Créer une application .NET MAUIBlazor Hybrid avec une application web Blazor

Article • 21/06/2024

Cet article vous montre comment créer une application .NET MAUIBlazor Hybrid avec une application web Blazor qui utilise une interface utilisateur partagée via une bibliothèque de classes (RCL) Razor.

Prérequis et étapes préliminaires

Pour connaître les prérequis et les étapes préliminaires, consultez Créer une application .NET MAUIBlazor Hybrid. Nous vous recommandons d'utiliser le didacticiel .NET MAUIBlazor Hybrid afin de configurer votre système local pour le développement de .NET MAUI avant d'utiliser les instructions de cet article.

.NET MAUIBlazor Hybrid et l'application d'exemple d'application web

Obtenir l'exemple d'application nommé `MauiBlazorWeb` à partir du dépôt GitHub d'exemples de Blazor (dotnet/blazor-samples) (.NET 8 ou version ultérieure).

L'exemple d'application est une solution de démarrage qui contient une application .NET MAUIBlazor Hybrid (native et multi-plateforme), une application web Blazor et une bibliothèque de classes Razor qui contient l'interface utilisateur partagée (composants Razor) utilisée par les applications web et natives.

Migration d'une solution .NET MAUIBlazor Hybrid

Au lieu d'utiliser l'exemple d'application, vous pouvez migrer une application .NET MAUIBlazor Hybrid existante avec les instructions de cette section à l'aide de Visual Studio.

Ajoutez un nouveau projet à la solution avec le modèle de projet application web **Blazor**. Sélectionnez les options suivantes :

- **Nom du projet** : Utilisez le nom de la solution avec `.Web` ajouté. Les exemples fournit dans cet article présument les désignations suivantes :

  - Solution : `MauiBlazorWeb`
  - Projet MAUI : `MauiBlazorWeb.Maui`
  - Application web Blazor : `MauiBlazorWeb.Web`
  - Bibliothèque de classes (RCL) Razor (ajoutée à une étape ultérieure) : `MauiBlazorWeb.Shared`
- **Type d'authentification** : **Aucun**
- **Configurer pour HTTPS** : Sélectionné (activé)
- **Mode de rendu interactif** : **Serveur**
- **Emplacement d'interactivité** : **Global**
- **Exemples de pages** : non sélectionné (désactivé)

Le paramètre d'**emplacement d'interactivité** sur **Global** est important car les applications MAUI s'exécutent toujours de manière interactive et renvoient des erreurs sur les pages de composants Razor qui spécifient explicitement un mode de rendu. Si vous n'utilisez pas de mode de rendu global, vous devez utiliser l'approche décrite dans la section Utiliser les modes de renduBlazor après avoir suivi les instructions dans cette section. Pour plus d'informations, consultez BlazorWebView a besoin d'un moyen d'activer la substitution de ResolveComponentForRenderMode (dotnet/aspnetcore #51235) .

Ajoutez un nouveau projet **Razor bibliothèque de classes** (RCL) à la solution. Les exemples de cet article supposent que le projet est nommé `MauiBlazorWeb.Shared`. Ne sélectionnez pas **Vues et pages de support** lorsque vous ajoutez le projet à la solution.

Ajoutez des références de projet au RCL depuis le projet MAUI et du projet d'application web Blazor.

Déplacez le dossier `Components` et tout son contenu depuis le projet MAUI vers la liste RCL. Vérifiez que le dossier `Components` est supprimé du projet MAUI.

> **Conseil**
>
> Lors du déplacement d'un dossier ou d'un fichier dans Visual Studio, utilisez des raccourcis clavier ou le menu contextuel en cliquant avec le bouton droit sur une opération couper-coller. Glisser le dossier dans Visual Studio copie uniquement depuis un emplacement vers un autre, ce qui nécessite une étape supplémentaire pour supprimer l'original.

Déplacez le dossier `css` depuis le dossier `wwwroot` du projet MAUI vers le dossier RCL `wwwroot`.

Supprimez les fichiers suivants du dossier RCL `wwwroot` :

- `background.png`
- `exampleJsInterop.js`

Dans la liste RCL, remplacez le fichier racine `_Imports.razor` par celui du dossier RCL `Components`, en remplaçant le fichier existant dans la liste RCL et en supprimant l'original dans le dossier `Components`. Après avoir déplacé le fichier, ouvrez-le et renommez les deux dernières états `@using` pour qu'ils correspondent à l'espace de noms du RCL. Dans l'exemple suivant, l'espace de noms RCL est `MauiBlazorWeb.Shared` :

razor

```razor
@using MauiBlazorWeb.Shared
@using MauiBlazorWeb.Shared.Components
```

À la racine du projet RCL, supprimez les fichiers suivants :

- `Component1.razor`
- `ExampleJsInterop.cs`

Dans la liste RCL, ouvrez le fichier `Components/Routes.razor` et remplacez `MauiProgram` par `Routes` :

diff

```
- <Router AppAssembly="@typeof(MauiProgram).Assembly">
+ <Router AppAssembly="@typeof(Routes).Assembly">
```

Ouvrez le fichier `MainPage.xaml` dans le projet MAUI. Ajoutez une référence `xmlns:shared` au RCL dans les attributs ContentPage. Dans l'exemple suivant, l'espace de noms RCL est `MauiBlazorWeb.Shared`. Définissez la bonne valeur pour les `clr-namespace` et les `assembly` :

XML

```
xmlns:shared="clr-
namespace:MauiBlazorWeb.Shared;assembly=MauiBlazorWeb.Shared"
```

Également dans le fichier `MainPage.xaml`, mettez à jour le composant racine BlazorWebView ComponentType de `local` à `shared` :

diff

```
- <RootComponent Selector="#app" ComponentType="{x:Type
local:Components.Routes}" />
+ <RootComponent Selector="#app" ComponentType="{x:Type
shared:Components.Routes}" />
```

Dans le projet MAUI, ouvrez le fichier `wwwroot/index.html` et modifiez les feuilles de style pour pointer vers le chemin d'accès des ressources statiques du RCL.

Supprimez les lignes suivantes :

diff

```
- <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
- <link rel="stylesheet" href="css/app.css" />
```

Remplacez les lignes précédentes par le code suivant. Dans l'exemple suivant, le chemin d'accès à la ressource statique de RCL est `_content/MauiBlazorWeb.Shared/` :

razor

```
<link rel="stylesheet"
href="_content/MauiBlazorWeb.Shared/css/bootstrap/bootstrap.min.css" />
<link rel="stylesheet" href="_content/MauiBlazorWeb.Shared/css/app.css" />
```

Dans l'application web Blazor, ouvrez le fichier `_Imports.razor` et ajoutez les deux instructions `@using` suivantes pour la liste RCL. Dans l'exemple suivant, l'espace de noms RCL est `MauiBlazorWeb.Shared` :

razor

```
@using MauiBlazorWeb.Shared
@using MauiBlazorWeb.Shared.Components
```

Dans le projet Web App Blazor, ouvrez le composant `App` (`Components/App.razor`). Supprimez la feuille de style `app.css` :

diff

```
- <link rel="stylesheet" href="app.css" />
```

Remplacez la ligne précédente par les références de feuille de style de ressource statique RCL. Dans l'exemple suivant, le chemin d'accès à la ressource statique de RCL est `_content/MauiBlazorWeb.Shared/` :

```
<link rel="stylesheet"
href="_content/MauiBlazorWeb.Shared/css/bootstrap/bootstrap.min.css" />
<link rel="stylesheet" href="_content/MauiBlazorWeb.Shared/css/app.css" />
```

Dans le projet Web App Blazor, supprimez le dossier et les fichiers suivants :

- `Components/Layout`
- `Components/Routes.razor`
- `Components/Pages/Home.razor`
- `wwwroot/app.css`

Ouvrez le fichier `Program.cs` de l'application web Blazor et ajoutez un assembly supplémentaire pour la bibliothèque RCL à l'application. Dans l'exemple suivant, l'espace de noms RCL est `MauiBlazorWeb.Shared` :

C#

```
app.MapRazorComponents<App>()
    .AddInteractiveServerRenderMode()

.AddAdditionalAssemblies(typeof(MauiBlazorWeb.Shared._Imports).Assembly);
```

Exécutez le projet MAUI en sélectionnant le projet dans **Explorateur de solutions** et en utilisant le bouton Démarrer de Visual Studio.

Exécutez le projet Web App Blazor en sélectionnant le projet d'Application web Blazor dans **Explorateur de solutions** et en utilisant le bouton Démarrer de Visual Studio avec la configuration de build `https`.

Si vous recevez une erreur de génération indiquant que l'assembly RCL ne peut pas être résolu, générez d'abord le projet RCL. Si des erreurs de ressource de projet MAUI surviennent lors de la génération, régénérez le projet MAUI afin de supprimer les erreurs.

Utiliser les modes de rendu Blazor

Utilisez les instructions de l'une des sous-sections suivantes qui correspond aux spécifications de votre application pour appliquer les Blazormodes de rendu pour un emplacement d'interactivité donné dans l'application web Blazor mais ignorez les affectations de mode de rendu dans le projet MAUI.

Sous-sections des spécifications de mode de rendu et d'interactivité :

- Interactivité de serveur global
- Interactivité globale auto ou WebAssembly
- Interactivité server par page/composant
- Interactivité automatique par page/composant
- Interactivité par page/composant WebAssembly

Interactivité de serveur global

- Mode de rendu interactif : **Serveur**
- Emplacement d'interactivité : **Global**
- Projets de solution
  - MAUI (`MauiBlazorWeb.Maui`)
  - Blazor Application web (`MauiBlazorWeb.Web`)
  - RCL (`MauiBlazorWeb.Shared`) : Renferme les composants partagés Razor sans définir de modes de rendu dans chaque composant.

Références de projet : `MauiBlazorWeb.Maui` et `MauiBlazorWeb.Web` ont une référence de projet à `MauiBlazorWeb.Shared`.

Interactivité globale auto ou WebAssembly

- Mode de rendu interactif : **Automatique** ou **WebAssembly**
- Emplacement d'interactivité : **Global**
- Projets de solution
  - MAUI (`MauiBlazorWeb.Maui`)
  - Application webBlazor
    - Projet de serveur : `MauiBlazorWeb.Web`
    - Projet client : `MauiBlazorWeb.Web.Client`
  - RCL (`MauiBlazorWeb.Shared`) : Renferme les composants partagés Razor sans définir de modes de rendu dans chaque composant.

Références du projet :

- Les projets `MauiBlazorWeb.Maui`, `MauiBlazorWeb.Web`, et `MauiBlazorWeb.Web.Client` ont une référence de projet à `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` a une référence de projet à `MauiBlazorWeb.Web.Client`.

Interactivité server par page/composant

- Mode de rendu interactif : **Serveur**
- Emplacement d'interactivité : **Par page/composant**
- Projets de solution
  - MAUI (`MauiBlazorWeb.Maui`) : Appels `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` dans `MauiProgram.cs`.
  - Web App Blazor (`MauiBlazorWeb.Web`) : ne définit pas d'attribut de directive `@rendermode` sur les composants `HeadOutlet` et `Routes` du composant `App` (`Components/App.razor`).
  - RCL (`MauiBlazorWeb.Shared`) : Renferme les composants partagés Razor qui définissent les de modes de rendu `InteractiveServer` dans chaque composant.

`MauiBlazorWeb.Maui` et `MauiBlazorWeb.Web` ont une référence de projet à `MauiBlazorWeb.Shared`.

Ajoutez la classe `InteractiveRenderSettings` suivante à la liste RCL. Les propriétés de classe sont utilisées pour définir les modes de rendu des composants.

Le projet MAUI est interactif par défaut. Par conséquent, aucune action n'est effectuée au niveau du projet MAUI autre que l'appel de `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

Pour l'application web Blazor sur le client web, les valeurs de propriété sont affectées à partir de RenderMode. Lorsque les composants sont chargés dans un BlazorWebView pour le client natif du projet MAUI, les modes de rendu ne sont pas attribués (`null`), car le projet MAUI définit explicitement les propriétés du mode de rendu sur `null` lorsque `ConfigureBlazorHybridRenderModes` est appelé.

`InteractiveRenderSettings.cs` :

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set; } =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; } =
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get; set; }
=
```

```
        RenderMode.InteractiveWebAssembly;

    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

Dans `MauiProgram.CreateMauiApp` de `MauiProgram.cs`, appelez `ConfigureBlazorHybridRenderModes` :

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

Dans le fichier RCL `_Imports.razor`, ajoutez la directive statique globale `@using` suivante pour rendre les propriétés de la classe disponibles pour les composants :

razor

```
@using static InteractiveRenderSettings
```

> **Notes**
>
> L'affectation des modes de rendu via les propriétés de classe RCL `InteractiveRenderSettings` diffère d'une application web Blazor autonome classique. Dans une application web Blazor, les modes de rendu sont fournis normalement par RenderMode via l'instruction `@using static Microsoft.AspNetCore.Components.Web.RenderMode` dans le fichier `_Import` de l'application web Blazor.

Interactivité automatique par page/composant

- Mode de rendu interactif : **Automatique**
- Emplacement d'interactivité : **Par page/composant**
- Projets de solution
  - MAUI (`MauiBlazorWeb.Maui`) : Appels `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` dans `MauiProgram.cs`.

- Application webBlazor
  - Projet de serveur: `MauiBlazorWeb.Web` : Ne définit pas d'attribut de directive `@rendermode` sur les composants `HeadOutlet` et `Routes` du composant `App` (`Components/App.razor`).
  - Projet client : `MauiBlazorWeb.Web.Client`
- RCL (`MauiBlazorWeb.Shared`) : Renferme les composants partagés Razor qui définissent les de modes de rendu `InteractiveAuto` dans chaque composant.

Références du projet :

- `MauiBlazorWeb.Maui`, `MauiBlazorWeb.Web` et `MauiBlazorWeb.Web.Client` ont une référence de projet à `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` a une référence de projet à `MauiBlazorWeb.Web.Client`.

Ajoutez la classe `InteractiveRenderSettings` suivante est ajoutée à la liste RCL. Les propriétés de classe sont utilisées pour définir les modes de rendu des composants.

Le projet MAUI est interactif par défaut. Par conséquent, aucune action n'est effectuée au niveau du projet MAUI autre que l'appel de `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

Pour l'application web Blazor sur le client web, les valeurs de propriété sont affectées à partir de RenderMode. Lorsque les composants sont chargés dans un BlazorWebView pour le client natif du projet MAUI, les modes de rendu ne sont pas attribués (`null`), car le projet MAUI définit explicitement les propriétés du mode de rendu sur `null` lorsque `ConfigureBlazorHybridRenderModes` est appelé.

`InteractiveRenderSettings.cs` :

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set; } =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; } =
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get; set; }
=
        RenderMode.InteractiveWebAssembly;
```

```
    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

Dans `MauiProgram.CreateMauiApp` de `MauiProgram.cs`, appelez `ConfigureBlazorHybridRenderModes` :

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

Dans le fichier RCL `_Imports.razor`, ajoutez la directive statique globale `@using` suivante pour rendre les propriétés de la classe disponibles pour les composants :

razor

```
@using static InteractiveRenderSettings
```

> **Notes**
>
> L'affectation des modes de rendu via les propriétés de classe RCL `InteractiveRenderSettings` diffère d'une application web Blazor autonome classique. Dans une application web Blazor, les modes de rendu sont fournis normalement par RenderMode via l'instruction `@using static Microsoft.AspNetCore.Components.Web.RenderMode` dans le fichier `_Import` de l'application web Blazor.

Interactivité par page/composant WebAssembly

- Mode de rendu interactif : **WebAssembly**
- Emplacement d'interactivité : **Par page/composant**
- Projets de solution
  - MAUI (`MauiBlazorWeb.Maui`)
  - Application webBlazor
    - Projet de serveur: `MauiBlazorWeb.Web` : Ne définit pas d'attribut de directive `@rendermode` sur les composants `HeadOutlet` et `Routes` du composant `App` (`Components/App.razor`).

    - Projet client : `MauiBlazorWeb.Web.Client`
  - Listes RCL
    - `MauiBlazorWeb.Shared`
    - `MauiBlazorWeb.Shared.Client` : Renferme les composants partagés Razor qui définissent les de modes de rendu `InteractiveWebAssembly` dans chaque composant. La liste RCL `.Shared.Client` est conservée séparément de la liste RCL `.Shared` car l'application doit conserver les composants requis pour s'exécuter sur WebAssembly de manière séparée des composants qui s'exécutent sur le serveur et qui restent sur le serveur.

Références du projet :

- `MauiBlazorWeb.Maui` et `MauiBlazorWeb.Web` ont des références de projet dans `MauiBlazorWeb.Shared`.
- `MauiBlazorWeb.Web` a une référence de projet à `MauiBlazorWeb.Web.Client`.
- `MauiBlazorWeb.Web.Client` et `MauiBlazorWeb.Shared` ont une référence de projet dans `MauiBlazorWeb.Shared.Client`.

Ajoutez le paramètre suivant AdditionalAssemblies à l'instance de composant `Router` pour l'assembly de projet `MauiBlazorWeb.Shared.Client` (via son fichier `_Imports`) dans le fichier `Routes.razor` du projet `MauiBlazorWeb.Shared` :

razor

```
<Router AppAssembly="@typeof(Routes).Assembly"
        AdditionalAssemblies="new [] {
typeof(MauiBlazorWeb.Shared.Client._Imports).Assembly }">
    <Found Context="routeData">
        <RouteView RouteData="@routeData"
DefaultLayout="@typeof(Components.Layout.MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
</Router>
```

Ajoutez l'assembly de projet `MauiBlazorWeb.Shared.Client` (via son fichier `_Imports`) avec l'appel suivant AddAdditionalAssemblies dans le fichier `Program.cs` du projet `MauiBlazorWeb.Web` :

C#

```
app.MapRazorComponents<App>()
    .AddInteractiveWebAssemblyRenderMode()
    .AddAdditionalAssemblies(typeof(MauiBlazorWeb.Shared._Imports).Assembly)
```

```
.AddAdditionalAssemblies(typeof(MauiBlazorWeb.Shared.Client._Imports).Assemb
ly);
```

Ajoutez la classe suivante `InteractiveRenderSettings` est ajoutée à la liste RCL `.Shared.Client`. Les propriétés de classe sont utilisées pour définir les modes de rendu des composants pour les composants basé sur le serveur.

Le projet MAUI est interactif par défaut. Par conséquent, aucune action n'est effectuée au niveau du projet MAUI autre que l'appel de `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes`.

Pour l'application web Blazor sur le client web, les valeurs de propriété sont affectées à partir de RenderMode. Lorsque les composants sont chargés dans un BlazorWebView pour le client natif du projet MAUI, les modes de rendu ne sont pas attribués (`null`), car le projet MAUI définit explicitement les propriétés du mode de rendu sur `null` lorsque `ConfigureBlazorHybridRenderModes` est appelé.

`InteractiveRenderSettings.cs` (`.Shared.Client` RCL) :

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared;

public static class InteractiveRenderSettings
{
    public static IComponentRenderMode? InteractiveServer { get; set; } =
        RenderMode.InteractiveServer;
    public static IComponentRenderMode? InteractiveAuto { get; set; } =
        RenderMode.InteractiveAuto;
    public static IComponentRenderMode? InteractiveWebAssembly { get; set; }
=
        RenderMode.InteractiveWebAssembly;

    public static void ConfigureBlazorHybridRenderModes()
    {
        InteractiveServer = null;
        InteractiveAuto = null;
        InteractiveWebAssembly = null;
    }
}
```

Une version légèrement différente de la classe `InteractiveRenderSettings` est ajoutée à la RCL `.Shared`. Dans la classe ajoutée au `.Shared` RCL, `InteractiveRenderSettings.ConfigureBlazorHybridRenderModes` du `.Shared.Client` RCL

est appelée. Cela garantit que le mode de rendu des composants WebAssembly rendus sur le client MAUI n'est pas attribué (`null`) car il est interactif par défaut sur le client natif.

`InteractiveRenderSettings.cs` (`.Shared` RCL) :

C#

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

namespace MauiBlazorWeb.Shared
{
    public static class InteractiveRenderSettings
    {
        public static IComponentRenderMode? InteractiveServer { get; set; }
=
            RenderMode.InteractiveServer;
        public static IComponentRenderMode? InteractiveAuto { get; set; } =
            RenderMode.InteractiveAuto;
        public static IComponentRenderMode? InteractiveWebAssembly { get;
set; } =
            RenderMode.InteractiveWebAssembly;

        public static void ConfigureBlazorHybridRenderModes()
        {
            InteractiveServer = null;
            InteractiveAuto = null;
            InteractiveWebAssembly = null;
            MauiBlazorWeb.Shared.Client.InteractiveRenderSettings
                .ConfigureBlazorHybridRenderModes();
        }
    }
}
```

Dans `MauiProgram.CreateMauiApp` de `MauiProgram.cs`, appelez `ConfigureBlazorHybridRenderModes` :

C#

```
InteractiveRenderSettings.ConfigureBlazorHybridRenderModes();
```

Dans le fichier `_Imports.razor` de la RCL `.Shared.Client`, ajoutez `@using static InteractiveRenderSettings` pour rendre les propriétés de la classe `InteractiveRenderSettings` disponibles pour les composants :

razor

```
@using static InteractiveRenderSettings
```

> **Notes**
>
> L'affectation des modes de rendu via les propriétés de classe RCL `InteractiveRenderSettings` diffère d'une application web Blazor autonome classique. Dans une application web Blazor, les modes de rendu sont fournis normalement par RenderMode via l'instruction `@using static Microsoft.AspNetCore.Components.Web.RenderMode` dans le fichier `_Import` de l'application web Blazor.

Utilisation d'interfaces pour prendre en charge différentes implémentations d'appareils

L'exemple suivant montre comment utiliser une interface pour appeler différentes implémentations dans l'application web et l'application MAUI (native). L'exemple suivant crée un composant qui affiche le facteur de forme de l'appareil. Utilisez la couche d'abstraction MAUI pour les applications natives et fournissez une implémentation pour l'application web.

Dans la bibliothèque de classes (RCL) Razor, un dossier `Interfaces` contient une interface `IFormFactor`.

`Interfaces/IFormFactor.cs` :

C#

```
namespace MauiBlazorWeb.Shared.Interfaces;

public interface IFormFactor
{
    public string GetFormFactor();
    public string GetPlatform();
}
```

Le composant `DeviceFormFactor`, suivant est présent dans le dossier `Components` de la RCL.

`Components/Pages/DeviceFormFactor.razor` :

razor

```
@page "/device-form-factor"
@using MauiBlazorWeb.Shared.Interfaces
@inject IFormFactor FormFactor

<PageTitle>Form Factor</PageTitle>

<h1>Device Form Factor</h1>

<p>You are running on:</p>

<ul>
    <li>Form Factor: @factor</li>
    <li>Platform: @platform</li>
</ul>

<p>
    <em>This component is defined in the MauiBlazorWeb.Shared library.</em>
</p>

@code {
    private string factor => FormFactor.GetFormFactor();
    private string platform => FormFactor.GetPlatform();
}
```

Dans la RCL, une entrée pour le composant `DeviceFormFactor` fait parti du menu de navigation dans le composant `NavMenu`.

Dans `Components/Layout/NavMenu.razor` :

razor

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="device-form-factor">
        <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span>
Form Factor
    </NavLink>
</div>
```

Les applications web et natives contiennent les implémentations pour `IFormFactor`.

Dans l'application web Blazor, un dossier nommé `Services` contient le fichier `FormFactor.cs` suivant avec l'implémentation `FormFactor` pour l'utilisation de l'application web.

`Services/FormFactor.cs` (Blazor projet application web) :

C#

```
using MauiBlazorWeb.Shared.Interfaces;

namespace MauiBlazorWeb.Web.Services;

public class FormFactor : IFormFactor
{
    public string GetFormFactor()
    {
        return "Web";
    }
    public string GetPlatform()
    {
        return Environment.OSVersion.ToString();
    }
}
```

Dans le projet MAUI, un dossier nommé `Services` contient le fichier `FormFactor.cs` suivant avec l'implémentation `FormFactor` pour une utilisation native. La couche d'abstractions MAUI est utilisée pour écrire du code qui fonctionne sur toutes les plateformes d'appareils natives.

`Services/FormFactor.cs` (Projet MAUI) :

C#

```
using MauiBlazorWeb.Shared.Interfaces;

namespace MauiBlazorWeb.Maui.Services;

public class FormFactor : IFormFactor
{
    public string GetFormFactor()
    {
        return DeviceInfo.Idiom.ToString();
    }
    public string GetPlatform()
    {
        return DeviceInfo.Platform.ToString() + " - " +
DeviceInfo.VersionString;
    }
}
```

Vous utilisez l'injection de dépendances pour obtenir les implémentations de ces services.

Dans le projet MAUI, ouvrez le fichier `MauiProgram.cs` et ajoutez les instructions `using` suivantes en haut du fichier :

C#

```
using MauiBlazorWeb.Maui.Services;
using MauiBlazorWeb.Shared.Interfaces;
```

Juste avant l'appel à `builder.Build()`, `FormFactor` est inscrit pour ajouter des services spécifiques à l'appareil utilisés par la RCL :

C#

```
builder.Services.AddSingleton<IFormFactor, FormFactor>();
```

Dans l'application Blazor, le fichier `Program` a les instructions `using` suivantes en haut du fichier :

C#

```
using MauiBlazorWeb.Shared.Interfaces;
using MauiBlazorWeb.Web.Services;
```

Juste avant l'appel à `builder.Build()`, `FormFactor` est inscrit pour ajouter des services spécifiques à l'appareil utilisés par l'application web Blazor :

C#

```
builder.Services.AddScoped<IFormFactor, FormFactor>();
```

Si la solution cible également WebAssembly via un projet `.Web.Client`, une implémentation de l'API précédente est également requise dans le projet `.Web.Client`.

Vous pouvez également utiliser des directives de préprocesseur du compilateur dans votre RCL pour implémenter une interface utilisateur différente en fonction de l'appareil sur lequel l'application s'exécute. Pour ce scénario, l'application doit cibler plusieurs cibles de la liste RCL comme l'application MAUI. Pour obtenir un exemple, consultez le BethMassi/BethTimeUntil dépôt GitHub .

Ressources supplémentaires

- Modes de rendu ASP.NET Core Blazor
- Réutilisez des composants Razor dans les applications ASP.NET Core Blazor Hybrid

Collaborer avec nous sur GitHub

La source de ce contenu se trouve sur GitHub, où vous pouvez également créer et examiner les problèmes et les demandes de tirage. Pour plus d'informations, consultez notre guide du contributeur.

Commentaires sur ASP.NET Core

ASP.NET Core est un projet open source. Sélectionnez un lien pour fournir des commentaires :

Ouvrir un problème de documentation

Indiquer des commentaires sur le produit