eBook Gratuit

APPRENEZ xamarin

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#xamarin

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xamarin

It is an unofficial and free xamarin ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official xamarin.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec xamarin

Remarques

Cette section fournit une vue d'ensemble de ce qu'est xamarin et pourquoi un développeur peut vouloir l'utiliser.

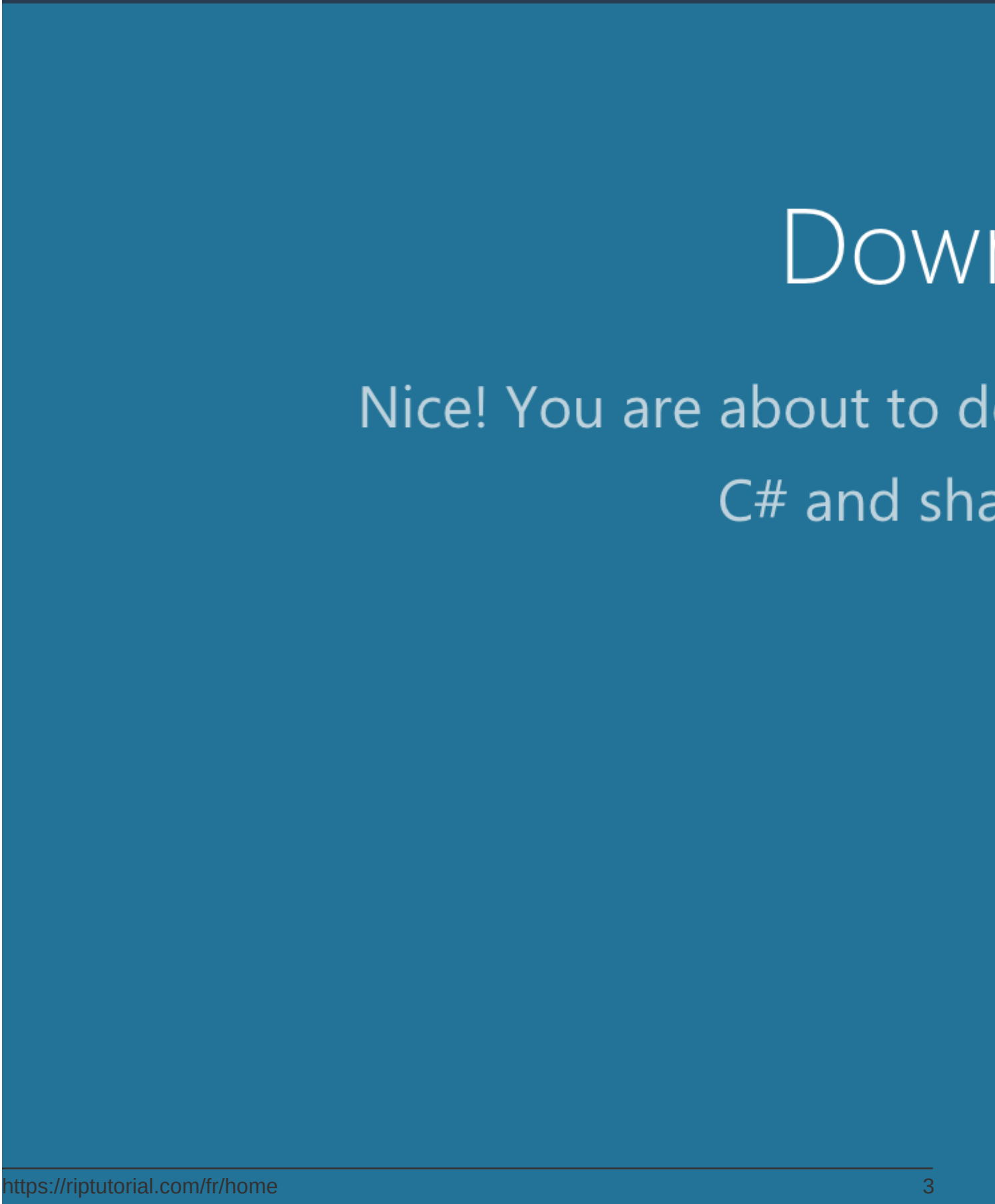
Il devrait également mentionner tous les grands sujets au sein de xamarin, et établir un lien avec les sujets connexes. La documentation de xamarin étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation de Xamarin Studio sur OS X

La première étape pour démarrer le développement de Xamarin sur une machine OS X consiste à télécharger et à installer la version de Xamarin Studio Community à partir du site officiel . Quelques champs doivent être remplis pour télécharger le programme d'installation, comme illustré ci-dessous.

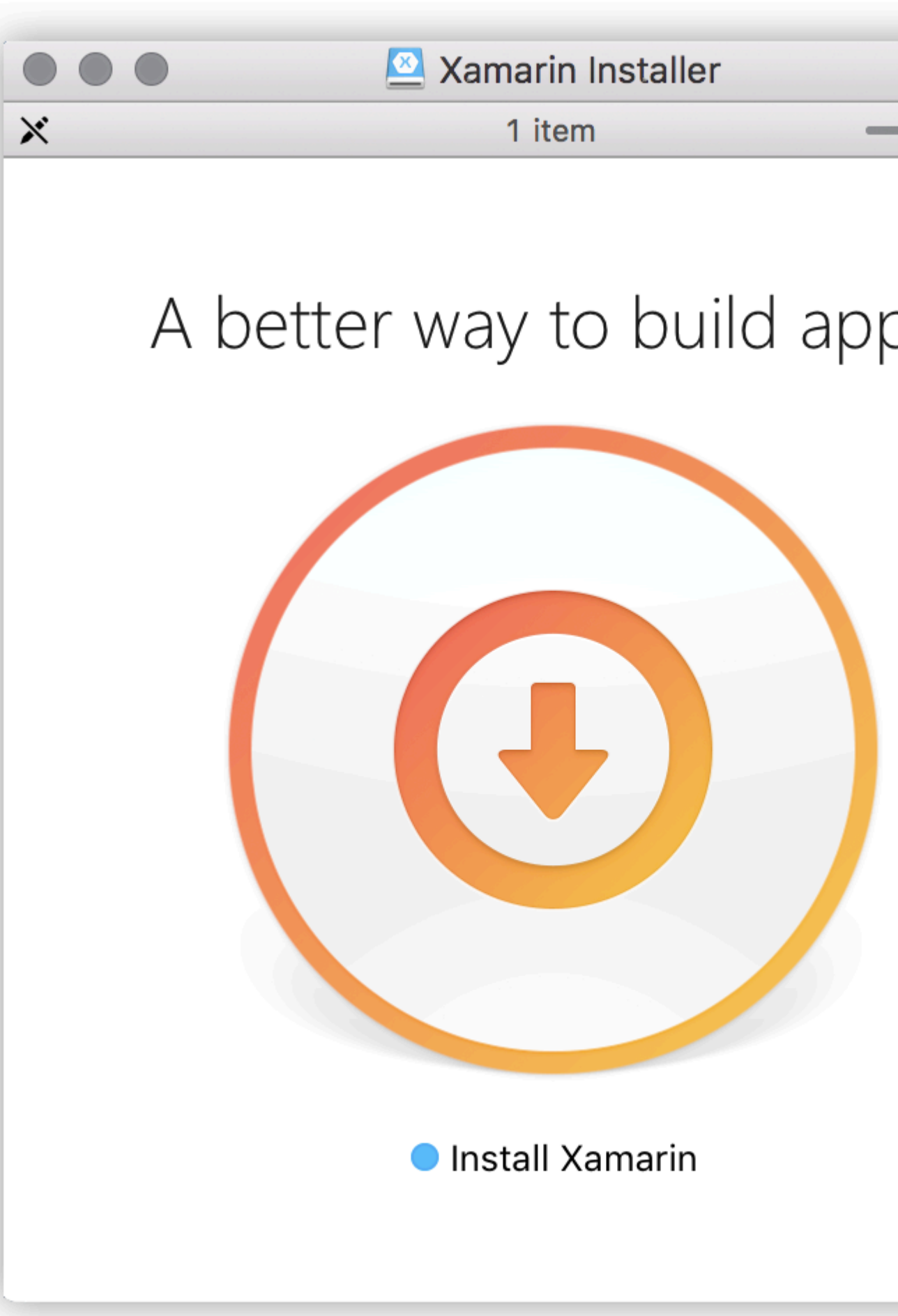

Xamarin
Products
Custom


Dowr
Nice! You are about to d
C# and sha

- La dernière version de Xcode sur le Mac App Store ou le site Web des développeurs Apple .
  site Web des développeurs Apple .
- Mac OS X Yosemite (10.10) et supérieur

Processus d'installation

Une fois les conditions préalables remplies, exécutez le programme d'installation Xamarin en double-cliquant sur le logo Xamarin.



~~OS X Gatekeeper peut afficher une boîte de dialogue vous demandant une confirmation pour ouv~~

https://riptutorial.com/fr/xamarin/topic/899/demarrer-avec-xamarin

Chapitre 2: Partage de code entre projets

Examples

Le pont

Le modèle Bridge est l'un des modèles de conception Inversion of Control les plus élémentaires. Pour Xamarin, ce modèle est utilisé pour référencer un code dépendant de la plate-forme à partir d'un contexte indépendant de la plate-forme. Par exemple: utiliser AlertDialog d'Android à partir d'une bibliothèque de classes portable ou de formulaires Xamarin. Aucun de ces contextes ne sait ce qu'est un objet AlertDialog, vous devez donc l'envelopper dans une boîte pour les utiliser.

```
// Define a common interface for the behavior you want in your common project (Forms/Other
PCL)
public interface IPlatformReporter
{
    string GetPlatform();
}


// In Android/iOS/Win implement the interface on a class
public class DroidReporter : IPlatformReporter
{
    public string GetPlatform()
    {
        return "Android";
    }
}


public class IosReporter : IPlatformReporter
{
    public string GetPlatform()
    {
        return "iOS";
    }
}


// In your common project (Forms/Other PCL), create a common class to wrap the native
implementations
public class PlatformReporter : IPlatformReporter
{
    // A function to get your native implemenation
    public static func<IPlatformReporter> GetReporter;

    // Your native implementation
    private IPlatformReporter _reporter;

    // Constructor accepts native class and stores it
    public PlatformReporter(IPlatformReporter reporter)
    {
        _reporter = GetReporter();
    }
```

```
    // Implement interface behavior by deferring to native class
    public string GetPlatform()
    {
        return _reporter.GetPlatform();
    }
}


// In your native code (probably MainActivity/AppDelegate), you just supply a function that
returns your native implementation
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.activity_main);

        PlatformReporter.GetReporter = () => { return new DroidReporter(); };
    }
}


public partial class AppDelegate : UIApplicationDelegate
{
    UIWindow window;

    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        window = new UIWindow(UIScreen.MainScreen.Bounds);
        window.RootViewController = new UIViewController();
        window.MakeKeyAndVisible();

        PlatformReporter.GetReporter = () => { return new IosReporter(); };

        return true;
    }
}


// When you want to use your native implementation in your common code, just do as follows:
public void SomeFuncWhoCares()
{
    // Some code here...

    var reporter = new PlatformReporter();
    string platform = reporter.GetPlatform();

    // Some more code here...
}
```

Le modèle de localisateur de services

Le modèle de conception du Service Locator est presque l'injection de dépendance. Comme le modèle de pont, ce modèle peut être utilisé pour référencer un code dépendant de la plate-forme à partir d'un contexte indépendant de la plate-forme. Plus intéressant encore, ce modèle repose sur le modèle singleton - tout ce que vous avez mis dans le localisateur de services sera un singleton de facto.

```
// Define a service locator class in your common project
public class ServiceLocator {
    // A dictionary to map common interfaces to native implementations
    private Dictionary<object, object> _services;

    // A static instance of our locator (this guy is a singleton)
    private static ServiceLocator _instance;

    // A private constructor to enforce the singleton
    private ServiceLocator() {
        _services = new Dictionary<object, object>();
    }

    // A Singleton access method
    public static ServiceLocator GetInstance() {
        if(_instance == null) {
            _instance = new ServiceLocator();
        }

        return _instance;
    }

    // A method for native projects to register their native implementations against the
common interfaces
    public static void Register(object type, object implementation) {
        _services?.Add(type, implementation);
    }

    // A method to get the implementation for a given interface
    public static T Resolve<T>() {
        try {
            return (T) _services[typeof(T)];
        } catch {
            throw new ApplicationException($"Failed to resolve type: {typeof(T).FullName}");
        }
    }


//For each native implementation, you must create an interface, and the native classes
implementing that interface
public interface IA {
    int DoAThing();
}


public interface IB {
    bool IsMagnificent();
}


public class IosA : IA {
    public int DoAThing() {
        return 5;
    }
}


public class DroidA : IA {
    public int DoAThing() {
        return 42;
    }
```

```
}


// You get the idea...


// Then in your native initialization, you have to register your classes to their interfaces
like so:
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle bundle)
    {
        base.OnCreate(bundle);
        SetContentView(Resource.Layout.activity_main);

        var locator = ServiceLocator.GetInstance();
        locator.Register(typeof(IA), new DroidA());
        locator.Register(typeof(IB), new DroidB());
    }
}


public partial class AppDelegate : UIApplicationDelegate
{
    UIWindow window;

    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        window = new UIWindow(UIScreen.MainScreen.Bounds);
        window.RootViewController = new UIViewController();
        window.MakeKeyAndVisible();

        var locator = ServiceLocator.GetInstance();
        locator.Register(typeof(IA), new IosA());
        locator.Register(typeof(IB), new IosB());

        return true;
    }
}


// Finally, to use your native implementations from non-native code, do as follows:
public void SomeMethodUsingNativeCodeFromNonNativeContext() {
     // Some boring code here


     // Grabbing our native implementations for the current platform
     var locator = ServiceLocator.GetInstance();
     IA myIA = locator.Resolve<IA>();
     IB myIB = locator.Resolve<IB>();


    // Method goes on to use our fancy native classes
}
```

Lire Partage de code entre projets en ligne: https://riptutorial.com/fr/xamarin/topic/6183/partage-de-code-entre-projets

Chapitre 3: Validation d'objet par des annotations

Introduction

mvc.net introduit des anotations de données pour la validation du modèle. Cela peut également être fait dans Xamarin

Examples

Exemple simple

Ajouter le package nuget `System.ComponentModel.Annotations`

Définir une classe:

```
public class BankAccount
{

   public enum AccountType
   {
       Saving,
       Current
   }

   [Required(ErrorMessage="First Name Required")]
   [MaxLength(15,ErrorMessage="First Name should not more than 1`5 character")]
   [MinLength(3,ErrorMessage="First Name should be more than 3 character")]
   public string AccountHolderFirstName { get; set; }

   [Required(ErrorMessage="Last Name Required")]
   [MaxLength(15,ErrorMessage="Last Name should not more than 1`5 character")]
   [MinLength(3,ErrorMessage="Last Name should be more than 3 character")]
   public string AccountHolderLastName { get; set; }

   [Required]
   [RegularExpression("^[0-9]+$", ErrorMessage = "Only Number allowed in AccountNumber")]
   public string AccountNumber { get; set; }

   public AccountType AcType { get; set; }
}
```

Définir un validateur:

```
public class GenericValidator
{
    public static bool TryValidate(object obj, out ICollection<ValidationResult> results)
    {
        var context = new ValidationContext(obj, serviceProvider: null, items: null);
        results = new List<ValidationResult>();
        return Validator.TryValidateObject(
```

```
            obj, context, results,
            validateAllProperties: true
        );
    }
}
```

utilisez le validateur:

```
var bankAccount = new BankAccount();
ICollection<ValidationResult> lstvalidationResult;

bool valid = GenericValidator.TryValidate(bankAccount, out lstvalidationResult);
if (!valid)
{
    foreach (ValidationResult res in lstvalidationResult)
    {
        Console.WriteLine(res.MemberNames +":"+ res.ErrorMessage);
    }

}
Console.ReadLine();
```

Sortie générée:

```
First Name Required
Last Name Required
The AccountNumber field is required.
```

Lire Validation d'objet par des annotations en ligne:
https://riptutorial.com/fr/xamarin/topic/9720/validation-d-objet-par-des-annotations

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec xamarin	Akshay Kulkarni, Community, Gil Sand, hankide, Joel Martinez, Marius Ungureanu, Sven-Michael Stübe, thomasvdb
2	Partage de code entre projets	kellen lask, valdetero
3	Validation d'objet par des annotations	Niek de Gooijer