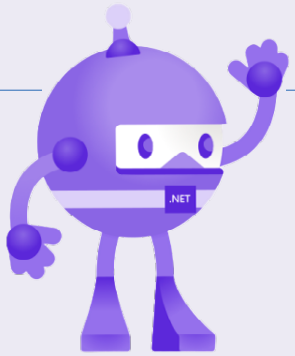


MAUI

programmation pour PC, tablettes et smartphones

le nouveau framework en théorie et en pratique

Dr. Veikko Krypczyk (Allemagne)

En développant une application PC, par exemple pour la simulation ou la capture de données, il est de plus en plus indispensable que l'application soit également disponible pour les appareils mobiles. Désormais, avec .NET MAUI, nous disposons d'un outil de développement intéressant qui permet aux développeurs de combler les écarts entre les plates-formes de manière transparente.

Les électroniciens ont souvent besoin de programmer non seulement des microcontrôleurs, mais aussi des applications compagnons pour les PC ou les appareils mobiles. Un exemple courant est le développement d'une application mobile qui vous permet de contrôler du matériel à distance grâce à des données cellulaires ou à une connexion Bluetooth Low Energy (BLE). Par exemple, vous pouvez contrôler les lumières extérieures de votre propriété, qui sont équipées d'une interface basée sur un microcontrôleur. La communication entre l'interface et un smartphone peut se faire via BLE. Il vous suffit de développer une application programmée sur mesure pour l'exécuter sur votre smartphone. C'est précisément l'objet de cet article. Nous allons explorer un nouveau cadre avancé appelé .NET MAUI [1] pour créer des applications pour différents appareils et systèmes d'exploitation.

Aperçu de .NET MAUI

Comme son nom l'indique, le cadre .NET MAUI repose sur le cadre .NET multiplateforme à partir de la version 6.0. Il permet de programmer sous Windows ou macOS en utilisant l'environnement de développement intégré Visual Studio. L'édition gratuite *Community Edition* répondra à nos besoins. Pour le développement sous Windows, vous viserez les systèmes Windows et Android. Pour créer des applications pour iOS et macOS, l'accès à un PC Mac sera nécessaire pour générer les paquets d'applications finaux. Cette contrainte n'est pas spécifique à .NET MAUI, mais s'applique en général au développement d'applications pour l'environnement iOS et macOS. En revanche, si vous développez sous macOS (*Visual Studio for Mac*), vous pouvez créer directement des applications pour tous les systèmes, à l'exception de Windows.
Nous expliquerons la configuration de l'environnement de développement, puis nous passerons en revue les étapes de la création d'un exemple d'application. Les applications pour tous les systèmes cibles sont générées à partir d'une base de code partagée, en C# pour la logique du programme et XAML pour l'interface utilisateur. Le cadre garantit que l'application fonctionne sur l'appareil correspondant et qu'elle soit adaptée, le cas échéant, à l'environnement du système (**figure 1**).

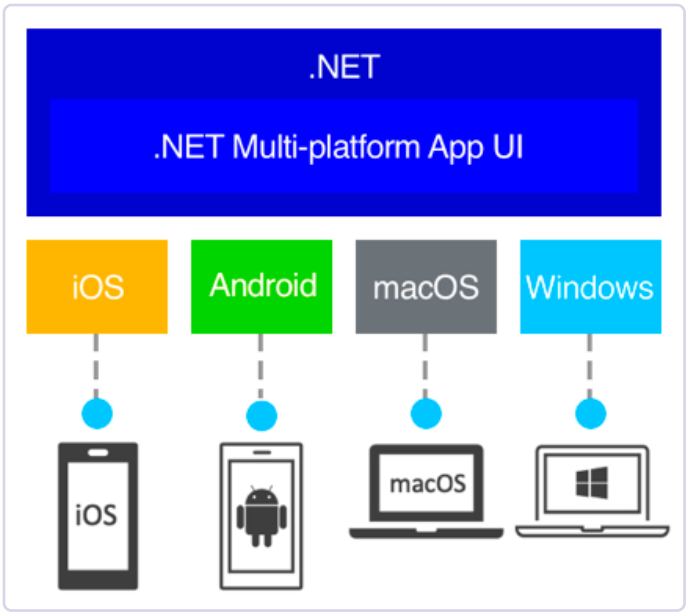


Figure 1. .NET MAUI génère une interface utilisateur multiplateforme (Source : Microsoft).

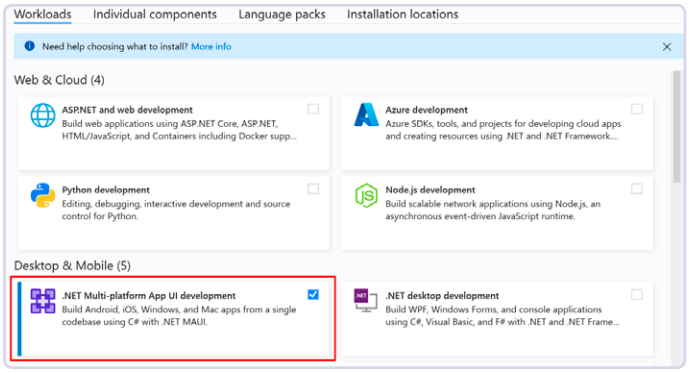


Figure 2. Sélectionnez le Workload .NET MAUI à installer.

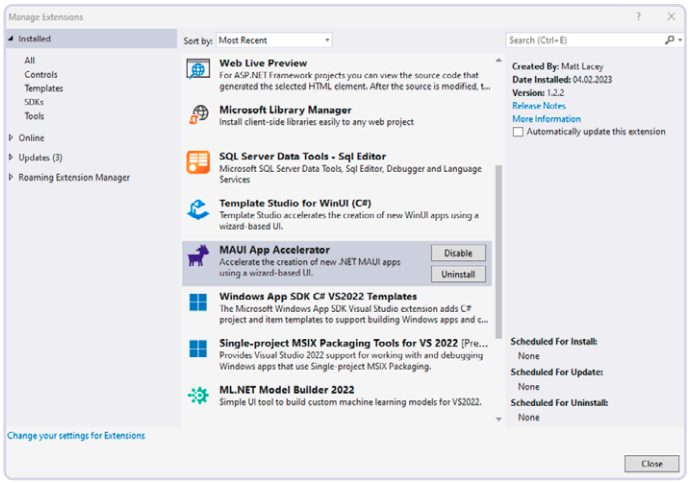


Figure 3. Installation de l'extension MAUI App Accelerator.

Vous n'aurez pas à vous préoccuper (au moins au début) de l'aspect et du fonctionnement des interfaces utilisateur sous Windows, macOS ou Android. Au lieu de cela, vous définissez l'interface utilisateur à un niveau fondamental avec XAML, et MAUI se charge de restituer correctement le bouton, le champ de texte et d'autres éléments sur les systèmes cibles. Cela ne vous empêche pas de procéder à des modifications individuelles pour chaque plateforme cible, comme l'utilisation de couleurs, d'icônes, d'espacements différents, et ainsi de suite, si vous le souhaitez, mais dans l'idéal, ces personnalisations seront minimes, voire inutiles.

Le même principe s'applique à l'utilisation des caractéristiques du matériel et de l'appareil. En particulier sur les appareils mobiles, les données provenant d'un capteur tel qu'un appareil photo, un GPS, etc. peuvent souvent jouer un rôle important. Chaque système d'exploitation utilise ces fonctions système différemment, généralement grâce à des bibliothèques de programmes dédiées (API). .NET MAUI intègre également ces bibliothèques système et fournit une interface de programmation générique destinée à un large éventail de fonctionnalités du système et de l'appareil. Cela signifie que vous n'avez qu'à programmer en fonction de cette interface multiplateforme et que vous n'avez pas à vous préoccuper de la manière d'utiliser les capteurs et d'autres fonctionnalités sur chaque système cible spécifique. Si une telle interface de programmation générique n'existe pas (encore), vous pouvez également accéder aux API spécifiques de chaque système à l'aide d'un code propre à la plateforme. Toutefois, cette méthode devrait être l'exception plutôt que la norme.

Bien que .NET MAUI soit encore relativement nouveau (il est sorti pour la première fois fin 2022), il est le successeur technologique de Xamarin, qui a bénéficié du soutien d'une communauté en ligne importante et active. La communauté propose activement des composants et des bibliothèques pour l'expansion et fournit une assistance étendue sous forme de documentation, d'articles de blog, de forums de discussion, etc.

Configuration de l'environnement de développement

La procédure est simple. Nous supposerons que vous travaillez sous Microsoft Windows. Pour commencer, assurez-vous d'avoir installé les mises à jour les plus récentes de Windows 10 ou 11. Ensuite, installez Visual Studio [2]. Dans l'assistant d'installation, sélectionnez le *Workload* appelée *.NET Multi-platform App UI development* (**figure 2**).

Vous pouvez accéder à l'assistant d'installation ultérieurement si vous avez besoin d'installer des composants supplémentaires. Nous disposons déjà d'exemples de projet pour développer une application .NET MAUI. Cependant, la création d'un nouveau projet est simplifiée grâce à une extension supplémentaire. Lancez Visual Studio et cliquez sur *Continue without code* en bas à droite de la boîte de dialogue de démarrage. Visual Studio démarre alors sans projet actif. Allez dans le menu *Extensions* et sélectionnez *Manage Extension*. Recherchez l'extension *MAUI App Accelerator* (**figure 3**) et installez-la.

Une fois l'installation terminée, Visual Studio doit être redémarré. Nous disposons à présent d'un assistant qui nous permet de créer la structure de base d'une application .NET MAUI en suivant un dialogue guidé. Nous pouvons le démontrer à l'aide d'un exemple spécifique.

Téléchargement

Le code source complet et d'autres informations (captures d'écran, schémas de câblage) sont disponibles sur le dépôt GitHub de l'auteur [4].

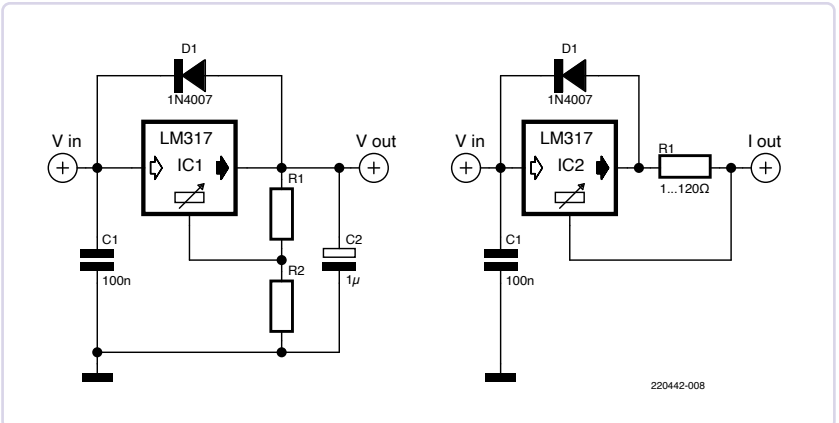


Figure 4. Schémas de circuit pour le régulateur de tension LM317.

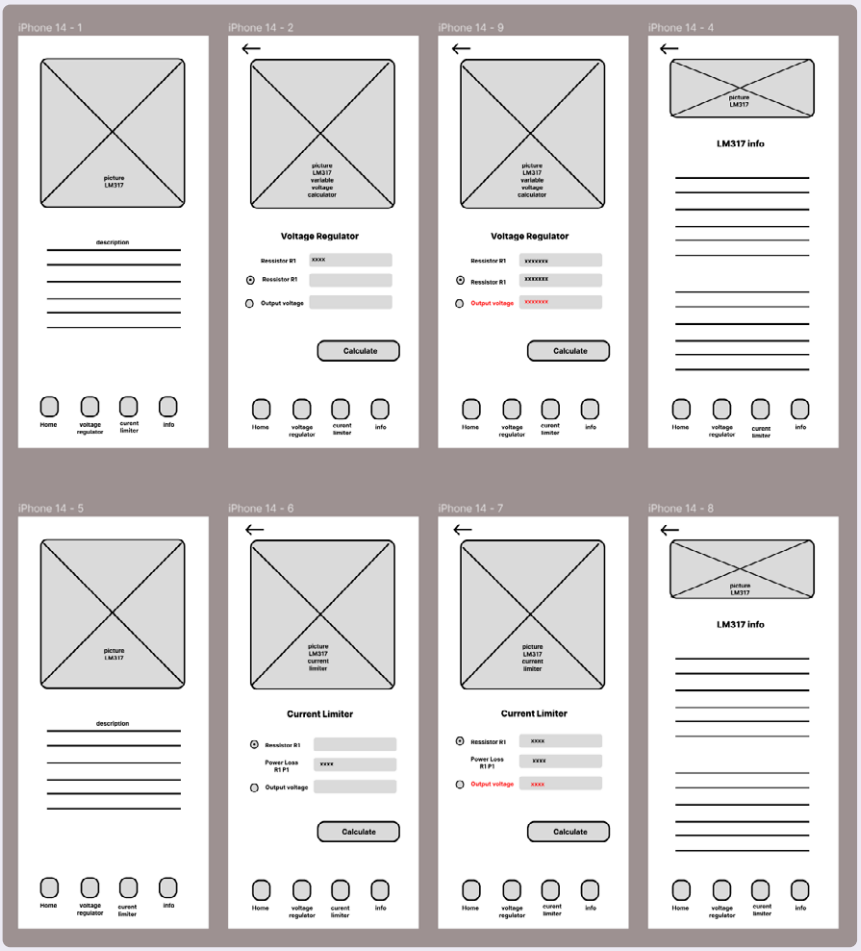
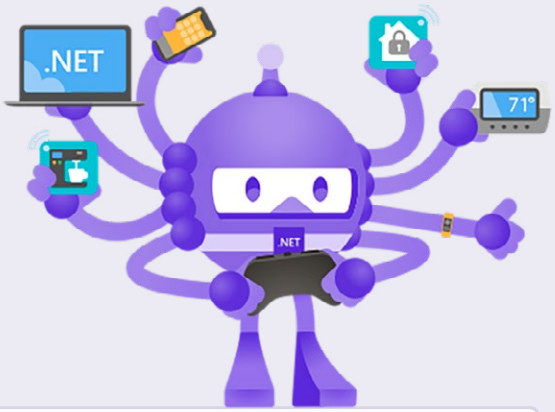


Figure 5. Wireframe pour l'application de calcul (iOS ou Android).



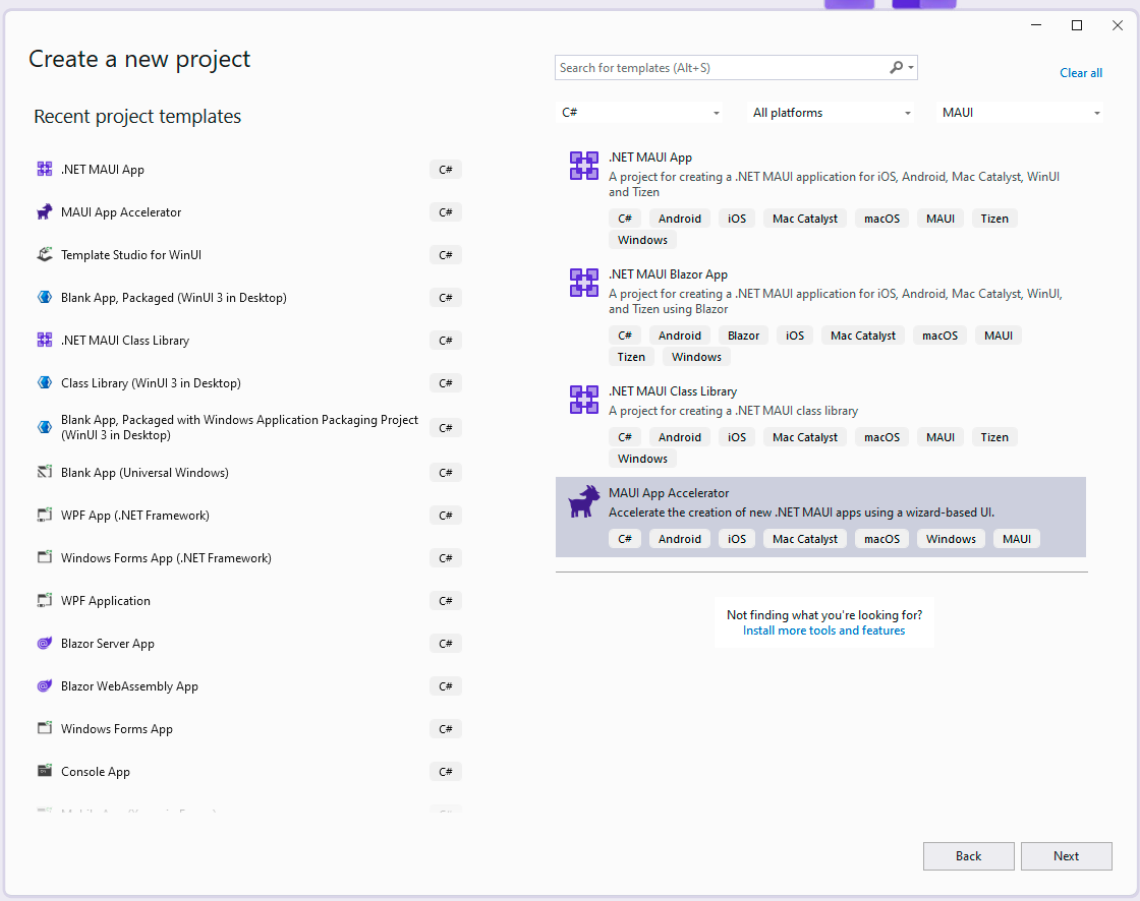


Figure 6. Sélectionnez le modèle MAUI App Accelerator pour le démarrage du projet.

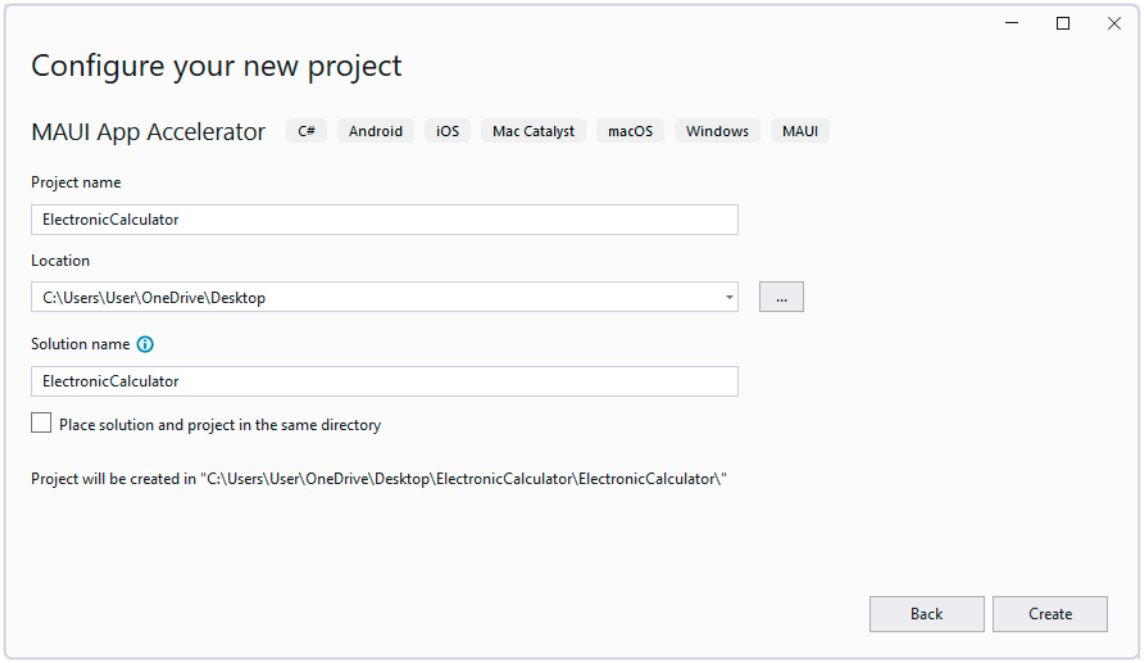


Figure 7. Saisissez le nom du projet et l'emplacement de stockage.

Exemple

Nous souhaitons développer une application mobile pour Android et iOS. Nous allons créer une application de capture de données et de calcul pour le fameux régulateur de tension LM317. La plupart de nos lecteurs devraient être familiers avec le fonctionnement et l'utilisation du LM317. Nous pouvons utiliser ce régulateur de tension pour générer soit une tension de sortie constante, soit une source de courant constant (**figure 4**).
Pour calculer la tension de sortie générée, nous pouvons appliquer la formule pratique simplifiée suivante :

$$V_{out} = 1.25 \cdot \left(1 + \frac{R_2}{R_1}\right)$$

Nous pouvons modifier cette équation pour calculer R_2 et déterminer la valeur de la résistance permettant de fournir une tension de sortie spécifique. Selon la fiche technique du LM317, la valeur de la résistance R_1 est de 240 Ω. Il est également possible d'utiliser ce circuit intégré régulateur pour créer une source de courant constant, avec la relation suivante :

$$I_{out} = \frac{1.25}{R_1}$$

Dans ce cas, la valeur de R_1 doit être comprise dans l'intervalle $1\,\Omega \le R_1 \le 120\,\Omega$. Bien qu'il existe déjà de nombreux outils de calcul pour ce régulateur de tension, il est pratique d'en créer un ici comme démonstration de la fonctionnalité de construction d'applications avec .NET MAUI.

Design

L'une des principales erreurs de débutant consiste à se lancer directement dans la programmation alors que l'on n'a pas encore défini de plan. Il est toujours utile d'avoir un prototype visuel, même s'il ne s'agit que d'un croquis, (**figure 5**).
Ce prototype représente une maquette pour une application mobile (iOS ou Android). Nous utiliserons quatre pages accessibles par des onglets de navigation situés dans la partie inférieure des pages. Ces quatre pages devraient fournir le contenu suivant :

- *Accueil* : schéma du circuit et informations générales sur le régulateur de tension LM317.
- *Régulateur de tension* : schéma du circuit et champs de saisie pour les résistances R_1 et R_2, ainsi que la tension de sortie souhaitée (U_{Out}). Des boutons radio permettent de choisir si le calcul doit être effectué pour R2 ou U_{Out}. Un bouton permettant de lancer le calcul.
- *CCR* = régulateur à courant constant : pour un fonctionnement à courant constant, nous disposons de champs d'entrée pour la résistance R1 et le courant de sortie. Des boutons radio vous permettant de choisir de calculer la valeur de R1 ou de I_{Out}. Un bouton permettant également d'exécuter le calcul.
- *Info* : nous affichons ici la fiche technique complète du régulateur de tension, par exemple, sur une page web.

Ce concept constitue la base de la mise en œuvre de l'application.

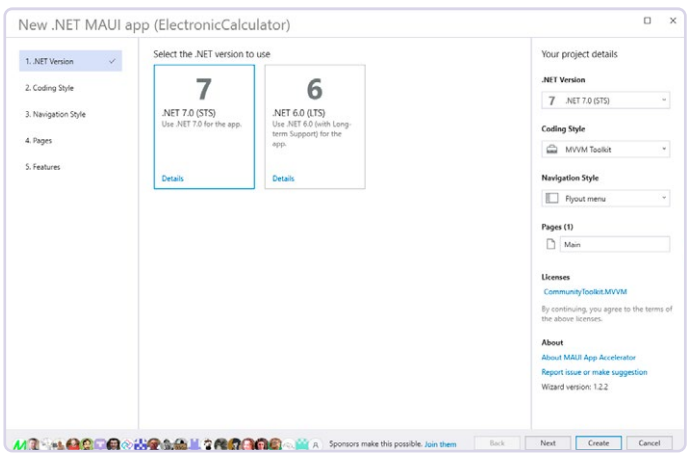


Figure 8. Sélectionnez la version .NET et la navigation.

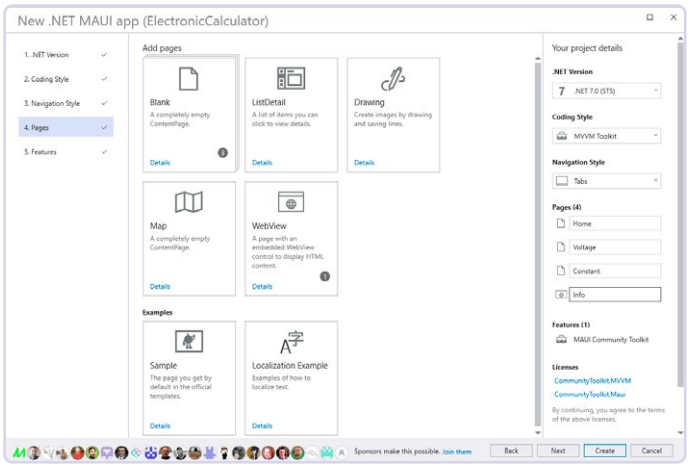


Figure 9. Définissez la structure et les pages de l'application.

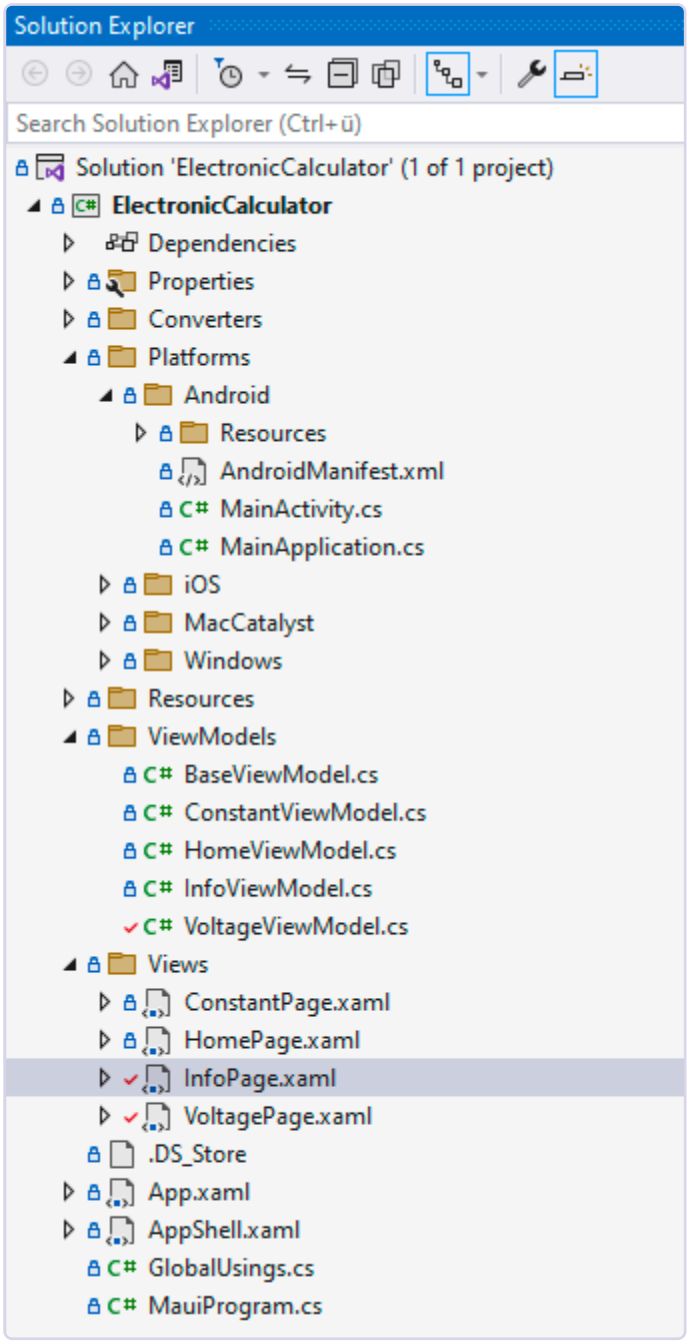


Figure 10. Structure du projet d'application MAUI.

Commencer le projet

Après avoir redémarré Visual Studio, sélectionnez le modèle *MAUI App Accelerator* (**figure 6**). À l'étape suivante, vous devez spécifier le nom de l'application et son emplacement ; voir (**figure 7**). Ensuite, vous serez invité à choisir la version de .NET (framework) - utilisez la plus récente, .NET 7.0 au moment de la rédaction (**figure 8**).). Dans l'étape suivante (**figure 9**), indiquez les informations suivantes :

- *Style de codage* : le modèle architectural de l'application est défini ici. Choisissez *MVVM-Toolkit*, ce qui signifie que vous intégrez une bibliothèque correspondante dans le projet et que vous utilisez le modèle architectural MVVM. Ce modèle garantit le découplage des couches (*Model*, *View*, *ModelView*) et est considéré comme la norme pour la mise en œuvre d'applications dont l'interface utilisateur est définie avec XAML. Vous pouvez trouver plus d'informations à ce sujet sur le site [3].
- *Style de navigation* : choisissez *Tabs* parmi les options disponibles pour utiliser des onglets pour la navigation.
- *Pages* : ici, vous pouvez ajouter trois pages de type *Blank* et les nommer *Home* (Accueil), *Voltage* (Tension) et *Constant* (Constante). De plus, ajoutez une page de *WebView* et nommez-la *Info*.

Une fois ces détails complétés, vous pouvez laisser Visual Studio créer l'application. Ce processus peut prendre un peu de temps. Ensuite, vous accéderez à l'environnement de développement avec votre projet ouvert.
En examinant la structure du projet (**figure 10**), nous pouvons faire les observations suivantes :

- *Multi-Targeting* : il n'y a qu'un seul projet pour toutes les plateformes, avec un dossier distinct pour chaque système cible.
- *ViewModels* : contient les classes de la logique du programme et sert de lien avec l'interface utilisateur.
- *Views* : classes composées de fichiers C# et XAML pour l'interface utilisateur.
- *App.xaml*, *MauiProgram.cs* : points d'entrée pour l'interface utilisateur et l'exécution de l'application.
- *Resources* : contient des images et d'autres ressources pour l'application.

Sans plus de programmation, vous pouvez maintenant lancer l'application pour la première fois (**figure 11**). Si nécessaire, vous pouvez configurer un émulateur Android sur votre système pour le premier lancement.

Créer l'interface utilisateur

L'interface utilisateur est déclarée en XAML, plus précisément dans les fichiers *View*. Prenons l'exemple de la page du deuxième onglet. Tout d'abord, il convient de définir la mise en page. Pour ce faire, il existe différents conteneurs de mise en page, tels que Grid, qui permet de définir les lignes et les colonnes. Dans une telle grille, les éléments individuels de l'interface utilisateur, tels qu'une image, un bouton ou une zone de texte, peuvent être placés dans des cellules. D'autres conteneurs de présentation permettent une disposition verticale ou horizontale des éléments. Le principe

consiste à utiliser autant que possible un positionnement et un espacement relatifs plutôt que des valeurs absolues. Cela est nécessaire pour garantir que l'interface utilisateur s'affiche correctement sur un maximum d'appareils ayant des tailles d'écran et des résolutions différentes.
Par exemple, notre page se compose d'une grille avec deux lignes au niveau supérieur, divisant la page en deux zones de taille égale (50:50). Dans la zone supérieure, nous plaçons le schéma du circuit (image). Dans la zone inférieure, nous plaçons les champs de texte pour la saisie, les champs d'étiquette pour les légendes, les boutons radio et un bouton dans une grille régulière. On peut adapter les différents éléments en termes de taille, d'alignement et de conception. Un extrait du code source de la page est présenté dans le **listage 1**.

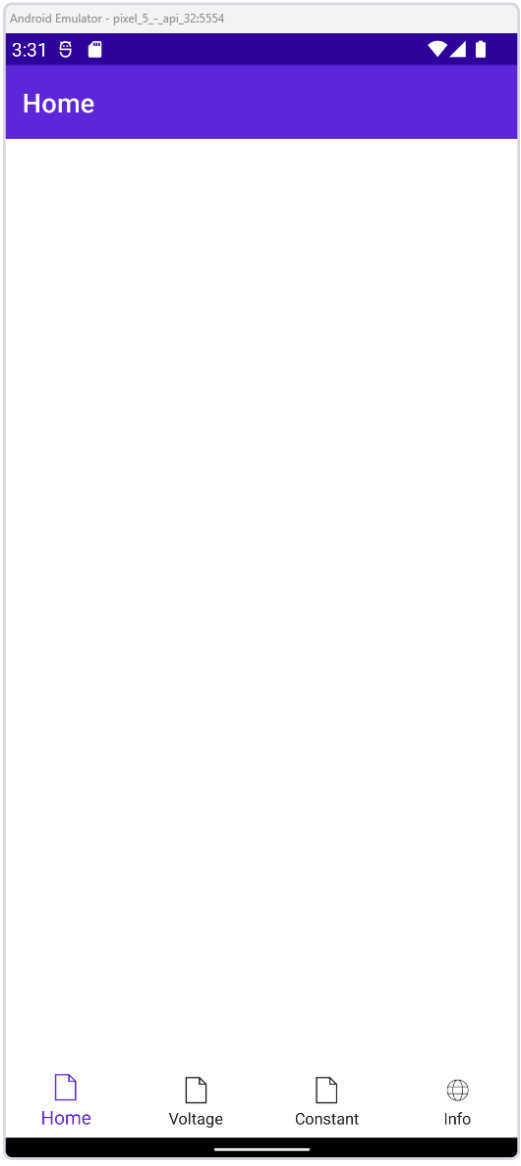


Figure 11. Premier lancement de l'application avec l'émulateur Android.

Listage 1.
Extrait du code source de la définition de la page.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    <Grid BackgroundColor="LightGray">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <VerticalStackLayout
            Grid.Row="1"
            HorizontalOptions="Center"
            VerticalOptions="Center">
            <Grid RowSpacing="20">
                <Grid.RowDefinitions>
                    <RowDefinition />
                    <RowDefinition />
                    <RowDefinition />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition />
                    <ColumnDefinition />
                </Grid.ColumnDefinitions>
                <Label
                    Margin="30,0"
                    HorizontalOptions="Start"
                    MaximumWidthRequest="100"
                    Text="Resistor R1 (240 Ohm):"
                    VerticalOptions="Center" />
                <Entry
                    Grid.Column="1"
                    HorizontalTextAlignment="Center"
                    Text=""
                    VerticalOptions="Center" />
                ...
                <Button
                    Grid.Row="3"
                    Grid.Column="1"
                    Command=""
                    Text="Calculation" />
            </Grid>
        </VerticalStackLayout>
        <VerticalStackLayout Margin="20"
         VerticalOptions="Center">
            <Border>
                <Border.StrokeShape>
                    <RoundRectangle CornerRadius="20" />
                </Border.StrokeShape>
                <Image Aspect="AspectFit" Source="lm317c.jpg" />
            </Border>
        </VerticalStackLayout>
    </Grid>
</ContentPage>
```

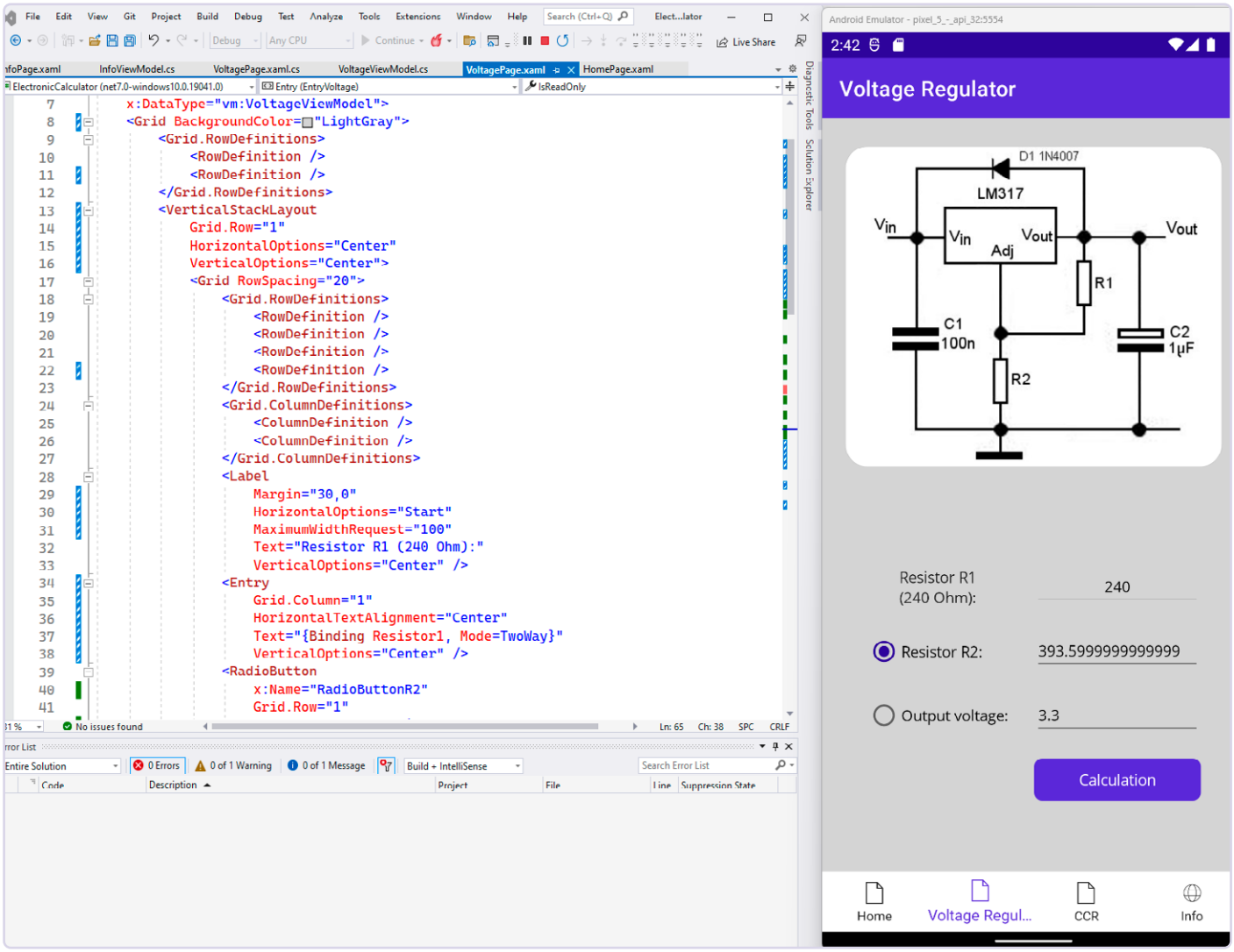


Figure 12. Le rechargement à chaud est idéal pour une construction rapide de l'interface utilisateur.

Pour mieux comprendre, vous pouvez lancer l'application sur un émulateur et placer le code source de cette page dans Visual Studio, côte à côte sur l'écran. En mode *Debug*, la fonction *Hot Reload* permet d'apporter des modifications au code XAML, qui seront immédiatement appliquées à l'application en cours d'exécution, c'est-à-dire qu'elles seront appliquées après l'enregistrement, sans qu'il soit nécessaire de redémarrer (!). Cela fonctionne également sur un appareil mobile et remplace un concepteur graphique (**figure 12**). Ainsi, vous pouvez commenter des éléments ou ajuster des propriétés dans le code XAML et voir immédiatement les résultats dans l'application.

Toutes les pages de l'application ont été créées de cette façon. Vous pouvez examiner le code source des fichiers d'affichage (XAML) pour plus de détails. La page d'info ne contient qu'un élément *WebView*. Ici, nous pouvons afficher un fichier HTML. Nous avons établi un lien statique vers une page web où l'on peut consulter la fiche technique du régulateur de tension. Il aurait également été possible d'intégrer un contrôle permettant d'afficher un fichier PDF.

Créer la logique du programme

Il s'agit des étapes ou des instructions qui dictent la manière dont les valeurs souhaitées pour la résistance et la tension de sortie sont calculées. Les concepts de base se composent des éléments suivants :

- **Variables :** les variables sont utilisées pour stocker et gérer les valeurs des données d'entrée. Cela se fait dans les fichiers *ViewModel* correspondants. Par exemple, dans le fichier *VoltageViewModel* :

```
[ObservableProperty]
private double voltage;

[ObservableProperty]
private double resistor1 = 240;

[ObservableProperty]
private double resistor2;

[ObservableProperty]
private bool voltageMode = true;
```

Les attributs de la variable indiquent sa visibilité (`private`), son type de données (`double`) et son nom (`resistor1`). En utilisant l'attribut `[ObservableProperty]`, l'environnement de développement est chargé de convertir automatiquement la variable locale en une caractéristique `public`. Cela permet également d'accéder à la valeur depuis l'extérieur de la classe.

- **Liaison des données :** les éléments des champs de l'interface utilisateur, tels que le champ de saisie des données (*Entry*), sont liés à des propriétés dans le code C#. Cela se fait directement dans le fichier XAML correspondant. En voici un exemple :

```
x:Name=»EntryR2»
Grid.Row=»1»
Grid.Column=»1»
MinimumWidthRequest=»150»
Text=»{Binding Resistor2, Mode=TwoWay}»
VerticalOptions=»Center» />
```

Il s'agit de la définition du champ de saisie de texte (*Entry*) pour la valeur de la résistance R2. La propriété `Text` est liée à la propriété définie (comme mentionné ci-dessus) avec la syntaxe de liaison. `Mode = TwoWay` indique que la liaison de données est bidirectionnelle, de sorte que les modifications apportées au champ de saisie (saisie de l'utilisateur) mettront à jour le code source sous-jacent et que les modifications apportées à la variable (après calcul) mettront également à jour la vue.

- **Commandes :** dans le code source, une méthode de calcul est définie, par exemple pour la tension de sortie ou la valeur de la résistance :

```
[RelayCommand]
private void Calculate()
{
 if (voltageMode)
 {
  Voltage = 1.25 * (1 + Resistor2 / Resistor1);
 }
 else
 {
  Resistor2 = Resistor1 * (Voltage / 1.25 - 1);
 }
}
```

L'ajout de l'attribut `[RelayCommand]` permet d'appeler la méthode en tant que commande directement à partir du fichier XAML, dans ce cas, avec le bouton. La propriété `Command` lui est liée.

```
Grid.Row=»3»
Grid.Column=»1»
Command=»{Binding CalculateCommand}»
Text=»Calculation» />
```

Ce sont les fonctions essentielles de l'application. Pour une meilleure compréhension, étudiez le code source dans Visual Studio et testez-le.

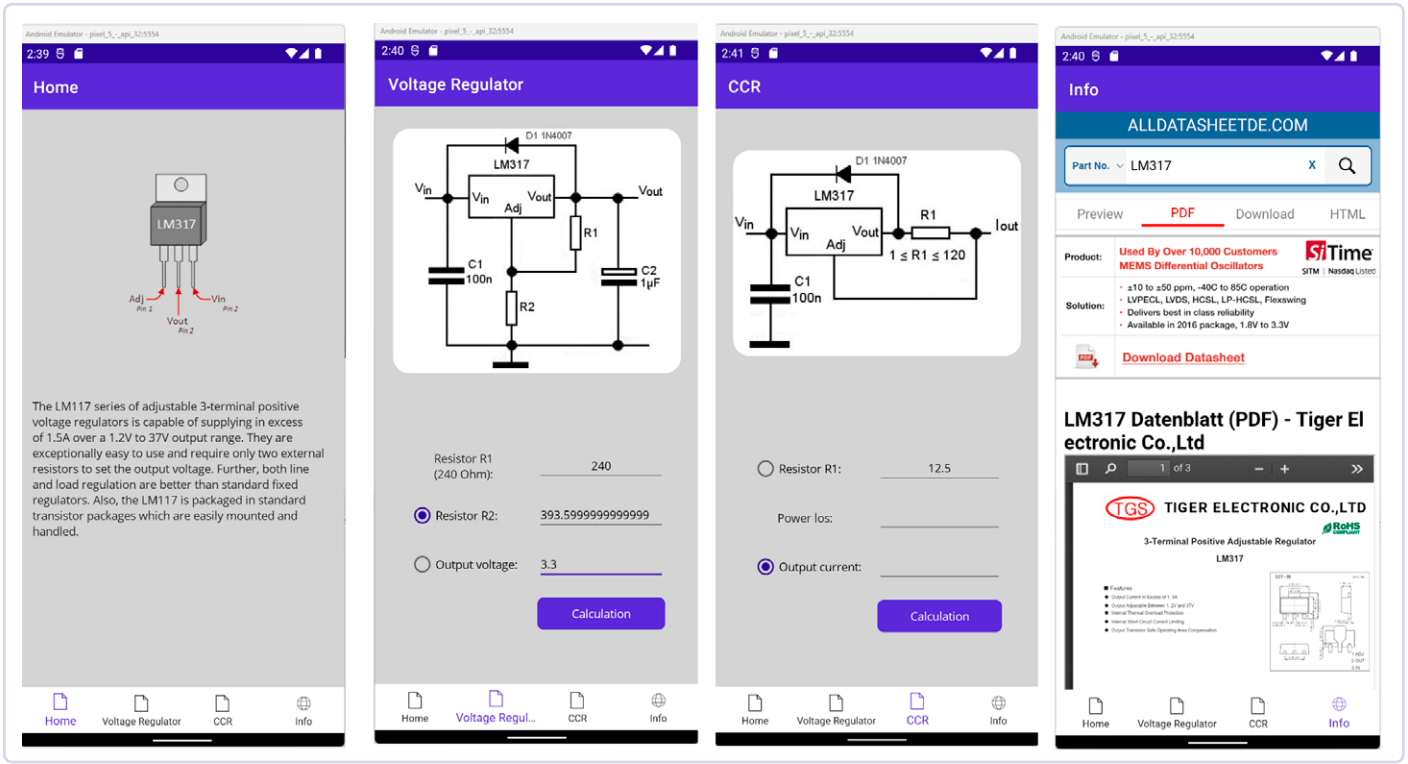


Figure 13. Première version de l'application avec la fonction de calcul représentée sur l'émulateur Android.

Résultats

Le résultat est une application mobile, initialement conçue pour Android (**figure 13**). Nous pouvons utiliser les onglets pour naviguer entre les pages et voir les calculs. Il est également possible de consulter la fiche technique. L'application a été initialement visualisée avec un émulateur, mais on peut relier un appareil Android au PC de développement via le port USB ou wifi pour installer l'application sur l'appareil. L'appareil Android nécessite l'activation du mode développeur avant le téléchargement du fichier APK.

Autres plateformes

Avec .NET MAUI, vous pouvez créer des applications natives pour différentes plateformes. L'exemple présenté ici est adapté à une application mobile (utilisant des onglets au contenu limité). Si vous codez sous Mac, avec la dernière version de l'environnement de développement Xcode, vous pouvez compiler l'application pour iOS directement depuis Visual Studio et l'exécuter sur un simulateur d'iPhone ou d'iOS. Des modifications mineures de la conception, telles que les couleurs, peuvent s'avérer nécessaires. Pour réduire les risques de problèmes de compatibilité, il est préférable de définir aussi peu d'éléments statiques et absolus que possible, en laissant le système se charger de l'alignement et de la conception conformément aux lignes directrices en la matière, ou de définir des caractéristiques spécifiques pour chaque système cible, le cas échéant. Si vous colorez explicitement l'arrière-plan d'une page et que vous y placez des champs de saisie de texte, cela peut sembler correct sur une plate-forme, mais le champ de texte peut avoir un style différent sur une autre, ce qui entraîne une mauvaise adaptation. La règle est donc la suivante : gardez la valeur par défaut ou procédez à des modifications pour les plates-formes sélectionnées. Toutefois, il est possible de partager l'intégralité du code source de l'interface utilisateur et de la logique de gestion ou de la fonctionnalité de base, ce qui vous épargne beaucoup de travail dans la plupart des cas.

Modifications

L'application n'est qu'un prototype fonctionnel. Vous devez apporter les modifications suivantes avant de distribuer l'application aux utilisateurs :

- Remplacez tous les graphiques par des images au format *.svg*. Il s'agit de graphiques vectoriels qui se mettent automatiquement à l'échelle sans perte, en fonction du système cible.
- Déterminez toutes les icônes pour la barre d'onglets au bas de l'écran. Les graphiques nécessaires sont stockés dans le dossier *Resources/Images* et affectés au fichier *AppShell.xaml*.
- Validez la plage d'entrée des résistances, la tension de sortie, etc., pour assurer des valeurs raisonnables (plage de valeurs) et mettez en œuvre une gestion des erreurs en cas d'entrées incorrectes. Cela se fait dans les fichiers *ViewModel*.
- Effectuez des tests complets sur différents appareils avec des émulateurs et sur les appareils physiques.
- Créez des paquets d'applications pour les plateformes cibles, par exemple Android et iOS.
- Enregstrerz les applications dans les magasins d'applications respectifs.

Autres fonctions de .NET MAUI

Dans cet exemple, nous n'avons abordé qu'une petite partie des fonctionnalités offertes par .NET MAUI pour la création d'applications destinées à des plates-formes multiples. Il convient également de mentionner les fonctionnalités suivantes, qui présentent un intérêt particulier dans le contexte de la création d'applications pour les projets électroniques :

- **Accès aux fonctions de l'appareil et aux capteurs :** .NET MAUI offre un accès intégré à une variété de fonctions du matériel. Dans la plupart des cas, vous devez demander les autorisations nécessaires dans les fichiers de la plate-forme avant de pouvoir accéder aux fonctions de l'appareil via une interface universelle.
- **Bibliothèques pour les fonctions du système :** des fonctions supplémentaires peuvent être fournies par des bibliothèques externes pour .NET ; elles peuvent être utiles pour au matériel spécifique, à des capteurs, etc. Ces bibliothèques sont fournies par des fournisseurs tiers et par la communauté. Cela vaut la peine de les découvrir.
- **Composants d'interface utilisateur :** .NET MAUI offre déjà par défaut une gamme de composants visuels. La sélection est continuellement enrichie par des fournisseurs tiers et par la communauté.
- **Composants d'interface utilisateur :** .NET MAUI offre déjà par défaut une gamme de composants visuels. La sélection est continuellement enrichie par des fournisseurs tiers et par la communauté.
- **Déploiement d'applications :** aujourd'hui, les applications à usage général sont disponibles sur les boutiques Google Play ou Apple App, en fonction de la plateforme cible. Les paquets d'applications peuvent être créés directement à partir de Visual Studio. Notez que vous devez vous conformer aux politiques appropriées de l'entreprise et que l'application doit être correctement signée par un certificat numérique. Pour iOS et macOS, un Mac sera nécessaire à cette fin.

Le choix de .NET comme base de cette méthode de développement est également avantageux. De nombreux services, API et autres ressources nécessaires au développement de logiciels modernes prennent directement en charge .NET en fournissant leurs propres bibliothèques, connues sous le nom de kits de développement logiciel (SDK), pour ce cadre. Si vous souhaitez stocker vos données dans le cloud, par exemple, il est fort probable qu'une bibliothèque .NET soit déjà disponible pour ce faire.

Conclusion

.NET MAUI offre une approche intéressante de création d'une application pour tous les systèmes cibles importants. Il en résulte des applications natives qui ne nécessitent pas de navigateur web pour fonctionner et qui permettent d'accéder aux fonctions du système. La combinaison du développement d'applications natives et de la programmation multiplateforme, à la fois pour les ordinateurs de bureau et les appareils mobiles, rend cette approche attrayante pour les projets électroniques. Il est toutefois un peu décevant

que .NET MAUI ne prenne pas actuellement en charge Linux, car il serait intéressant de pouvoir l'utiliser dans un environnement Raspberry Pi. Microsoft a confié la responsabilité de la prise en charge de Linux à la communauté. Il faudra peut-être attendre un peu avant qu'il ne soit disponible.
Si cet article a suscité votre intérêt pour .NET MAUI, vous pouvez trouver une bonne introduction avec des tutoriels vidéo utiles sur [5] !

220442-04

Questions ou commentaires ?

Contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Veikko Krypczyk est développeur de logiciels, formateur et auteur technique, spécialisé dans des sujets tels que WinUI 3 et .NET MAUI (en savoir plus [6]). Pendant son temps libre, il se sert d'un fer à souder et s'amuse à construire divers projets électroniques. Il est fasciné par la combinaison de l'électronique et de la programmation qui, ensemble, permettent souvent de réaliser des solutions surprenantes.

Produits

- **John Allwork, *C# Programming for Windows and Android*, Elektor 2015 (E-Book)**
  https://elektor.fr/18220

LIENS

[1] .NET MAUI : https://dotnet.microsoft.com/en-us/apps/maui
[2] Visual Studio : https://visualstudio.microsoft.com/
[3] MVVM Toolkit : https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm
[4] Dépôt GitHub de l'auteur de l'application ElectronicCalculator : https://github.com/veikkoEF/ElectronicCalculator
[5] Tutoriels vidéo : .NET MAUI for Beginners by dotnet : https://youtube.com/playlist?list=PLdo4fOcmZ0oUBAdL2NwBpDs32zwGqb9DY
[6] Site web de l'auteur : https://larinet.com