 3iL INGENIEURS </les sciences informatiques>	Technologie .Net Maui (3iL I3 - B. Chervy) TP Maui N° 1	Année : 2023/2024 Temps : 3 séances Nombre de page : 6
---	---	---

TP .Net Maui n° 1 : Utilisation de l'API système

Objectif : Utiliser les fonctionnalités accessibles des APIs Système à travers .Net Maui

Partie 1 : Création du projet et tests

Lancez « Visual Studio » et créez un nouveau projet de type « Application .NET MAUI ».

Lancez l'exécution en sélectionnant « Windows Machine ». Testez l'application.

Allez sous « Windows Machine » et faites « Gestionnaire d'appareils Android », faites « Nouveauté », sélectionnez « Pixel 2 (+Store), x86, Android 9 - API28 » et laissez les autres paramètres par défaut, faites ensuite « Créer ».

Démarrez votre émulateur.

Revenez à Visual Studio et lancez votre application sur votre émulateur.

Ouvrez le fichier « App.xaml et App.xaml.cs » de quel code s'agit-il ? :

Ouvrez le fichier « AppShell.xaml et AppShell.xaml.cs » de quel code s'agit-il ? :

Ouvrez le fichier « MainPage.xaml et MainPage.xaml.cs » de quel code s'agit-il ? :

Allez dans « Ressources/Styles » et ouvrez le fichier « Colors.xaml », que contient-il ? :

Modifiez la couleur Primary à #005067 et la couleur Secondary à #E84D0D. Ceux sont les couleurs officielles de la charte de 3iL.

Allez dans « Ressources/Styles » et ouvrez le fichier « Styles.xaml », que contient-il ? :

Démarrez votre application en mode « Windows Machine ». Lorsqu'elle est affichée, revenez à Visual Studio et modifiez le texte du premier Label (fichier MainPage.xaml). Que pouvez constater ? :

Partie 2 : Création des premières pages

- Création de la page A Propos

Dans l'Explorateur de solutions, cliquez avec le bouton droit sur le projet et faites « Ajouter un nouvel élément ». Dans la fenêtre, sélectionnez .NET MAUI et .NET MAUI ContentPage (XAML). Nommez votre page « APropos.xaml ». Changez le code de la façon suivante :

```
<VerticalStackLayout Spacing="10" Margin="10">
    <Image Source="logo_3il.png"
        SemanticProperties.Description="Cette application est une production 3iL"
        HeightRequest="64" HorizontalOptions="CenterAndExpand"/>
    <Label FontSize="22" FontAttributes="Bold" Text="Notes 3iL" VerticalOptions="End"
        TextColor="{StaticResource Secondary}"/>
    <Label FontSize="22" Text="v1.0" VerticalOptions="End" TextColor="{StaticResource
        Secondary}"/>
    <Label Text="Application XAML/C# écrite en .NET MAUI."
        TextColor="{StaticResource Primary}"/>
    <Button Text="En savoir plus sur 3iL" Clicked="APropos_Clicked" />
</VerticalStackLayout>
```

Que fait d'après vous le « VerticalStackLayout » ? : _____

Avec le bouton droit cliquez sur le répertoire « Resources/Images » et faites « Ajoutez/Elément Existant », et ajoutez le fichier « logo3il.png ».

Sur le nom de la méthode « `APropos_Clicked` », cliquez avec le bouton droit pour faire « Atteindre la définition ». Dans la méthode créée, ajoutez le code :

```
await Launcher.Default.OpenAsync("https://www.3il-ingenieurs.fr/");
```

Attention la méthode devra être passée en « `async` »

Ajoutez les 2 images « icon_about.png » et « icon_notes.png » au répertoire « Images ».

Ouvrez « AppShell.xaml » et remplacez la balise « `ShellContent` » par le code par :

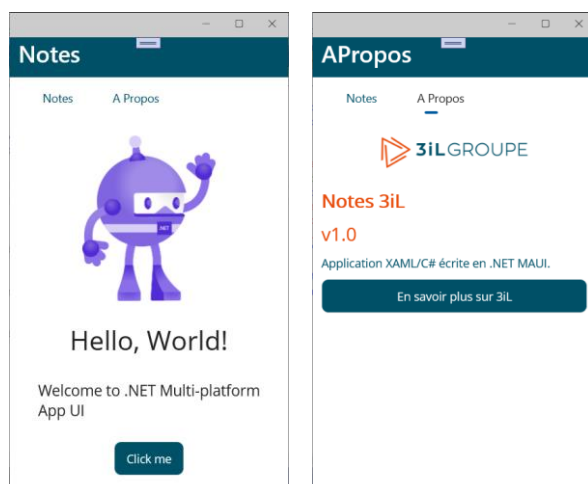
```
<TabBar>
    <ShellContent
        Title="Notes"
        ContentTemplate="{DataTemplate local:MainPage}"
        Icon="icon_notes" />

    <ShellContent
        Title="A Propos"
        ContentTemplate="{DataTemplate local:APropos}"
        Icon="icon_about" />
</TabBar>
```

Notez que <Shell> est l'objet racine du balisage XAML, <TabBar> est une barre de commande avec 2 éléments de type <ShellContent>

Lancez votre application et vérifiez que la page APropos s'ouvre et que le bouton « En savoir plus sur 3iL » ouvre bien le navigateur.

Vous devez avoir le rendu suivant :



Partie 3 : Quelques infos système

Ajoutez un répertoire « Views » et un répertoire « Models » à votre projet.

Dans View, ajoutez une nouvelle page « .NET MAUI ContentPage (XAML) » que vous nommez « vPhone.xaml ». Toujours dans View, ajoutez une nouvelle page « .NET MAUI ContentPage (XAML) » que vous nommez « vGeolocalisation.xaml ».

Ouvrez « AppShell.xaml » et ajoutez 2 boutons « Infos » pointant sur « vPhone » et « Géo » pointant sur « vGeolocalisation » :

```
<TabBar>
  <ShellContent
    Title="Information"
    ContentTemplate="{DataTemplate views:vPhone}"
    Icon="icon_info" />
  ...
```

En précisant dans la balise « Shell » le namespace : `xmlns:views="clr-namespace:TPMau1.Views"`

- Infos système :

Ouvrez « vPhone.xaml », mettez les propriétés « `Spacing="20" Margin="10"` » au « `VerticalStackLayout` ». Ajoutez-y 2 labels centrés horizontalement (`HorizontalOptions`), le premier affichant « Info Téléphone » avec « `FontSize="22"` », le deuxième nommé « `lbPhoneInfo` » avec pour celui-ci « `HeightRequest="240" FontSize="22"` ».

Ouvrez « vPhone.xaml.cs ». Créez une méthode « `Phone_Info()` » que vous appellerez au démarrage de la fenêtre. Dans cette méthode :

- instanciez un objet de type « `System.Text.StringBuilder` » (opérateur new)
- ajoutez à cet objet avec la méthode « `AppendLine` » successivement les valeurs disponibles dans « `DeviceInfo` » qui sont « `Current.Model` », « `Current.Manufacturer` », « `Name` », « `VersionString` » et « `Current.Platform` » avec des libellés descriptifs. Par exemple : `object.AppendLine($"Modèle: {DeviceInfo.Current.Model}");`
- affichez cet objet dans le Label « `lbPhoneInfo` ».

Ajoutez une méthode « `Get_Idiom()` ». Cette méthode doit renvoyer « Téléphone », « PC » ou « Tablette ». Pour cela il suffit de tester la valeur de « `DeviceInfo.Current.Idiom` » en la comparant avec les constantes disponibles dans « `DeviceIdiom. ...` ». Affichez ensuite cette valeur dans votre Label.

- Infos écran :

Dans View, ajoutez une nouvelle page « .NET MAUI ContentPage (XAML) » que vous nommez « vEcran.xaml ». Ouvrez « vPhone.xaml », et juste après la « `VerticalStackLayout` » ajoutez une « `ToolBarItems` » comme ceci :

```
<ContentPage.ToolbarItems>
  <ToolbarItem Text="Ecran" Clicked="Ecran_Clicked"
    IconImageSource="{FontImage Glyph='+', Color=White, Size=22}" />
</ContentPage.ToolbarItems>
```

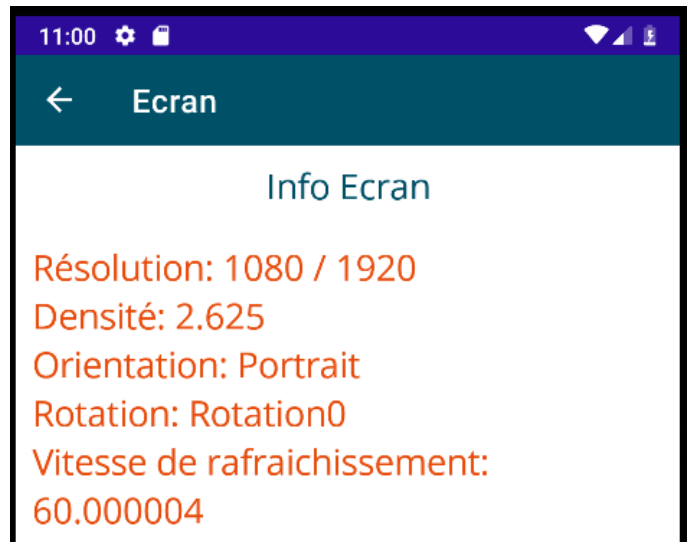
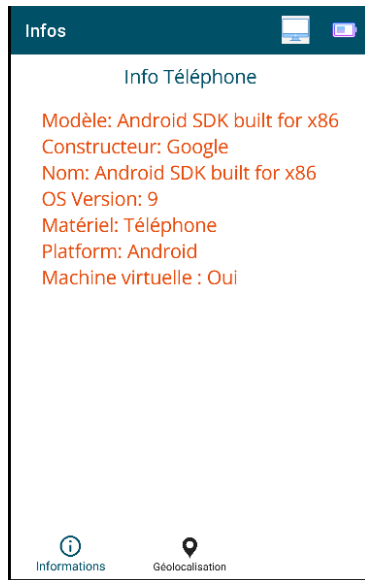
Interceptez la méthode associée et ajoutez lui le code :

```
await Shell.Current.GoToAsync(nameof(vEcran));
```

Enfin dans le constructeur de la fenêtre ajoutez la définition de la route :

```
Routing.RegisterRoute($" {nameof(vEcran)}", typeof(vEcran));
```

Dans le code de votre fenêtre « vEcran », ajoutez 2 labels comme pour la page « vPhone ». Dans le 2^o affichez la résolution de l'écran, sa densité, son orientation, sa vitesse de rafraichissement. Ces caractéristiques sont disponibles dans « `DeviceDisplay.Current.MainDisplayInfo` ».



Partie 4 : Jouer avec la batterie

Dans View, ajoutez une nouvelle page « .NET MAUI ContentPage (XAML) » que vous nommez « vBatterie.xaml ». Ouvrez « vPhone.xaml », et ajoutez un item à la toolbar permettant d'y accéder.



Dans « vBatterie.xaml », ajoutez un label de titre « Alimentation », un bouton avec le titre « Afficher », un label « Etat Batterie » et label vide pouvant contenir 2 lignes.

Lors d'un clic sur le bouton, votre page doit afficher le type d'alimentation en cours. Pour cela, créez une méthode qui renvoie « AC 220V », « Câble USB », « Chargeur sans fils », « Sur Batterie » ou « Inconnu ».

Faites un Switch sur la valeur « Battery.Default.PowerSource » en comparant avec les constantes disponibles dans « BatteryPowerSource ».

Attention sous « Android », l'accès aux informations de la batterie demande une autorisation. Pour passer cette demande, ouvrez « AndroidManifest.xml » et ajoutez la ligne :

```
<uses-permission android:name="android.permission.BATTERY_STATS" />
```

Faites le test sur le simulateur Android en cliquant sur le bouton  puis sur  Battery.

Affichez votre page batterie, changez le type de charge dans le simulateur (AC Charger puis None avec Charging), puis réaffichez votre page.

Concernant le niveau de charge de la batterie, vous allez afficher les informations en temps réel. Pour cela créez une méthode :

```
private void Battery_InfoChanged(object sender, BatteryInfoChangedEventArgs e)
```

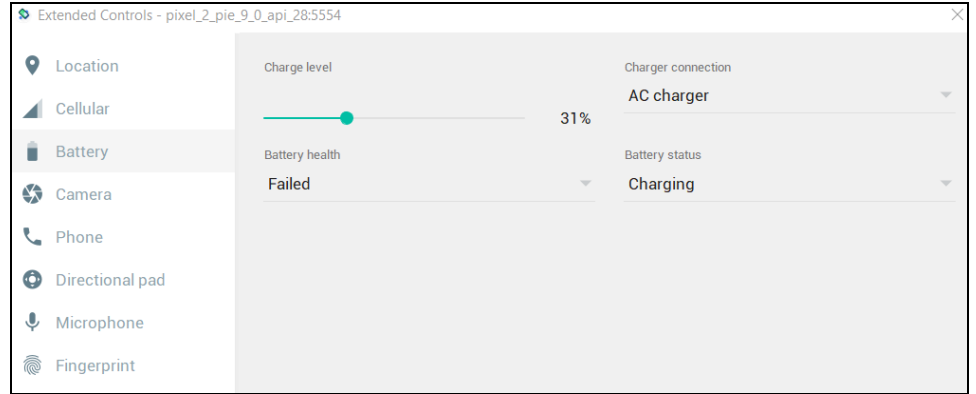
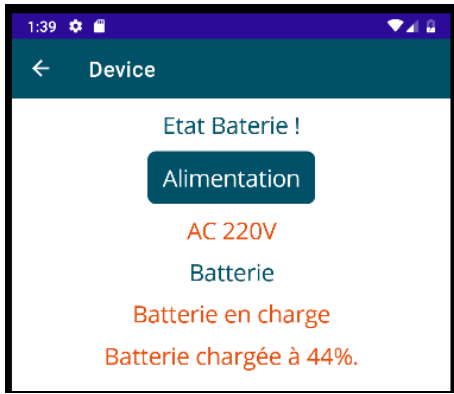
Dans cette méthode affichez l'état de la batterie (Batterie en charge, Batterie déchargée, Batterie pleine, ...) en comparant la valeur de « e.State » avec les constantes disponibles dans « BatteryState ». Affichez aussi le niveau de charge en % que vous trouverez dans « e.ChargeLevel ».

Pour lancer cette méthode, dans le constructeur de la classe inscrivez cette méthode comme événement à intercepter sur « BatteryInfoChanged »

```
Battery.Default.BatteryInfoChanged += Battery_InfoChanged;
```

Sélectionnez une nouvelle valeur à « Chargeur Connexion » et allez sur votre page « Batterie ».

Ensuite vous pouvez changer l'état de la batterie et le niveau de charge qui doivent s'afficher en temps réel sur votre application.



Les appareils sur batterie présentent généralement un mode faible consommation d'énergie lorsque la batterie tombe en dessous de 20% de sa capacité. Dans ce mode, le système d'exploitation réduit les activités qui ont tendance à épuiser la batterie. Toute application peut participer à ce mode en évitant toute activité énergivore (le traitement en arrière-plan, ...). Pour détecter le mode, il faut abonner l'application à l'événement associé pour qu'elle soit prévenue de l'activation du mode. Pour cela il faut créer une méthode de type

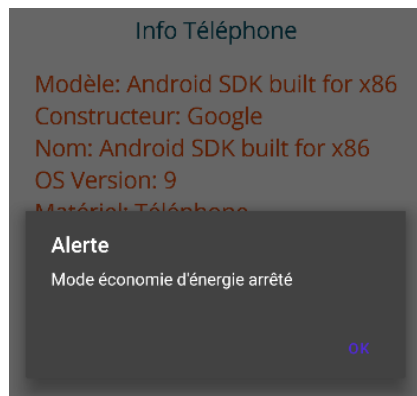
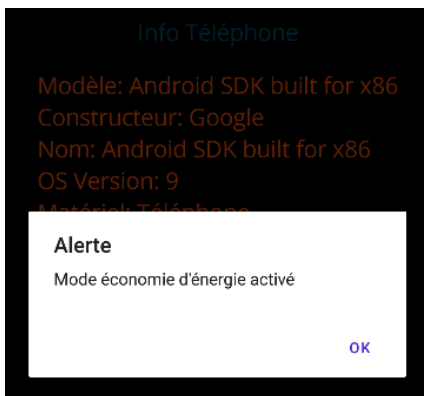
```
private async void Battery_ModeEcoChanged(object sender, EnergySaverStatusChangedEventArgs e)
```

puis l'ajouter à l'événement : `Battery.Default.EnergySaverStatusChanged`

Dans cette méthode, il suffit de tester « `Battery.Default.EnergySaverStatus` » qui vaudra

« `EnergySaverStatus.On` » lorsque le mode est activé. Complétez le code pour afficher une boîte de dialogue qui signale l'activation ou l'arrêt du mode économie d'énergie (`await DisplayAlert()`).

Pour le tester, mettez votre simulateur sans chargeur avec une batterie non en charge. Diminuez ensuite votre charge de batterie jusqu'à ce que le système du simulateur demande s'il doit passer en économie d'énergie. Répondez oui. A ce moment-là, votre application va recevoir l'information. Quand vous repassez l'alimentation en « AC Charger », l'application va recevoir l'information de l'arrêt du mode économie.



Partie 5 : Utilisation de la Géolocalisation

Ouvrez « `vGeolocalisation.xaml` », et ajoutez 2 boutons « Géolocalisation » et « Annulation » puis 3 Labels (longitude, latitude, altitude).

La lecture de la géolocalisation peut être assez lente voir provoquer un blocage si celle-ci n'est pas lisible (GPS hors de portée). Pour cela, il faut donner à l'utilisateur la possibilité d'annuler la demande.

Créez 2 variables globales à la classe de la fenêtre :

```
private CancellationTokenSource _cancelTokenSource;
private bool _isCheckingLocation;
```

Créez une méthode « `Get_Localisation` » qui est appelée par la méthode du bouton :

```
public async Task GetCurrentLocation()
```

Dans cette méthode :

- ajoutez un try/catch/finally
- mettez « `isCheckingLocation` » à vrai
- créez une variable de type « `GeolocationRequest` » et appelez son constructeur (opérateur `new`) en passant comme paramètre « `GeolocationAccuracy.Medium` » et « `TimeSpan.FromSeconds(10)` »
- instanciez la variable « `_cancelTokenSource` » en appelant son constructeur (opérateur `new`)
- créez une variable « `Location` » et initialisez-la en appelant
`Geolocation.Default.GetLocationAsync(votre variable GeolocationRequest, _cancelTokenSource.Token);`
- testez si cette variable « `Location` » n'est pas nulle, et dans ce cas affectez ses propriétés longitude, latitude, altitude aux Labels prévus à cet effet.
- dans la clause « `finally` », passez « `isCheckingLocation` » à faux



Pour terminer rajoutez le code suivant au bouton « Annulation » :

```
if (_isCheckingLocation && _cancelTokenSource != null
    && _cancelTokenSource.IsCancellationRequested == false)
    _cancelTokenSource.Cancel();
```

Pour tester la géolocalisation sur Android vous devez donner des autorisations à votre application. Ouvrez « `AndroidManifest.xml` » et ajoutez les lignes :

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.location" android:required="false" />
<uses-feature android:name="android.hardware.location.gps" android:required="false" />
<uses-feature android:name="android.hardware.location.network" android:required="false" />
```

Démarrez l'application sur le simulateur Android.

Sur le simulateur, cliquez sur le bouton  puis sur  Location.

Sélectionnez une position sur la partie « Map » et dans votre application cliquez sur votre bouton « Géolocalisation ».

