# Table des matières

[0 Table des matières 1](#_Toc120489456)

[1 Introduction 4](#_Toc120489457)

[1.1 Résumé du cahier des charges 4](#_Toc120489458)

[1.1.1 Fonctionnalités de base 4](#_Toc120489459)

[1.1.2 A réaliser au minimum 4](#_Toc120489460)

[1.2 Description du projet 4](#_Toc120489461)

[2 Mise en œuvre du projet 4](#_Toc120489462)

[3 Maquettes 5](#_Toc120489463)

[3.1 Menu 5](#_Toc120489464)

[3.2 Tâches 6](#_Toc120489465)

[3.2.1 To Do List 6](#_Toc120489466)

[3.2.2 Ma journée 6](#_Toc120489467)

[3.2.3 Ajout d’une tâche 6](#_Toc120489468)

[3.2.4 Affichage d’une tâche 6](#_Toc120489469)

[3.3 Catégories 7](#_Toc120489470)

[3.3.1 Gérer les catégories 7](#_Toc120489471)

[3.3.2 Ajouter une catégorie 7](#_Toc120489472)

[4 Environnement du projet 8](#_Toc120489473)

[4.1 Installation 8](#_Toc120489474)

[4.1.1 VS 2019 8](#_Toc120489475)

[4.1.2 VS 2022 8](#_Toc120489476)

[4.2 Problèmes d’installation 9](#_Toc120489477)

[5 Création d’un projet 9](#_Toc120489478)

[6 Explication de l’environnement 11](#_Toc120489479)

[6.1 Fichiers source 11](#_Toc120489480)

[6.2 Fichiers de ressources 11](#_Toc120489481)

[6.3 Modélisation 12](#_Toc120489482)

[6.4 Code d’une activité 12](#_Toc120489483)

[7 Emulateur 13](#_Toc120489484)

[7.1 Création d’un émulateur 13](#_Toc120489485)

[7.2 Utilité d’un émulateur 14](#_Toc120489486)

[8 Activités 14](#_Toc120489487)

[8.1 Layouts 14](#_Toc120489488)

[8.1.1 LinearLayout 14](#_Toc120489489)

[8.1.2 RelativeLayout 15](#_Toc120489490)

[8.1.3 GridLayout 16](#_Toc120489491)

[8.2 Création de composants 18](#_Toc120489492)

[8.2.1 Dans le fichier XML 18](#_Toc120489493)

[8.2.1.1 Balises 18](#_Toc120489494)

[8.2.1.2 Boite à outils 18](#_Toc120489495)

[8.2.1.3 Attributs 19](#_Toc120489496)

[8.2.2 Dynamiquement 20](#_Toc120489497)

[8.2.2.1 Attributs dans le corps du bouton 20](#_Toc120489498)

[8.2.2.2 Attributs après la déclaration de la variable 21](#_Toc120489499)

[8.2.2.3 Affichage de l’élément dans un layout 21](#_Toc120489500)

[8.2.3 Lien entre un composant et le code 21](#_Toc120489501)

[8.2.3.1 Créer des évènements 21](#_Toc120489502)

[8.3 Lier le fichier CS avec le fichier XML 22](#_Toc120489503)

[8.4 Lien entre 2 activités 23](#_Toc120489504)

[9 Cycle de vie d’une application 24](#_Toc120489505)

Application mobile Android - Xamarin



|  |  |
| --- | --- |
| **Auteur** | Damien Loup |
| **Lieu** | ETML - Vennes |
| **Classe** | Cin-Cid3B |
| **Date** | 12.10.2022 |
| **Maître de projet** | Ferrari Roberto |

# Introduction

Ce document est fait pour des informaticiens et plus précisément des personnes ayant des notions en C#, celui-ci fait référence à toutes les étapes mises en œuvre pour créer un projet d’application mobile en C# avec Microsoft Visual Studio Xamarin.

Au final, l’application réalisée devra être exploitable et livrable, ainsi que la réalisation de la documentation du projet

## Résumé du cahier des charges

### Fonctionnalités de base

* Des tâches
  + Chaque tâche doit contenir :
    - Un titre
    - Un mini descriptif
    - Une date d’échéance
  + La possibilité à l’utilisateur d’en créer
  + La possibilité de sélectionner plusieurs tâches et de les ajouter à ma journée
* Des activités
  + Une activité comprenant toutes les tâches créées
    - Tri avec les catégories
  + Une activité comprenant toutes les tâche d’aujourd’hui
    - Tri avec les catégories
  + Une activité permettant d’ajouter une tâche
* Utiliser un sensor
  + Récupérer les données et les utiliser pour exécuter quelque chose
* Des catégories
  + Chaque catégorie doit au moins contenir un nom
  + Il doit être possible d’ajouter une tâche dans une catégorie

### A réaliser au minimum

* Un lien entre 2 activités différentes
* Utiliser les ressources
* Utiliser des méthodes événementielles
* Faire un layout dynamique
  + Générer du code xml dans le code c#
* Stocker des informations en base de données
* Utiliser un sensor du téléphone

## Description du projet

Réaliser une application mobile en c# avec Xamarin gérant des tâches. Des tâches peuvent être ajoutées ou retirées. Chaque tâche peut être ajouter à la liste de tâche à faire aujourd’hui et peuvent être retirées manuellement ou en secouant le téléphone.

# Mise en œuvre du projet

Pour créer une application de base, il faut réaliser certaines étapes obligatoires pour son bon fonctionnement, qui est ici **un gestionnaire de tâches**.

Les fonctionnalités ont été citées dans le cahier des charges ci-dessus.

Dans ce document vont être énuméré chaque étape et chaque tâches important au bon déroulement selon les suivantes :

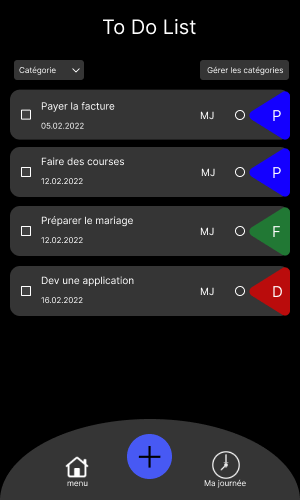
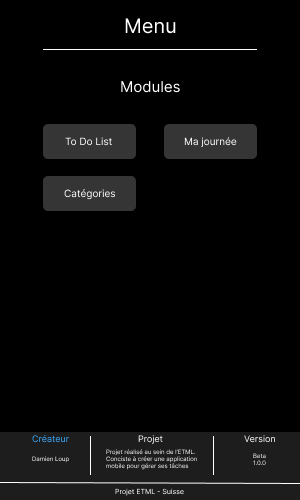
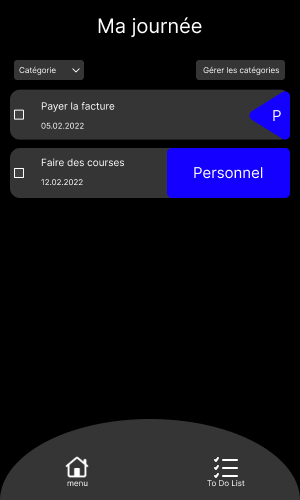
* Création des maquettes de l’application
* L’environnement de projet
* Création d’un projet
* Création d’une activité
* Utilisation d’un fichier de ressources
* Lien entre deux activités
* Les méthodes événementielles
* Les différents types de layout
* Les layouts dynamiques
* Persistance de données
* L’utilisation d’un sensor

# Maquettes

Tout d’abord, avant de commencer le projet, il faut y créer les maquettes de l’application.

Les maquettes ont été faites sur Figma à l’aide des outils à disposition sur leur application/site

## Menu

Le menu est la première activité à s’ouvrir au lancement de l’application, elle permet de se déplacer dans les différentes autres activités comme la liste de tâches, ma journée et les catégories.

## Tâches

### To Do List

La To Do List (Liste de tâches) permet d’afficher toutes les tâches crées dans l’application et entrées en base de données avec une liste scrollable. Il est possible de filtrer les tâches par catégorie ou de les gérer.

Il sera affiché le nom de la tâche, sa date d’échéance, un checkbox qui définit si la tâche est terminée ou non et un radio bouton qui définit si la tâche doit se trouver dans les tâches du jour.

Il est possible aussi de créer des tâches avec le bouton « + » en bas.

Il est possible de supprimer la tâche en appuyant longtemps sur elle et d’appuyer sur le bouton « - »

### Ma journée

L’activité « Ma journée » permet de voir toutes les tâches ajoutées à « aujourd’hui » c’est-à-dire les tâches que l’utilisateur veut pouvoir faire le jour-même avec une liste scrollable ! Comme pour la To Do List, Il est possible de filtrer les tâches par catégorie ou de les gérer.

Il sera affiché le nom de la tâche, sa date d’échéance, un checkbox qui définit si la tâche est terminée ou non et un radio bouton qui définit si la tâche doit se trouver dans les tâches du jour.

Il est possible de supprimer la tâche pour le jour en appuyant longtemps sur elle et d’appuyer sur le bouton « - »

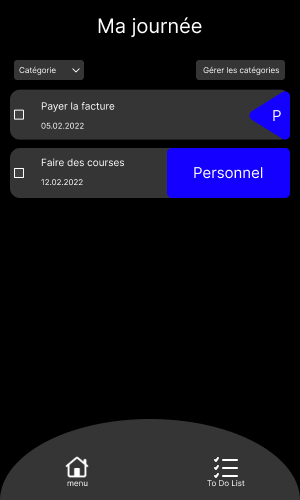
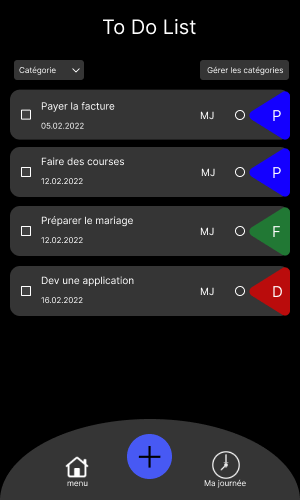
**C’est sur cette activité que sera utilisé le sensor :**Le but est de faire que lorsque l’on secoue le téléphone, toutes les tâches de la journée actuelle se suppriment (Uniquement dans « aujourd’hui »).

### Ajout d’une tâche

L’activité d’ajout d’une tâche permet d’ajouter une tâche avec un titre, une description, une catégorie et une date.

### Affichage d’une tâche

L’activité d’affichage d’une tâche permet d’afficher son titre, sa description, sa catégorie et sa date d’échéance.



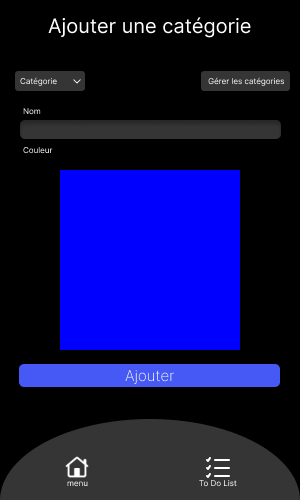


## Catégories

### Gérer les catégories

Cette activité est le sous-menu des catégories, ce menu permet de voir toutes les catégories créées dans l’application, il permet aussi d’en créer

### Ajouter une catégorie

Cette activité permet de créer une catégorie avec un nom et une couleur donnée¨¨

# Environnement du projet

L’environnement de projet utilisé est Xamarin, pour l’utiliser, il faut installer l’application (IDE) Visual Studio.

## Installation

En premier lieu il faut lancer « Visual Studio Installer »

Une image contenant texte, capture d’écran, écran

Description générée automatiquement

Une fois sur cette page, il faudra choisir (S’il y a plusieurs versions de Visual Studio) la version de Visual studio préférée.

Il faudra venir modifier celle-ci avec le bouton « Modifier » et trouver en fonction de la version :

### VS 2019

« Développement mobile en .NET » et cocher la checkbox en haut à droit et installer la modification.

Une image contenant texte

Description générée automatiquement

Et il faudra installer les packages SDK et un émulateur Android.

### VS 2022

« Développement .NET Multi-Platform App UI » et cocher la checkbox en haut à droite et installer la modification.

Une image contenant texte

Description générée automatiquement

Et ensuite aller dans l’onglet « Composants individuels » et rechercher « Xamarin » et cocher ces 3 checkboxs.

Une image contenant texte

Description générée automatiquement

## Problèmes d’installation

Etant donné que Xamarin va bientôt être obsolète et remplacé par Maui, il peut y avoir des problèmes d’installation ou ne pas trouver Xamarin dans les dernières versions de Visual Studio.

Si cela se passe, il faudra aller chercher dans les versions antérieures aux versions qui ne disposent plus de cet environnement ou de suivre le point précédent en installant différemment l’environnement.

# Création d’un projet

Au démarrage de Visual Studio il faut créer un projet.

Une image contenant texte

Description générée automatiquement

Une fois ce bouton cliqué, il faudra choisir le bon déroulé pour avoir accès au projet.

Il faut choisir les filtres séparément : C#, Android et Tous les types de projet.

Le type de projet « Application Android (Xamarin) est celui qui nous intéresse pour ce projet.

Une image contenant texte

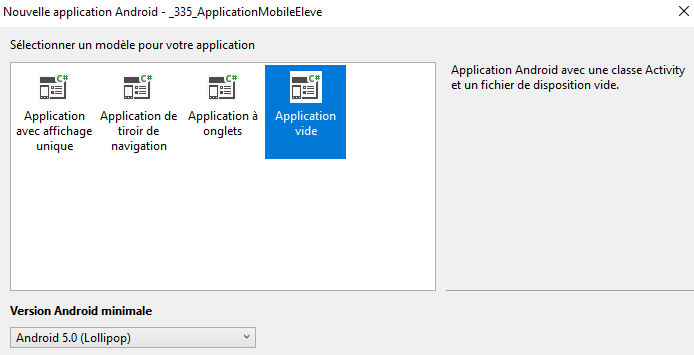
Description générée automatiquement

Ensuite il faudra venir donner un titre et choisir l’emplacement stocké du projet

Une image contenant texte, capture d’écran, écran, argent

Description générée automatiquement

Une fois cela fait et le bouton « Créer » appuyé, il faudra choisir le modèle de base du projet.  
Ici nous allons utiliser une application Vide pour la faire de A à Z.



Vient ensuite, le choix de la version de notre application, ici nous allons choisir la ou une des plus anciennes versions d’Android, car plus la version est récente, plus il y aura de fonctionnalités à notre disposition, mais il y aura aussi plus de téléphones qui ne pourront pas l’utiliser et la supporter, du à leur version d’Android.

Une image contenant texte

Description générée automatiquement

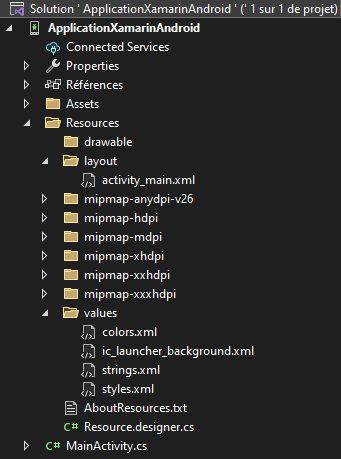
# Explication de l’environnement

L’environnement ressemble à un projet C# classique en console, etc… Cependant, il y a quelques subtilités qui en font une différence considérable.

## Fichiers source

Les fichiers sources s’apparentent à ceci.

* Les interface de l’application, avec les fichiers de modélisation en XML se trouvent dans « layout »
* Les images ou les éléments à ajouter dans la modélisation de l’application se trouveront dans « drawable »
* Les fichiers de ressources permettant de gérer les styles, les couleurs et des chaines de caractères de l’application sont dans « values »



## Fichiers de ressources

Les fichiers de ressources se trouvant dans « values » sont considéré comme des fichiers contenant des constantes accessibles par tout l’environnement.

Ils peuvent permettre de changer la couleur du thème, changer des couleurs ou en ajouter autant que l’on veut.

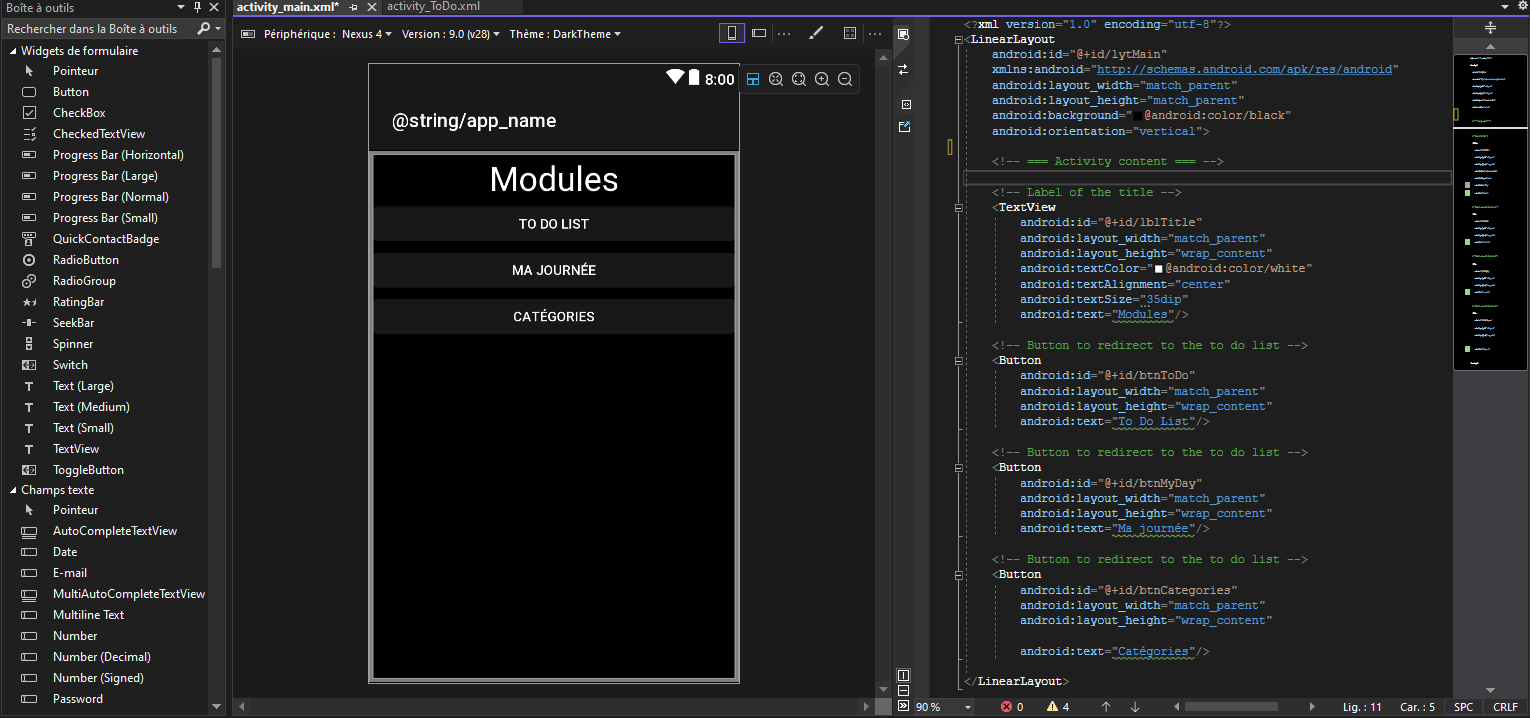
Les fichiers de ressources peuvent aussi être adaptés pour rendre l’application multilingue avec un fichier par langue différente. Ils peuvent aussi faire en sorte d’adapter l’application avec des fichiers différents en fonction de la taille de l’écran, cela permet de séparer le code source des éléments de design, etc…

## Modélisation

Le fichier de modélisation étant, dans ce cas-là, le fichier « activity\_main.xml »

En double cliquant sur le fichier de modélisation, l’interface ci-dessous apparait, le code XML est à droite et la visualisation de l’interface de l’application à gauche.

Tout à gauche de l’interface, se trouve la boîte à outils, elle permet d’y placer des éléments graphiques dans l’activité, comme un bouton, du texte, etc…



## Code d’une activité

Le code de base d’une activité s’apparente à une classe classique.

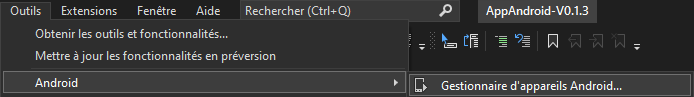
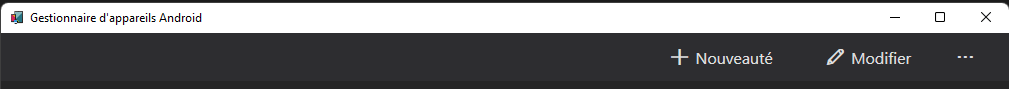
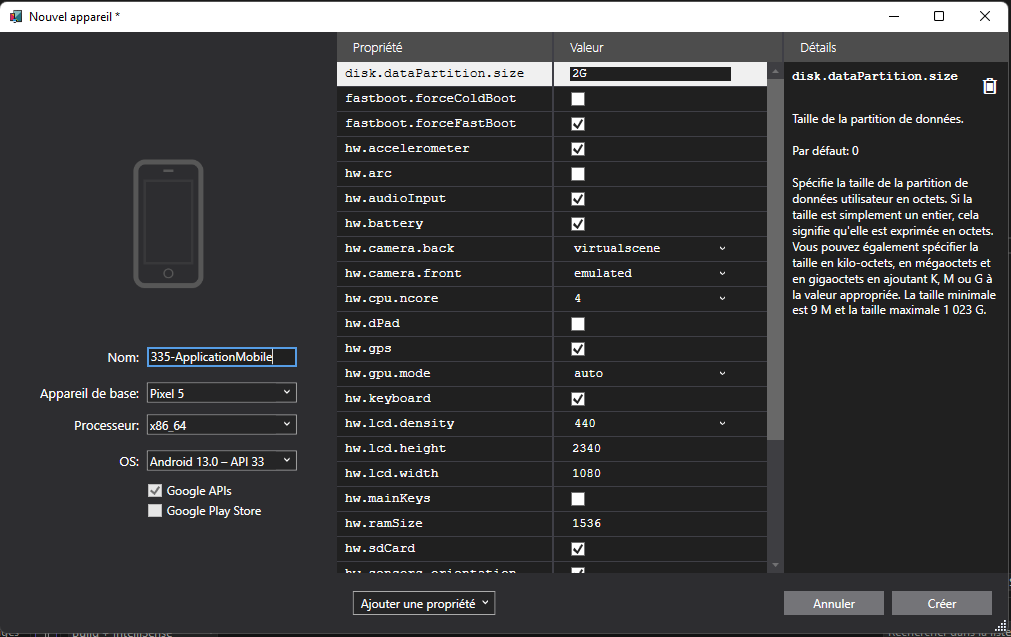
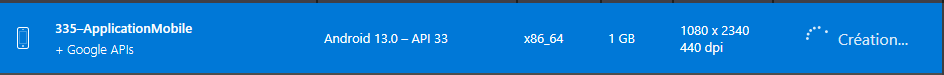
Quelques éléments à relever sont conçus pour l’environnement de Xamarin :  
La ligne En dessus de la classe permet de définir :  
- Label -> Le nom à afficher par défaut dans l’activité.  
- Theme -> Le thème à utiliser qui a été défini dans un des fichiers ressources (styles.xml).  
- MainLauncher -> Défini si l’activité est la principale de l’application.

# Emulateur

L’émulateur est un point important pour tester son application. Le but est de simuler un vrai téléphone sur le PC.

Il est possible aussi de tester son application en la déployant sur un élément physique comme un téléphone. Pour ceci, il faut aller ***paramètres -> À propos du téléphone -> informations sur le logiciel***   
Une fois ceci fait, il faut appuyer 7 fois sur ***Numéro de version*** pour activer le mode développeur pour ensuite se rendre dans ***Options de développement*** activer le débogage USB et connecter son téléphone en USB à votre PC et lancer le débogage dans visual studio.

## Création d’un émulateur

En premier lieu, il faut aller dans ***outils -> android -> Gestionnaire d’appareils Android*** et être admin sur la machine.   
  
Une fois sur la page d’appareils Android, il faut cliquer sur le bouton « Nouveauté »  
  
Après ceci, une page de configuration va s’afficher. Elle permet de configurer le téléphone comme par exemple : La taille de stockage, la ram, le nom, le type de téléphone, l’OS, etc…  
  
Lorsque la configuration est effectuée, il faut appuyer sur « Créer » et notre émulateur va s’installer dans notre gestionnaire.  
Une fois que l’émulateur est installé, il est possible de le démarrer et l’utiliser « comme » un vrai téléphone et pour déboguer le code effectué, il suffit de choisir **l’émulateur** ou le **téléphone physique** dans le menu de débogage.  
Une image contenant texte

Description générée automatiquement

## Utilité d’un émulateur

Un émulateur permet de tester l’application et de voir si elle fonctionne correctement sur d’autres types de téléphones sans les avoirs physiquement et de les simuler. Cela permet aussi de tester selon la version de l’OS étant donné que chaque version contient des fonctionnalités en plus ou en moins.

# Activités

Au début du projet de base une activité est créée. Une activité est composée de 2 fichiers différents :

* 
* 

Le fichier MainActivity.cs contient tout le code C# de l’activité de son nom. Il permet de créer des objets comme des boutons, etc… et d’en récupérer aussi. Il permet aussi de gérer les méthodes événementielles.

Le fichier Activity\_main.xml contient tout le code XML avec ses balises, c’est ce qui permet de créer l’interface de l’application.

## Layouts

Un layout est un des composants principaux d’une activité, il est en quelque sorte, un conteneur qui contient une série d’éléments comme des **boutons**, des **textes** et même d’autres **layouts**.

Les layouts ne donnent pas la possibilité de scroller, c’est pourquoi s’il y a trop d’éléments il faudra rajouter un scrollView.

Il en existe plusieurs qui effectuent des tâches différentes, en voici une petite liste.

### LinearLayout

Le linear layout est un des layouts les plus utilisés. Il permet d’afficher, comme son nom l’indique, de manière linéaire tous les éléments qu’il contient. Il est possible de le paramétrer dans les attributs pour changer son orientation, afin d’afficher son contenu verticalement ou horizontalement.  
Il est possible aussi de changer la couleur de fond, la taille et d’autre attributs dont l’ID qui est le plus important, afin de le récupérer dans le code plus tard.  
Une image contenant texte

Description générée automatiquement

### RelativeLayout

Le relative layout est aussi un des layouts les plus utilisés et permet d’afficher des éléments aux positions exactes que l’on veut.

Les éléments se placent toujours de la même manière dans le code XML, mais ne se posent pas automatiquement en dessous de celui d’avant.

Ici pour placer un élément, il faut se baser sur les autres ou l’élément parent qui peut être un autre layout.  
Une image contenant texte

Description générée automatiquement  
Ici, on place un TextView contenant un texte « Ajouter des catégories » ce texte étant le premier élément, il est affiché au début.

Ensuite, on place un LinearLayout en horizontal en spécifiant qu’il soit en dessous du texte placé juste avant à l’aide de l’attribut « android:layout\_below="@id/txtCategoryTitle" »  
Ensuite, on y place deux boutons avec des attributs qui permettent de gérer le style, ainsi que respectivement un poids de 1 pour qu’ils aient la même taille et se placent correctement sur l’horizontal comme défini dans le layout parent, ainsi que des marges.

Il y a aussi un bouton placé après le linearLayout avec l’attribut « android:layout\_alignParentBottom=’’true’’ » qui permet de le placer tout en bas de l’activité  
Une image contenant texte

Description générée automatiquement  
Une image contenant texte

Description générée automatiquement

### GridLayout

Le gridLayout permet de gérer son activité à l’aide d’une grille et de placer les éléments sur des lignes et des colonnes.

Tout d’abord, les premiers réglages se font dans la balise « <GridLayout> » et d’y placer les attributs habituels ainsi que le nombre de colonnes et de lignes de celui-ci à l’aide des attributs « android:columnCount=“ “ » et « android:rowCount=“ “ »  
Une image contenant texte

Description générée automatiquement

Pour l’exemple, il y a 4 TextView placés l’un après l’autre. 3 de ces 4 éléments contiennent un attribut définissant combien de ligne il occupe « android:layout\_rowSpan“ “ ».  
Une image contenant texte

Description générée automatiquementUne image contenant texte

Description générée automatiquement

Il y a normalement un moyen de faire en sorte que chacun des ces éléments remplissent totalement avec les attributs « android:layout\_rowWeight="1" » et « android:layout\_columnWeight="1" ». Malheureusement dans la version actuelle, Visual Studio génère des erreurs et demande une unité qui désactive ces attributs.  
Une image contenant texte

Description générée automatiquement

## Création de composants

Un composant peut être créé de différentes manière, que ça soit dans le fichier XML en dur directement ou dans le code dynamiquement.

Il est important de bien différencier ces 2 manières de faire, car elles ne sont pas forcément utilisées pour le même but. Créer des éléments dynamiquement pourrait servir à créer plusieurs fois le même élément, mais avec une utilité différente.

### Dans le fichier XML

Il y a 2 manières de base de placer un composant dans le fichier. Il est possible de directement les écrire via le clavier ou d’utiliser une boîte à outils qui placera automatiquement les balises et les attributs importants à l’intérieur.

#### Balises

Pour créer un composant, il y a toujours besoin d’une balise XML. Le nom de la balise est équivalent au nom de la classe C#. Cela permet de définir le type d’élément et le récupérer dans le code.

Un composant peut avoir deux écritures de balise, mais dans tous les cas, il doit y avoir un début et une fin.  
1.  
  
2.  
  
Il existe plusieurs types d’éléments et chaque élément à ses propres balises :  
- Button  
- TextView  
- LinearLayout  
- etc...

#### Boite à outils

La boîte à outils permet directement de placer les balises dans le code XML en double cliquant sur un des éléments dans la liste. Depuis cet endroit, tous les éléments possibles sont affichés dans la liste et sont répertoriés dans des catégories. Les attributs de base sont automatiquement écrits dans la balise.  
Une image contenant texte

Description générée automatiquement Une image contenant texte

Description générée automatiquement

#### Attributs

Les attributs sont les bases d’un élément.   
Ceux-ci permettent de donner le style de celui-ci : changer la couleur du fond, du texte, d’y inscrire un texte et d’y changer son alignement, sa taille, etc...   


Ils permettent aussi d’attribuer un ID aux éléments, ce qui permet plus tard de les retrouver dans le code et de les répertorier dans le fichier de ressources.

Un id se déclare d’une certaine manière, il faut faire « **@+id/** », car il se créée automatiquement dans le fichier de ressources. Chaque Id est de type « **int** » et contient un nombre supérieur de 1 au dernier ID créé. Un ID n’est pas forcément ceux que nous avons créé, il peut être un Id système permettant le bon fonctionnement du programme interne.

Celui-ci contient un nombre et son nom de variable qui est placé après le « **/** ».  
  
Voici comment un ID est représenté dans le fichier de ressource en C#   
Une image contenant texte

Description générée automatiquement

Devant un attribut, le mot clef « android » est placé pour faire référence à l’application qui est de type android.  
Une image contenant texte

Description générée automatiquement  


### Dynamiquement

Pour créer un élément dynamiquement, il suffit de le faire dans le code. Comme dans les points précédents, un élément est un « objet » du nom de sa balise. Ici pour créer un élément, cela requiert d’instancier une variable à l’aide de la classe appropriée.

Pour l’exemple, nous allons prendre des boutons. Afin de créer un bouton il faut commencer à l’aide de la classe « Button » et lui donner un nom de variable.  
Il est aussi obligatoire d’ajouter le contexte dans lequel le bouton va s’afficher qui est le layout que l’on veut. Ici on y met le mot clef « this » afin que le contexte soit le layout actuel, étant donné que ce bouton à été déclaré dans la classe d’une activité.  
Une image contenant texte, périphérique, mètre, jauge

Description générée automatiquement

Il est aussi possible de créer des layouts dynamiques et donc d’ajouter des éléments dynamiques à l’intérieur ce layout-ci.

Désormais, il y a 2 manières de changer les attributs d’un élément.

#### Attributs dans le corps du bouton

La première manière est d’instancier un bouton de la même manière que ci-dessus, mais en ajoutant des accolades avant le point-virgule.  
En faisant ceci, nous pouvant ajouter certains attributs directement avec leur nom à l’intérieur du corps de « méthode » (qui ressemble à un corps de méthode) du bouton.   
Nous ne pouvons pas définir tous les attributs dans le corps et il faudra les définir après la déclaration de la variable pour certains.

Ici on donne un texte au bouton et une visibilité qui fera en sorte que celui-ci sera invisible à l’affichage  
Une image contenant texte

Description générée automatiquement

#### Attributs après la déclaration de la variable

La deuxième manière d’ajouter des attributs à l’élément est d’utiliser le nom de variable suivit d’un point et de l’attribut que nous voulons définir  
Une image contenant texte

Description générée automatiquement

Ici on définit la couleur du fond et du texte.

#### Affichage de l’élément dans un layout

Désormais, nous pouvons ajouter notre élément à un layout. Pour ceci, il faut tout d’abord récupérer le layout auquel nous voulons ajouter l’élément avec la méthode « FindviewById » qui permet de récupérer tout type d’éléments dont les layouts.  


Après ceci fait, nous pouvons ajouter notre élément au layout à l’aide de la variable déclarée qui récupère le layout principal de l’activité grâce à la méthode « AddView() » qui permet d’y entrer l’élément que nous venons de créer.  


### Lien entre un composant et le code

Lorsque l’on crée un élément dans le fichier XML, nous pouvons faire le lien entre le code et cet élément en le récupérant dans un objet dans le code. Prenons l’exemple de la création d’un bouton dans le XML.  
Une image contenant texte

Description générée automatiquement   
Dans le code, nous pouvons désormais récupérer ce bouton à l’aide de la méthode « FindViewById » en déclarant un objet de type « Button » étant donné que l’élément créé dans le XML est un bouton.  


#### Créer des évènements

Pour terminer, nous pouvons ajouter des méthodes événementielles à ce bouton en ajoutant « .Click += » après la variable du bouton en spécifiant le nom de la méthode que nous voulons qu’elle s’exécute lorsque l’utilisateur appuie dessus.  


Lorsque l’on clique sur ce bouton, la méthode événementielle liée s’exécute. Pour l’exemple, ici nous allons faire en sorte que le texte change et que le bouton créé plus tôt s’affiche. Pour ceci le bouton à besoin d’être instancié dans les variables de classes.

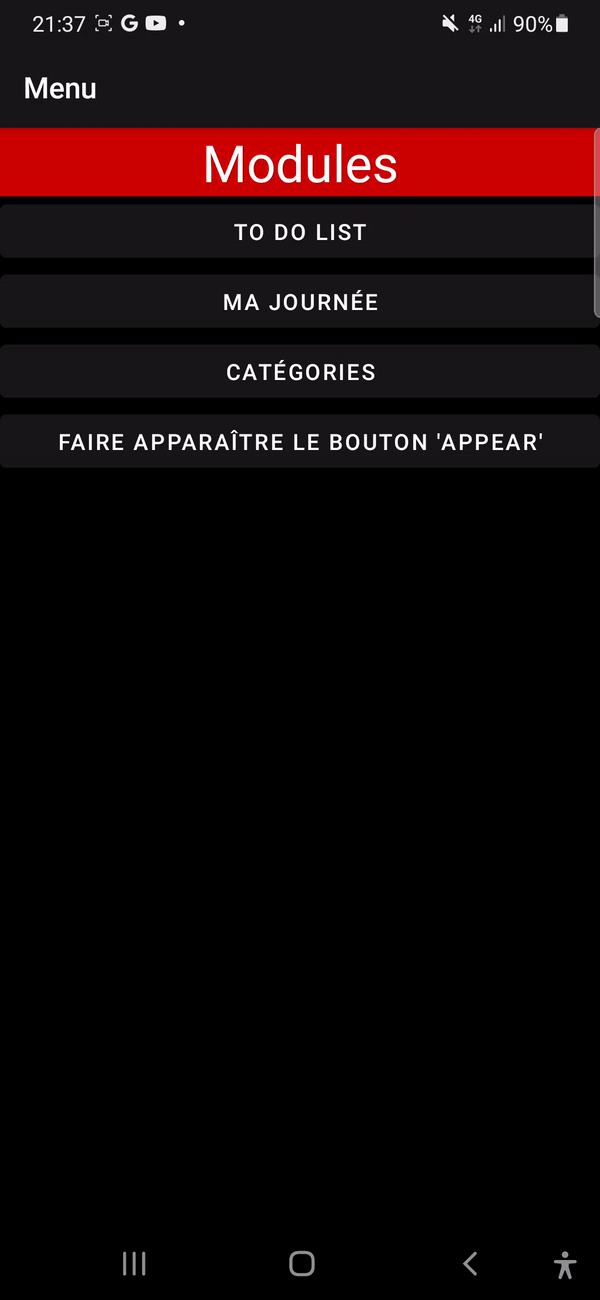
Maintenant, pour changer la visibilité du bouton, il suffit de reprendre l’objet du bouton créé plutôt.

Ensuite, il y a 2 éléments important dans la signature de méthode qui sont le « sender » et le « e » ces deux éléments permettent de récupérer des informations en rapport au clic du bouton.

Le « sender » est l’élément qui à été cliqué, donc l’objet du bouton en lui-même, ce qui permet d’affecter des changements à celui-ci à l’aide d’attributs, ici on y change le texte.

Le « e » est l’événement de clic du bouton et donnent des informations par rapport à l’événement qui a été effectuer.   
Une image contenant texte

Description générée automatiquement

Voici le résultat de cet événement.  


## Lier le fichier CS avec le fichier XML

La simple ligne de code « SetContentView() ; » permet d’associer le fichier .cs au fichier .xml comme ci-dessous.

Une image contenant texte

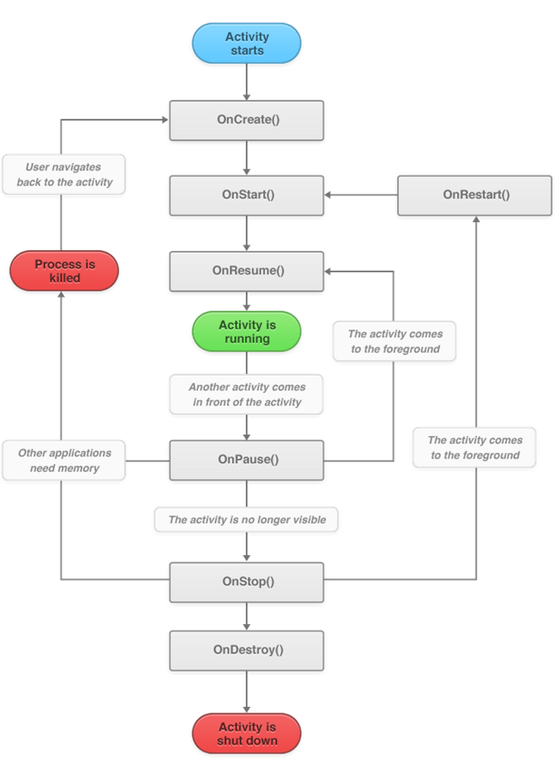
Description générée automatiquement

À sa création, le fichier XML de l’activité s’inscrit dans les ressources à l’aide d’un ID et se retrouve à l’aide de « Ressource.Layout.activity\_name »

## Lien entre 2 activités

# Cycle de vie d’une application

Le cycle de vie d’une application contient plusieurs états qui permettent de créer, démarrer, afficher et gérer les activités. Chacun de ces états ont leur équivalent en méthodes dans le code qui sont automatiquement créées et appelées par le programme en fonction des actions de l’utilisateur sur l’application.



Les méthodes sont définies dans la classe « Activity », elles permettent d’effectuer ces états à chacune des activités dû au fait que toutes les classes de celles-ci héritent de « Activity ». Afin de pouvoir effectuer certains états à une activité, il suffit de déclarer ces méthodes dans la classe de l’activité voulue. Etant donné l’héritage, il faut réécrire ces méthodes avec le mot clef « override » et « protected ».

Une image contenant texte

Description générée automatiquement