

Programmation mobile avec Xamarin (1)

Présentation de Xamarin :

Lire l'article programmez « Pourquoi Xamarin ? »

1. Réalisation d'un projet Xamarin Forms :

Pour débiter l'apprentissage de la programmation Xamarin, nous allons opter pour un projet « Xamarin Forms », moins flexible que le développement natif mais plus simple de prime abord pour se lancer avec ce framework.

L'objectif ici sera de construire une petite application pour gérer ses contacts téléphoniques.

1.1. Préparation du projet

Nous allons gérer le code source sous github. Pour cela, créez un nouveau projet public depuis votre compte github comme illustré ci-dessous :

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: edondelinger / Repository name: IntroAXamarin

Great repository names are short and memorable. Need inspiration? How about silver-waffle.

Description (optional): Dépôt application découverte de Xamarin

☒ Public: Anyone can see this repository. You choose who can commit.

☐ Private: You choose who can see and commit to this repository.

☒ Initialize this repository with a README: This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

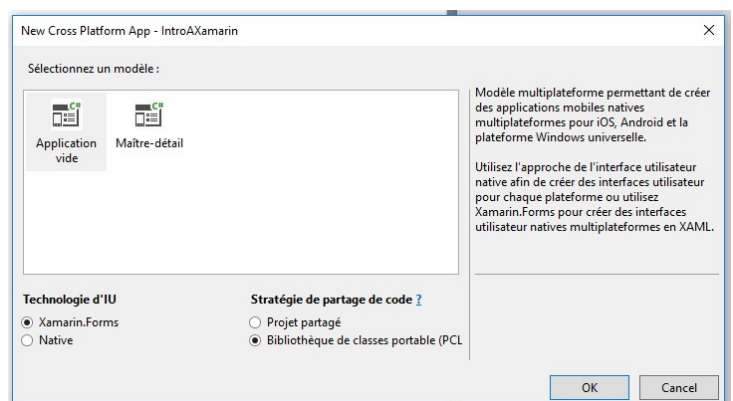
Add .gitignore: Android / Add a license: GNU Lesser General Public License v3.0

Create repository

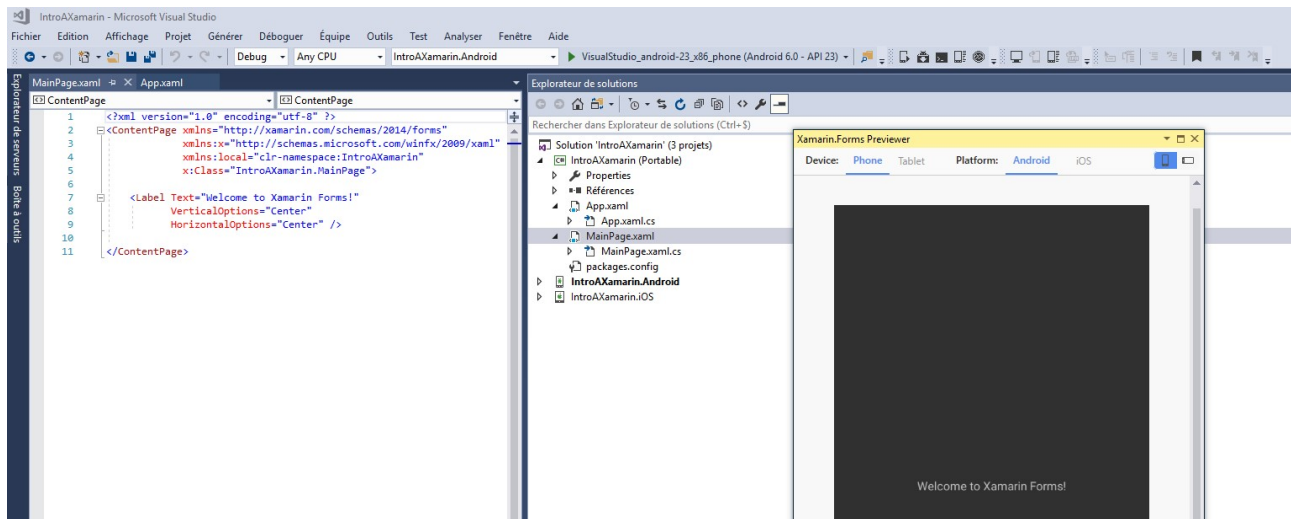
Clonez ensuite ce projet en local sur votre poste avec un outil comme Git Desktop.

1.2. Création du projet

Depuis l'interface de Visual Studio 2017 (version entreprise 15.3.1), créez un nouveau projet de type « Cross platform App » dans le dossier où vous avez cloné le projet github. Choisissez le type « Xamarin Forms » et prenez une application vide.



L'interface rassemble alors les éléments suivants :



La fenêtre Xamarin.Forms Previewer s'obtient depuis le menu affichage / autres fenêtres.

Plusieurs choses sont à remarquer ici :

- Les « écrans » se créent à l'aide d'un dérivé du langage XML nommé XAML qui permet de positionner des éléments graphiques. Ces éléments sont nombreux et la documentation en ligne en donne un bon aperçu : <https://developer.xamarin.com/guides/xamarin-forms/getting-started/>
- L'explorateur de solution montre plusieurs projets, nous interviendrons sur le premier (de type Portable), les 2 autres Android et IOS étant destinés aux éléments spécifiques qui doivent être écrits différemment pour chacune des 2 plateformes.
- Le Previewer vous permettra de bien visualiser le placement de vos éléments à l'écran et peut être utile car il n'existe pas d'interface visuelle comme pour une application Windows Forms pour placer graphiquement nos éléments d'interface.

Afin de vérifier que tout est en ordre de marche, je vous propose d'exécuter votre application sur l'émulateur Android.

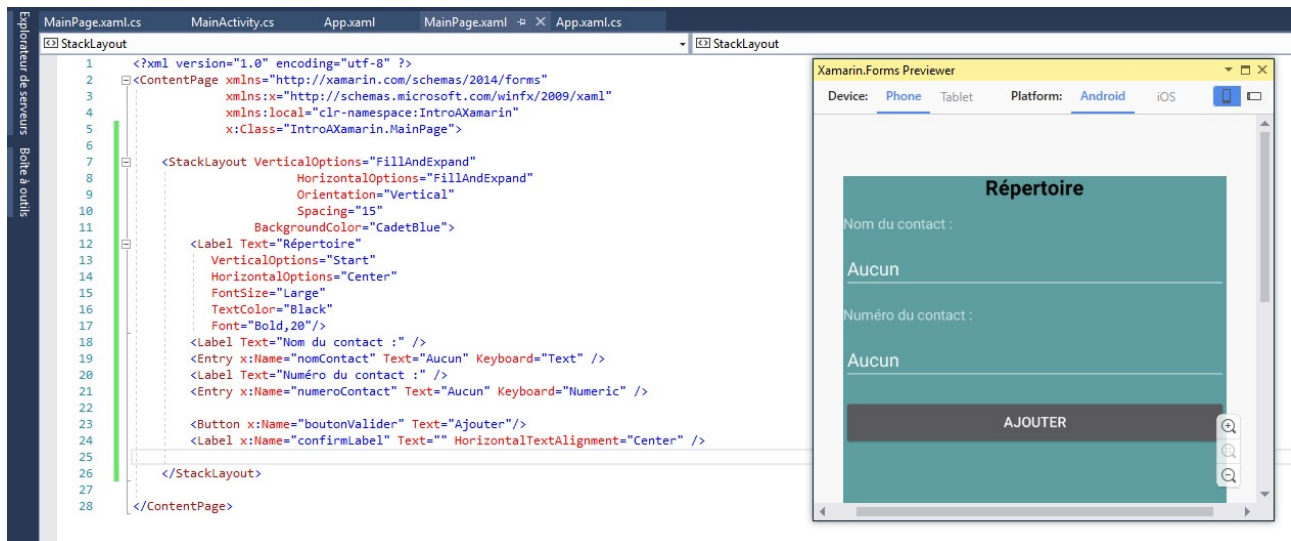
1.3. Création de la première page

Nous allons créer une Forms qui permettra d'ajouter un nouveau contact, ce sera notre page d'accueil. Rappelons que la conception de nos écrans repose sur le langage XAML comme nous l'avons vu dans la section précédente.

Ajoutez le code suivant à votre page principale MainPage.xaml

```
<StackLayout VerticalOptions="FillAndExpand"
              HorizontalOptions="FillAndExpand"
              Orientation="Vertical"
              Spacing="15"
              BackgroundColor="CadetBlue">
  <Label Text="Répertoire"
        VerticalOptions="Start"
        HorizontalOptions="Center"
        FontSize="Large"
        TextColor="Black"
        Font="Bold,20"/>
  <Label Text="Nom du contact :" />
  <Entry x:Name="nomContact" Text="Aucun" Keyboard="Text" />
  <Label Text="Numéro du contact :" />
  <Entry x:Name="numeroContact" Text="Aucun" Keyboard="Numeric" />
  <Button x:Name="boutonValider" Text="Ajouter"/>
  <Label x:Name="confirmLabel" Text="" HorizontalTextAlignment="Center" />
</StackLayout>
```

L'aperçu dans Xamarin.Previewer vous donne cela :



1.4. Création des objets métiers

Nous allons maintenant donner vie à ce formulaire en préparant les objets qui stockeront en mémoire nos contacts du répertoire.

Ajoutez une classe Contact à votre projet dont la description figure ci-dessous :

```
public class Contact
{
    private int id;
    private string nom;
    private string numero;

    public int Id { get => id; set => id = value; }
    public string Nom { get => nom; set => nom = value; }
    public string Numero { get => numero; set => numero = value; }

    public Contact() { }

    public Contact(int id, string nom, string num)
    {
        this.Id = id;
        this.Nom = nom;
        this.Numero = num;
    }

    public Contact(string nom, string num)
    {
        this.Nom = nom;
        this.Numero = num;
    }
}
```

Une fois cette classe créée, il nous faut une collection de contacts afin de les stocker. Ajoutez dans le code CS un nouvel attribut collection :

```
public partial class MainPage : ContentPage
{
    private List<Contact> lesContacts;
```

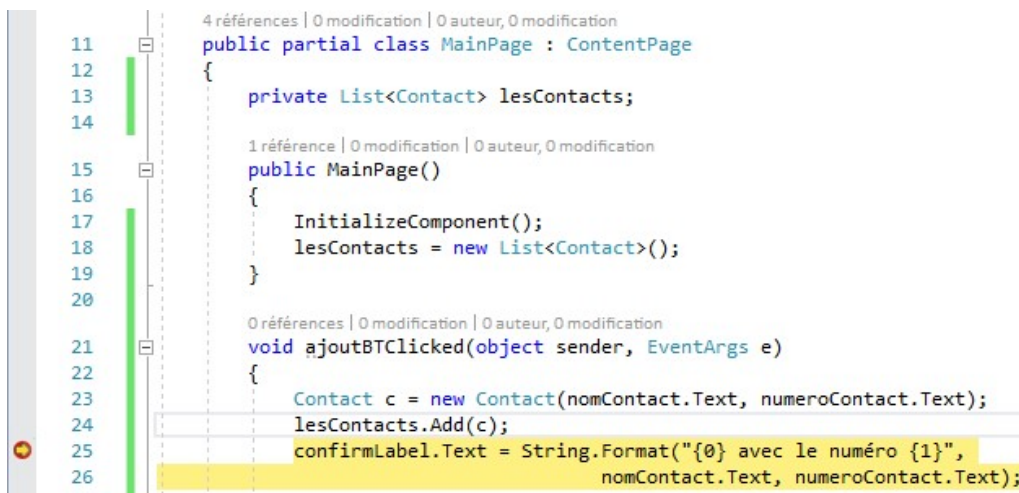
1.5. Stockage des contacts créés

Nous allons maintenant stocker nos contacts ajoutés sur le formulaire dans notre collection.

La première chose à faire est d'associer une méthode au clic sur notre bouton du formulaire. Pour cela, il y a plusieurs manières de faire. Par programmation ou directement sur le composant bouton dans le fichier XAML. Nous allons opter pour cette seconde option en ajoutant un attribut « Clicked » au bouton :

```
<Button x:Name="boutonValider" Text="Ajouter" Clicked="ajoutBTClicked"/>
```

La méthode ajoutBTClicked doit ensuite être écrite dans le code. On voit ci-dessous qu'on a créé cette méthode qui pour le moment va simplement instancier un contact à partir des informations entrées par l'utilisateur, ajouter ce contact à la collection puis afficher un message de confirmation dans la zone d'interface « confirmLabel ».



```

11 4 références | 0 modification | 0 auteur, 0 modification
12 public partial class MainPage : ContentPage
13 {
14     private List<Contact> lesContacts;
15
16     1 référence | 0 modification | 0 auteur, 0 modification
17     public MainPage()
18     {
19         InitializeComponent();
20         lesContacts = new List<Contact>();
21     }
22
23     0 références | 0 modification | 0 auteur, 0 modification
24     void ajoutBTClicked(object sender, EventArgs e)
25     {
26         Contact c = new Contact(nomContact.Text, numeroContact.Text);
27         lesContacts.Add(c);
28         confirmLabel.Text = String.Format("{0} avec le numéro {1}",
29                                         nomContact.Text, numeroContact.Text);
26

```

Vous exécuterez ce code pour tester son fonctionnement en mettant un point d'arrêt après l'ajout dans la collection. Vous pourrez alors voir que la collection se remplit correctement à chaque nouvel ajout. Nous mémorisons donc nos contacts en mémoire.

1.6. Affichage des contacts créés

Nous ajoutons nos contacts à l'application. Reste maintenant à afficher ces contacts à l'écran. Plusieurs changements vont s'avérer nécessaires.

Tout d'abord, nous allons modifier notre XAML pour ajouter une ListView dont voici le code :

```

<ListView x:Name="ContactView">
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <StackLayout BackgroundColor="#eee"
                    Orientation="Vertical">
                    <StackLayout Orientation="Horizontal">
                        <Label Text="{Binding Nom}"
                            TextColor="#f35e20" />
                        <Label Text="{Binding Numero}"
                            HorizontalOptions="EndAndExpand"
                            TextColor="#503026" />
                    </StackLayout>
                </StackLayout>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>

```

Notre ListView se compose d'un ListView.ItemTemplate qui décrit la façon dont devra apparaître chacun de ses éléments. Ici, on fait en sorte que chaque ligne apparaisse avec 2 labels dont la valeur prendra la valeur des attributs de nos objets Contact (Binding Nom et Binding Numero).

Comme les choses ne sont pas totalement magiques pour faire apparaître les données, il va falloir écrire quelques lignes de code. Rendons nous dans le constructeur de notre page et ajoutons les lignes suivantes :

```
ContactView.ItemsSource = lesContacts;
ContactView.ItemSelected += ContactView_ItemSelected;
```

On indique que la source de nos données de la ListView est la collection lesContacts et que lorsqu'un élément sera sélectionné, on exécutera la fonction « ContactView_ItemSelected » qu'il va nous falloir écrire.

Pour le moment, cette méthode se contentera d'afficher un texte :

```
private void ContactView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
{
    DisplayAlert("Alerte", "Vous avez choisi un élément de la liste", "OK");
}
```

Notre code est presque fonctionnel mais malheureusement, le Binding tel qu'il a été défini ne peut pas fonctionner avec un objet List classique. Nous allons devoir remplacer le type de notre collection List par une collection de type ObservableCollection qui en reprend les grands principes à la différence près qu'elle englobe des fonctionnalités lui permettant d'informer d'autres objets qui seraient à son écoute (c'est le cas de notre ListView). Modifiez donc votre code de la manière suivante et testez le résultat :

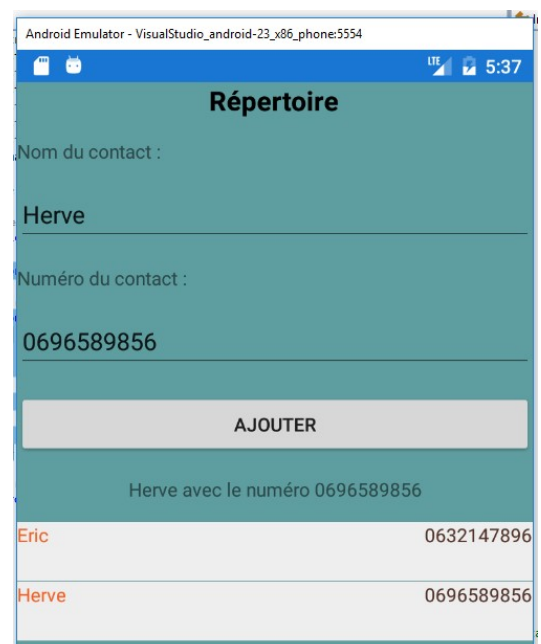
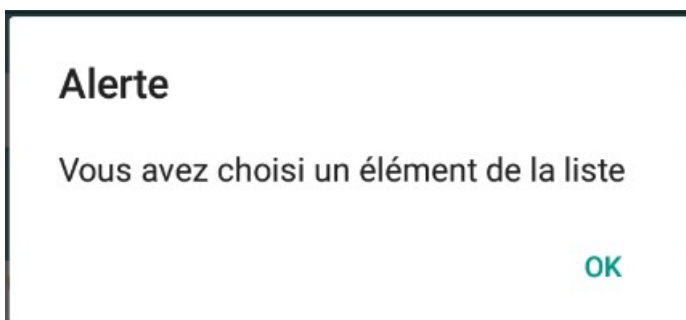
```
private ObservableCollection<Contact> lesContacts;

public MainPage()
{
    InitializeComponent();
    lesContacts = new ObservableCollection<Contact>();

    ContactView.ItemsSource = lesContacts;

    ContactView.ItemSelected += ContactView_ItemSelected;
}
```

Le résultat est conforme à nos attentes ...



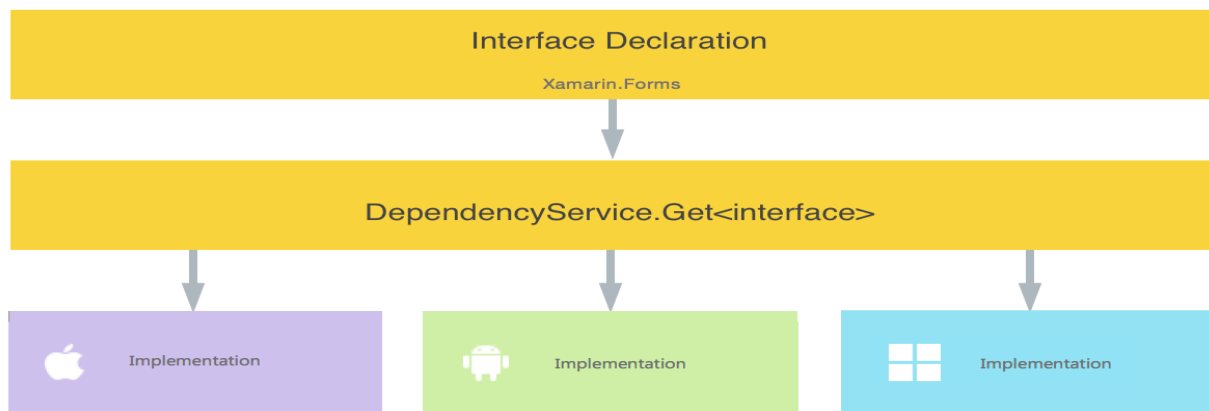
1.7. Enregistrement des contacts créés

Nos contacts sont mémorisés mais jusqu'à un certain point. En effet, si nous relançons l'application, tout est remis à zéro et nous perdons ce que nous avons saisi.

Nous allons y remédier en gérant une persistance dans un simple fichier texte.

Tout d'abord, il faut savoir que même si Xamarin permet de développer avec un même code sur plusieurs supports, certaines fonctionnalités doivent être codées nativement pour chacun de ces supports. Selon l'approche utilisée, l'accès au système de fichier peut en faire partie. Nous allons donc mettre ceci en œuvre, ce qui sera formateur pour bien appréhender cet aspect des choses.

Une des techniques à disposition pour cela est d'utiliser le DependencyService qui à partir d'un code commun va pouvoir accéder à des fonctionnalités natives.



Le principe est illustré par le schéma ci-dessus : il faut créer une interface dans le projet Xamarin.Forms et définir des implémentations différentes dans les projets natifs de chacun des supports (iOS, Android, Windows Phone).

Pour sauvegarder nos contacts dans un fichier texte, nous aurons besoin de créer une interface dans notre projet commun. Nous partons du principe que nous ferons 2 choses :

- Enregistrer des données dans le fichier
- Charger les données contenues dans le fichier

Nous créons donc l'interface suivante :

```
namespace IntroAXamarin.classes
{
    public interface ISaveAndLoad
    {
        void SaveText(string filename, string text);
        string LoadText(string filename);
    }
}
```

A noter que la page suivante explique très bien la marche à suivre pour davantage de détails :

<https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/files/>

Ceci effectué, nous allons définir le code natif du projet Android qui utilisera cette interface :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using IntroAXamarin.classes;
using System.IO;
using Xamarin.Forms;
using IntroAXamarin.Droid;

[assembly: Dependency(typeof(SaveAndLoad))]
namespace IntroAXamarin.Droid
{
    public class SaveAndLoad : ISaveAndLoad
    {
        public void SaveText(string filename, string text)
        {
            var documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            var filePath = Path.Combine(documentsPath, filename);
            System.IO.File.WriteAllText(filePath, text);
        }
        public string LoadText(string filename)
        {
            var documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            var filePath = Path.Combine(documentsPath, filename);
            return System.IO.File.ReadAllText(filePath);
        }
    }
}
```

Ceci effectué, nous allons invoquer le DependencyService dans une méthode de sauvegarde et une méthode de chargement. Voyons la partie sauvegarde tout d'abord :

```
void ajoutBTClicked(object sender, EventArgs e)
{
    Contact c = new Contact(nomContact.Text, numeroContact.Text);
    lesContacts.Add(c);
    confirmLabel.Text = String.Format("{0} avec le numéro {1}",
                                     nomContact.Text, numeroContact.Text);
    this.saveContacts();
}

public void saveContacts()
{
    string toSaveText = "";
    foreach(Contact c in this.lesContacts)
    {
        toSaveText += c.Nom + ";" + c.Numero + "\n";
    }
    DependencyService.Get<ISaveAndLoad>().SaveText("temp.txt", toSaveText);
}
```

A chaque ajout de contact, on va appeler la méthode saveContacts() qui va écrire à nouveau tous nos contacts dans le fichier texte (les informations nom et numéro y sont séparées par un point virgule). Il est idiot de réécrire tout le fichier à chaque ajout mais nous nous en contenterons pour ce premier exemple.

Passons à la partie chargement :

```
public void loadContacts()
{
    String recupere = DependencyService.Get<ISaveAndLoad>().LoadText("temp.txt");

    String[] lesLignesContact = recupere.Split(new[] { Environment.NewLine },
StringSplitOptions.None);
    foreach (var item in lesLignesContact)
    {
        if (item.Length > 0)
        {
            string[] leContact = item.Split(';');
            Contact c = new Contact(leContact[0], leContact[1]);
            lesContacts.Add(c);
        }
    }
}
```

On appellera cette méthode depuis le constructeur de la page principale afin que les contacts soient chargés dès le lancement de l'application.

1.8. De page en page

Nous allons maintenant découvrir comment naviguer entre plusieurs pages de l'application. Pour cela, nous allons créer une nouvelle page « Aide.xaml » avec le XAML suivant :

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="IntroAXamarin.Aide">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Bienvenue sur l'aide du logiciel - Composez le numéro du SAV pour toute question
..." />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Ajoutons un bouton sur la page principale qui permettra d'ouvrir cette page d'aide.

Pour pouvoir démarrer la navigation de page en page, il faut commencer par modifier le code du constructeur du fichier « App.cs » :

```
public App()
{
    InitializeComponent();

    MainPage = new NavigationPage(new MainPage());
    //MainPage = new IntroAXamarin.MainPage();
}
```

Créons ensuite un bouton sur le XAML de la page principale qui servira à ouvrir la nouvelle page.

```
<Button x:Name="boutonAide" Text="Aide" Clicked="OnButtonAideClicked"/>
```

Puis le code associé à l'évènement du clic sur ce bouton :


```
void OnButtonAideClicked(object sender, EventArgs e)
{
    Navigation.PushAsync(new Aide());
}
```

Testez le bon fonctionnement de l'application.

A noter que pour transmettre des informations d'une page à l'autre, l'une des techniques à disposition est de faire figurer ces données en tant qu'attributs statiques d'une des classes de l'application. On pourra ainsi les récupérer au besoin depuis n'importe quelle page.

A ce stade, n'oubliez pas de valider le source du projet avec github Desktop pour remonter votre code sur github.

2. Evolution du projet :

Les choses furent relativement simples jusqu'à présent puisque nous avons suivi pas à pas le début du développement du projet. Votre mission va se compliquer puisqu'on vous demande de réaliser les choses suivantes :

- Ajout d'informations sur les contacts : adresse mail et séparation du nom et du prénom dans 2 champs différents
- Ajout de la saisie des coordonnées GPS du contact : longitude et latitude
- Modification de la liste avec affichage seul du nom et prénom dans une nouvelle page
- Lors du clic sur un contact dans la liste, on ouvrira une boîte de dialogue pour afficher les détails et proposer la suppression du contact éventuelle.
- Evidemment, l'enregistrement dans le fichier txt devra tenir compte des nouvelles informations

Bon courage !