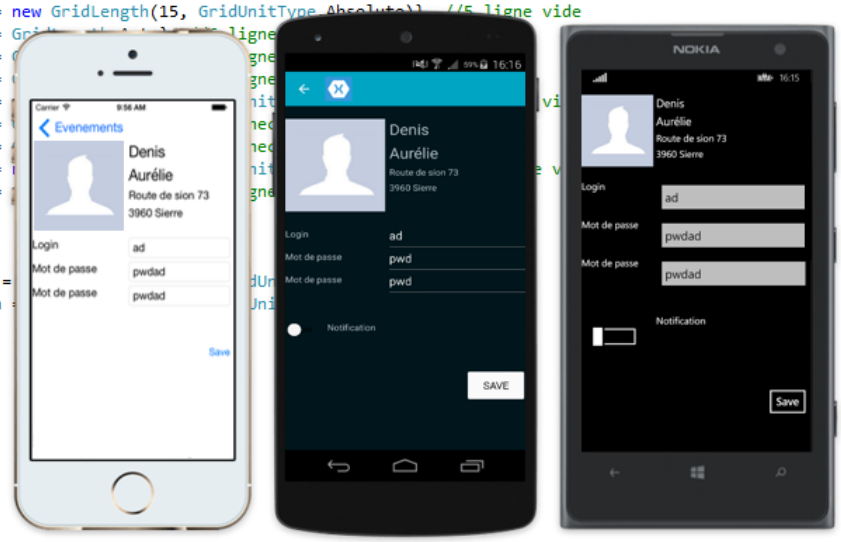
Par la suite, l'utilisateur aura la capacité de changer le login et le mot de passe de son compte. Pour se faire, il devra uniquement cliquer sur le champ texte du login ou du mot de passe. Pour le mot de passe, il est nécessaire d'insérer deux fois le même mot de passe afin qu'aucun message d'erreur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

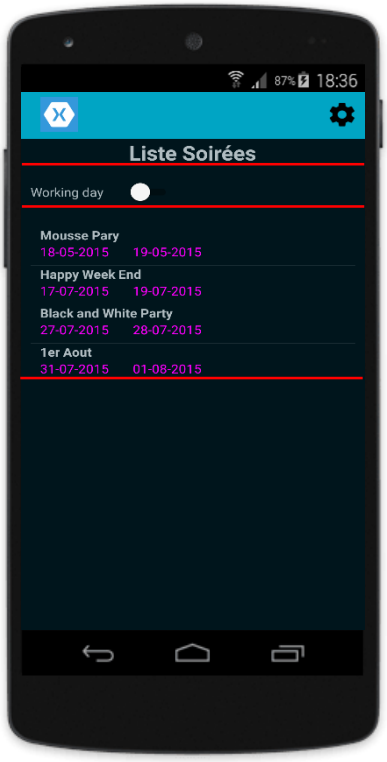
La première se trouve être l'accès à la galerie image du smartphone. Pour accéder à la galerie, le plugin XLabs a été utilisé. Ce dernier fut obtenu dans la bibliothèque *NuGet*. Ce dernier nous permet facilement d'accéder à de nombreux éléments natifs de chaque plateforme (cf. chapitre 4.5.1 XLabs).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

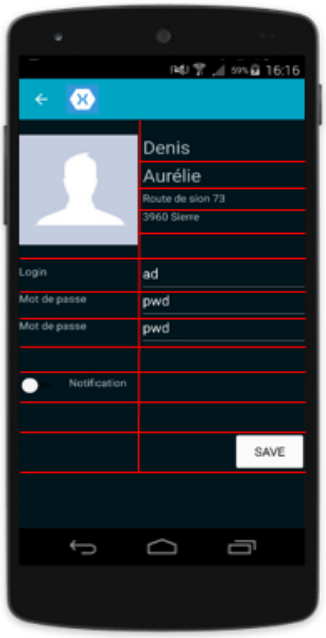
Le programme va ainsi donc comprendre que l'image sera positionnée la colonne 0 à gauche jusqu'à la colonne 1 en partant sur la droite (Colonne non comprise). Puis, il partira de la première ligne pour aller jusqu'à la cinquième ligne afin de fusionner ces cellules.

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#*

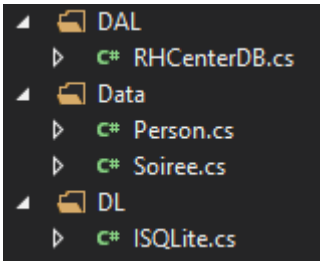
Ce fut la page que j'ai eu le plus de mal à écrire. Tout d'abord, le mélange des éléments graphiques et de leurs actions font de cette page un élément non structuré. Et enfin, le rajout de plugin, spécialement le XLabs, fut laborieux car les explications concernant son application sont toujours en cours 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

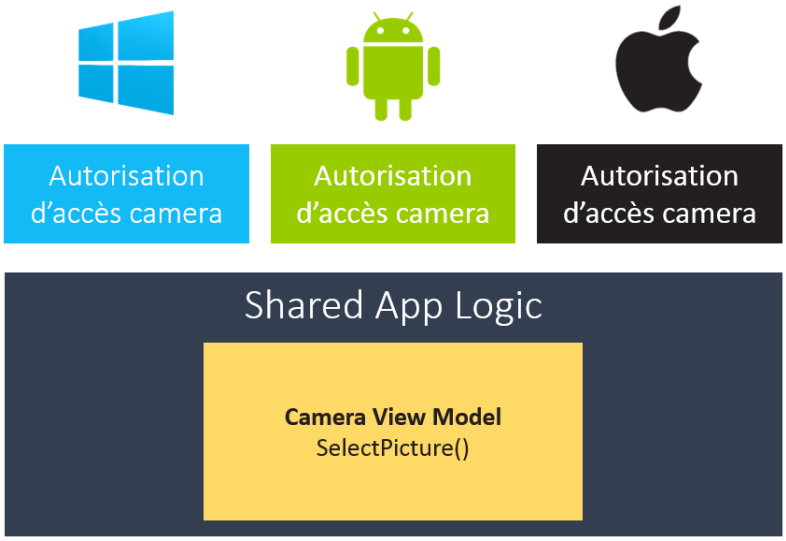
- L'accès à la caméra
- L'accès au GPS
- L'accès à la galerie

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.

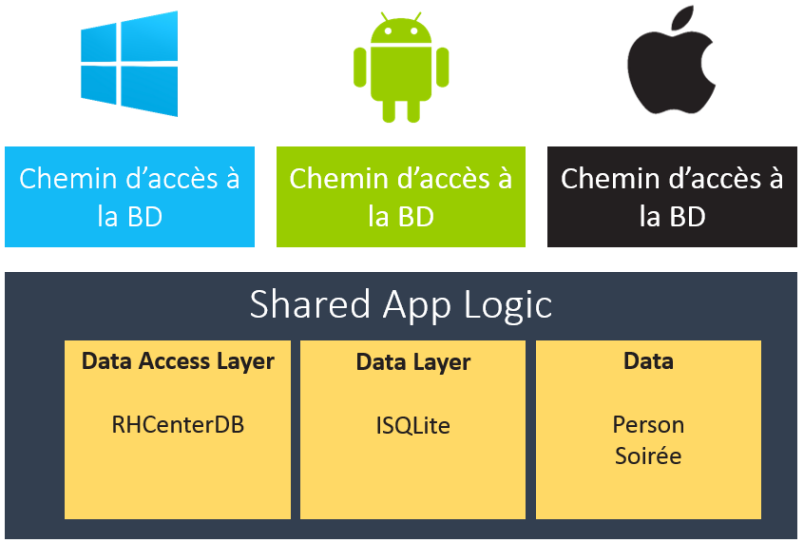
Comme expliqué dans le chapitre « 4.4 Base de données », tous les chemins d'accès vers la base de données sont définis dans les projets plateforme. Autrement, tout le reste de la gestion de la base de données se trouve dans le projet partagé. Cette gestion comprend :

- 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 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

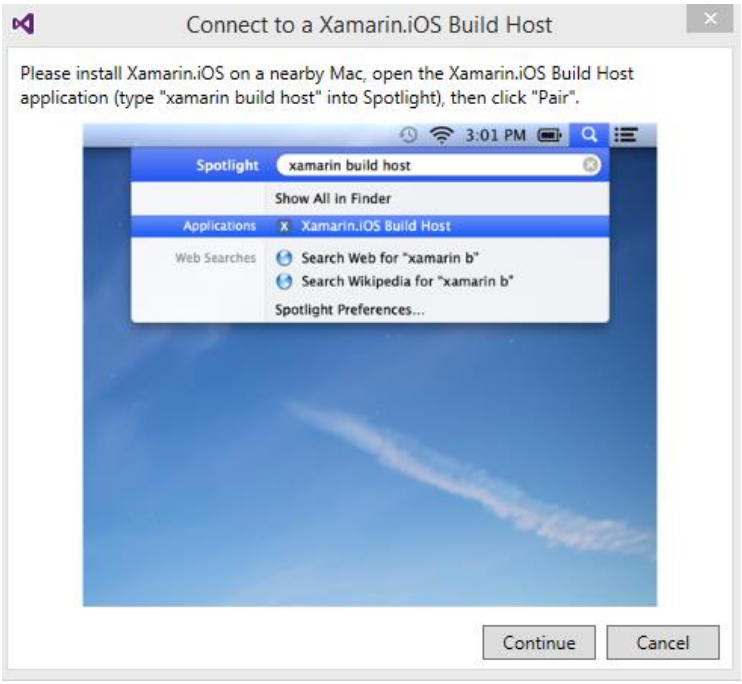
En temps normal, le développement d'une application iPhone se fait en *Objective*-C et dans certains cas avec le nouveau langage de développement d'Apple (Le *Swift*). Pour pouvoirs compiler ce code, nous avons besoin de l'*IDE* d'Apple qu'est X-Code. C'est pour cela que la programmation d'application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

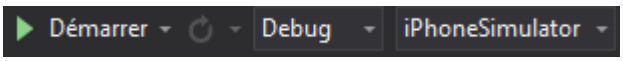
Dans le cadre de ce travail, la deuxième option fut choisie.

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

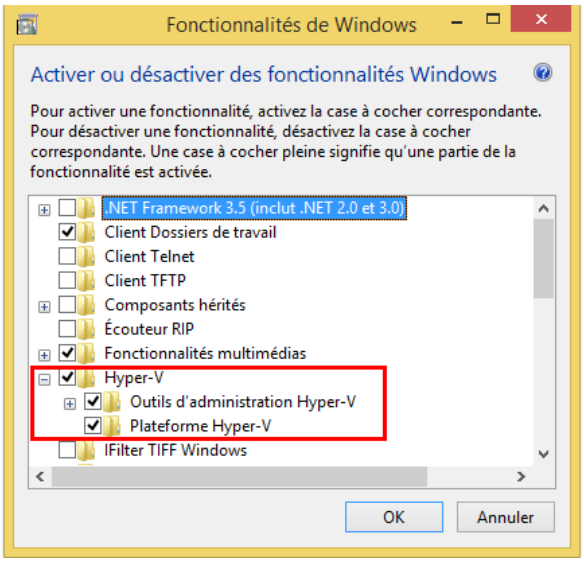
remarqués et corrigés.

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.

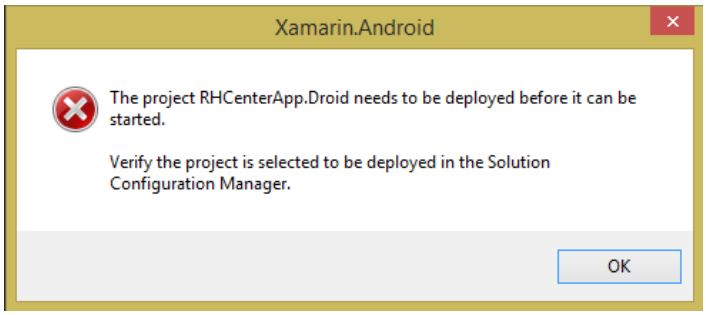
5.4.3 Tests sur Android

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.