Universal Windows Platform

Sommaire

- Introduction
- Création d’interface utilisateur
- Styles et ressources
- Implémenter la navigation
- MVVM et MVVM Light
- Gestion du périphérique
- Accès aux données locales
- Accès aux données distantes

Introduction

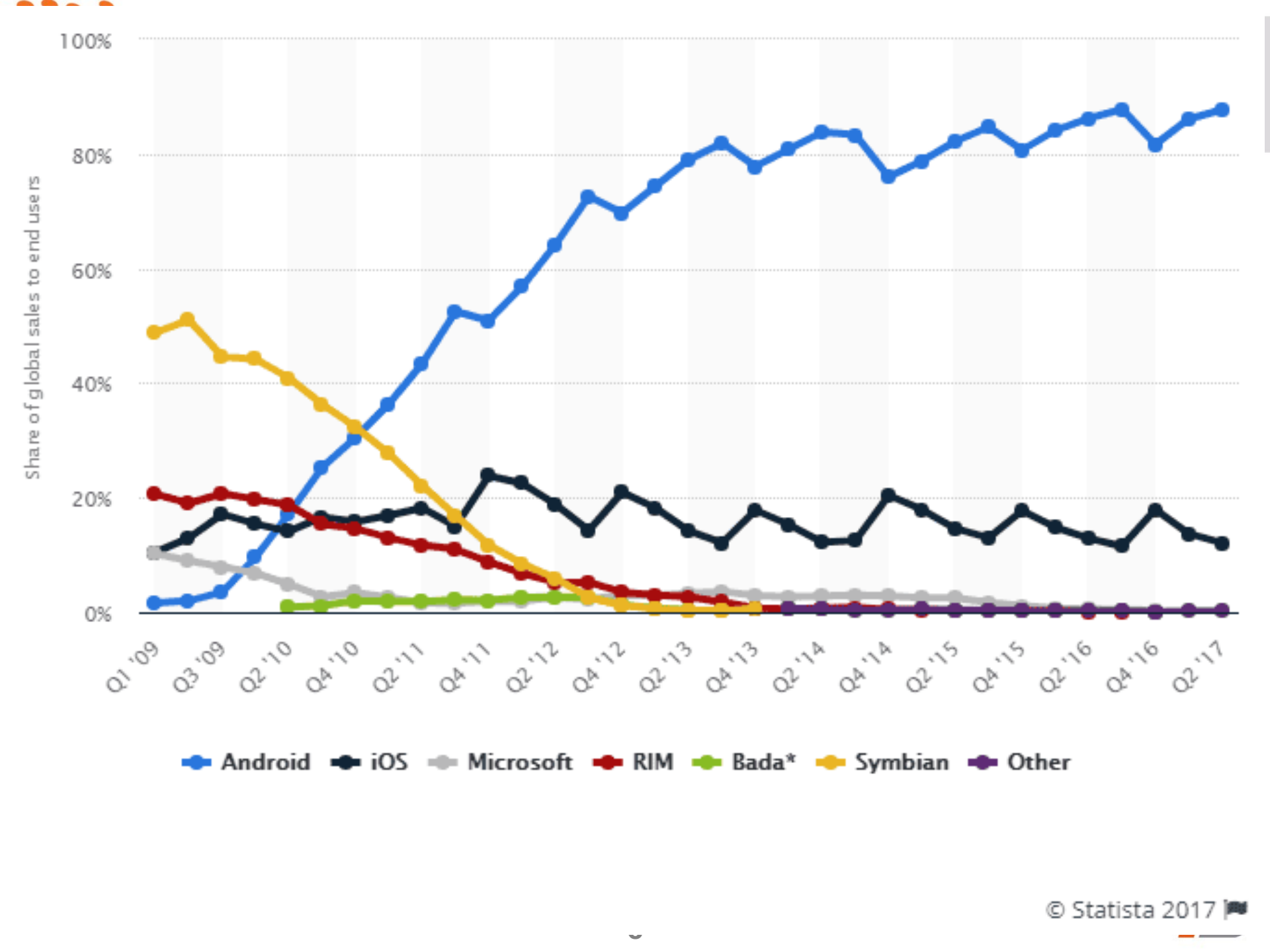
- Smartphone : Téléphone intelligent
- Historique Smartphone :
  - Simon (IBM – 1992) > BlackBerry (RIM – 1999) > Windows Mobile (HTC – 2003) > iPhone (Apple – 2007) > Android (HTC – 2008) > Windows Phone 7 (HTC – 2010) > Firefox OS (zte – 2014) | Windows Phone 8 (Nokia – 2014) > Windows 10 Mobile (2015)
- Historique Tablette :
  - Linus Write-Top (1987) > iPad (2010) > Samsung Galaxy Tab (2010) > iPad 2 (2011) > PlayBook (2011) > iPad 3 (2012) > Nexus 7/10 (2012)

Introduction

- Smart Watch : Montre intelligente
- Historique Smart Watch :
  - Tizen (Samsung - 2011) > Android Wear (LG – 2014) > Apple Watch (Apple – 2014)

Introduction

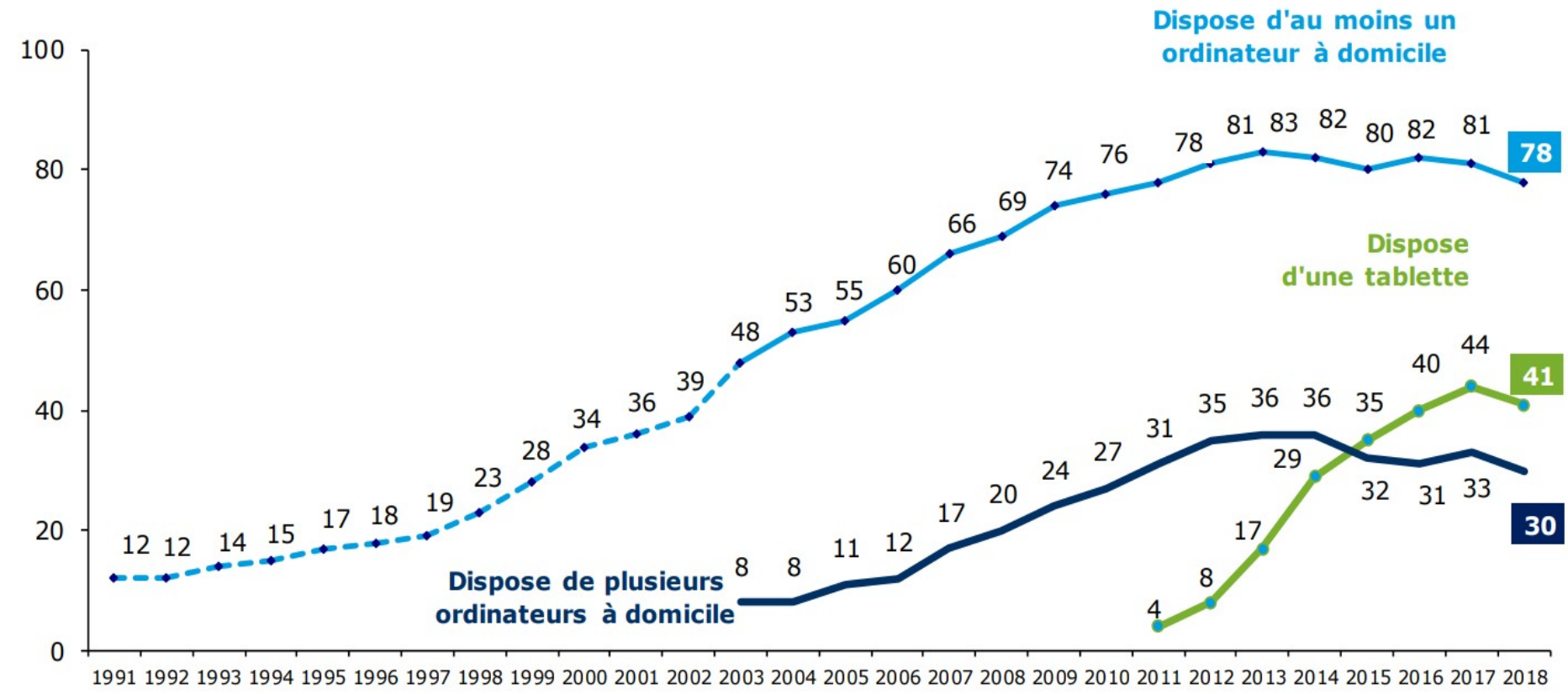
Système d'exploitation	2009 (1 trimestre)		2010 (1 trimestre)		2011 (1 trimestre)		2014 (année)		2015 (année)	
	Unités	Part de marché	Unités	Part de marché	Unités	Part de marché	Unités	Part de marché	Unités	Part de marché
Android	755900	1,8 %	10606100	17,2 %	36267800	36 %	1059000000	81,5 %	1161000000	81.0 %
iOS	5325000	13 %	8743000	14,2 %	16883200	16,8 %	193000000	14,8 %	226000000	15.8 %
Windows Mobile	3829700	9,4 %	3096400	5 %	3658700	3,6 %	34000000	2,6 %	31300000	2,2 %
BlackBerry OS	7782200	19 %	11228500	18,2 %	13004000	12,9 %	6000000	0,5 %	NC	N/A
Symbian Nokia	20880800	51 %	25386800	41,2 %	27598500	27,4 %	0	N/A	0	0
Linux (hors Android)	1901100	4,6 %	1503100	2,4 %	0	N/A	0	N/A	0	0
Autres	497100	1,2 %	1804800	1,8 %	3357200	3,3 %	7000000	0,6 %	11300000	0,8 %


100%
80%
60%
40%
20%
0%
Share of global sales to end users
Q1 '09
Q3 '09
Q2 '10
Q4 '10
Q2 '11
Q4 '11
Q2 '12
Q4 '12
Q2 '13
Q4 '13
Q2 '14
Q4 '14
Q2 '15
Q4 '15
Q2 '16
Q4 '16
Q2 '17
Android
iOS
Microsoft
RIM
Bada*
Symbian
Other
© Statista 2017

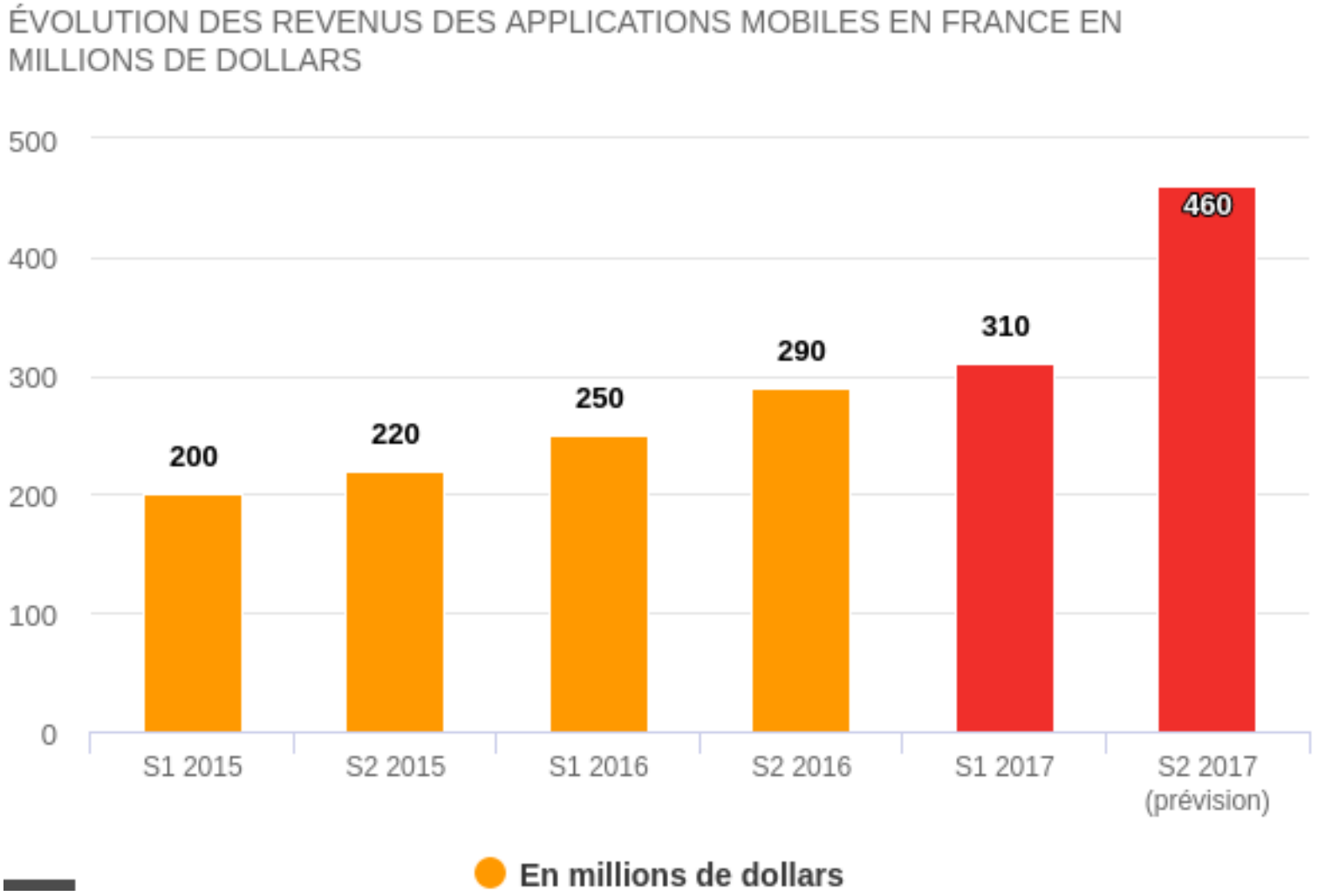
Introduction

Le taux d'équipement en ordinateur et en tablettes diminue (en %)

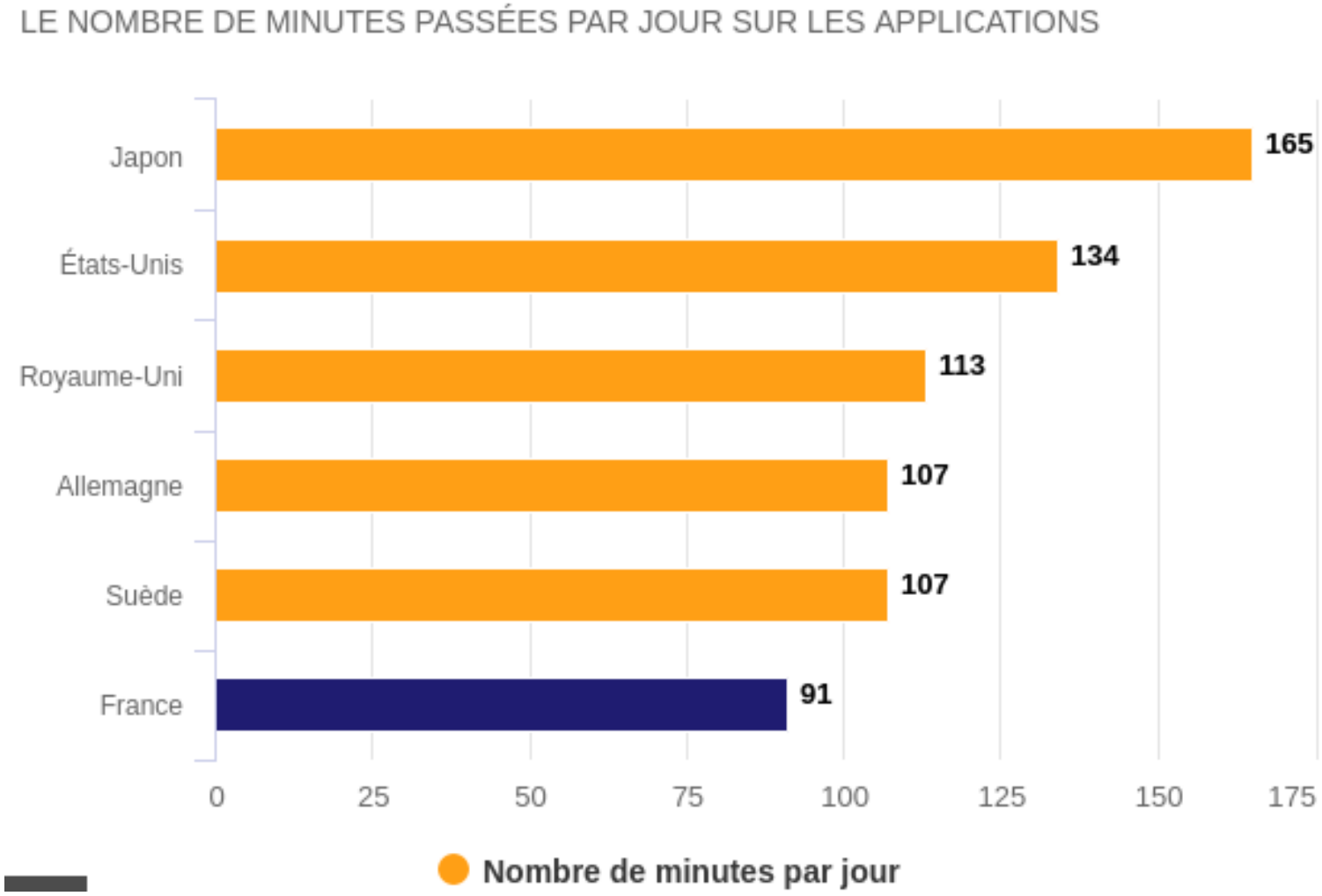
- Champ : ensemble de la population de 12 ans et plus, en % -



Introduction



Introduction

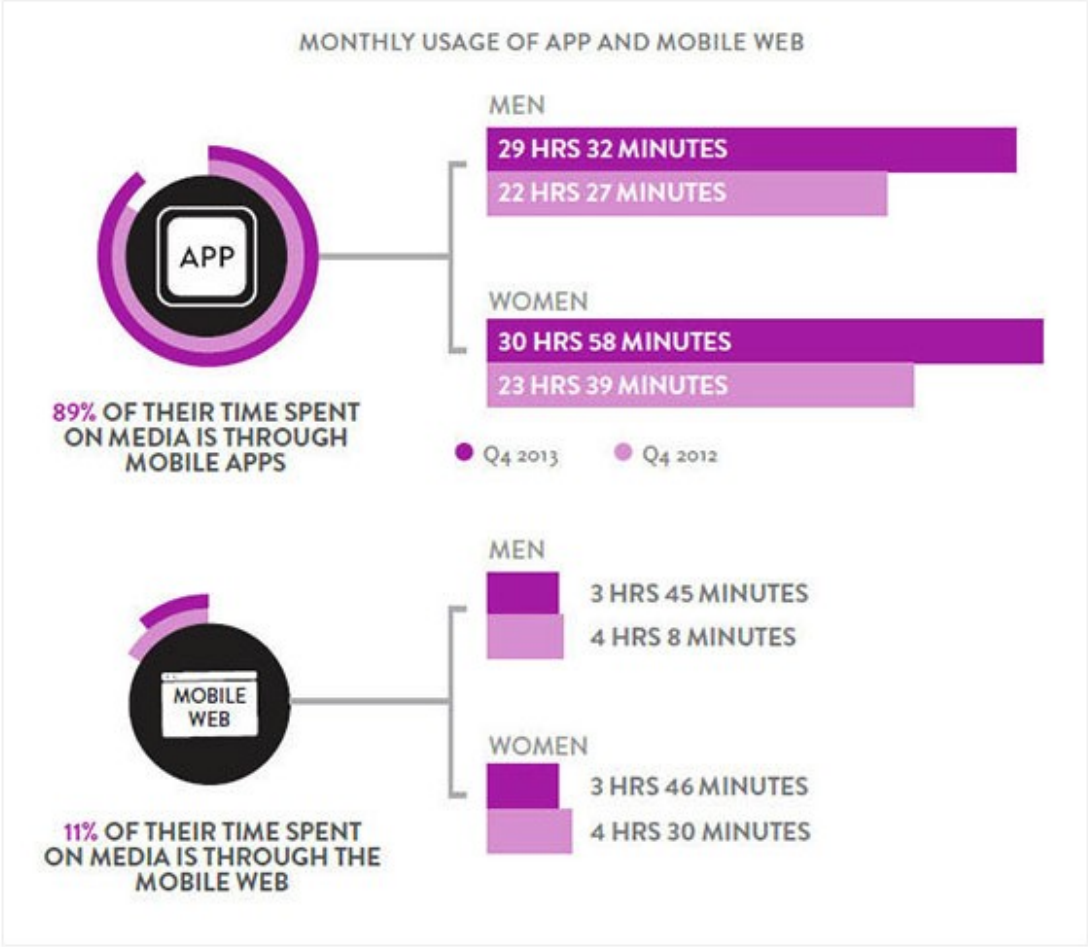


Introduction

- Annonce arrêt de WindowsPhone Juillet 2017
  - Guerre de la mobilité perdu
    - Arrivé trop tardive sur le marché
    - Communauté de la mobilité déjà experte en Android et iOS
  - Fin du support de Windows 10 mobile 2019
  - Reconditionnement des développements vers UWP
    - Utilisable sur Windows, Windows TV, Windows Mobile
  - Réécriture et création de .NET-Core
    - Multi-plateforme

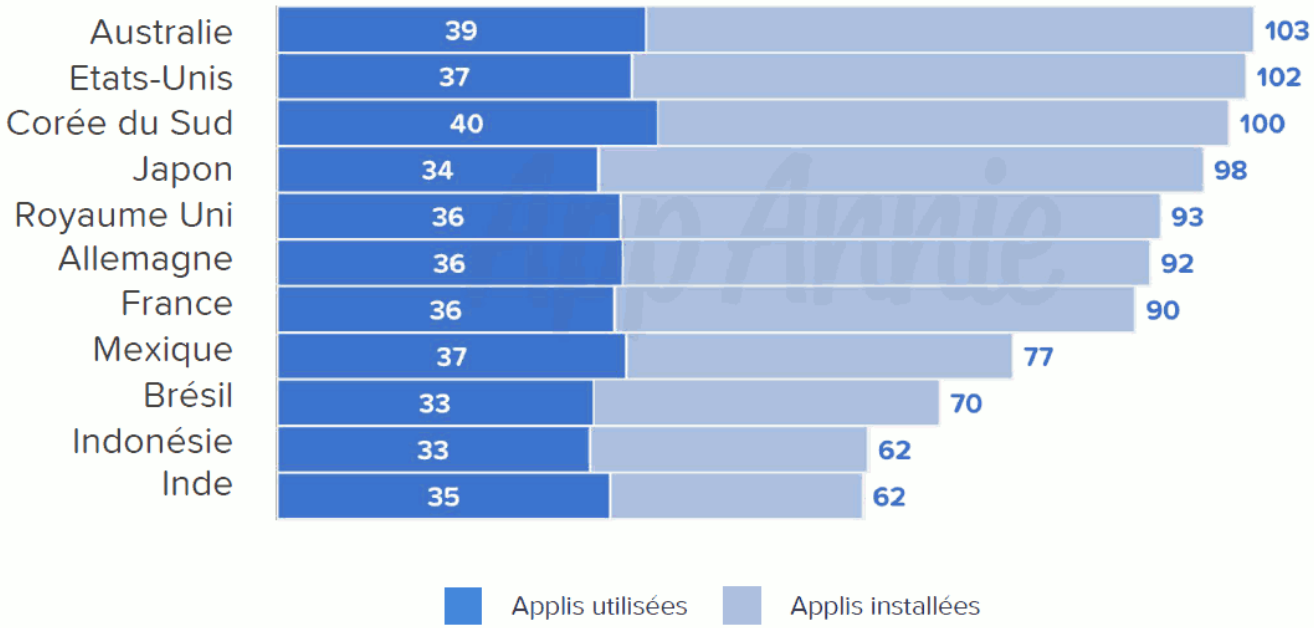
Introduction

- Le point de vue de l'utilisateur



Introduction

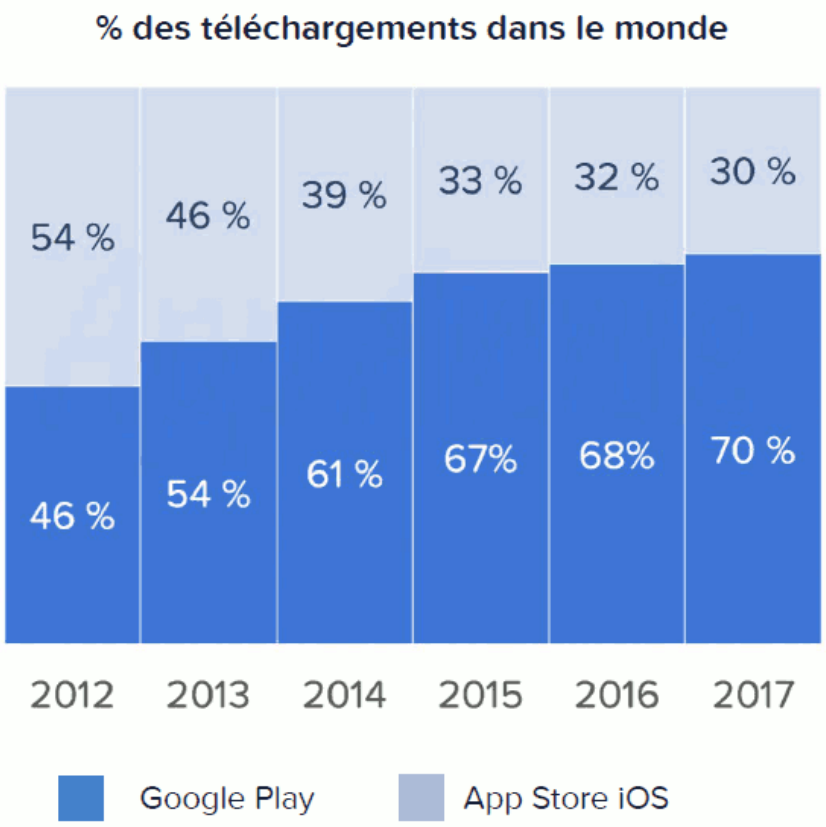
Nombre moyen d'applis installées et utilisées sur les terminaux Android des principaux marchés au 1er sem. 2018

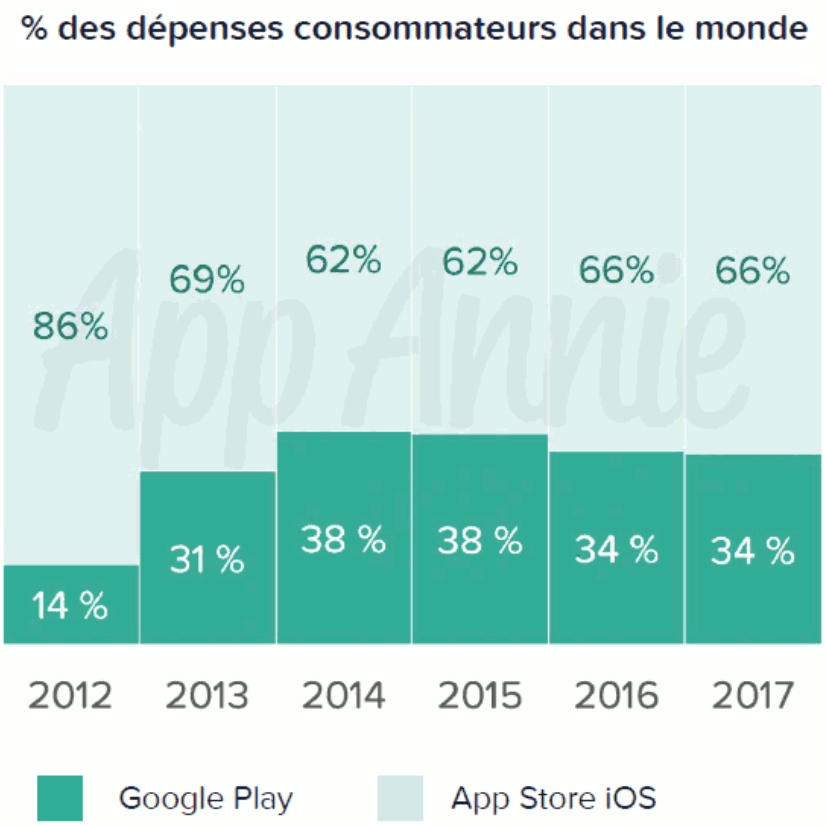


Introduction

- Le point de vue du magasin d'applications
- Store => magasin d'applications
- Google Play (Google)
  - Issue de l'Android Market publié le 28 août 2008
  - Créé le 6 mars 2012
  - Fusion de l'Android Market, Google Movies, Google ebookstore et Google Music
- App Store (Apple)
  - Créé le 11 juillet 2008
  - Utilisable pour l'iPod touch, l'iPhone et l'iPad

Introduction

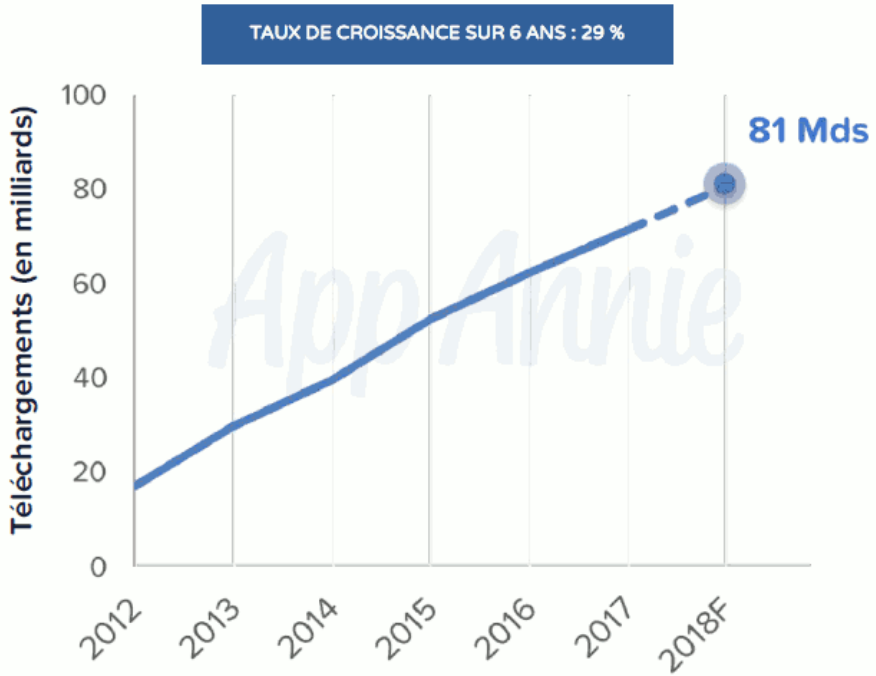




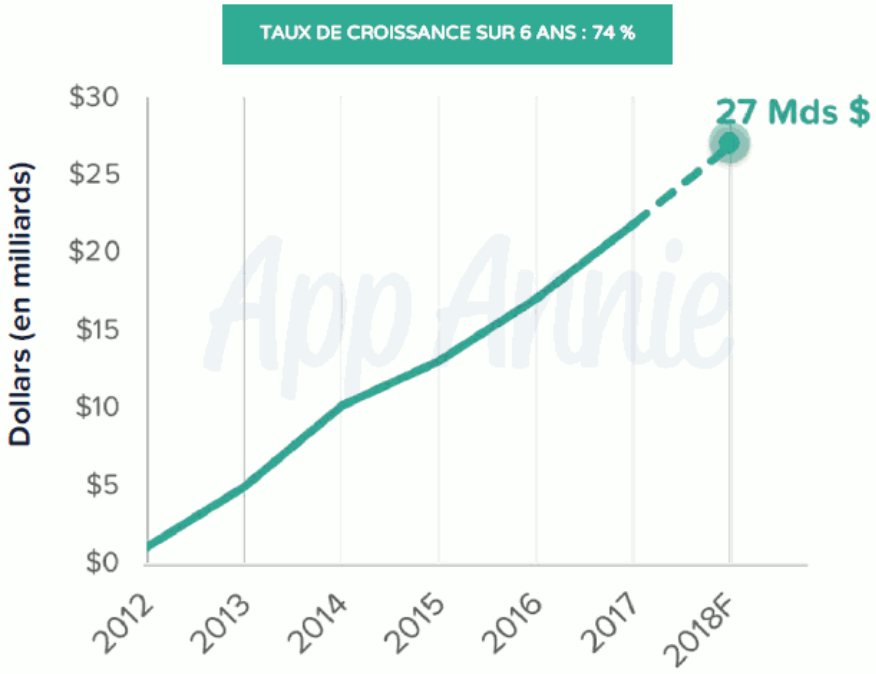
Note : Google Play n'est pas disponible en Chine

Introduction

Téléchargements Google Play dans le monde*



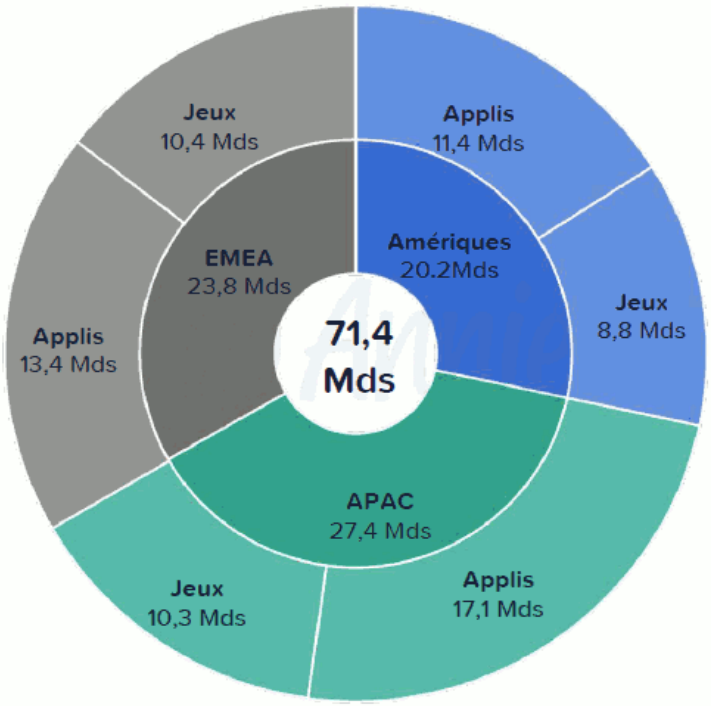
Dépenses consommateurs Google Play dans le monde*



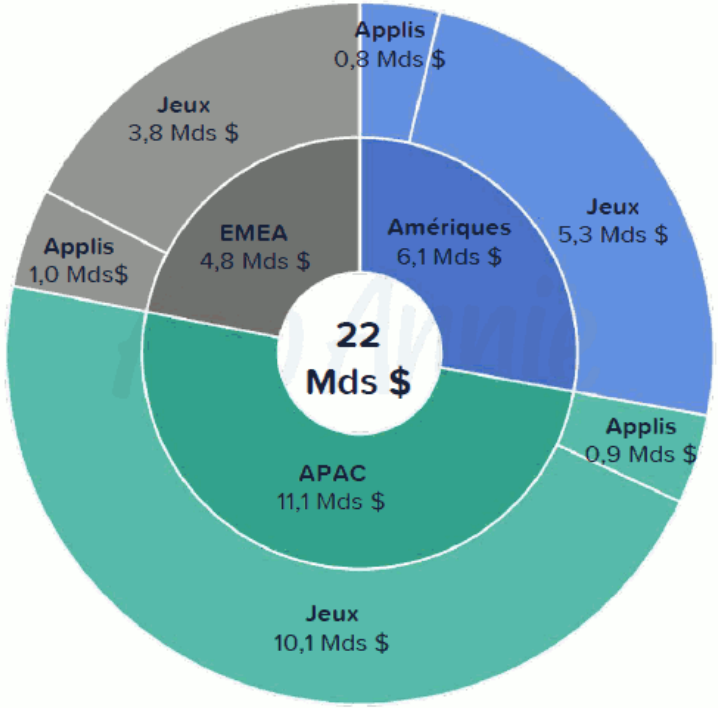
*Dépenses brutes, qui incluent les frais Google Play
Note : Google Play n'est pas disponibles en Chine.

Introduction

Téléchargements Google Play en 2017

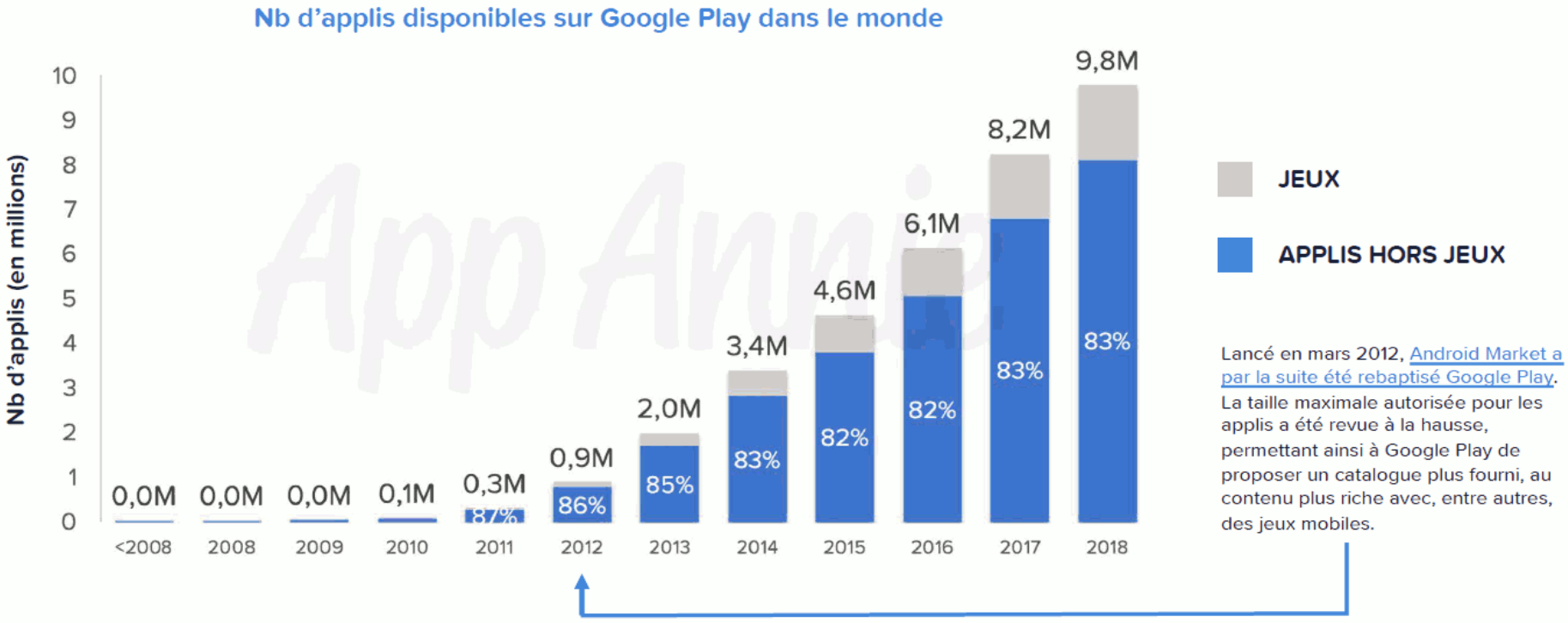


Dépenses consommateurs Google Play en 2017



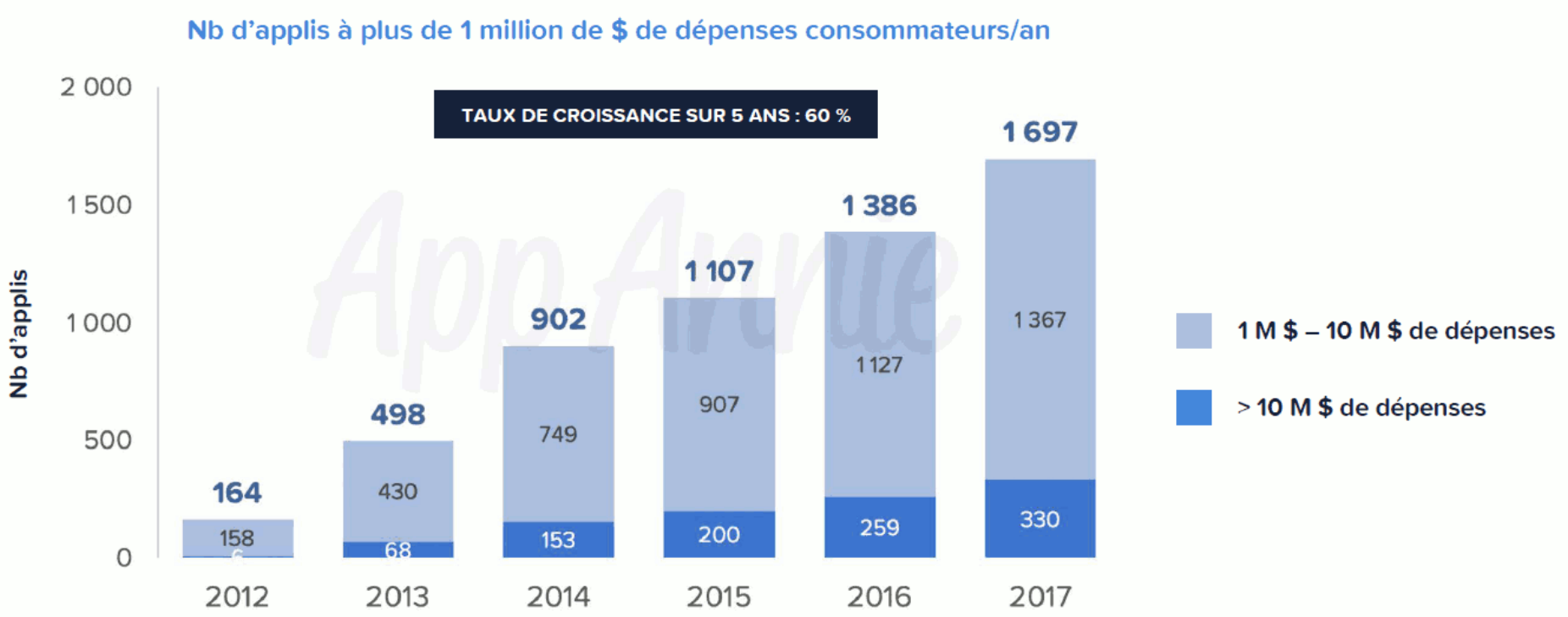
Note: Google Play n'est pas disponible en Chine

Introduction



Note : Le tableau recense le nombre d'applications sorties au 31 août 2018. La date de sortie d'une appli correspond à sa date d'entrée dans le classement des téléchargements ou des chiffres d'affaires Google Play d'un pays.

Introduction



Introduction

Dépenses consommateurs dans le monde liées aux applis Google Play (hors jeux)

Dépenses consommateurs dans le monde (hors jeux)

	0,2 Mds $	0,8 Mds $	1,0 Mds $	1,2 Mds $	1,7 Mds $	2,7 Mds $
1	GREE	LINE	LINE	LINE	LINE	LINE
2	LINE	LINE PLAY	LINE PLAY	LINE Manga	Tinder	Tinder
3	SwiftKey Keyboard	GREE	LINE Manga	LINE PLAY	HBO NOW	Pandora Music
4	Documents To Go	KakaoTalk	KakaoTalk	Pandora Music	Pandora Music	Netflix
5	Poweramp	Pandora Music	Pandora Music	Pokecolo	LINE Manga	HBO NOW
	2012	**2013**	**2014**	**2015**	**2016**	**2017**

Top 5 des applis par dépenses consommateurs dans le monde (hors jeux)

Applis comprenant un abonnement intégré au moment du classement

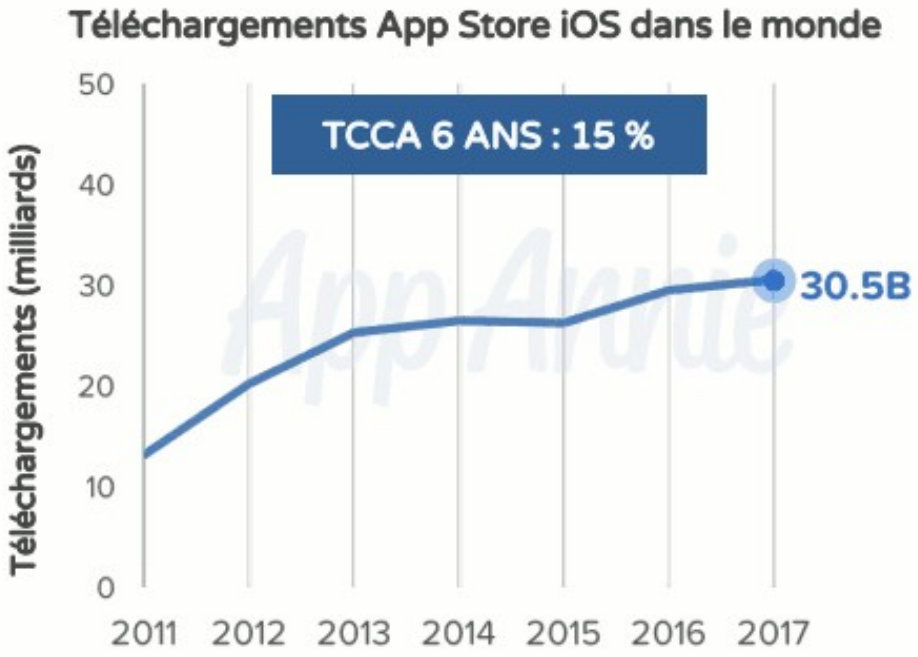
Introduction des abonnements intégrés en mai 2012

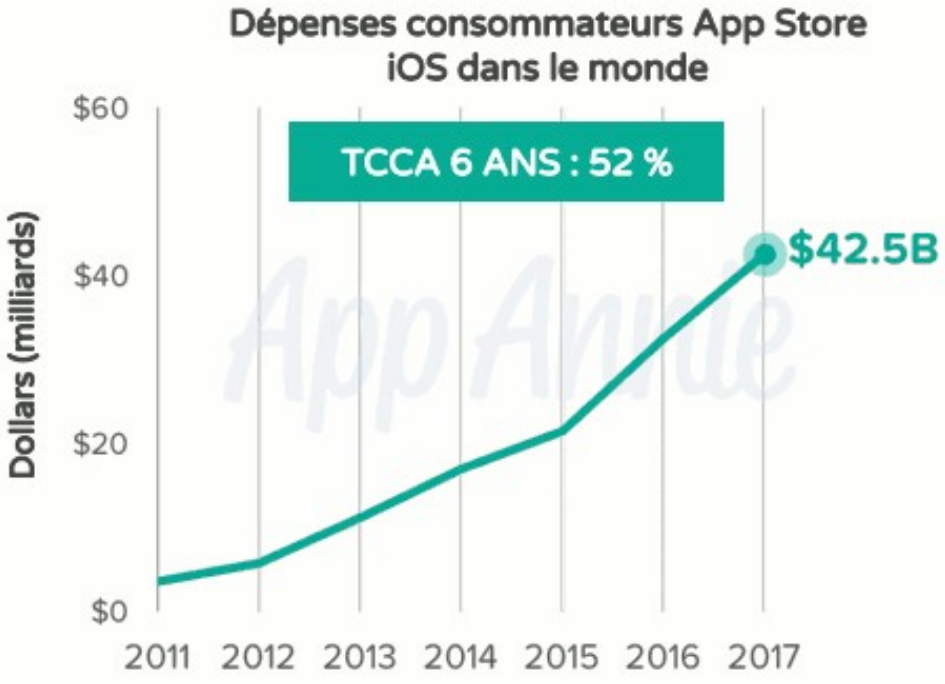
Google a modifié son modèle d'abonnement en octobre2017 pour encourager les développeurs à commercialiser les abonnements sous forme d'achat intégrés. Les frais d'App Store prélevés sur les transactions sont ainsi passes de 30% à 15% pour les abonnements à long terme. Croissance des dépenses de 55% entre 2016 et 2017 – la plus forte croissance annuelle depuis 2012.

Introduction

- Commission « Google Play »
  - 30 % du prix de l'application à l'achat est reversé à Google
  - Google propose aussi une bibliothèque pour les achats « in-app »
  - Publier des applications demande un compte développeur Google associé à une adresse mail Google
    - 25 USD, paiement unique et compte disponible à vie
    - Pas de limitation pour le nombre d'application publié
- Store alternatifs
  - Amazon Appstore : 2011
  - F-Droid : 2010

Introduction

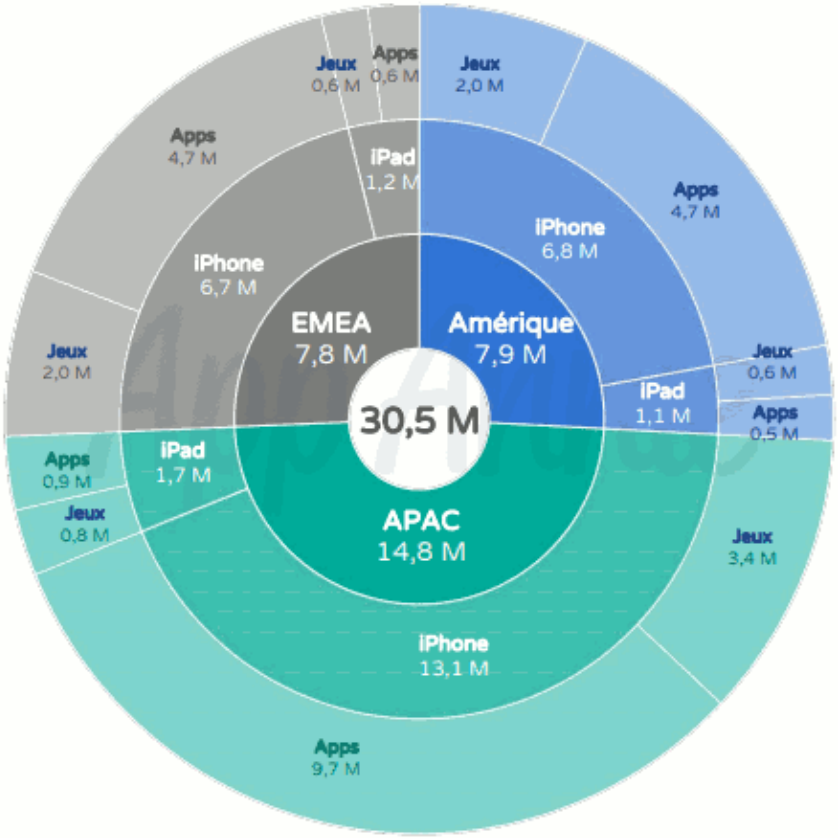




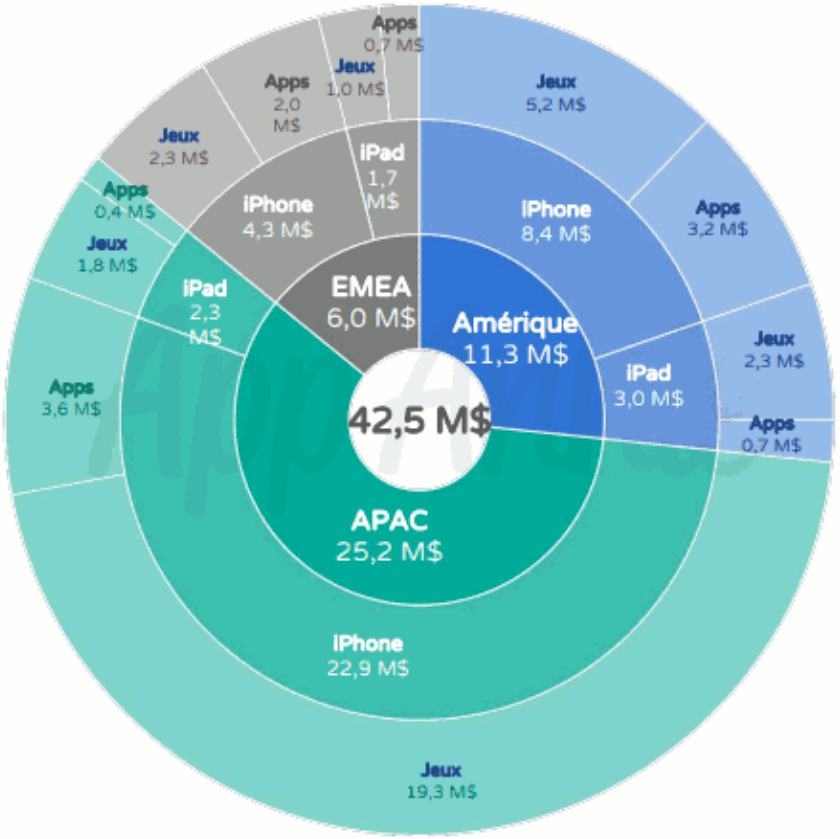
*Les dépenses sont brutes, les frais d'App Store sont inclus

Introduction

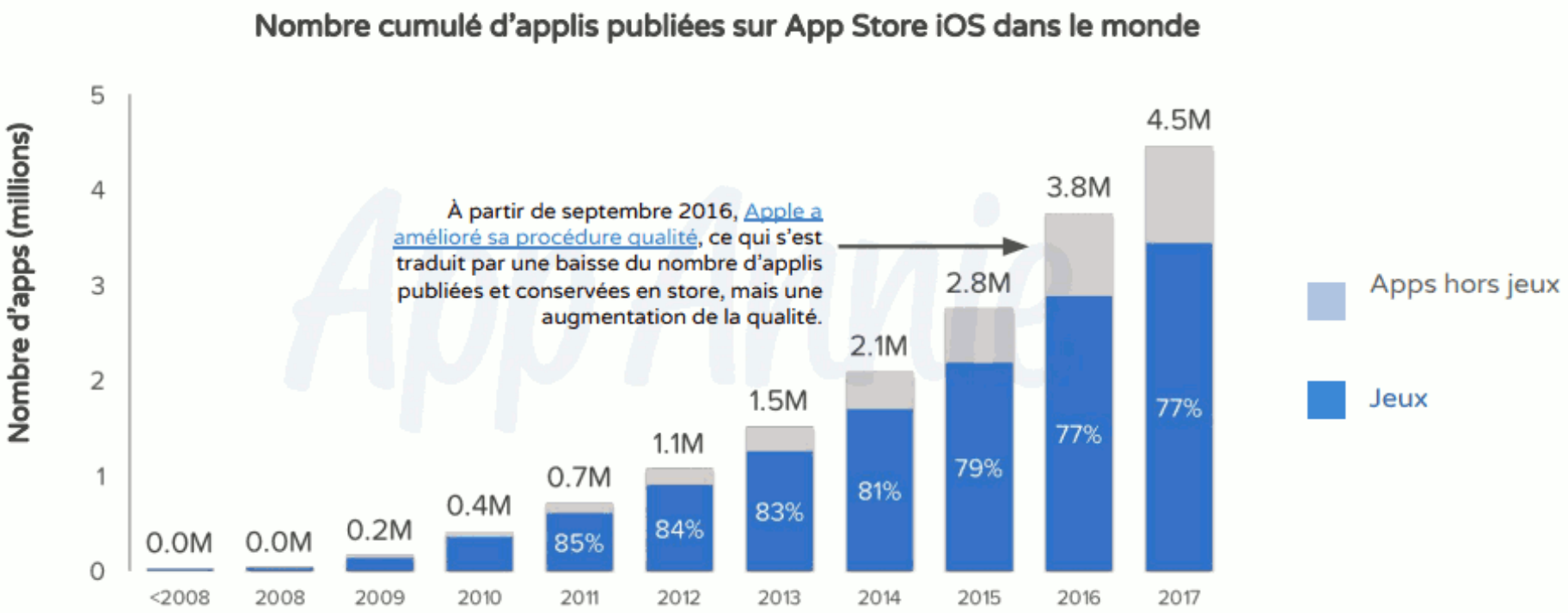
Téléchargements App Store iOS en 2017



Dépenses consommateurs App Store iOS en 2017

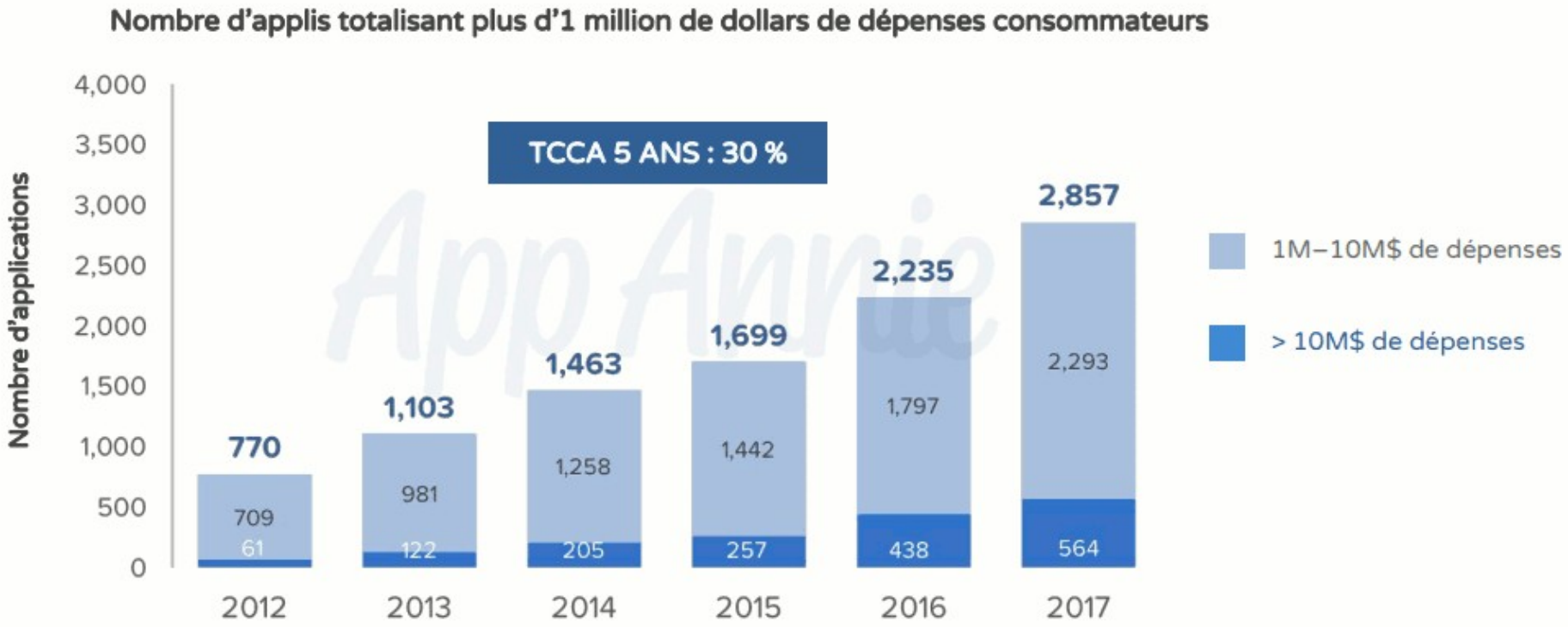


Introduction



Remarque : la date de publication est la date du premier classement de l'app dans l'App Store iOS, pour les téléchargements ou le chiffre d'affaires, dans n'importe quel pays

Introduction



Introduction

- Commission « App Store »
  - 30 % du prix de l'application à l'achat est reversé à Apple
  - Apple propose aussi une bibliothèque pour les achats « in-app »
  - Publier des applications demande un compte développeur Apple selon 1 des 2 programmes possibles
    - Apple Developer Program 99 USD / an
    - Apple Developper Entreprise Program 299 USD / an
- Store alternatifs
  - AppValley
  - vShare

Introduction

- Le point de vue du développeur
- Financement d'une application
  - Application payante
    - A privilégier pour les applications iPhone / iPad
    - Peut efficace pour les utilisateurs Android
  - Publicité
    - Alternative à l'achat d'application
    - Régies : Madvertise (ex Mobile Network Group), Mobvalue, AdMob (Google), Audience Network (Facebook), Orange Advertising, SFR Régie, Webedia
  - Freemium
    - Version Lite gratuite
    - Version Full payante

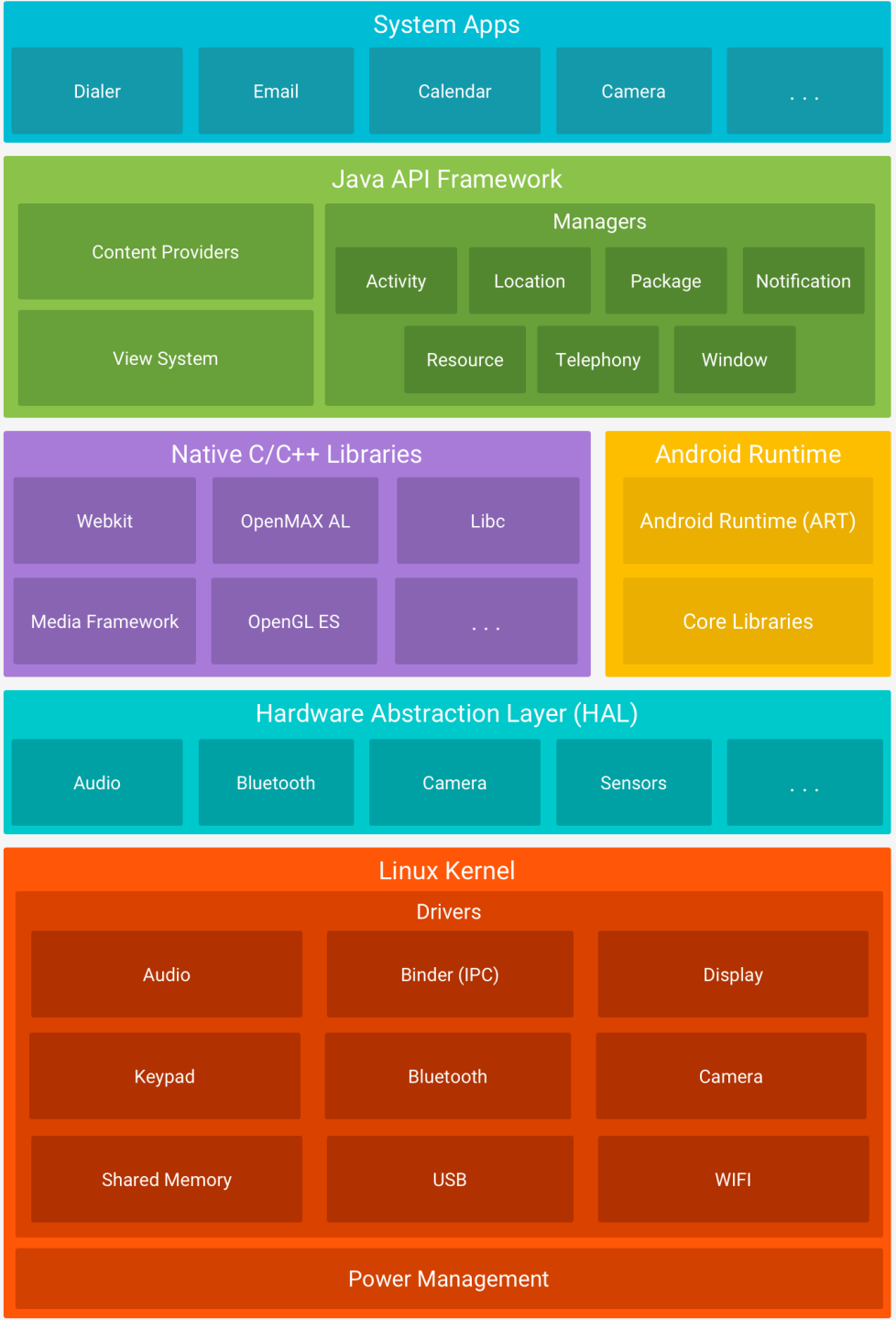
Introduction

- Achat « in-app »
  - Achat dans l'application (réapprovisionnables ou non / abonnements)
  - Très utilisé dans les jeux (monnaie virtuelle – add-on - bonus)
- Sponsoring
  - Espace publicitaire réservé à un annonceur
  - Partenariat entre éditeur et marque limité dans le temps
- Location de BDD « opt-in »
  - Collecte de données utilisateurs louées aux annonceurs
  - Newsletter envoyé aux utilisateurs (autorisation)
  - Peu utilisé

Introduction

- Plateforme de développement
- Android
  - Smartphone & Tablette
  - Langage Java Android
  - Utilise l'Android SDK (system development kit)
    - Contient les outils nécessaires au développement Android
  - Peut utiliser l'Android NDK (native development kit)
    - Permet d'accéder aux fonctionnalités native de la plateforme Android (langage C)

Intro


System Apps
Dialer
Email
Calendar
Camera
. . .
Java API Framework
Content Providers
View System
Managers
Activity
Location
Package
Notification
Resource
Telephony
Window
Native C/C++ Libraries
Webkit
OpenMAX AL
Libc
Media Framework
OpenGL ES
. . .
Android Runtime
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
. . .
Linux Kernel
Drivers
Audio
Binder (IPC)
Display
Keypad
Bluetooth
Camera
Shared Memory
USB
WIFI
Power Management

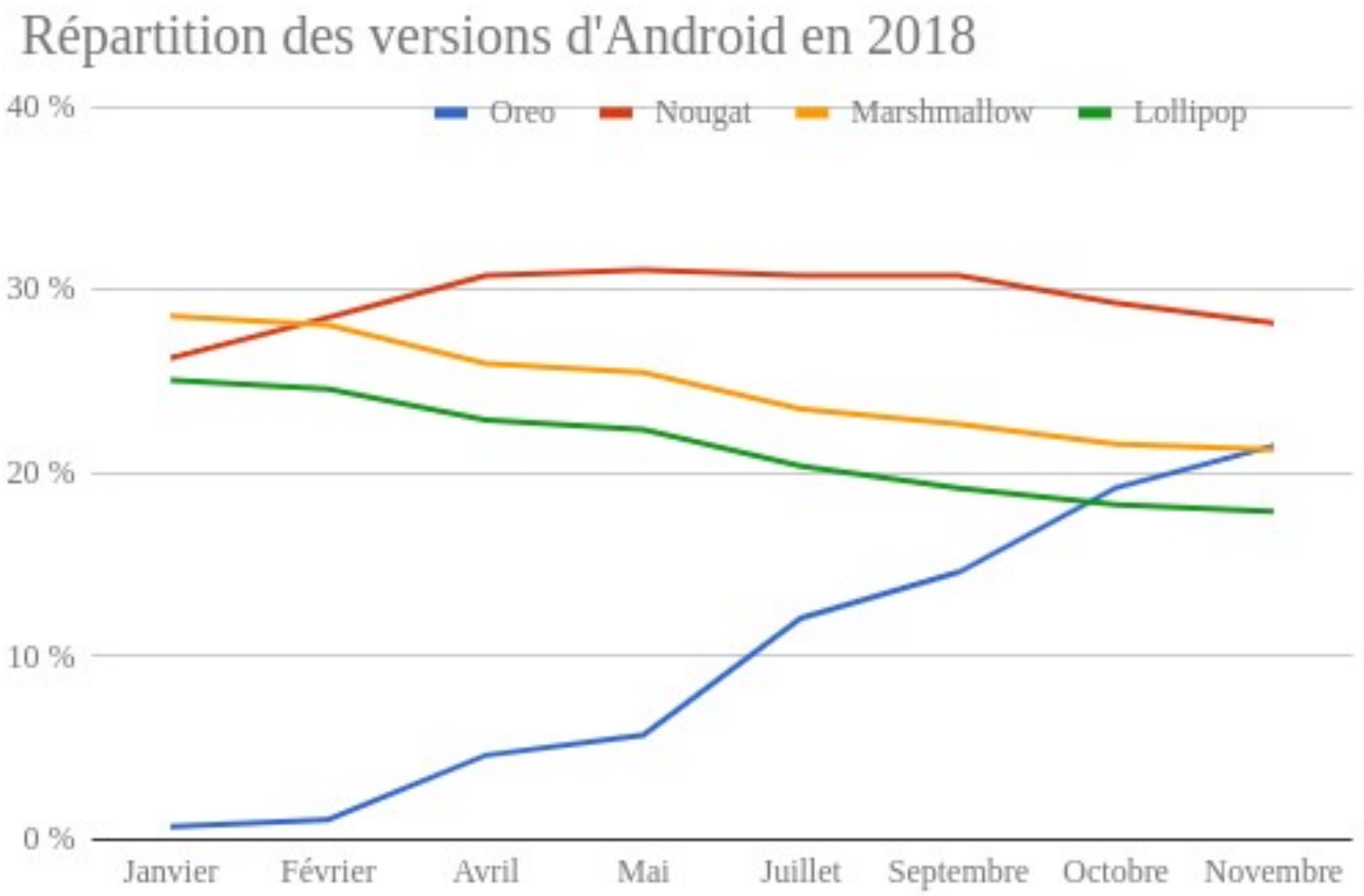
Introduction

- Android
  - Smartphone & Tablette
  - Beaucoup de résolution disponible
  - WVGA: 800 x 480 pixels
  - HD: 1280 x 720 pixels
  - Full HD: 1920 x 1080 pixels
  - QHD ou Quad HD ou WQHD: 2560 x 1440 pixels
  - Quad HD+: 2960 x 1440 pixels

Introduction

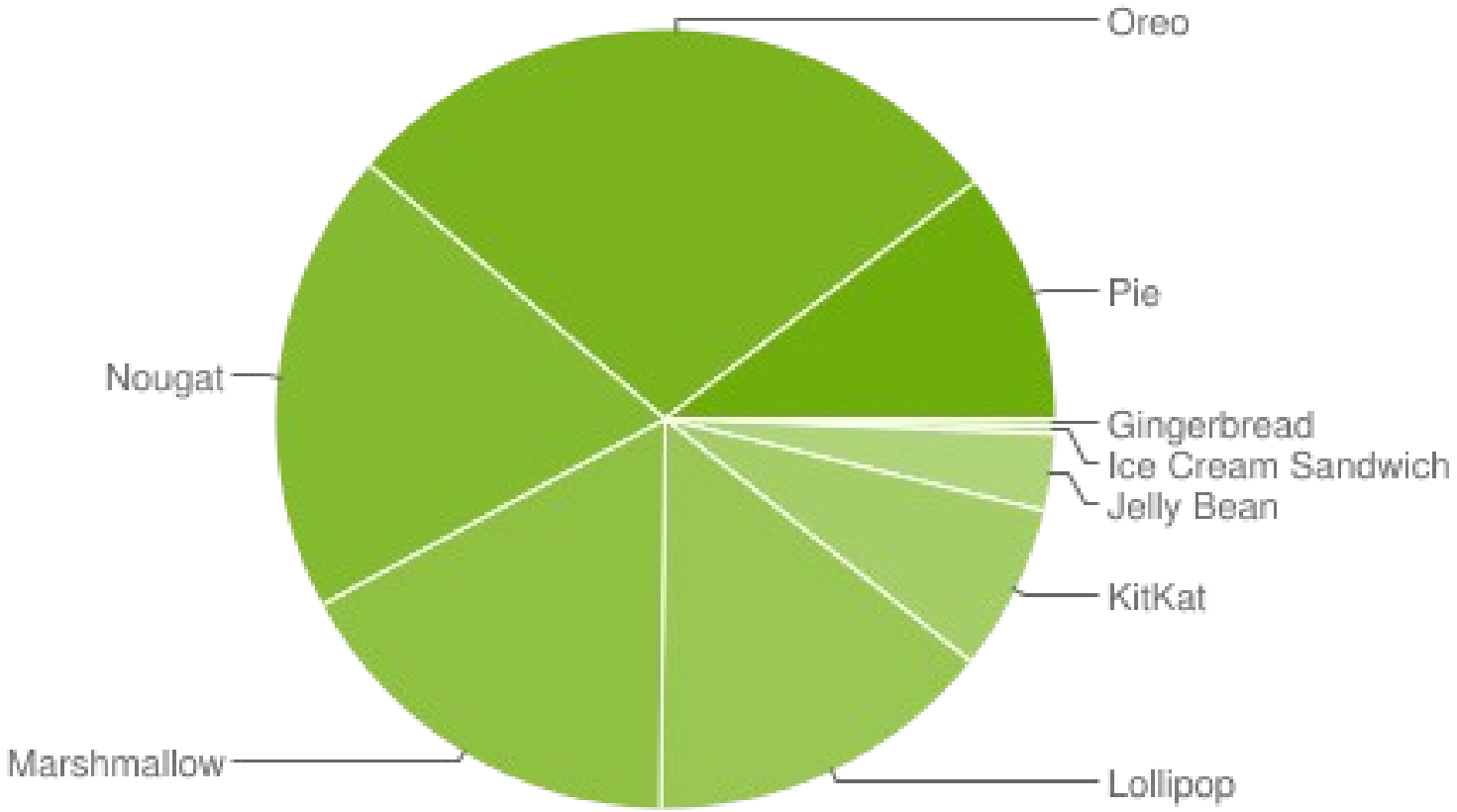
- Plateforme de développement
  - Un grand écosystème de version actives
    - Dont 90 % regroupé en 4 versions en 2018
    - 90 % regroupé en 5 version en 2019
    - Et 10 % couvrant les autres versions existante
    - Librairie de rétrocompatibilité pour supporter les anciennes versions
      - Android support-v7
      - Android support-v4
      - …
    - Historique des versions

Introduction



Introduction

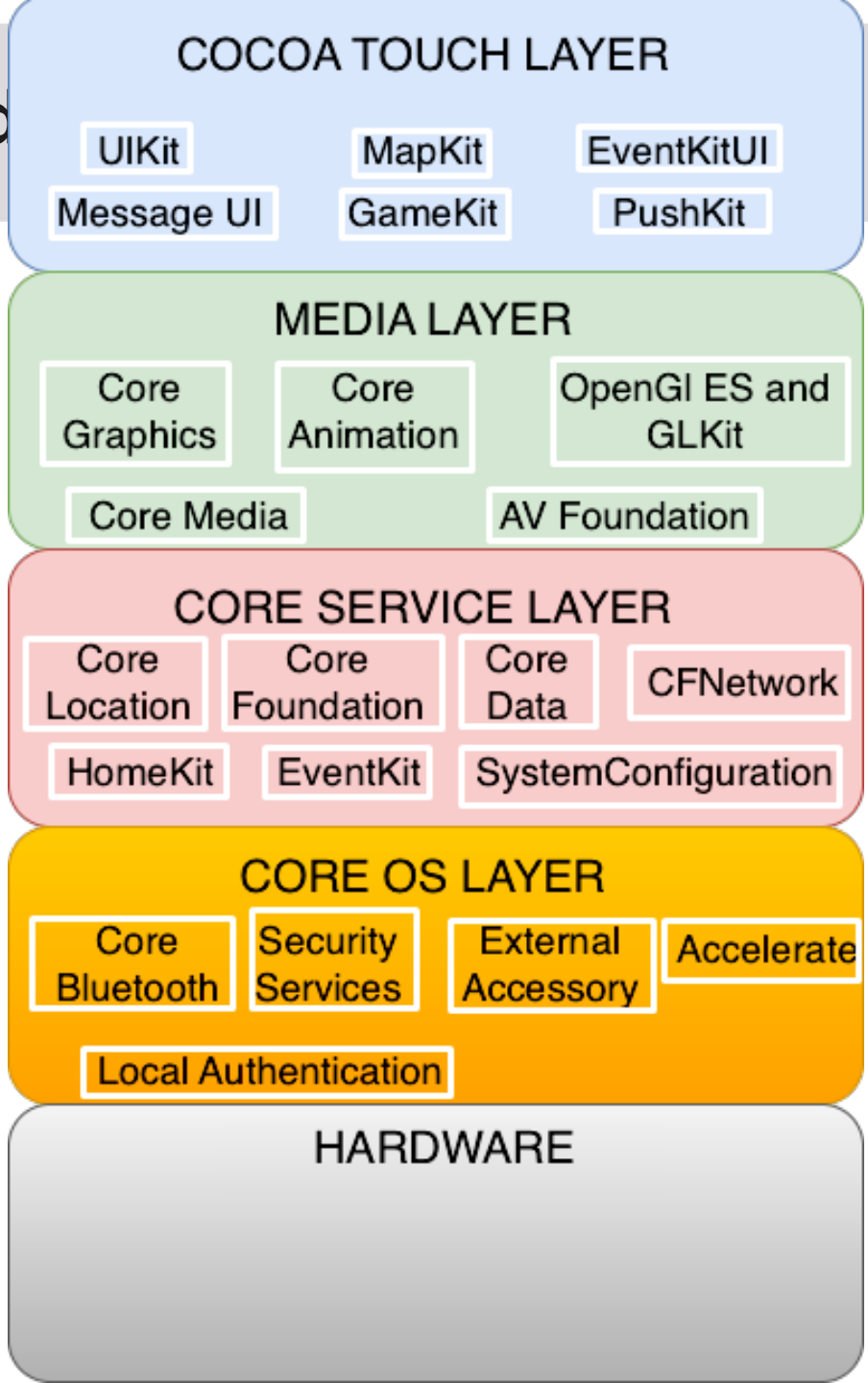
- Répartition des versions installé en 2019



Introduction

- IOS
  - iPhone & iPad
    - Langage Objective-C, Swift
    - Utilise l’iOS SDK
    - Peut accéder directement aux fonctionnalités native C

Introd


COCOA TOUCH LAYER
UIKit
MapKit
EventKitUI
Message UI
GameKit
PushKit
MEDIA LAYER
Core Graphics
Core Animation
OpenGl ES and GLKit
Core Media
AV Foundation
CORE SERVICE LAYER
Core Location
Core Foundation
Core Data
CFNetwork
HomeKit
EventKit
SystemConfiguration
CORE OS LAYER
Core Bluetooth
Security Services
External Accessory
Accelerate
Local Authentication
HARDWARE

Introduction

- IOS
  - iPhone
    - Résolutions
      - 1st Gen, 3G & 3GS : 320x480
      - 4 & 4S : 640x960
      - 5, 5C & 5S : 640x1136
      - 6 : 750x1334
      - 6 plus : 1242x2208 (downsampled : 1080x1920)

Introduction

- IOS
  - iPad
    - Résolutions
      - iPad Pro 12.9-inch (2em génération) : 2048x2732
      - iPad Pro 10.5-inch : 2224x1668
      - iPad Pro 12.9-inch : 2048x2732
      - iPad Pro (9.7-inch) : 1536x2048
      - iPad Air 2 : 1536x2048
      - iPad Mini 4 : 1536x2048
  - Tableau des résolutions iPhone et iPad

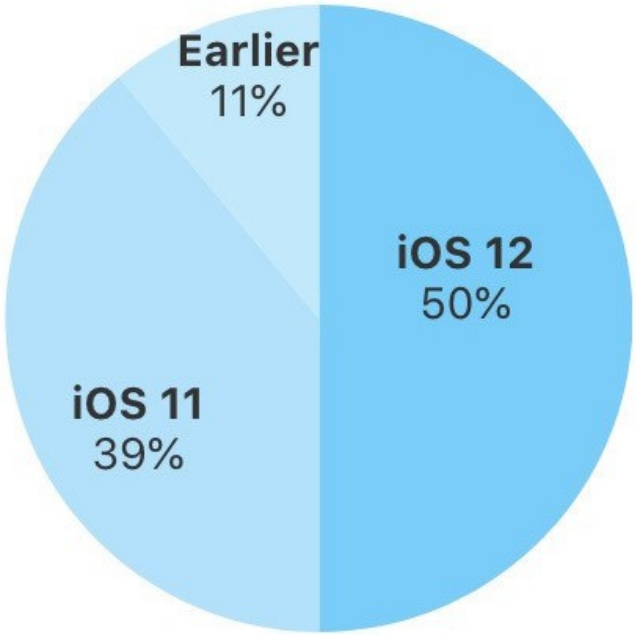
Introduction

- Les versions d’iOS couramment installé sur les iPhone et iPad suivent les dernières versions d’iOS
- C’est un écosystème forçant l’évolution des versions d’OS et des appareils physiques

Introduction

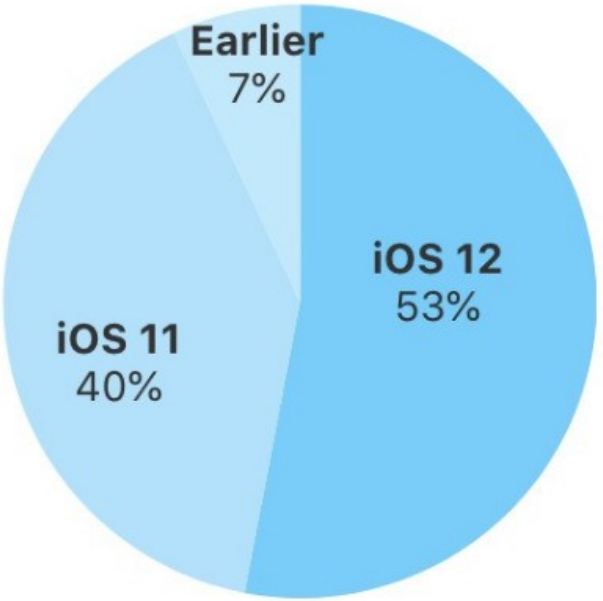
Introduction

50% of all devices are using iOS 12.



As measured by the App Store on October 10, 2018.

53% of devices introduced in the last four years are using iOS 12.



As measured by the App Store on October 10, 2018 based on devices introduced since September 2014.

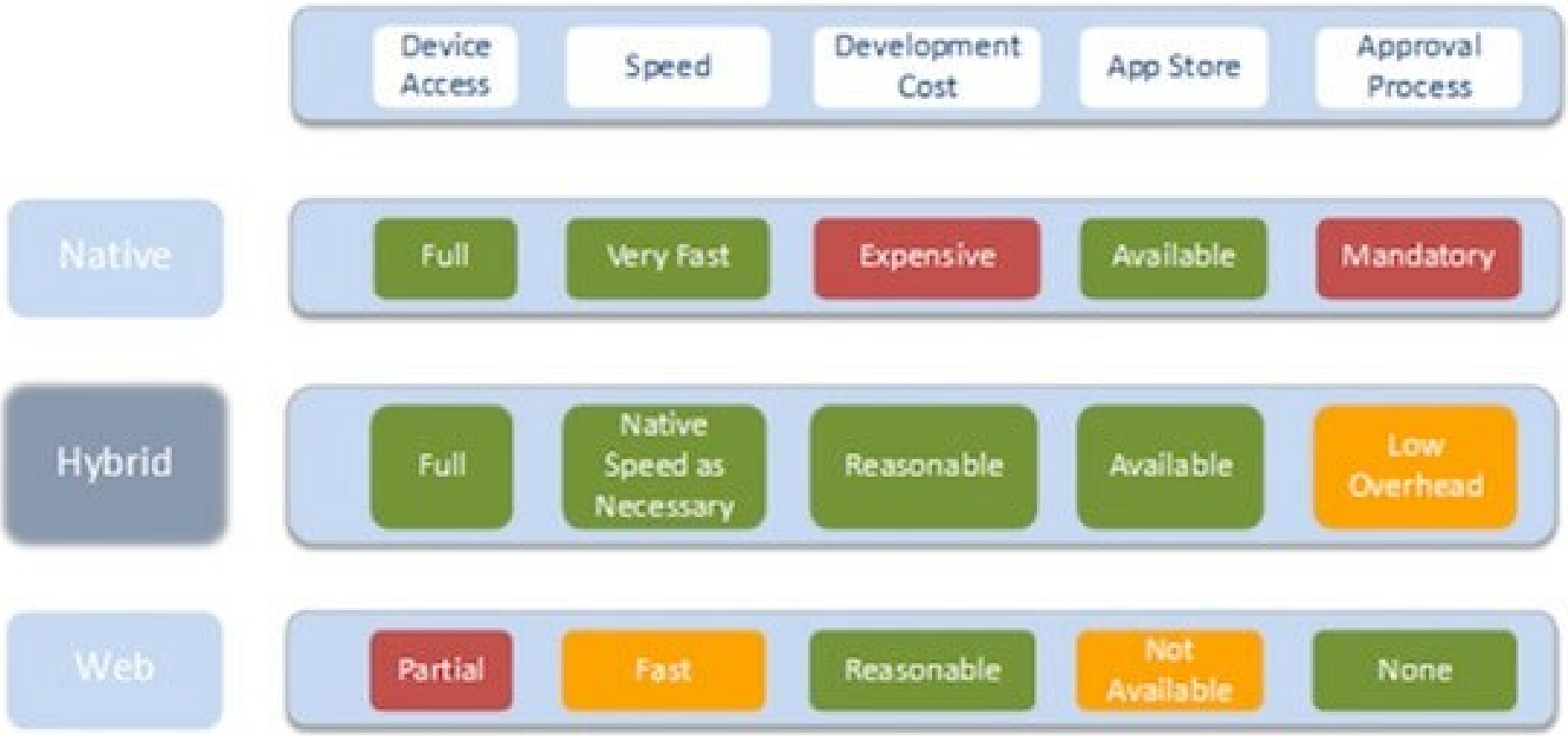
Introduction

- Type d’application
- Le développement d’application mobile est réparti selon trois grandes catégories
  - Les applications « native »
  - Les applications « Web App »
  - Les applications « hybride »

Introduction

- Type d’application
- Le développement d’application mobile est réparti selon trois grandes catégories
  - Les applications « native »
    - Utilise directement le langage de développement et les outils dédiés à la plateforme de développement
  - Les applications « Web App »
    - Une application « Web App » est une application Web intégré à une coquille native permettant de l’afficher
    - La mécanique utilisé historiquement est basé sur les composants WebView, WebBrowser intégré dans les applications natives
    - Se type d’application est aujourd’hui interdite sur les Stores officiel de Google et Apple à cause de l’expérience utilisateur jugé déplorable
    - La réponse à se type d’application est un site web responsive pour mobile
  - Les applications « hybride »
    - Les applications hybrides ont pour concept de base la création d’un seul code permettant la génération de code natif générique pour l’ensemble des plateformes disponibles

Introduction

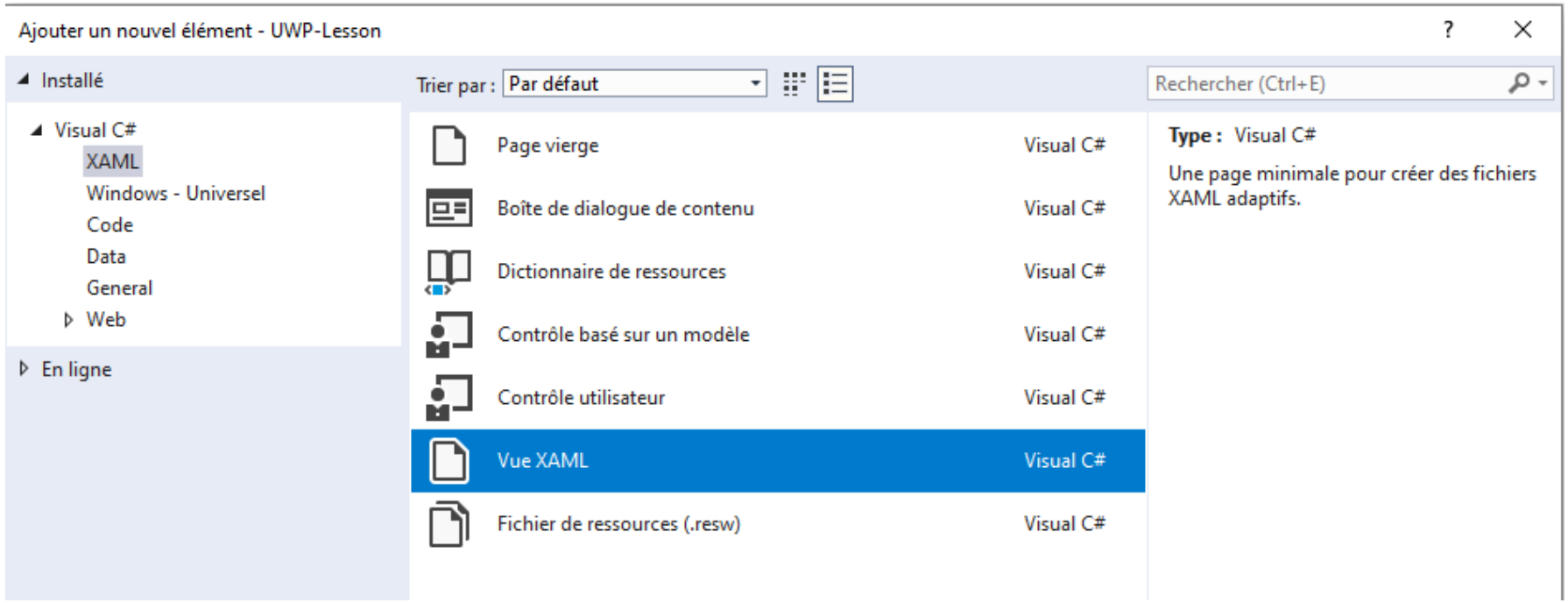


Création de l’interface utilisateur

- Extensible Application Markup Language (XAML)
  - Langage déclaratif
  - Basé sur XML
  - Utilisé pour séparer la déclaration graphique et le code source d’un élément de la couche de présentation
  - Utilisé pour les applications :
    - WPF
    - ASP.NET (avec Silverlight)
    - UWP
  - Standard « .NET » pour la description d’IHM
  - Outils de conceptions : Concepteur XAML de Visual Studio, Blend

Création de l’interface utilisateur

- Création d’un élément XAML (UWP)



Création de l’interface utilisateur

- Création d’un élément XAML (UWP)
- Page vierge :
  - Génération de deux fichiers
  - « BlankPage1.xaml » : représentation du code XAML pour l’affichage utilisateur
  - Hérite de « Page » : modélisé par une balise englobante « <Page> »
  - Les balises XAML sont définis dans l’espace de nom :
    - xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  - Associé au fichier « BlankPage1.xaml.cs » par l’attribut :
    - « x:Class="UWP_Lesson.XamlViews.BlankPage1" »
  - Une page XAML doit posséder un conteneur comme premier élément, dans son corps :
    - « <Grid></Grid> »

Création de l’interface utilisateur

- Création d’un élément XAML (UWP)
- Page vierge : représente un écran de l’application
  - Génération de deux fichiers
  - « BlankPage1.xaml.cs » : Logique de code associé à la vue utilisateur
  - Hérite de « Page »
  - Possède la fonction « InitializeComponent(); » dans sont constructeur vide
  - Est est classe « sealed » et « partial » :
    - « sealed » indique que l’on ne peut pas hériter de la classe courante
    - « partial » indique que la classe possédera un bout de représentation ailleurs qui viendra la compléter
  - Attention : de manière standard le système utilisera uniquement le constructeur vide de la classe pour son initialisation, tout autre constructeur ne sera jamais nativement utilisé

Création de l'interface utilisateur

- Création d'un élément XAML (UWP)
  - Page vierge : représente un écran de l'application

BlankPage1.xaml

```
<Page
   x:Class="UWP_Lesson.XamlViews.BlankPage1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:local="using:UWP_Lesson.XamlViews"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

   <Grid>

   </Grid>
</Page>
```

BlankPage1.xaml.cs

```
namespace UWP_Lesson.XamlViews
{
   public sealed partial class BlankPage1 : Page
   {
      public BlankPage1()
      {
         this.InitializeComponent();
      }
   }
}
```

Création de l'interface utilisateur

- Création d'un élément XAML (UWP)
- Boite de dialogue de contenu : représente une « popup » avec des boutons
  - Génération de deux fichiers
  - « ContentDialog1.xaml » : représentation du code XAML pour l'affichage utilisateur
    - Fonctionne sur le même principe qu'une page dans la description XAML
    - Inclue par défaut deux boutons et un titre
      - Les boutons sont automatiquement lié à la classe de code pour les évènements de clique
      - Le contenu supplémentaire doit être placé dans un conteneur XAML

Création de l'interface utilisateur

- Création d'un élément XAML (UWP)
- Boite de dialogue de contenu : représente une « popup » avec des boutons
  - Génération de deux fichiers
  - « ContentDialog1.xaml.cs » : logique de code associé à la vue utilisateur
    - Fonctionne comme une page
    - Hérite de « ContentDialog »
    - Rajoute la gestion de deux évènements pour les cliques bouton
- Il est possible de ne pas utiliser la mécanique de base pour les évènements des boutons

Création de l’interface utilisateur

- Création d’un élément XAML (UWP)
- Boite de dialogue de contenu : représente une « popup » avec des boutons

ContentDialog1.xaml

```
<ContentDialog
    x:Class="UWP_Lesson.XamlViews.ContentDialog1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Lesson.XamlViews"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Title="TITLE"
    PrimaryButtonText="Button1"
    SecondaryButtonText="Button2"
    PrimaryButtonClick="ContentDialog_PrimaryButtonClick"
    SecondaryButtonClick="ContentDialog_SecondaryButtonClick">

    <Grid>
    </Grid>
</ContentDialog>
```

ContentDialog1.xaml.cs

```
namespace UWP_Lesson.XamlViews
{
    public sealed partial class ContentDialog1 : ContentDialog
    {
        public ContentDialog1()
        {
            this.InitializeComponent();
        }

        private void
ContentDialog_PrimaryButtonClick(ContentDialog sender,
ContentDialogButtonClickEventArgs args)
        {
        }

        private void
ContentDialog_SecondaryButtonClick(ContentDialog sender,
ContentDialogButtonClickEventArgs args)
        {
        }
    }
}
```

Création de l’interface utilisateur

- Création d’un élément XAML (UWP)
- Dictionnaire de ressources : représente des templates graphique réutilisable au travers de l’application
  - Génération d’un fichier
  - « Dictionary1.xaml »
    - Permet de définir des styles pouvant être réutilisé en XAML
    - Peut être importé et utilisé pour un ou des composants XAML

```
<ResourceDictionary
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

   <SolidColorBrush x:Key="brush" Color="Red"/>

</ResourceDictionary>
```

Création de l’interface utilisateur

Implémentation

```
<Page
   x:Class="UWP_Lesson.XamlViews..MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
       <ResourceDictionary>
          <ResourceDictionary.MergedDictionaries>
             <ResourceDictionary Source="Dictionary1.xaml"/>
          </ResourceDictionary.MergedDictionaries>

          <x:String x:Key="greeting">Hello world</x:String>

       </ResourceDictionary>
    </Page.Resources>

    <TextBlock Foreground="{StaticResource brush}" Text="{StaticResource
greeting}" VerticalAlignment="Center"/>
</Page>
```

Création de l'interface utilisateur

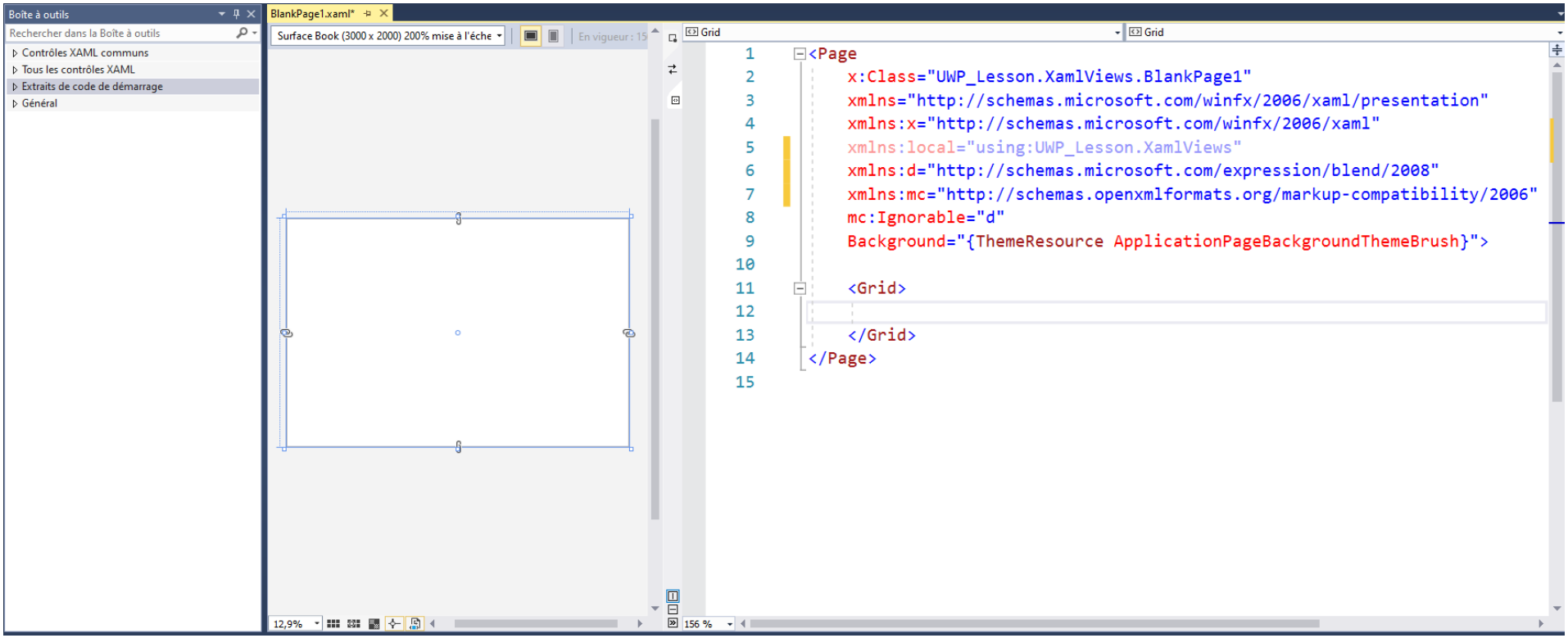
- Création d'un élément XAML (UWP)
- Contrôle Utilisateur : décrit un élément graphique « autonome » pouvant être utilisé dans plusieurs vue XAML
  - Génération de deux fichiers
  - « MyUserControl1.xaml » : représentation du code XAML pour l'affichage utilisateur
    - Fonctionne sur le même principe qu'une page dans la description XAML
  - « MyUserControl1.xaml.cs » : logique de code associé à la vue utilisateur
    - Fonctionne comme une page
    - Hérite de «UserControl»
  - Un Contrôle Utilisateur seul ne peut pas avoir d'existence graphique il doit forcément être contenu dans une page

Création de l’interface utilisateur

- Conception d’IHM en XAML
  - Utilisation du concepteur de vue de Visual Studio
  - Une vue XAML pourra être séparé en deux parties
    - Code XAML
    - Rendu graphique automatique du code XAML
  - Le concepteur de vue embarque une boite à outils contenant les contrôles XAML standard disponibles

Création de l'interface utilisateur

- Conception d'IHM en XAML



Création de l’interface utilisateur

- Conception d’IHM en XAML (UWP)
  - Tout les éléments XAML possède des attributs (directives) de base fournit par
    - « http://schemas.microsoft.com/winfx/2006/xaml »

```
<Grid x:Uid="" x:Name="" x:Load="True" x:FieldModifier="" x:DefaultBindMode="TwoWay"> </Grid>
```

  - « x:Uid » : nom unique de l’élément graphique dans la page toutes inclusions comprises
  - « x:Name » : nom unique de l’élément graphique dans la page selon le namespace utilisé
  - « x:Load » : peut prendre les valeur « True » ou « False » permet d’indiquer si l ‘élément doit être chargé au rendu de la page
  - « x:FieldModifier » : permet d’indiquer la visibilité d’un élément XAML (privé par défaut)
  - « x:DefaultBindMode » : permet d’indiquer le mode de binding à utiliser « OneTime », « OneWay » ou « TwoWay »

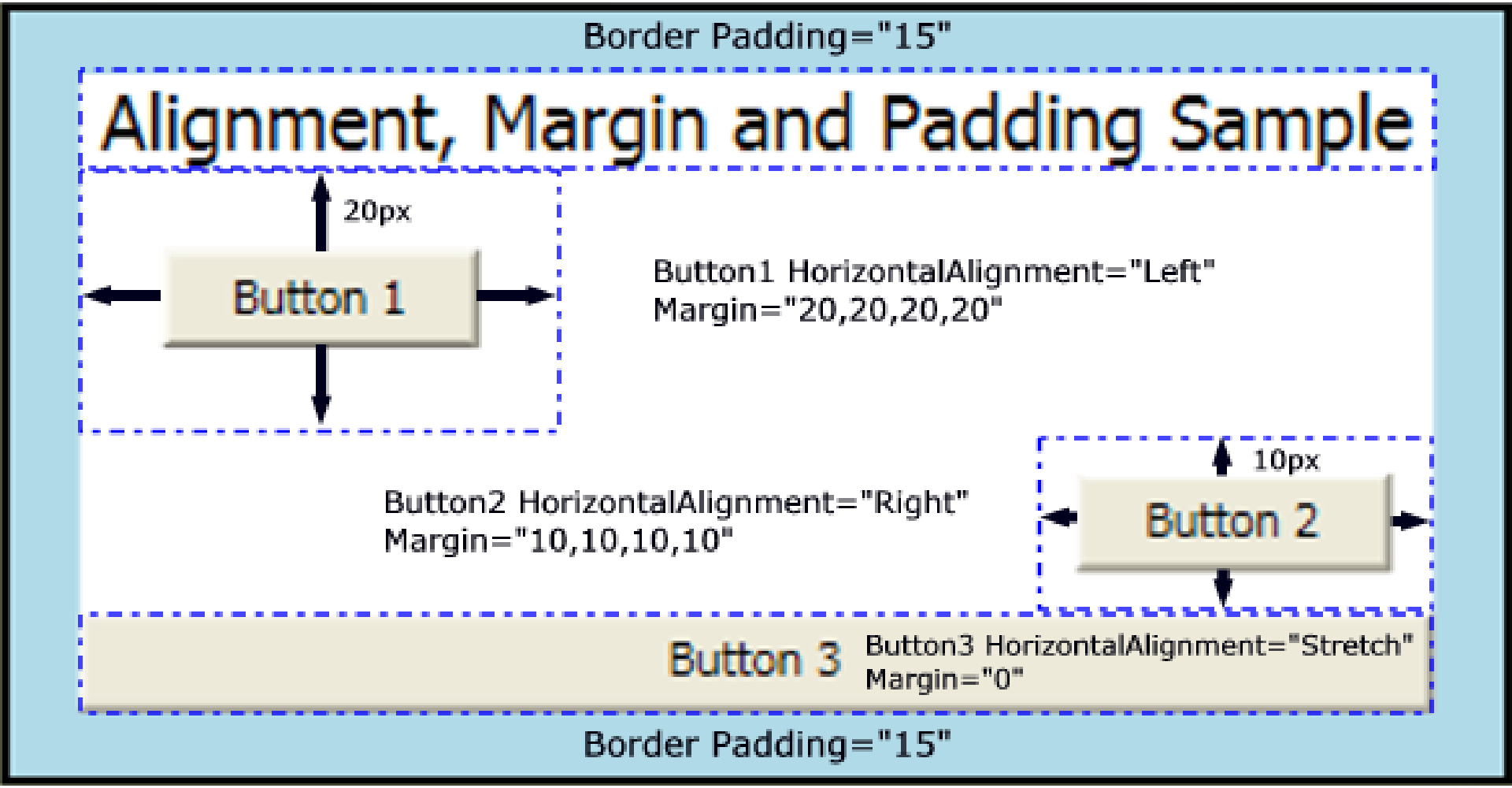
Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Beaucoup d'élément hériteront aussi de « FrameworkElement » et posséderont les propriétés non exhaustives suivantes :

```
<Grid Width="40" Height="40" Name="" VerticalAlignment="Bottom"
HorizontalAlignment="Center" Margin="10,10,10,20" >  </Grid>
```

  - « Width | Height » : définit la taille en px de l'élément (bloque le redimensionnement automatique)
  - « MaxWidth | MinWidth | MaxHeight | MinHeight » définit les tailles maximales et minimales si le redimensionnement automatique est activé
  - « Name » : permet de nommer un élément graphique
  - « VerticalAlignment » : Bottom, Center, Strech, Top
  - « HorizontalAlignment » : Center, Left, Right, Strech
  - « Margin » : permet de définir les marges de l'élément soit de manière uniforme soit en précisant chaque bornes

Création de l'interface utilisateur



Création de l’interface utilisateur

- Conception d’IHM en XAML
- Un fichier XAML doit toujours posséder un conteneur principal dans lequel tout les autres éléments graphiques seront positionnés
  - Il existe plusieurs conteneur de base avec des comportements différent :
  - « Grid » : composant de base
    - Permet la gestion automatique du redimensionnement de sont contenu
    - On peut y définir des « Row » et des « Column »
    - Les tailles sont exprimé en Pixel
    - On peut définir les tailles en ratio de place disponible avec « * »
    - Si aucune taille n’est données on peut indiquer les tailles minimal et maximal

```
<Grid>
   <Grid.RowDefinitions>
      <RowDefinition Height="10"/>
      <RowDefinition MaxHeight="60" MinHeight="40"/>
   </Grid.RowDefinitions>
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="2*"/>
   </Grid.ColumnDefinitions>
</Grid>
```

Création de l’interface utilisateur

- « Grid » : composant de base
- On peut placer les sous éléments de la grille à des coordonnées de cellule

```
<TextBlock Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2" Grid.RowSpan="3"/>
```

- Les « Row » et « Column » commence à 0:0
- Les « RowSpan » et « ColumnSpan » commence à 1
- Attention un élément ne peut pas être positionné en dehors de la grille
- Si plusieurs éléments sont sur la même case alors le dernier élément décrit dans le sens de lecture du fichier sera au premier plan
- Si un élément ne possède aucune contrainte de positionnement il apparaîtra dans le coin haut gauche de la grille

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
- « StackPanel » : empileur d'élément
  - Empile les éléments graphique les un après les autres
  - Peut gérer l'orientation de l'empilement des éléments
- « RelativePanel » : permet de positionner les éléments relativement aux autres
  - On utilise « RelativePanel.xxx » pour positionner les éléments

```
<RelativePanel>
   <TextBlock x:Name="txtB1" RelativePanel.AlignTopWithPanel="True"
      RelativePanel.AlignHorizontalCenterWithPanel="True" >1</TextBlock>
   <TextBlock x:Name="txtB2" RelativePanel.Below="txtB1"
      RelativePanel.AlignHorizontalCenterWith="txtB1">2</TextBlock>
   <TextBlock x:Name="txtB3" RelativePanel.RightOf="txtB2">3</TextBlock>
   <TextBlock x:Name="txtB4" RelativePanel.LeftOf="txtB2"
      RelativePanel.AlignVerticalCenterWith="txtB2">4</TextBlock>
</RelativePanel>
```

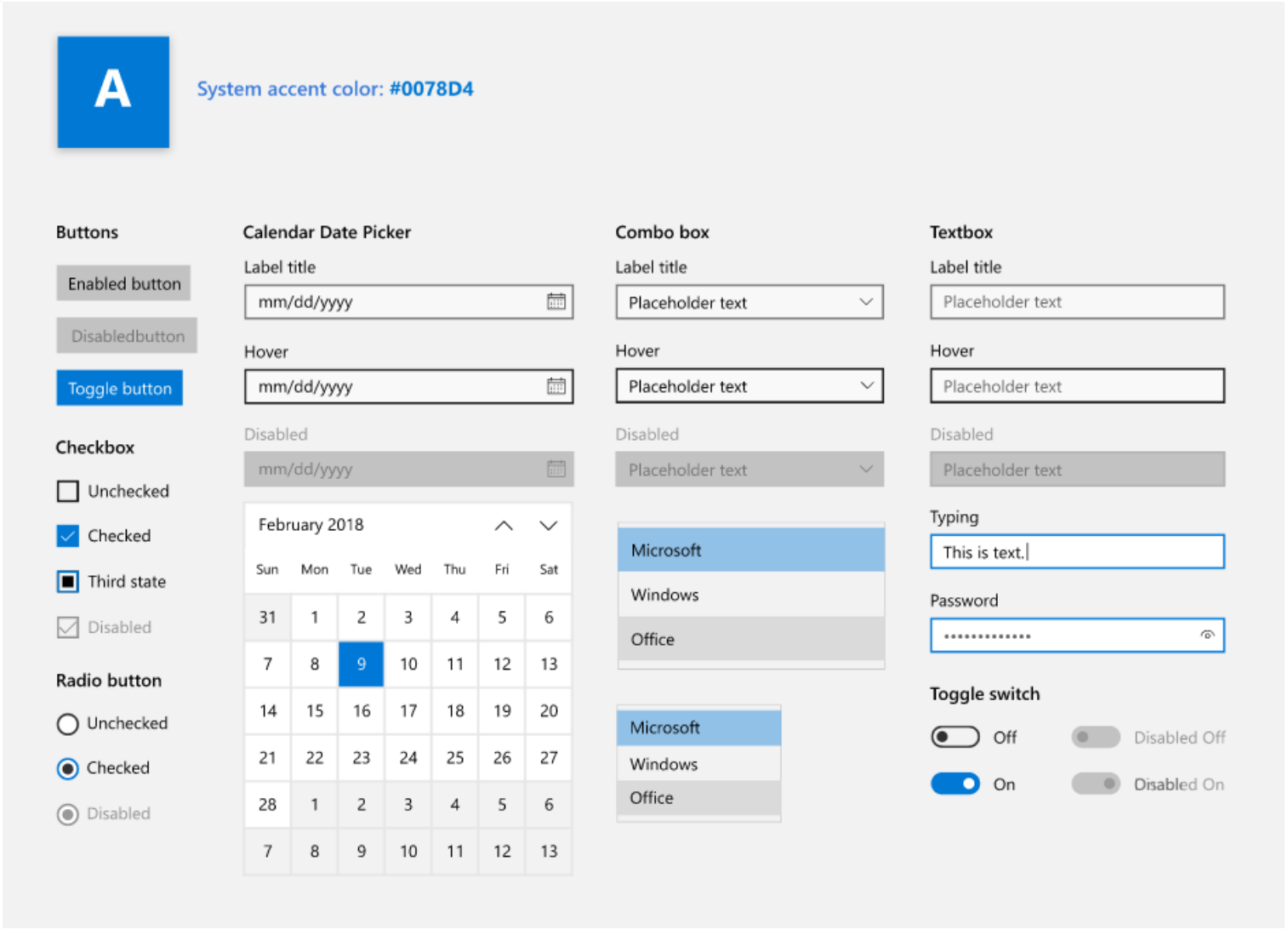
Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
- « WebView » : Conteneur web
  - Possède une propriété « Source » pouvant accueillir :
    - Une « Uri »
    - Un « ms-appdata » : stockage local de l'application
    - Un « ms-appx-web » : stockage dans le package de l'application
  - Une WebView à la capacité de naviguer vers une ressource web
  - Permet un pont JavaScript pour la récupération et l'envoie d'évènement
  - !!! Attention à la faible compatibilité de la WebView avec les standard Web

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Éléments graphiques usuels :
    - « TextBlock » : représente une zone de texte sur une ligne non éditable
    - « TextBox » : représente une zone de texte sur une ligne éditable
    - « RichTextBlock » : représente une zone de texte multi-lignes non éditable
    - « RichEditBox » : représente une zone de texte multi-lignes éditable
    - « PasswordBox » : représente une zone de saisie de mot de passe
  - Attention « TextBlock » et « TextBox » peuvent être rendu multi-ligne
  - Par défaut seul certaine « font-family » sont disponible

Création de l'interface utilisateur



Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Éléments graphiques usuels :
    - « CalendarDatePicker » : permet la sélection d'une date par un calendrier
    - « DatePicker » : permet la sélection d'une date avec « année/mois/jour »
    - « TimePicker » : permet la sélection d'un temps avec « heures/minutes/AM:PM »
    - « CheckBox » : élément indépendant pouvant être dans 3 positions (non cliqué, oui, non)
    - « RadioButton » : élément de sélection d'option dépendantes pouvant être dans 3 positions (non cliqué, oui, non)
    - « ToggleSwitch » : élément indiquant qu'un booléen est vrai ou faux

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Éléments graphiques usuels :
    - « Image » : représente une image chargé depuis une « Source »

```
<Image Source="ms-appx:///Assets/Images/logo.png"/>

<Image Source="http://mysite/logo.png"/>
```

    - « MediaPlayerElement » : permet de jouer vidéo et musique
      - Peut fonctionner en mode streaming

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Manipulation du « Style » d'un « FrameworkElement »
  - Dans un fichier de ressource graphique ou des les ressources d'un conteneur on peut utiliser une balise « Style »
    - Équivalent à une feuille CSS pour la page courante
    - On indique le type d'élément à cibler
    - On définit toutes les propriétés à configurer

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Manipulation du « Style » d'un « FrameworkElement »

```
<Page.Resources>
   <Style TargetType="Button">
      <Setter Property="BorderThickness" Value="5" />
      <Setter Property="Foreground" Value="Black" />
      <Setter Property="BorderBrush" >
         <Setter.Value>
            <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
               <GradientStop Color="Yellow" Offset="0.0" />
               <GradientStop Color="Red" Offset="0.25" />
               <GradientStop Color="Blue" Offset="0.75" />
               <GradientStop Color="LimeGreen" Offset="1.0" />
            </LinearGradientBrush>
         </Setter.Value>
      </Setter>
   </Style>
</Page.Resources>

<StackPanel Orientation="Horizontal">
   <Button Content="Button"/>
   <Button Content="Button"/>
   <Button Content="Button"/>
</StackPanel>
```

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Manipulation du « Style » d'un « FrameworkElement »
    - La balise « Setter » permet d'accéder à l'ensemble des propriétés de l'élément définit dans le « TargetType »
      - C'est la seule façon de définir en XAML des propriétés non disponible comme attribut de l'élément

Création de l'interface utilisateur

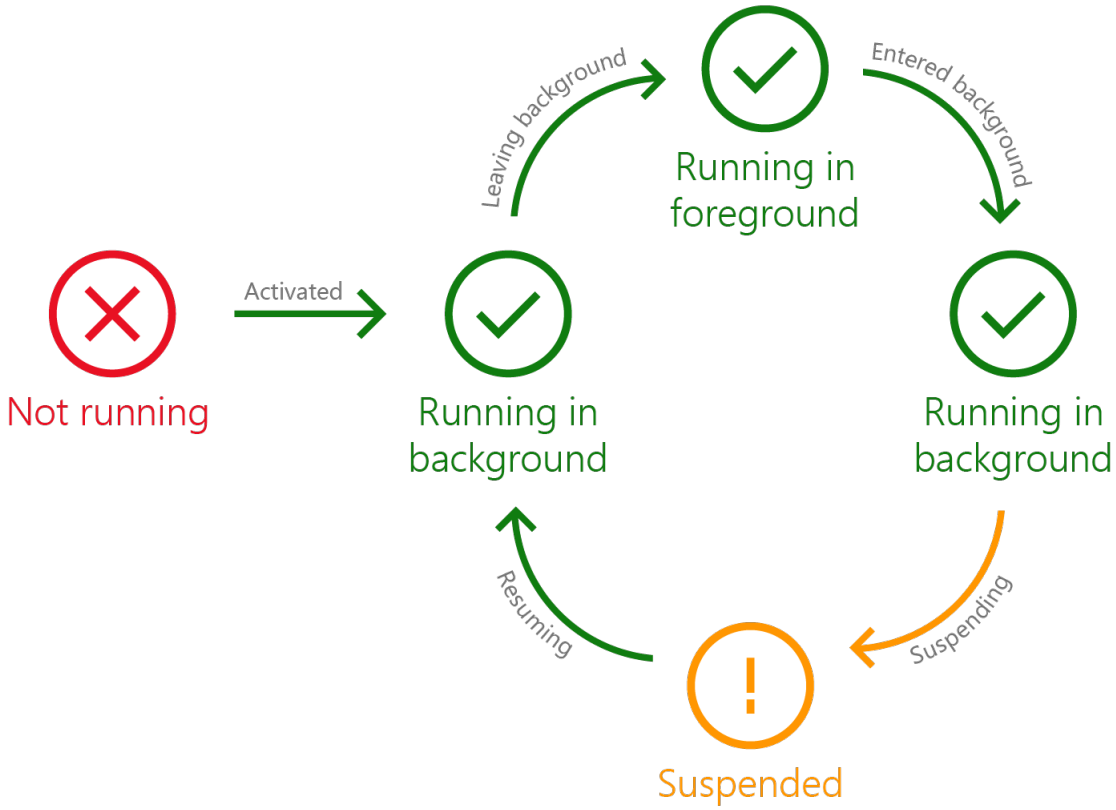
- Conception d'IHM en XAML (UWP)
  - Empilement graphique en UWP
    - Page > Frame > (Border > ScrollViewer > « Non modifiable » Window)
  - Récupération de la « Frame » principale en code avec :

```
Frame rootFrame = Window.Current.Content as Frame;
```

  - Lancement de la première vue depuis le fichier « App.xaml.cs »
    - Représente le point d'entré de l'application
    - Hérite d' « Application »

Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Cycle de vie graphique d'une application UWP



Création de l'interface utilisateur

- Conception d'IHM en XAML (UWP)
  - Cycle de vie d'une page UWP
  - Une page UWP va respecter l'empilement d'évènement suivant :
    - Début du chargement de la page
    - Page chargé
    - Page attaché à la « Frame » parente
    - Mise à jour du contenu graphique
    - Déchargement de la page
    - Si la page courante à lancé une navigation
      - Début de la navigation vers une autre page
      - Fin de la navigation vers une autre page

Création de l’interface utilisateur

```
this.Loading += BlankPage1_Loading;
this.Loaded += BlankPage1_Loaded;
this.OnNavigatedTo();
this.LayoutUpdated += BlankPage1_LayoutUpdated;
this.Unloaded += BlankPage1_Unloaded;
this.OnNavigatingFrom();
this.OnNavigatedFrom();
```

Création de l’interface utilisateur

- Conception d’IHM en XAML
  - Code behind
  - Le code behind est le code C# associé à la vue XAML
  - Attention la classe de vue aura une existence graphique partielle qu’après avoir exécuter la fonction « this.InitializeComponent(); » dans son constructeur
  - Afin d’effectuer des manipulations graphique on se placera dans l’évènement représentant la page entièrement chargé

Création de l'interface utilisateur

- Conception d'IHM en XAML
  - Code behind

```
public BlankPage1()
{
   this.InitializeComponent();

   this.Loaded += BlankPage1_Loaded;
}

private void BlankPage1_Loaded(object sender, RoutedEventArgs e)
{
   //UI code here
}
```

Création de l'interface utilisateur

- Conception d'IHM en XAML
  - Code behind
  - Il est possible d'injecter des éléments graphique directement depuis le code behind à partir du moment ou il existe un conteneur nommé déjà présent dans la vue
  - Lors de l'ajout d'un élément par code behind il est rajouté après les éléments XAML
  - On ajoute ou enlève un élément d'un conteneur en utilisant sa propriété « Children »

```
Button btn = new Button();

//UIElement
btn.Visibility = Visibility.Visible;
btn.Opacity = 0.3;
btn.Rotation = 10;
btn.CanDrag = false;
```

```
//FrameworkElement
btn.Width = 100; btn.Height = 100;
btn.MaxWidth = 200; btn.MaxHeight = 200;
btn.MinWidth = 5; btn.MinHeight = 5;
btn.Name = "myButton";
btn.VerticalAlignment = VerticalAlignment.Center;
btn.HorizontalAlignment = HorizontalAlignment.Stretch;
btn.Margin = new Thickness(5, 10, 15, 20);
```

```
// Button
btn.Content = "click me";

this.mainContent.Children.Add(btn) ;
```

Création de l'interface utilisateur

- Événement et code behind
  - Association d'un événement à une fonction
  - Sur un objet on peut visualiser les différents évènements qui lui sont associés
  - On pourra alors associer l'évènement à une fonction

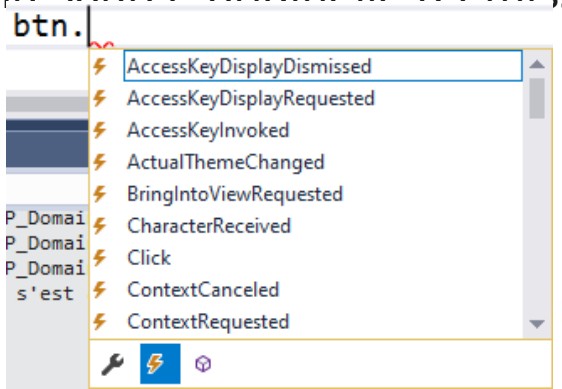
    avec le symbole d'association « += »

    btn.Click += Btn_Click;

  - Le membre de droite prend un délégué
    - Le sender représente l'UIElement ayant déclenché l'action
    - Le « RoutedEventArgs » contient la source originelle d'émission (si contrôle empilé)

```
private void Btn_Click(object sender, RoutedEventArgs e)
{
   // Code here
}

Debug.WriteLine(sender);
Debug.WriteLine(e.OriginalSource);
```



Création de l’interface utilisateur

- Événement et code behind (UWP)
  - « Click » vs « Tapped »
  - Click représente un clique souris, lors d’un clique seul l’élément graphique au plus haut niveau enregistre l’évènement
  - Tapped représente un clique par un pointeur (souris, tactile, …), tout les éléments graphiques empilés reçoive l’évènement (du plus haut au plus bas)
    - Utilise un « TappedRoutedEventArgs »
    - Contient les même informations qu’un « RoutedEventArgs »
    - Rajoute « RoutedEventArgs » qui indique le type de pointeur ayant cliqué
    - Rajoute « Handled » qui si il est à « True » empêche la propagation du signale de clique (les sous éléments ne seront pas appelés)

Création de l’interface utilisateur

- Événement et code behind (UWP)
  - « Click » vs « Tapped »
  - Attache des évènements
  - Création des délégués associé

```
btn.Click += Btn_Click;
btn.Tapped += Btn_Tapped;
```

```
private void Btn_Click(object sender, RoutedEventArgs e)
{
    Debug.WriteLine("Btn_Click");
    Debug.WriteLine(sender);
    Debug.WriteLine(e.OriginalSource);
}


private void Btn_Tapped(object sender, TappedRoutedEventArgs e)
{
    Debug.WriteLine("Btn_Tapped");
    Debug.WriteLine(e.OriginalSource);
    Debug.WriteLine(e.PointerDeviceType);
    e.Handled = true;
    Debug.WriteLine(e.Handled);
}
```

Création de l’interface utilisateur

- Événement et code behind (UWP)
  - « Click » vs « Tapped »
  - Avec « Tapped » il est possible de détecter spécifique un type de clique
    - Clique droit

      btn.RightTapped += Btn_RightTapped;

    - Double clique

      btn.DoubleTapped += Btn_DoubleTapped;

Création de l’interface utilisateur

- Événement et code behind
  - Cycle de vie d’un composant et évènement
  - Chaque composant graphique possède un cycle de vie qui lui est propre
    - Chargement : l’élément n’existe pas encore graphiquement
    - Chargé : l’élément existe graphiquement
    - Mise à jour : l’élément a été modifié
    - Déchargé : l’élément n’existe plus

```
// Manipulation du cycle de vie de l'objet
btn.Loading += Btn_Loading;
btn.Loaded += Btn_Loaded;
btn.LayoutUpdated += Btn_LayoutUpdated;
btn.Unloaded += Btn_Unloaded;
```

Création de l’interface utilisateur

- Événement et code behind
  - Manipulation graphique et évènement
  - « Pointer » : représente les actions d’un pointeur vis à vis de l’objet courant

```
// Manipulation du pointeur de l'objet
btn.PointerEntered += Btn_PointerEntered;
btn.PointerExited += Btn_PointerExited;
btn.PointerMoved += Btn_PointerMoved;
btn.PointerPressed += Btn_PointerPressed;
btn.PointerReleased += Btn_PointerReleased;
btn.PointerWheelChanged += Btn_PointerWheelChanged;
```

  - « Focus » : représente le contexte courant de sélection
    - Un élément peut prendre ou perdre le focus
    - C’est l’élément courant qu’on est entrain de manipuler

```
// Manipulation du focus de l'objet
btn.FocusEngaged += Btn_FocusEngaged;
btn.FocusDisengaged += Btn_FocusDisengaged;
btn.GettingFocus += Btn_GettingFocus;
btn.GotFocus += Btn_GotFocus;
btn.LosingFocus += Btn_LosingFocus;
btn.LostFocus += Btn_LostFocus;
```

Styles et Ressources

- Réutilisation de style sur plusieurs contrôles
  - La réutilisation de style passe par l'utilisation de « ResourceDictionary »
  - On cré un fichier XAML de type « ResourceDictionary »
  - On y définit les « Style » souhaité

```
<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Lesson.Resources">

    <Style x:Key="DefaultTextBlockStyle" TargetType="TextBlock">
        <Setter Property="FontFamily" Value="Arial"/>
        <Setter Property="FlowDirection" Value="RightToLeft"/>
    </Style>
</ResourceDictionary>
```

Styles et Ressources

- Réutilisation de style sur plusieurs contrôles
  - On inclus le style dans la « Page » ou le conteneur souhaité
    - Attention la propriété « Source » doit être exprimé comme étant un chemin relatif au fichier courant

```
<Page.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="../Resources/Dictionary1.xaml"/>
    </ResourceDictionary.MergedDictionaries>
    <Style TargetType="TextBlock" BasedOn="{StaticResource DefaultTextBlockStyle}"/>
  </ResourceDictionary>
</Page.Resources>
```

- Les éléments peuvent ensuite se voir appliquer un style avec :

```
<TextBlock Style="{StaticResource DefaultTextBlockStyle}">Hello world</TextBlock>
```

Styles et Ressources

- Réutilisation de style sur plusieurs contrôles
  - On inclus le style dans la « Page » ou le conteneur souhaité
    - Pour appliquer le style à tout les éléments de la catégorie il suffit de faire hériter le style directement dans un nouveau style dans la page

```
<Page.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="../Resources/Dictionary1.xaml"/>
    </ResourceDictionary.MergedDictionaries>
    <Style TargetType="TextBlock" BasedOn="{StaticResource DefaultTextBlockStyle}"/>
  </ResourceDictionary>
</Page.Resources>
```

    - Les styles peuvent être surchargé directement dans la page ou les conteneur présent dans la page

Styles et Ressources

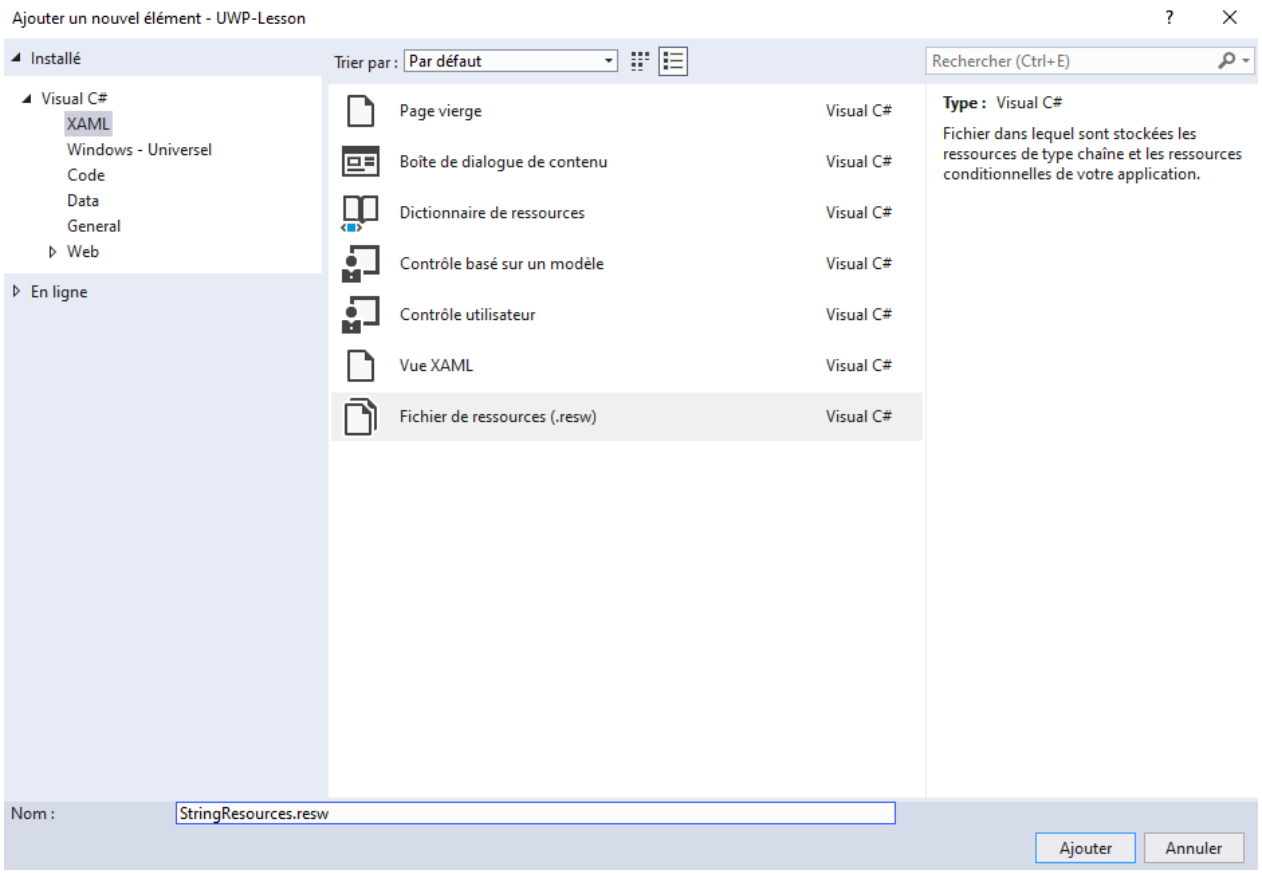
- Réutilisation de style sur plusieurs contrôles (UWP)
  - On peut enregistrer un dictionnaire de manière globale dans « App.xaml »
    - Si on fait un override du style il s'appliquera directement partout

```
<Application
    x:Class="UWP_Lesson.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Lesson">

    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="/Resources/Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>
            <Style TargetType="TextBlock" BasedOn="{StaticResource DefaultTextBlockStyle}"/>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

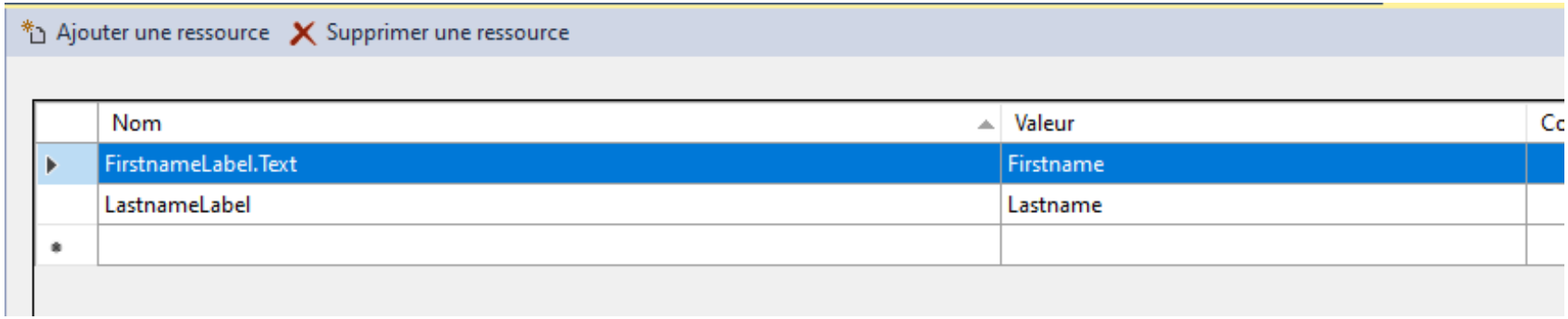
Styles et Ressources

- Partage de ressources
  - On peut créer un fichier de ressource de « String »



Styles et Ressources

- Partage de ressources (UWP)
  - On crée alors les strings souhaité
  - On peut avoir un empilement objet avec « . »



  - On exploite en code behind la ressource en la chargeant
    - Attention si la ressource est en mode objet il faut remplacer « . » par « / »

```
var resources = new Windows.ApplicationModel.Resources.ResourceLoader("StringResources");
this.FirstnameLabel.Text = resources.GetString("FirstnameLabel/Text");
```

Styles et Ressources

- Partage de ressources
  - Utilisation de ressources statique à la page
    - Définition par « x:TypeAUtiliser »
    - Nommage par « x:Key »

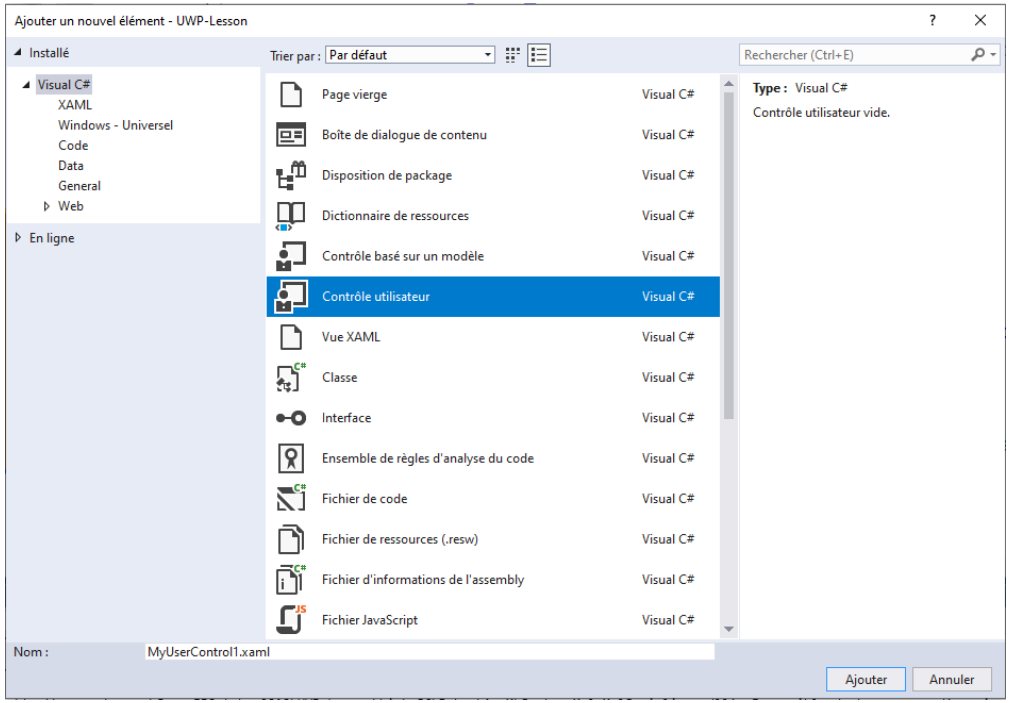
```
<Page.Resources>
   <x:String x:Key="LocalResource">Lastname</x:String>
</Page.Resources>
```

  - Utilisation avec « StaticResource »

```
<TextBlock x:Name="LastnameLabel" Text="{StaticResource LocalResource}"></TextBlock>
```

Styles et Ressources

- Contrôles personnalisés
  - Création de « UserControl » personnalisé
  - L'objectif est de créer un composant par UserControl
  - Un composant peut contenir plusieurs éléments graphiques
  - Il peut y avoir une logique applicative dans le composant



Styles et Ressources

- Contrôles personnalisés
  - On peut définir tout l'aspect graphique XAML pour embarquer le style par défaut du composant

```
<UserControl
   x:Class="UWP_Lesson.CustomControls.CustomTextBox"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <UserControl.Resources>
      <Storyboard x:Name="ColorBorderStoryBoard">
         <DoubleAnimation
            Storyboard.TargetName="TextBox"
            Storyboard.TargetProperty="Opacity"
               From="0.0" To="1.0" Duration="0:0:2"/>
         <ColorAnimation Duration="0:0:1"
                  To="Red"
                  Storyboard.TargetProperty="(Control.Background).(SolidColorBrush.Color)"
                  Storyboard.TargetName="TextBox" />
      </Storyboard>
   </UserControl.Resources>
  <TextBox x:Name="TextBox" x:FieldModifier="public" Loaded="TextBox_Loaded"/>
</UserControl>
```

Styles et Ressources

- Contrôles personnalisés
  - On pourra ajouter un style par défaut

```
<Style TargetType="TextBox">
    <Setter Property="Background" Value="Chocolate"/>
    <Setter Property="Foreground" Value="Wheat"/>
    <Setter Property="Text" Value="Chocolate"/>
    <Setter Property="TextAlignment" Value="Center"/>
    <Setter Property="TextWrapping" Value="WrapWholeWords"/>
    <Setter Property="HorizontalContentAlignment" Value="Center"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
    <Setter Property="BorderThickness" Value="20"/>
    <Setter Property="BorderBrush" >
       <Setter.Value>
          <LinearGradientBrush StartPoint="0.8,0" EndPoint="0.5,1">
             <GradientStop Color="Yellow" Offset="0.0" />
             <GradientStop Color="Red" Offset="0.25" />
             <GradientStop Color="Blue" Offset="0.75" />
             <GradientStop Color="LimeGreen" Offset="1.0" />
          </LinearGradientBrush>
       </Setter.Value>
    </Setter>
    <Setter Property="TextWrapping" Value="WrapWholeWords"/>
 </Style>
```

Styles et Ressources

- Contrôles personnalisés
  - Dans le code behind on définira l'ensemble des actions de contrôles nécessaires au composant

```
namespace UWP_Lesson.CustomControls
{
   public sealed partial class CustomTextBox : UserControl
   {
      public CustomTextBox()
      {
         this.InitializeComponent();
         this.TextBox.TextChanged += TextBox_TextChanged;
         // TODO add controls events
      }

      private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
      {
         // TODO controls
      }

      private void TextBox_Loaded(object sender, RoutedEventArgs e)
      {
         this.ColorBorderStoryBoard.Begin();
      }
   }
}
```

XAML Liste et objets

- « Combobox » : sélecteur d'élément
  - Un combobox est un menu déroulant permettant de sélectionner 0, 1 ou plusieurs éléments
  - Il peut être décrit en XAML ou en code behind
  - Un combobox possède
    - Un « Header » : texte a afficher au dessus du combobox
    - Un « PlaceholderText » : qui affiche un texte non sélectionnable directement dans le combobox

```
<ComboBox PlaceholderText="Select it" Header="Selecteur" x:Name="combo1">
</ComboBox>
```

XAML Liste et objets

- « Combobox » : sélecteur d'élément
  - On peut définir les éléments qui le compose directement en XAML
    - Les éléments XAML peuvent être de différent type

```
<ComboBox PlaceholderText="Select it" Header="Selecteur" x:Name="combo1">
   <ComboBoxItem>
      <StackPanel Orientation="Horizontal">
         <TextBlock>Hey</TextBlock>
         <TextBox PlaceholderText="Hoooooo"></TextBox>
      </StackPanel>
   </ComboBoxItem>
   <ComboBoxItem>
      <StackPanel Orientation="Horizontal">
         <TextBlock>Plop</TextBlock>
         <TextBox PlaceholderText="Hiiiiii"></TextBox>
      </StackPanel>
   </ComboBoxItem>
</ComboBox>
```

XAML Liste et objets

- « Combobox » : sélecteur d'élément
  - Si on déclare à la fois par XAML et en code behind les éléments qui compose le combobox alors se sont les éléments du code behind qui remplaceront les éléments XAML

    this.combo1.ItemsSource = new List<String> { "hi", "ho", "he" };

  - On peut se binder à l'évènement de changement de la sélection en code behind afin d'effectuer des traitements

    this.combo1.SelectionChanged += Combo1_SelectionChanged;

    - On utilisera un « SelectionChangedEventArgs » qui permet de représenter les éléments
      - Ajoutés
      - Supprimés
    - Pour connaître l'élément couramment sélectionné en mono sélection on utilisera les propriétés « SelectedItem » et « SelectedValue »

XAML Liste et objets

- « Combobox » : sélecteur d'élément (UWP)

```
    private void Combo1_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        Debug.WriteLine(sender);
        Debug.WriteLine(e.OriginalSource);
        Debug.WriteLine(e.AddedItems);
        Debug.WriteLine(e.RemovedItems);

        Debug.WriteLine(this.combo1.SelectedItem);
        Debug.WriteLine(this.combo1.SelectedValue);
    }
```

Data binding

- Le « Data Binding » ou liaison de données permet de :
  - Synchroniser les données du code behind vers la vue
  - Synchroniser les données de la vue vers le code behind
- Le Data Binding est lié au « DataContext » du composant
  - Le composant graphique doit posséder un contexte de données
    - La propriété « DataContext » du composant doit être définit par une classe
  - Seul les propriétés peuvent être binder au contexte de données

XAML

```
<StackPanel>
   <TextBlock Text="{Binding TextData}"/>
</StackPanel>
```

Code Behind

```
public String TextData { get; set; }
public BlankPage2()
{
   this.InitializeComponent();
   this.TextData = "Code Behind TEXT";
   this.DataContext = this;
}
```

Data binding

- Dans le XAML
  - On bind une donnée sur une propriété de l'objet XAML
  - Les types de données doivent correspondre
  - Le binding se fait par le nom de la propriété présente dans le DataContext
  - On peut aussi utiliser « x:Bind » pour les manipulations simple (plus performant que « Binding ») lorsque la propriété est présente dans le code-behind du composant
- Dans le code behind
  - Il faut définir le « DataContext » de l'objet qui doit faire le binding
    - On peut utiliser juste une valeur ou une classe possédant des propriétés

Data binding

- Il existe 3 modes de binding
  - « OneTime » : Chargement des données au chargement du contexte
  - « OneWay » : Mise à jour de l'IHM quand la source de donnée est mise à jour
  - « TwoWay » : Mise à jour bi-directionnelle

Data binding

- Soit le XAML suivant

```
<StackPanel>
   <TextBox PlaceholderText="firstname" Text="{x:Bind Firstname, Mode=TwoWay}"/>
   <TextBox PlaceholderText="lastname" Text="{x:Bind Lastname, Mode=TwoWay}"/>
   <Button Click="Button_Click" Height="60" Width="120">Validate</Button>
</StackPanel>
```

- Le code behind suivant

```
public String Firstname { get; set; }
public String Lastname { get; set; }
public BlankPage2()
{
   this.InitializeComponent();
   this.Firstname = "";
   this.Lastname = "";
   this.DataContext = this;
}
private void Button_Click(object sender, RoutedEventArgs e)
{
   Debug.WriteLine(Firstname);
   Debug.WriteLine(Lastname);
}
```

Data binding

- Afin de monter un data-binding performant il faut que les propriétés utilisées utilise une implémentation de « INotifyPropertyChanged »
- La fonction « OnPropertyChanged » doit être utilisé sur toutes les propriétés participant au data-binding

```
public class EntityBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string name)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

Data binding

- « OnPropertyChanged » permet de notifier le système qu'un élément voit sa valeur être mise à jour

```
public int Id
{
    get { return id;}
    set
    {
        id = value;
        OnPropertyChanged("Id");
    }
}
```

XAML Liste & objets, Data Binding

- « Combobox » : sélecteur d'élément
  - On peut définir un combobox avec le DataBinding et un contexte d'affichage de données
  - On utilisera en XAML la propriété « ItemTemplate » de la combobox et la propriété « DataTemplate » pour définir le conteneur de données à utiliser

```
<ComboBox PlaceholderText="Select it binding">
   <ComboBox.ItemTemplate>
      <DataTemplate>
         <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Info}"/>
            <TextBox PlaceholderText="{Binding PlaceHolder}" Text="{Binding Data}"/>
         </StackPanel>
      </DataTemplate>
   </ComboBox.ItemTemplate>
</ComboBox>
```

Implémenter la navigation

- Navigation Frame à Frame
- Navigation vers la première page
  - Le fichier « App.xaml.cs » contient dans l'évènement « OnLaunched » le chargement de la première Frame de l'application
  - On remplacera dans l'exemple « BlankPage1 » par la page souhaité
  - Il faut faire attention au chargement de la page on peut être dans 2 modes
    - Application non démarré
    - Application relancé / réactivé

Implémenter la navigation

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
    {
        rootFrame = new Frame();
        rootFrame.NavigationFailed += OnNavigationFailed;
        Window.Current.Content = rootFrame;
    }
    if (e.PrelaunchActivated == false)
    {
        if (rootFrame.Content == null)
        {
            rootFrame.Navigate(typeof(BlankPage1), e.Arguments);
        }
        Window.Current.Activate();
    } }
```

Implémenter la navigation

- Navigation Frame à Frame
- Navigation vers une nouvelle page
  - Il suffit de récupérer la Frame courante avec
    - « (Window.Current.Content as Frame) »
  - On utilise la fonction « Navigate() » afin de déclencher la navigation vers une nouvelle page
    - « (Window.Current.Content as Frame).Navigate(typeof(BlankPage1)) »

Implémenter la navigation

- Navigation Frame à Frame
- Passage de paramètre
  - Envoie de paramètre par la fonction « Navigate() »
    - « Navigate(typeof(BlankPage1), new object()) »
    - Envoie d'un paramètre de type « object »
    - Utilisation de « Dictionary<String,object> » pour stocker plusieurs information
  - Récupération du paramètre après navigation
    - Override de la fonction « OnNavigatedTo(NavigationEventArgs e) »

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    object passedData = e.Parameter;
    base.OnNavigatedTo(e);
}
```

Implémenter la navigation

- Navigation par Pivot (UWP)
- Création du pivot
- Pour intercepter l'évènement de changement de vue on utilisera
  - « this.rootPivot.SelectionChanged += RootPivot_SelectionChanged; »

```
private void RootPivot_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
   PivotItem item = this.rootPivot.SelectedItem as PivotItem;
   if (item != null)
   {
      // TODO send data to other pivot
   }
}
```

Implémenter la navigation

```
<Pivot x:Name="rootPivot" Title="Category Title">
   <PivotItem Header="Section 1" HorizontalAlignment="Center">
      <TextBlock Text="Content of section 1."/>
   </PivotItem>
   <PivotItem Header="Section 2" HorizontalAlignment="Center">
      <TextBlock Text="Content of section 2."/>
   </PivotItem>
   <PivotItem Header="Section 3" HorizontalAlignment="Center">
      <TextBlock Text="Content of section 3."/>
   </PivotItem>
</Pivot>
```

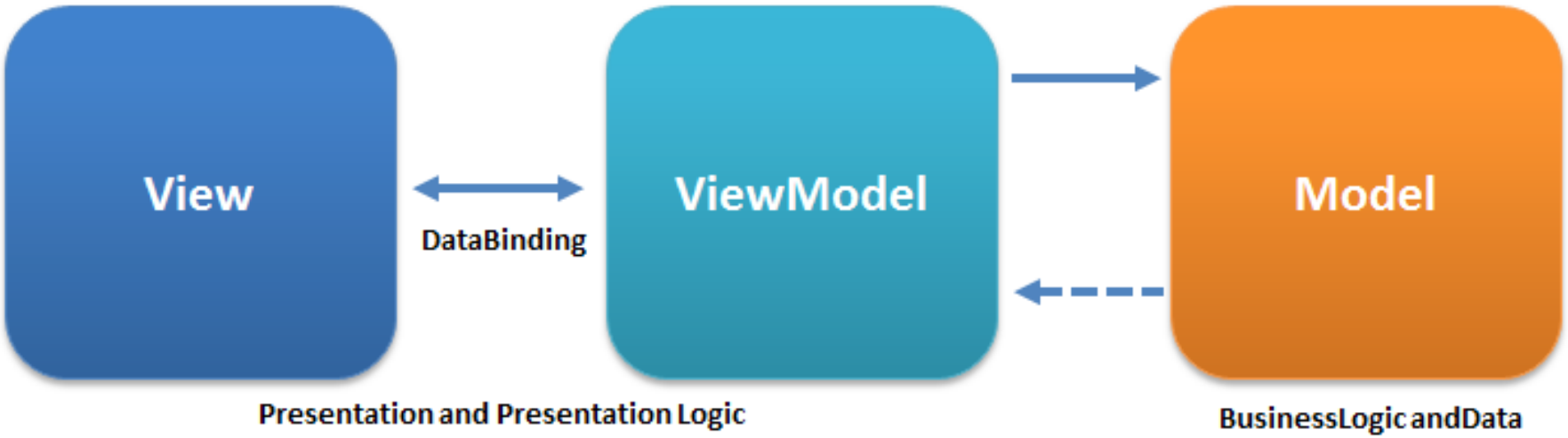
Implémenter la navigation

- Navigation par Pivot (UWP)
  - Les éléments du « Pivot » peuvent être gérer directement dans la « Page » ou le « UserControl » qui le possède
  - Les « PivotItem » peuvent avoir chacun un « DataContext » qui leur est propre
  - On pourrai charger le « DataContext » de l'item courant lors de la navigation vers celui-ci

MVVM et MVVM Light

- Architecture MVVM
- Model :
  - Entité représentant le conteneur de données à utiliser
- View :
  - Couche présentation, contient les éléments d'IHM d'écrit en XAML ou autre, contient le data-binding
- View-Model :
  - Contient la logique de la vue, le code métier
- MVVM est l'architecture de base préconisé pour les applications .NET

MVVM et MVVM Light



MVVM et MVVM Light

- Le MVVM nécessite un contexte de données que l'on pourra fournir grâce à la propriété « DataContext » des différents objets
  - On définira le composant visuel XAML + C#
  - On créer le ViewModel
  - Dans le code-behind de la vue on instancie le ViewModel et on l'associe au DataContext
  - Dans le XAML on mettra en place le data-binding
- Le ViewModel peut ou non connaître le composant pour lequel il sert de DataContext
  - La version la plus propre serai un ViewModel qui n'a aucune connaissance du composant

MVVM et MVVM Light

Page3 ViewModel C#

```
public class Page3ViewModel
   {
      public String Firstname { get; set; }
      public String Lastname { get; set; }
      public String Surname { get; set; }

      public Page3ViewModel(BlankPage3 page3)
      {
         this.Firstname = "john";
         this.Lastname = "Connord";
         this.Surname = "johny";
      }
   }
```

Page3 C#

```
public sealed partial class BlankPage3 : Page
{
   public Page3ViewModel ViewModel { get; set; }

   public BlankPage3()
   {
      this.InitializeComponent();
      this.ViewModel = new Page3ViewModel(this);
      this.DataContext = this.ViewModel;
   }
}
```

Page3 XAML

```
<StackPanel>
   <TextBox x:Name="txtB1" Text="{Binding Firstname}"/>
   <TextBox x:Name="txtB2" Text="{Binding Lastname}"/>
   <TextBox x:Name="txtB3" Text="{Binding Surname}"/>
</StackPanel>
```

MVVM et MVVM Light

- MVVM Light (UWP)
  - Permet une implémentation standardisé de MVVM
  - Paquet NuGet « MvvmLight »
  - Implémentation de base
    - Création d'une page de manière standard
    - Création du ViewModel de la page héritant de « ViewModelBase »
    - Création d'un ViewModelLocator
      - Le ViewModelLocator à pour rôle d'enregistrer l'ensemble des ViewModels qui seront utilisés dans l'application
    - Implémentation du ViewModel pour chaque page par le DataContext en XAML

MVVM et MVVM Light

```
public class ViewModelLocator
{
    /// <summary>
    /// Initializes a new instance of the ViewModelLocator class.
    /// </summary>
    public ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

        //Register your services used here
        SimpleIoc.Default.Register<INavigationService, NavigationService>();
        SimpleIoc.Default.Register<BlankPageViewModel>();
        SimpleIoc.Default.Register<OtherPageViewModel>();
    }

    public BlankPageViewModel BlankPageInstance
    {
        get { return ServiceLocator.Current.GetInstance<BlankPageViewModel>(); }
    }

    public OtherPageViewModel MyProperty
    {
        get { return ServiceLocator.Current.GetInstance<OtherPageViewModel>(); }
    }
}
```

MVVM et MVVM Light

```
public class BlankPageViewModel : ViewModelBase
{
   public String Firstname { get; set; }
   public String Lastname { get; set; }
   public String Surname { get; set; }

   public BlankPageViewModel()
   {
      this.Firstname = "john";
      this.Lastname = "Connord";
      this.Surname = "johny";
   }
}

public class OtherPageViewModel : ViewModelBase
{
   public String Field1 { get; set; }
   public String Field2 { get; set; }
   public String Field3 { get; set; }

   public OtherPageViewModel()
   {
      this.Field1 = "Field1";
      this.Field2 = "Field2";
      this.Field3 = "Field3";
   }
}
```

MVVM et MVVM Light

```
<Page
    x:Class="UWP_Lesson.Views.BlankPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Lesson.Views"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    DataContext="{Binding BlankPageInstance, Source={StaticResource Locator}}"
    >

    <StackPanel>
        <TextBox x:Name="txtB1" Text="{Binding Firstname}"/>
        <TextBox x:Name="txtB2" Text="{Binding Lastname}"/>
        <TextBox x:Name="txtB3" Text="{Binding Surname}"/>
    </StackPanel>
</Page>
```

MVVM et MVVM Light

```
<Page
    x:Class="UWP_Lesson.Views.OtherPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Lesson.Views"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    DataContext="{Binding OtherPageInstance, Source={StaticResource Locator}}"
    >

    <StackPanel>
        <TextBox x:Name="txtB1" Text="{Binding Field1}"/>
        <TextBox x:Name="txtB2" Text="{Binding Field2}"/>
        <TextBox x:Name="txtB3" Text="{Binding Field3}"/>
    </StackPanel>
</Page>
```

MVVM et MVVM Light

```
<Application
    x:Class="UWP_Lesson.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="using:UWP_Lesson.ViewModels"
    xmlns:local="using:UWP_Lesson">

    <Application.Resources>
        <vm:ViewModelLocator xmlns:vm="using:UWP_Lesson.ViewModels"
x:Key="Locator" />
    </Application.Resources>
</Application>
```

MVVM et MVVM Light

- MVVM Light (UWP)
  - Avec cette implémentation il n’y pas de réécriture de DataContext pouvant être ambigu
  - Pour l’ensemble des actions d’évènement (Button par exemple) nous devons réaliser un data-binding en XAML sur la propriété d’évènement qui prendra en paramètre un élément de type « ICommand »

MVVM et MVVM Light

- MVVM Light (UWP)
  - Dans le ViewModel on aura :

```
public ICommand NavigationCommand { get; set; }

public BlankPageViewModel()
{
   NavigationCommand = new RelayCommand() ;
}

public class MyCommand : ICommand
{
   public event EventHandler CanExecuteChanged;

   public bool CanExecute(object parameter)
   {
      throw new NotImplementedException();
   }

   public void Execute(object parameter)
   {
      throw new NotImplementedException();
   }
}
```

MVVM et MVVM Light

- MVVM Light Navigation (UWP)
  - Il faut configurer un objet NavigationService et l'enregistrer dans la configuration de MVVM Light
    - Déclaration dans « ViewModelLocator »

```
SimpleIoc.Default.Register<INavigationService>(() =>
{
    var navigationService = new NavigationService();
    navigationService.Configure("BlankPage", typeof(BlankPage));
    navigationService.Configure("OtherPage", typeof(OtherPage));
    return navigationService;
});
```

MVVM et MVVM Light

- MVVM Light Navigation (UWP)
  - Pour récupérer le NavigationService il faut le récupérer entant que paramètre dans les constructeur de ViewModel souhaités
    - Il ne doit y avoir qu'un seul constructeur dans le ViewModel
    - Tout élément « Register » dans la configuration de MVVM Light est disponible par injection
    - On utilisera la classe « RelayCommand » pour ne pas définir un ICommand manuellement

```
public BlankPageViewModel(INavigationService navigationService)
{
   NavigationCommand = new RelayCommand(() =>
   {
      navigationService.NavigateTo("OtherPage");
   });
}
```

MVVM et MVVM Light

- MVVM Light injection
  - On peut créer n'importe quelle classe issue d'une interface ou non et l'enregistrer dans la configuration de MVVM Light
  - On pourra alors récupérer la ressource par injection dans un constructeur d'un ViewModel
  - Attention l'objet sera créer qu'au premier appel de l'injection
  - Cela permet aussi de n'avoir qu'une seul instance de l'objet
  - Si on souhaite en avoir plusieurs instance il faudra alors préciser une clef pour chaque instance afin de les dissocier

MVVM et MVVM Light

- MVVM Light injection
  - Création du service

```
public class MyService
{
    private int myIncrementalInt;
    public int MyIncrementalInt
    {
        get { return myIncrementalInt; }
    }

    public MyService()
    {
        Task.Factory.StartNew(() =>
        {
            while (true)
            {
                Task.Delay(TimeSpan.FromSeconds(1)).Wait();
                myIncrementalInt++;
            }
        });
    }
}
```

MVVM et MVVM Light

- MVVM Light injection
  - Paramétrage des configuration

```
SimpleIoc.Default.Register<MyService>(() =>
 {
    return new MyService();
 });
```

  - Récupération par le constructeur d'un ViewModel

```
private INavigationService navigationService;
private MyService myService;

public String Field3 { get { return myService.MyIncrementalInt.ToString(); } }

public OtherPageViewModel(INavigationService navigationService, MyService myService)
{
   this.navigationService = navigationService;
   this.myService = myService;
}
```

MVVM et MVVM Light

- MVVM Light Messenger
  - Permet de notifier les éléments héritant de « ViewModelBase »
  - Envoie de notification avec

```
MessengerInstance.Send<NotificationMessage<String>>(
        new NotificationMessage<string>(Field3,"Event from OtherPageViewModel"),"Field3Update");
```

  - Abonnement au résultat avec

```
MessengerInstance.Register<NotificationMessage<String>>(this, "Field3Update", NotifyMe);


public void NotifyMe(NotificationMessage<String> notificationMessage)
{
   string notification = notificationMessage.Notification;
   this.Surname = notificationMessage.Content;
}
```

MVVM et MVVM Light

- MVVM Light Messenger
  - Plusieurs type de message sont disponible
    - NotificationMessage<T>
      - Envoie un message de notification et une valeur de type T
    - NotificationMessageAction
      - Envoie un message de notification et une action
  - Il est possible d'hériter de classe de Message afin de les customiser

MVVM et MVVM Light

- Réception d'évènement de Page dans un ViewModel
  - Création d'une interface pour récupérer les actions souhaité dans les ViewModel

```
public interface INavigationEvent
{
    void OnNavigatedTo(NavigationEventArgs e);
    void OnNavigatedFrom(NavigationEventArgs e);
}
```

  - Dans une Page liée au ViewModel implémentant l'interface on attache les évènement et appel le DataContext qui est notre ViewModel afin de déclencher les actions

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
    (this.DataContext as INavigationEvent).OnNavigatedTo(e);
}
```

MVVM et MVVM Light

- Réception d'évènement de Page dans un ViewModel (UWP)
  - Dans le ViewModel associé il suffit alors que d'implémenter les fonctions

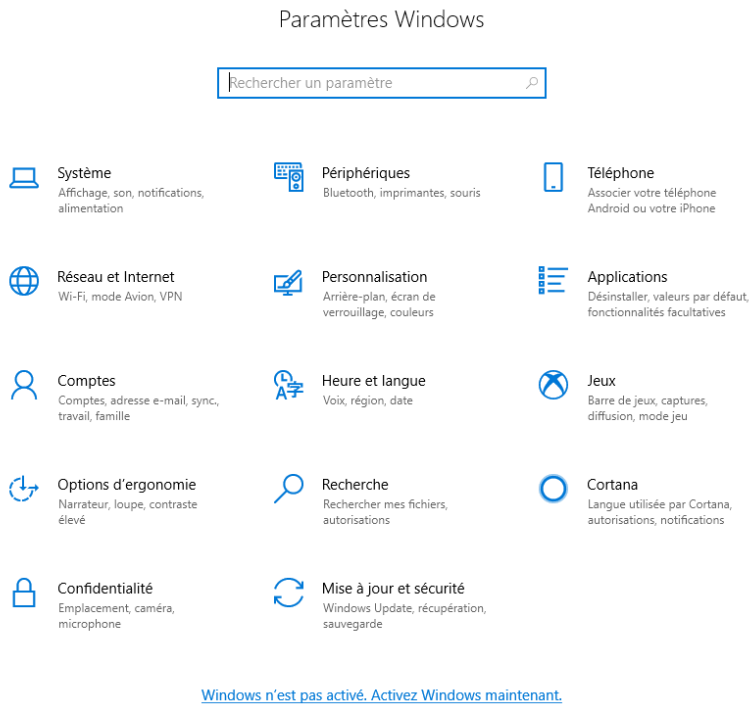
```
public class OtherPageViewModel : INavigationEvent
{
    public OtherPageViewModel()
    {
    }

    public void OnNavigatedTo(NavigationEventArgs e)
    {
        Debug.WriteLine("Enter OtherPage");
    }

    public void OnNavigatedFrom(NavigationEventArgs e)
    {
        Debug.WriteLine("Leave OtherPage");
    }
}
```

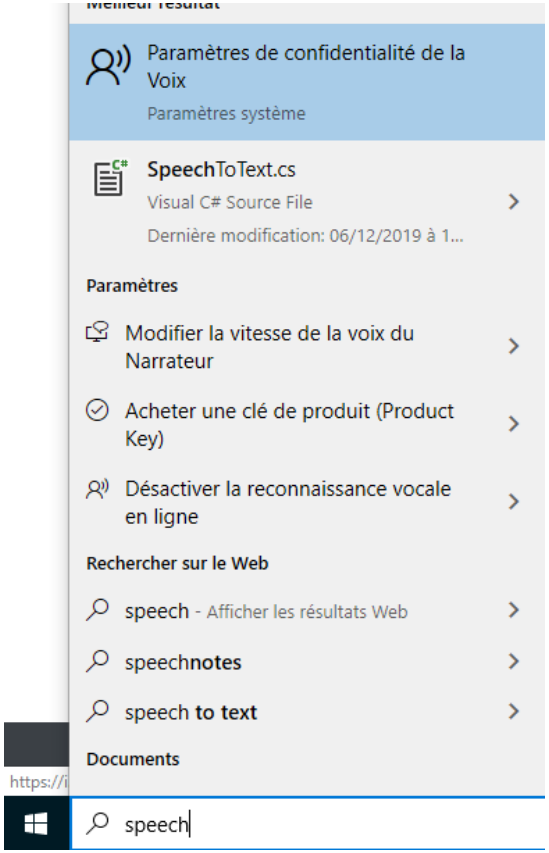
Gestion des périphérique

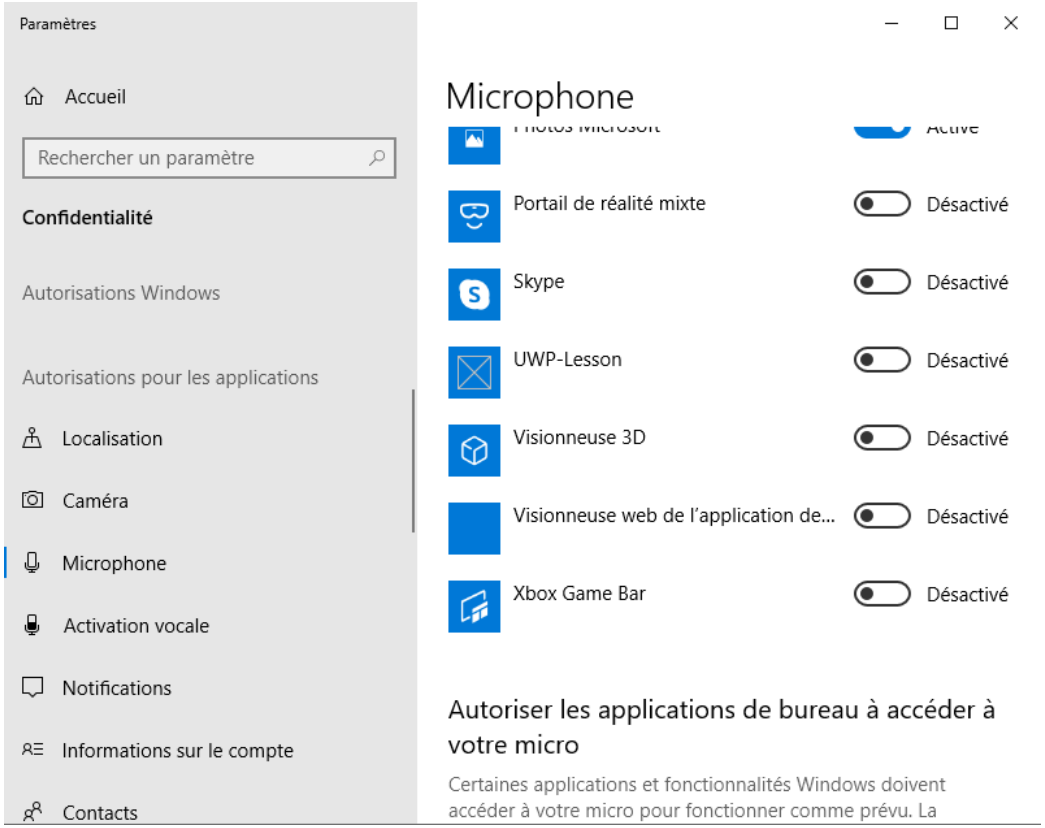
- Cortana
  - Ajout dans le « Manifest » de la capacité « microphone »
  - Configuration de Windows pour la reconnaissance vocale « Heure et langue »



Gestion des périphérique

- Cortana
  - Ajouter dans « Microphone » l'autorisation pour l'application d'utiliser le microphone





Gestion des périphérique

- Cortana
  - Lecture de texte
    - Utilisation d'un « MediaPlayer »

```
public class MediaPlayerManager
{
   private MediaPlayer mediaPlayer;
   private ManualResetEvent mre;
   private String text;
   private String[] slicedText;
   private Int32 indexOfRead;

   public MediaPlayerManager(String toRead)
   {
      indexOfRead = -1;
      text = toRead;
      Slicer();
   }
}
```

Gestion des périphérique

- Cortana
  - Lecture de texte
    - Création d’un slicer

```
public void Slicer(String[] spliters = null)
{
   if (spliters is null)
   {
      spliters = new String[1];
      spliters[0] = " ";
   }
   slicedText = text.Split(spliters, StringSplitOptions.RemoveEmptyEntries);
}
```

Gestion des périphérique

- Cortana
  - Lecture de texte
  - Lecteur de texte slicé
    - Création d’un « SpeechSynthesizer »
    - Lecture de tout les textes slicé « phrase par phrase » avec le MediaPlayer
    - La lecture doit être asynchrone
    - Il faut attendre que la tâche courante s’exécute après la précédente avec le « ManualResetEvent »

Gestion des périphérique

```
public void PlaySlicedText()
{
   Task.Factory.StartNew(() =>
   {
      foreach (var item in this.slicedText)
      {
         mre = new ManualResetEvent(false);
         SpeechSynthesizer reader = new SpeechSynthesizer();
         reader.Voice = SpeechSynthesizer.AllVoices.First(gender => gender.Gender ==
                  VoiceGender.Female);
         SpeechSynthesisStream stream = reader.SynthesizeTextToStreamAsync(item).AsTask().Result;
         Task t = new Task(() =>
         {
            this.mediaPlayer = new MediaPlayer();
            this.mediaPlayer.Source = MediaSource.CreateFromStream(stream, stream.ContentType);
            this.indexOfRead++;
            mediaPlayer.MediaEnded += MediaPlayer_MediaEnded;
            mediaPlayer.Play();
         });
         t.Start();
         mre.WaitOne();
      }
   });
}
```

Gestion des périphérique

- Cortana
  - Écoute de voix
    - Création de « SpeechToText »

```
public class SpeechToText
{
   private SpeechRecognizer speechRecognizer;
   private SpeechToTextEventArgs speechToTextEventArgs;

   private static SpeechToText instance;

   private SpeechToText()
   {
      speechToTextEventArgs = new SpeechToTextEventArgs();
   }

   public static SpeechToText Instance
   {
      get
      {
         if (instance == null)
         {
            instance = new SpeechToText();
         }
         return instance;
      }}}
```

Gestion des périphérique

- Cortana
  - Écoute de voix, création de la fonction de reconnaissance vocale

```
public async void StartRecognization()
{
   OnstartEvent(new EventArgs());
   // Create an instance of SpeechRecognizer.
   speechRecognizer = InitSpeechRecognizer();

   // Listen for audio input issues.
   speechRecognizer.RecognitionQualityDegrading +=
       speechRecognizer_RecognitionQualityDegrading;

           // Compile the constraint.
   await speechRecognizer.CompileConstraintsAsync();

   speechRecognizer.ContinuousRecognitionSession.ResultGenerated +=
                     ContinuousRecognitionSession_ResultGenerated;
   speechRecognizer.ContinuousRecognitionSession.Completed +=
                     ContinuousRecognitionSession_Completed;

   if (speechRecognizer.State == SpeechRecognizerState.Idle)
   {
      await speechRecognizer.ContinuousRecognitionSession.StartAsync();
   }
}
```

Gestion des périphérique

- Cortana
  - Ajout de contrainte

```
// Add a web search grammar to the recognizer.
var webSearchGrammar = new SpeechRecognitionTopicConstraint(
    SpeechRecognitionScenario.WebSearch, "webSearch");

speechRecognizer.UIOptions.AudiblePrompt = "Say what you want to search for...";
speechRecognizer.UIOptions.ExampleText = @"Ex. 'weather for London'";
speechRecognizer.Constraints.Add(webSearchGrammar);
```

Gestion des périphérique

- Cortana
  - Évènement de résultat d’écoute obtenu

```
private void ContinuousRecognitionSession_ResultGenerated(
        SpeechContinuousRecognitionSession sender,
        SpeechContinuousRecognitionResultGeneratedEventArgs args)
{
    if (args.Result.Confidence == SpeechRecognitionConfidence.Medium ||
        args.Result.Confidence == SpeechRecognitionConfidence.High)
    {
        speechToTextEventArgs.SpeechResult = args.Result.Text;
        OnHaveResultEvent(speechToTextEventArgs);
    }
}
```

Gestion des périphérique

- Cortana
  - Création de l’évènement de résultat renvoyé par la classe

```
public event SpeechToTextEventHandler HaveResult;
public delegate void SpeechToTextEventHandler(object sender, SpeechToTextEventArgs e);

protected virtual void OnHaveResultEvent(SpeechToTextEventArgs e)
{
   SpeechToTextEventHandler handler = HaveResult;
   if (handler != null)
   {
      handler(this, e);
   }
}
```

Accès aux données locales

- Le stockage local (UWP)
  - Stockage dans les préférences de l'application
    - Chaque application UWP possède un dictionnaire de données pour l'application courante
    - On peut y écrire des valeurs en mode « Dictionary<String,Object> »

```
private void StoreDefaultAppValues()
{
   ApplicationDataContainer localSettings = ApplicationData.Current.LocalSettings;
   if (!localSettings.Values.Keys.Contains("appName"))
   {
      localSettings.Values["appName"] = "UWP-Lesson";
   }
   else
   {
      Object value = localSettings.Values["appName"];
   }
}
```

Accès aux données locales

- Le stockage local (UWP)
  - Stockage dans les préférences de l'application
    - On peut aussi stocker des données composites

```
private void StoreDefaultAppCompositeValues()
{
   ApplicationDataContainer localSettings = ApplicationData.Current.LocalSettings;
   ApplicationDataCompositeValue composite = new ApplicationDataCompositeValue();
   composite["part1"] = -1;
   composite["part2"] = "test";
   localSettings.Values["compositeVal1"] = composite;

   ApplicationDataCompositeValue compositeRetrieve =
            localSettings.Values["compositeVal1"] as ApplicationDataCompositeValue;

   foreach (KeyValuePair<String,Object> item in compositeRetrieve)
   {
      // TODO
   }
}
```

Accès aux données locales

- Le stockage local (UWP)
  - Stockage fichier
    - Enregistrement avec « LocalFolder »

```
public ICommand SaveFile
{
   get
   {
      return new RelayCommand(async () =>
      {
         StorageFolder localFolder = ApplicationData.Current.LocalFolder;
         StorageFile myFile = await localFolder.CreateFileAsync("myFile.txt",
            CreationCollisionOption.ReplaceExisting);
         await FileIO.WriteTextAsync(myFile, this.Field1 + "." + this.Field2);
      });
   }
}
```

Accès aux données locales

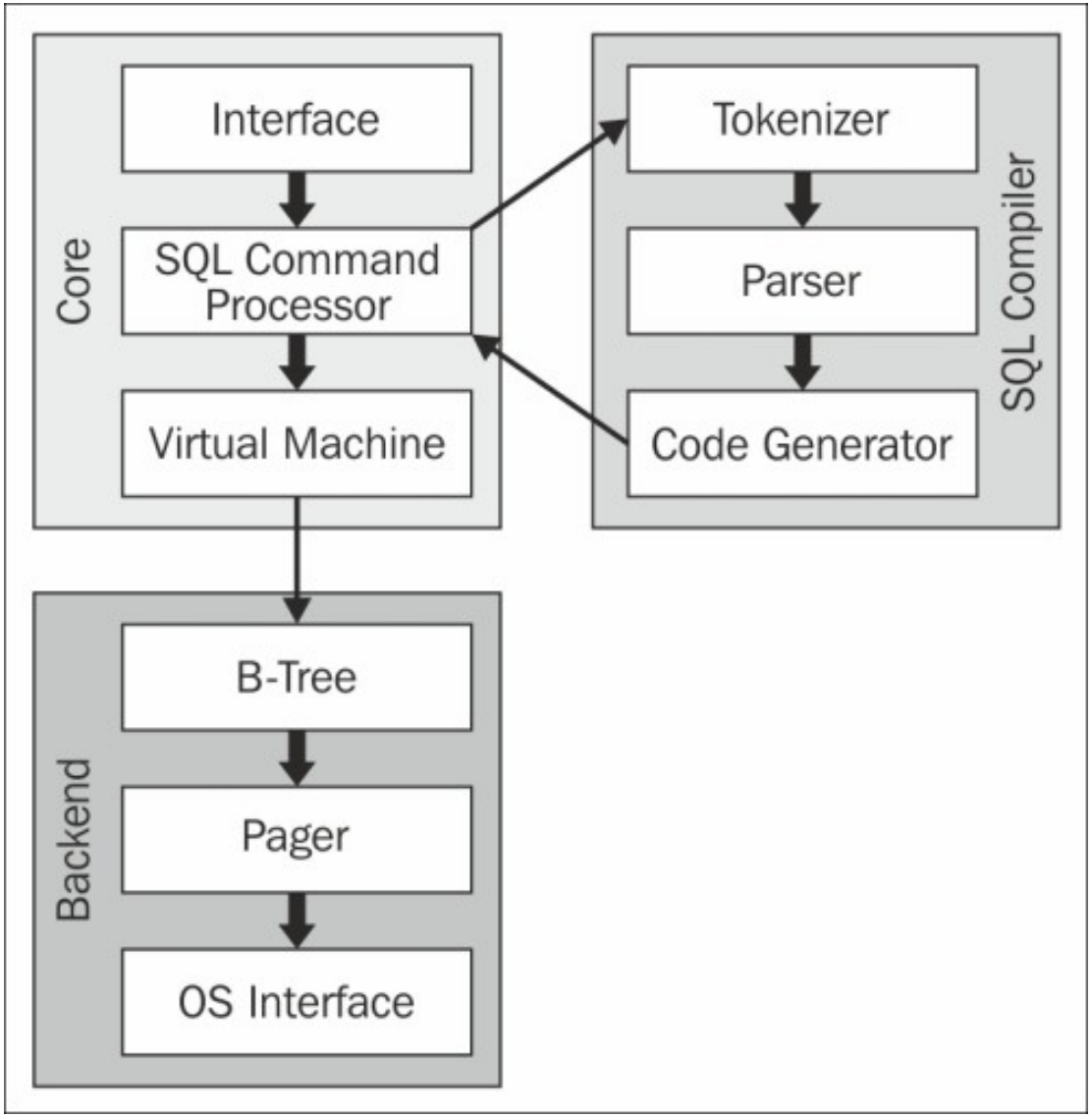
- Le stockage local (UWP)
  - Stockage fichier
    - Chargement avec « LocalFolder »

```
public ICommand LoadFile
{
   get
   {
      return new RelayCommand(async () =>
      {
         StorageFolder localFolder = ApplicationData.Current.LocalFolder;
         StorageFile myFile = await localFolder.GetFileAsync("myFile.txt");
         String data = await FileIO.ReadTextAsync(myFile);
         String[] datas = data.Split('.');
         Field1 = datas[0];
         Field2 = datas[1];
      });
   }
}
```

Accès aux données locales

- SQLite est un SGBD écrit en C
- C’est une base de données qui s’embarque directement dans les applications
- Elle n’a pour but que d’être consommé par une unique application
- Il n’y a pas de gestion de droit dans la base de données, ce sont les droits fichiers qui en régissent l’accès

Accès aux données locales



Accès aux données locales

Example Typenames From The CREATE TABLE Statement or CAST Expression	Resulting Affinity	Rule Used To Determine Affinity
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2

Accès aux données locales

BLOB *no datatype specified*	BLOB	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

Accès aux données locales

- On utilisera des paquets NuGet afin de gérer les données avec SQLite
  - « SQLite » par SQLite Development Team
  - « SQLiteNetExtensions » par TwinCoders

Accès aux données locales

- SQLite tags :
  - [PrimaryKey, AutoIncrement]
  - [Indexed]
  - [Column(String name)]
  - [Ignore]
  - [NotNull]
  - [Table(String name)]
  - [Unique]

Accès aux données locales

- Utilisation pour une table unique
  - Annotation d'une classe C# avec l'ensemble des descriptions souhaité
  - Annotation des propriétés pour les options de colonne
  - Obligation de définir une clef primaire

```
[Table("mytable")]
public class Class1
{
   //private attributes

   [PrimaryKey, AutoIncrement]
   public long Id { get => id; set => id = value; }

   public string Field1 { get => field1; set => field1 = value; }
   public string Field2 { get => field2; set => field2 = value; }
   public string Field3 { get => field3; set => field3 = value; }
}
```

Accès aux données locales

- Enregistrement des données dans la base de données
  - Ouverture et création du fichier « mydb.sqlite » dans le dossier local de l'application, sans l'écraser si il existe déjà
  - Déclaration d'une connexion à la base de données
  - Création de la table basé sur la classe si elle n'existe pas

```
StorageFolder localFolder = ApplicationData.Current.LocalFolder;
StorageFile myDb = await localFolder.CreateFileAsync("mydb.sqlite",
   CreationCollisionOption.OpenIfExists);
using (var db = new SQLiteConnection(myDb.Path))
{
   db.CreateTable<Class1>();

   Class1 c11 = new Class1() { Field1 = "testF11", Field2 = "testF21", Field3 = "TestF31" };
   db.Insert(c11);
}
```

Accès aux données locales

- Récupération des données dans la base de données
  - Ouverture et création du fichier « mydb.sqlite » dans le dossier local de l'application, sans l'écraser si il existe déjà
  - Déclaration d'une connexion à la base de données
  - Création de la table basé sur la classe si elle n'existe pas

```
StorageFolder localFolder = ApplicationData.Current.LocalFolder;
StorageFile myDb = await localFolder.CreateFileAsync("mydb.sqlite",
    CreationCollisionOption.OpenIfExists);
using (var db = new SQLiteConnection(myDb.Path))
{
    db.CreateTable<Class1>();
    Class1s.Clear();
    foreach (Class1 item in db.Table<Class1>().ToList())
    {
        Class1s.Add(item);
    }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Annotation de liaison
    - ManyToMany (N-N)
      - Nécessite la création d'une table d'association
    - OneToMany (N-1)
    - ManyToOne (1-N)
    - OneToOne (1-1)

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Class2A

```
public class Class2A
{
   // Private attributes

   [PrimaryKey, AutoIncrement]
   public int Id { get => id; set => id = value; }

   public DateTime MyDate { get => myDate; set => myDate = value; }
   public string Data { get => data; set => data = value; }

   [ManyToOne]
   public Class2A SubClassA { get => subClassA; set => subClassA = value; }

   [ForeignKey(typeof(Class2A))]
   public int SubClassAId { get; set; }

   [ManyToMany(typeof(Class2AB))]
   public List<Class2B> ListClass2B { get => listClass2B; set => listClass2B = value; }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Class2B

```
public class Class2B
{
   // Private attributes

   [PrimaryKey, AutoIncrement]
   public int Id { get => id; set => id = value; }

   public string Data { get => data; set => data = value; }

   [ManyToOne]
   public Class2A SubClassA { get => subClassA; set => subClassA = value; }

   [ManyToMany(typeof(Class2AB))]
   public List<Class2A> ListClass2A { get => listClass2A; set => listClass2A = value; }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Classe de liaison entre Class2A et Class2B
  - Lors d'un ManyToMany la table d'association doit être représenté et doit contenir entant que clef étrangère les clefs primaires des tables liées

```
public class Class2AB
{
   [ForeignKey(typeof(Class2A))]
   public int IdClass2A { get; set; }

   [ForeignKey(typeof(Class2B))]
   public int IdClass2B { get; set; }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Création d'un service de base de données en créant toute les tables

```
public class DatabaseService
{
   private SQLiteConnection sqliteConnection;

   public DatabaseService()
   {
      Task.Factory.StartNew(async () =>
      {
        StorageFolder localFolder = ApplicationData.Current.LocalFolder;
        StorageFile myDb = await localFolder.CreateFileAsync("mydb.sqlite",
           CreationCollisionOption.OpenIfExists);
        this.sqliteConnection = new SQLiteConnection(myDb.Path);
        this.sqliteConnection.CreateTable<Class1>();

        this.sqliteConnection.CreateTable<Class2AB>();
        this.sqliteConnection.CreateTable<Class2A>();
        this.sqliteConnection.CreateTable<Class2B>();
      });
   }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Création de l'équivalent des DbSet (EF6) pour accéder aux données des tables

```
public TableQuery<Class1> Class1
{
   get { return this.sqliteConnection.Table<Class1>(); }
}

public TableQuery<Class2A> Class2A
{
   get { return this.sqliteConnection.Table<Class2A>(); }
}

public TableQuery<Class2B> Class2B
{
   get { return this.sqliteConnection.Table<Class2B>(); }
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Ajout des fonctions spécifiques de sauvegarde d'objet avec l'ensemble des sous éléments

```
public void SaveWithChildren(Class2A item)
{
   this.Save(item.SubClassA);
   this.sqliteConnection.InsertOrReplaceAllWithChildren(item.ListClass2B);
   this.sqliteConnection.InsertOrReplaceWithChildren(item, true);
}

public void SaveWithChildren(Class2B item)
{
   this.Save(item.SubClassA);
   this.sqliteConnection.InsertOrReplaceAllWithChildren(item.ListClass2A);
   this.sqliteConnection.InsertOrReplaceWithChildren(item);
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Ajout des fonctions spécifiques de récupération des éléments avec tout leurs sous éléments

```
public List<Class2A> Class2AEager
{
   get { return this.sqliteConnection.GetAllWithChildren<Class2A>(); }
}

public List<Class2B> Class2BEager
{
   get { return this.sqliteConnection.GetAllWithChildren<Class2B>(); }
}
```

  - Création de la fonction standard de sauvegarde

```
public int Save(object item)
{
   return this.sqliteConnection.InsertOrReplace(item);
}
```

Accès aux données locales

- Utilisation avec des liaisons entre plusieurs tables
  - Sauvegarde d'un objet Class2A complet

```
Class2A class2A = new Class2A();
class2A.Data = "my data";
class2A.MyDate = DateTime.Now;
Class2A class2ASub = new Class2A();
class2ASub.Data = "sub data";
class2A.SubClassA = class2ASub;
List<Class2B> class2Bs = new List<Class2B>();
class2Bs.Add(new Class2B() { Data = "2B data 1" });
class2Bs.Add(new Class2B() { Data = "2B data 2" });
class2Bs.Add(new Class2B() { Data = "2B data 3" });
class2A.ListClass2B = class2Bs;

this.databaseService.SaveWithChildren(class2A);
```

  - Chargement des éléments Class2A

```
List<Class2A> class2As = this.databaseService.Class2AEager;
```

Accès à des données distantes

- Les services Web API
  - REST (Representational State Transfer)
  - Utilisation de données JSON
  - Définition des verbes HTTP utilisables
    - Sûre : ne modifie pas l'état du serveur
    - Idempotente : jouer plusieurs fois la même requête à le même effet et laisse le serveur dans le même état

Verbe	La requête a un corps	Une réponse de succès a un corps	Sûre	Idempotente	Peut être mise en cache	Autorisée dans les formulaires HTML
GET	Non	Oui	Oui	Oui	Oui	Oui
POST	Oui	Oui	Non	Non	Non	Oui
PUT	Oui	Non	Non	Oui	Non	Non
DELETE	Non	Non	Non	Oui	Non	Non

Accès à des données distantes

- Les services Web API
  - Récupération d'une réponse par l'objet « HttpResponseMessage »
  - Utilisation de la librairie NewtonSoft.JSON pour parser le contenu JSON en un objet C#

```
private async Task<TItem> HandleResponse<TItem>(TItem item, HttpResponseMessage response)
{
    if (response.IsSuccessStatusCode)
    {
        String result = await response.Content.ReadAsStringAsync();
        item = JsonConvert.DeserializeObject<TItem>(result);
    }

    return item;
}
```

Accès à des données distantes

- Les services Web API
  - Communication web par l'objet « HttpClient »
  - Mise en place d'une requête « GET »
  - « url » représente sur le site « UrlDeMonSite » la ressource spécifique à atteindre

```
private async Task<TItem> HttpClientCaller<TItem>(String url, TItem item)
{
   using (HttpClient client = new HttpClient())
   {
      client.BaseAddress = new Uri("UrlDeMonSite");
      client.DefaultRequestHeaders
        .Accept
        .Add(new MediaTypeWithQualityHeaderValue("application/json"));

      HttpResponseMessage response = await client.GetAsync(url);
      item = await HandleResponse(item, response);
   }

   return item;
}
```

Accès à des données distantes

- Les services Web API
  - Mise en place d'une requête « POST »
    - Sérialisation de l'objet C# en JSON et envoie par le body de la requête

```
private async Task<TItem> HttpClientSender<TItem>(String url, TItem item, TItem result)
{
   using (HttpClient client = new HttpClient())
   {
      client.BaseAddress = new Uri("UrlDeMonSite");
      client.DefaultRequestHeaders
        .Accept
        .Add(new MediaTypeWithQualityHeaderValue("application/json"));

      HttpResponseMessage response = await client.PostAsync(url,
         new StringContent(JsonConvert.SerializeObject(item),
         Encoding.UTF8, "application/json"));

      result = await HandleResponse(item, response);
   }

   return result;
}
```

Accès à des données distantes

- Les services Web API
  - Mise en place d'une requête « PUT »

```
private async Task<TItem> HttpClientPuter<TItem>(string url, TItem item, TItem result)
{
   using (HttpClient client = new HttpClient())
   {
      client.BaseAddress = new Uri("UrlDeMonSite");
      client.DefaultRequestHeaders
        .Accept
        .Add(new MediaTypeWithQualityHeaderValue("application/json"));

      HttpResponseMessage response = await client.PutAsync(url,
         new StringContent(JsonConvert.SerializeObject(item),
         Encoding.UTF8, "application/json"));

      result = await HandleResponse(item, response);
   }

   return result;
}
```

Accès à des données distantes

- Les services Web API
  - Mise en place d'une requête « DELETE »

```
private async Task<TResult> HttpClientDeleter<TItem, TResult>(string url, TItem item, TResult result)
{
   using (HttpClient client = new HttpClient())
   {
      client.BaseAddress = new Uri("UrlDeMonSite");
      client.DefaultRequestHeaders
        .Accept
        .Add(new MediaTypeWithQualityHeaderValue("application/json"));

      using (HttpRequestMessage message = new HttpRequestMessage(HttpMethod.Delete, url))
      {
         message.Content = new StringContent(JsonConvert.SerializeObject(item),
         Encoding.UTF8, "application/json");
         HttpResponseMessage response = await client.SendAsync(message);

         result = await HandleResponse(result, response);
      }
   }

   return result;
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Utilisation de différentes route sur l'API
    - GET

Classe C#	Racine	Liste des élément	Un élément par id	Liste de sous élément	Élément par id de sous élément
Post	/posts	/posts	/posts/{id}	/posts/{id}/{liste}	/posts?souselement=1

    - POST => /posts
    - PUT et DELETE => /posts/{id}

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
- Création de l'interface « DbItem »

```
public interface DbItem
{
   int Id { get; set; }
}
```

- Création de la classe « Post »

```
public class Post : DbItem
{
   public int userId { get; set; }
   [JsonProperty("id")]
   public int Id { get; set; }
   public string title { get; set; }
   public string body { get; set; }
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Création de la classe WebServiceManager

```
public class WebServiceManager<T> where T : class, DbItem
{
   public String DataConnectionResource { get; set; }

   public WebServiceManager(String dataConnectionResource)
   {
      this.DataConnectionResource = dataConnectionResource;
   }
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Ajout des fonctions GET à WebServiceManager

```
public async Task<T> Get(Int32 id)
{
   T item = default(T);
   String url = typeof(T).Name.ToLower() + "s" + "/" + id + "/";
   item = await HttpClientCaller<T>(url, item);
   return item;
}

public async Task<List<T>> Get()
{
   List<T> item = default(List<T>);
   String url = typeof(T).Name.ToLower() + "s" + "/";
   item = await HttpClientCaller<List<T>>(url, item);
   return item;
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Ajout des fonctions POST et PUT à WebServiceManager

```
public async Task<T> Post(T item)
{
   T result = default(T);
   String url = typeof(T).Name.ToLower() + "s" + "/";
   result = await HttpClientSender<T>(url, item, result);

   return result;
}

public async Task<T> Put(T item)
{
   T result = default(T);
   String url = typeof(T).Name.ToLower() + "s" + "/" + item.Id + "/";
   result = await HttpClientPuter<T>(url, item, result);

   return result;
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Ajout de la fonction DELETE à WebServiceManager

```
public async Task<T> Delete(T item)
{
    T result = default(T);
    String url = typeof(T).Name.ToLower() + "s" + "/" + item.Id + "/";
    result = await HttpClientDeleter<T, T>(url, item, result);

    return result;
}
```

Accès à des données distantes

- Les services Web API mise en pratique avec JSONPlaceholder
  - Mise en œuvre

```
WebServiceManager<Post> postManager = new
        WebServiceManager<Post>("https://jsonplaceholder.typicode.com/");
List<Post> posts = await postManager.Get();
Post post = await postManager.Get(10);
Post putPost = new Post() { body="testbody", title="testtitle", userId=2 };
Post resultPutPost = await postManager.Put(putPost);
Post deletePost = new Post() { Id=2 };
Post resultDeletePost = await postManager.Delete(deletePost);
```

Changelog

Date	Author	Comment
2019	Crônier Antoine	Create initial document