Macoratti.net Xamarin Forms - Usando o Local File Storage (PCL Storage) - I

Neste artigo vou mostrar como usar a **API PCL Storage** para acessar informações em uma aplicação Xamarin Forms usando no Visual Studio 2017 e a linguagem C#.

Sabemos que o código Xamarin.Forms é executado em várias plataformas e que cada uma tem seu próprio sistema de arquivos e cada uma usa diferentes APIS para ler, escrever, apagar e realizar outras operações com arquivos. Dessa forma temos um problema quando o assunto é acessar arquivos no ambiente multiplataforma, pois não temos uma API para compartilhar os recursos do File System no Xamarin Forms.

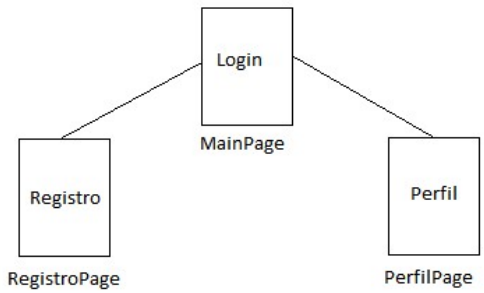
Existe no entanto uma forma bem simples de contornar esse problema que é usar uma bilblioteca de terceiros chamada PCL Storage através da qual podemos compartilhar o compartlhamento de arquivos para todas as plataformas sem ter que escrever código separado para cada plataforma. A PCL Storage fornece um conjunto consistente de APIs IO para arquivos locais para Android, iOS , Windows Store e Windows Phone tornando mais simples o tratamento de arquivos no ambiente multiplataforma.

As API do Plugin do Sistema de Arquivos para Xamarin e Windows são as interfaces **IFile, IFolder e IFileSystem**. A interface **IFileSystem** é o principal ponto de entrada da API. Você pode obter uma instância da implementação para a plataforma atual com a propriedade **FileSystem.Current**.

Para mostrar como usar os recursos da PCL Storage vamos criar uma aplicação Xamarin Forms simulando uma página de **Login**, **Registro e Perfil** onde iremos ler e persistir informações de arquivos no sistema local de arquivos.

Nossa aplicação terá 3 páginas :

1. **MainPage.xaml** - Página para Login do usuário;
2. **RegistroPage.xaml** - página para registrar um novo usuário;
3. **PerfilPage.xaml** - Página de perfil do usuário;



As informações serão lidas e persistidas em arquivos textos armazenados no sistema de arquivos usando o plugin PCL Storage.

Vamos criar uma classe **PCLHelper** contendo os métodos para realizar as operações com arquivos no projeto PCL.

Recursos usados:

- Visual Studio Community 2017 ou Xamarin Studio
- Xamarin

Criando o projeto no Visual Studio 2017 Community e definindo a página principal

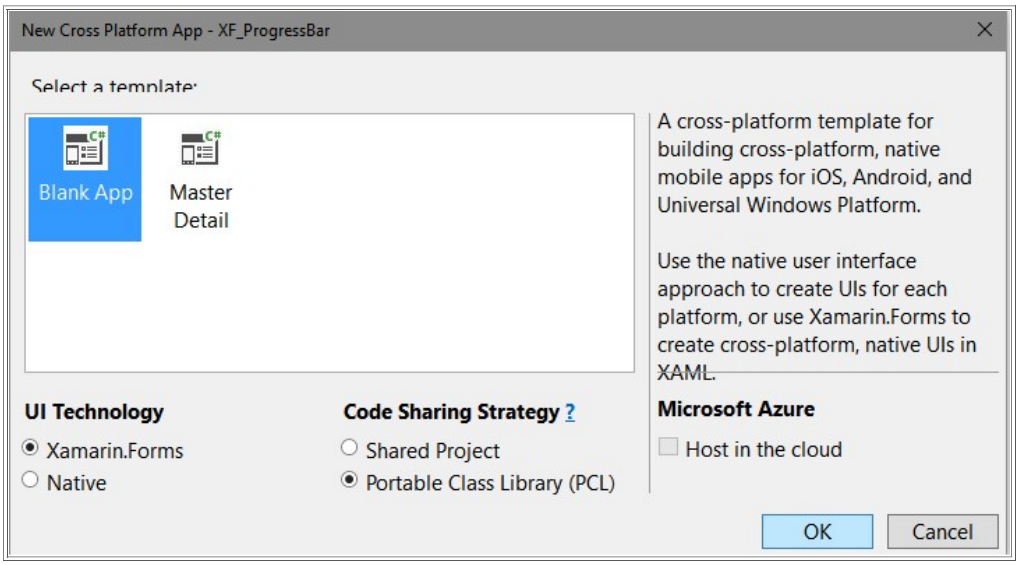
Abra o Visual Studio Community 2017 e clique em **New Project;**

Selecione Visual C#, o template **Cross Plataform** e a seguir **Cross Plataform App(Xamarin.Forms or Native);**

Informe o nome **XF_PclStorage** e clique no botão OK;

A seguir selecione **Blank App** e marque as opções - **Xamarin.Forms** e **Portable Class Library (PCL**) e clique em OK;

Nota: Verifique a necessidade de atualizar o Xamarin Forms no seu projeto.



Será criado um projeto contendo no projeto **Portable** as páginas **App.xaml e MainPage.xaml.**

No code-behind do arquivo **App.xaml** temos a classe **App.cs** que irá conter o código compartilhado e que vamos usar neste artigo.

No arquivo App.cs inclua o código a seguir onde definimos a página principal da aplicação como sendo a página **MainPage**:

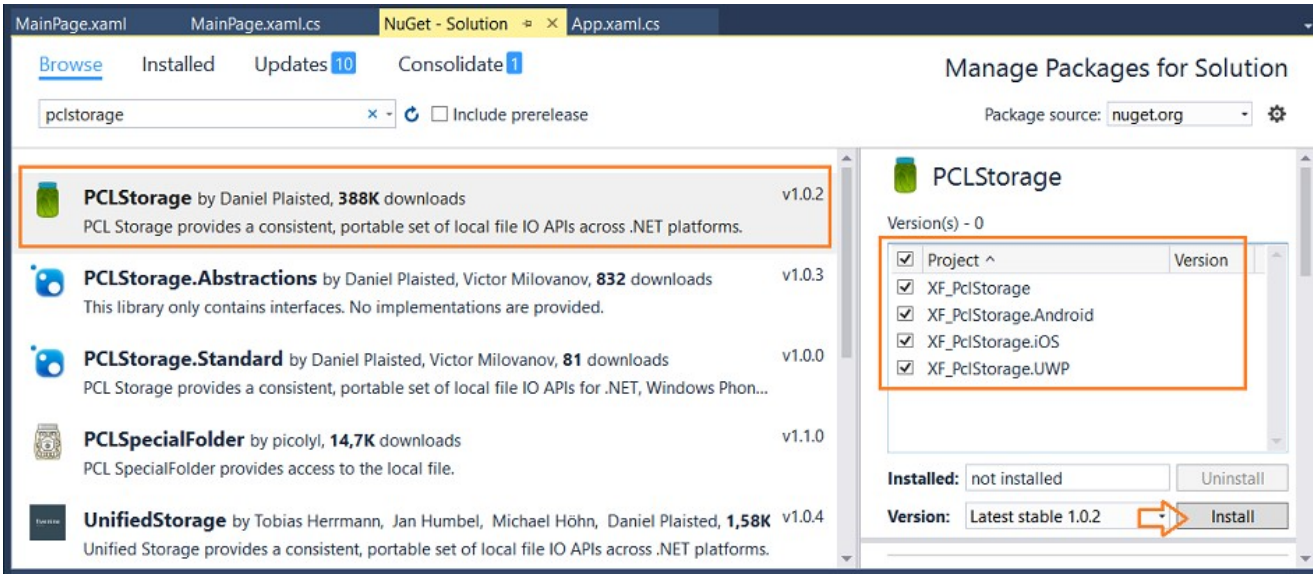
```
using Xamarin.Forms;

namespace XF_RealmDB
{
   public partial class App : Application
   {
      public App()
      {
         InitializeComponent();

         MainPage = new NavigationPage(new XF_PclStorage.MainPage());
      }

      protected override void OnStart()
      {
         // Handle when your app starts
      }

      protected override void OnSleep()
      {
         // Handle when your app sleeps
      }

      protected override void OnResume()
      {
         // Handle when your app resumes
      }
   }
}
```

Ao final teremos as referências incluídas em todos os projetos da nossa solução. Iremos usar a página **MainPage.xaml** como página principal da nossa aplicação.

Incluindo a referência a API PCL Storage no projeto

Agora vamos incluir uma referência ao pacote do PCL Storage via Nuget.

No menu **Tools** clique em **Nuget Package Manager -> Manage Nuget Packages for Solution** e selecione o pacote **PCLStorage**:



Selecione todos os projetos e clique no botão **Install**.

Criando a classe PCLHelper no projeto PCL

Vamos criar uma a classe **PCLHelper** onde iremos definir os métodos para realizar as operações com os arquivos do sistema usando os recursos da PCL Storage.

Crie um arquivo **PCLHelper.cs** e defina o código abaixo neste arquivo:

```
using PCLStorage;
using System;
using System.Threading.Tasks;

namespace XF_PclStorage
{
   public static class PCLHelper
   {
      public async static Task<bool> ArquivoExisteAsync(this string nomeArquivo, IFolder pastaRaiz = null)
      {
         IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
         ExistenceCheckResult pastaExiste = await pasta.CheckExistsAsync(nomeArquivo);
         // não sobrescreve se existir
         if (pastaExiste == ExistenceCheckResult.FileExists)
         {
            return true;
         }
         return false;
      }

      public async static Task<bool> PastaExisteAsync(this string nomeDaPasta, IFolder pastaRaiz = null)
      {
         IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
         ExistenceCheckResult pastaExiste = await pasta.CheckExistsAsync(nomeDaPasta);
         // não sobrescreve se existir
         if (pastaExiste == ExistenceCheckResult.FolderExists)
         {
            return true;
         }
         return false;
      }
```

```
public async static Task<IFolder> CriarPasta(this string nomeDaPasta, IFolder pastaRaiz = null)
{
   IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
   pasta = await pasta.CreateFolderAsync(nomeDaPasta, CreationCollisionOption.ReplaceExisting);
   return pasta;
}

public async static Task<IFile> CriarArquivo(this string nomeArquivo, IFolder pastaRaiz = null)
{
   IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
   IFile arquivo = await pasta.CreateFileAsync(nomeArquivo, CreationCollisionOption.ReplaceExisting);
   return arquivo;
}

public async static Task<bool> WriteTextAllAsync(this string nomeArquivo, string content = "", IFolder pastaRaiz = null)
{
   IFile arquivo = await nomeArquivo.CriarArquivo(pastaRaiz);
   await arquivo.WriteAllTextAsync(content);
   return true;
}

public async static Task<string> ReadAllTextAsync(this string nomeArquivo, IFolder pastaRaiz = null)
{
   string conteudo = "";
   IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
   bool existe = await nomeArquivo.ArquivoExisteAsync(pasta);
   if (existe == true)
   {
      try
      {
         IFile arquivo = await pasta.GetFileAsync(nomeArquivo);
         conteudo = await arquivo.ReadAllTextAsync();
      }
      catch
      {
         throw new Exception("Erro ao ler arquivo.");
      }
   }
   return conteudo;
}
public async static Task<bool> DeleteFile(this string nomeArquivo, IFolder pastaRaiz = null)
{
   IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;
   bool existe = await nomeArquivo.ArquivoExisteAsync(pasta);
   if (existe == true)
   {
      IFile arquivo = await pasta.GetFileAsync(nomeArquivo);
      await arquivo.DeleteAsync();
      return true;
   }
   return false;
}
public async static Task SaveImagem(this byte[] imagem, String nomeArquivo, IFolder pastaRaiz = null)
{
   IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;

   // cria o arquivo e sobrescreve
   IFile arquivo = await pasta.CreateFileAsync(nomeArquivo, CreationCollisionOption.ReplaceExisting);

   // popula o arquivo com dados da imagem
   using (System.IO.Stream stream = await arquivo.OpenAsync(FileAccess.ReadAndWrite))
   {
      stream.Write(imagem, 0, imagem.Length);
   }
}
```

```
        public async static Task<byte[]> LoadImagem(this byte[] imagem, String nomeArquivo, IFolder pastaRaiz = null)
        {
            IFolder pasta = pastaRaiz ?? FileSystem.Current.LocalStorage;

            //abre se o arquivo existir
            IFile arquivo = await pasta.GetFileAsync(nomeArquivo);
            //carrega o stream para o buffer
            using (System.IO.Stream stream = await arquivo.OpenAsync(FileAccess.ReadAndWrite))
            {
                long length = stream.Length;
                byte[] streamBuffer = new byte[length];
                stream.Read(streamBuffer, 0, (int)length);
                return streamBuffer;
            }
        }
    }
}
```

No código estamos temos os métodos para verificar se um arquivo ou pasta existem e também métodos para escrever e ler arquivos textos e imagens.

Definindo o código a página principal : MainPage

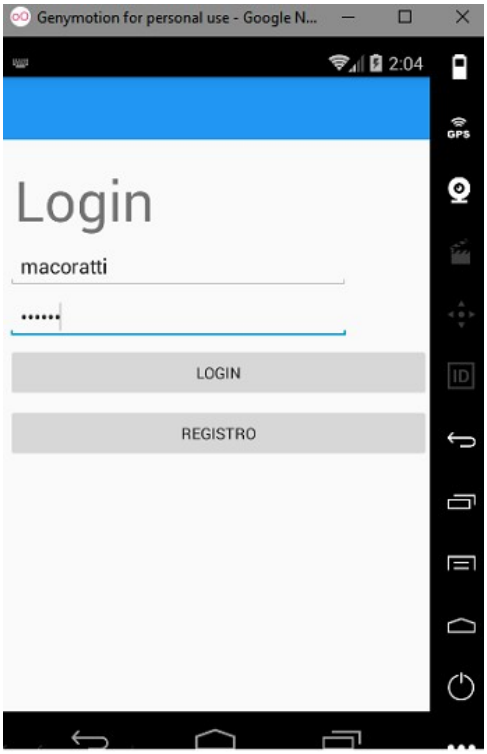
Agora vamos definir o código XAML da página **MainPage** :

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:XF_PclStorage"
             x:Class="XF_PclStorage.MainPage"
             Padding="0, 20, 0, 0">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="300" />
        </Grid.ColumnDefinitions>

        <Label Text="Login" Grid.Row="0" Grid.Column="2" FontSize="50" />
        <Entry Placeholder="Usuário" x:Name="txtUsuario" Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="2"/>
        <Entry IsPassword="True" x:Name="txtSenha" Placeholder="Senha" Grid.Row="2" Grid.Column="1" Grid.ColumnSpan="2"/>

        <StackLayout Grid.Row="3" Grid.Column="1" Grid.ColumnSpan="3">
            <Button Text="Login" Clicked="btnLogin_Click" />
            <Button Text="Registro" Clicked="btnRegistro_Click" />
        </StackLayout>
    </Grid>
</ContentPage>
```

Neste código estamos usando a View **Entry** para o usuário informar o **nome e senha** do usuário. Definimos também os seguintes eventos dos botões de comando :

- **bntLogin_Click -** Trata o login do usuário;
- **btnRegistro_Click -** Trata do registro de um novo usuário:

O leiaute do código XAML pode ser visto na figura baixo:



Agora vamos definir o código do arquivo **MainPage.xaml.cs** é mostrado a seguir:

```
using System;
using Xamarin.Forms;

namespace XF_PclStorage
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }

        string[] valor;
        string nome;
        string usuario;
        string senha;

        private async void btnLogin_Click(object sender, EventArgs e)
        {
            bool existe = await PCLHelper.ArquivoExisteAsync(txtUsuario.Text);

            if (existe != true)
            {
                await DisplayAlert("Login", "Login falhou... Tente Novamente...", "OK");
            }
            else
            {
                LeInfoUsuario(txtUsuario.Text);
                if (usuario == txtUsuario.Text && senha == txtSenha.Text)
                {
                    await DisplayAlert("Login", "Login feito com sucesso ... \nAgora Edite o seu perfil", "OK");
                    await Navigation.PushModalAsync(new PerfilPage(txtUsuario.Text));
                }
                else
                {
                    await DisplayAlert("Login", "Login falhou... Tente Novamente...", "OK");
                    txtUsuario.Text = "";
                    txtSenha.Text = "";
```

```
                txtUsuario.Focus();
            }
        }
    }

    public async void LeInfoUsuario(string arquivo)
    {
        string conteudo = await PCLHelper.ReadAllTextAsync(arquivo);
        valor  = conteudo.Split('\n');
        nome = valor[0].ToString();
        usuario = valor[1].ToString();
        senha = valor[2].ToString();
    }

    private async void btnRegistro_Click(object sender, EventArgs e)
    {
        await Navigation.PushModalAsync(new RegistroPage());
    }
  }
}
```

Neste código temos a implementação dos eventos dos botões de comando onde no evento **Click** do botão **Registro** chamamos a página **RegistroPage** e se o login foi feito com sucesso chamamos a página **PerfilPage**.

Primeiro verificamos se o arquivo existe usando o método **ArquivoExisteAsync()** da classe **PCLHelper** e seguir lemos as informações do arquivo usando o método **ReadAllTextAsync**.

Como os métodos são assíncronos estamos usando as cláusulas **await e asycn.**

Na **segunda parte do artigo** vamos implementar as páginas **RegistroPage** e **PerfilPage**.

Porque todos quantos fostes batizados em Cristo já vos revestistes de Cristo.
Nisto não há judeu nem grego; não há servo nem livre; não há macho nem fêmea; porque todos vós sois um em Cristo Jesus.
Gálatas 3:27-28

ReportViewer no VS 2013 ?

- **Curso - Gerando Relatórios com o ReportViewer no VS 2013 - Vídeo Aulas** NEW

Referências:

- Seção VB .NET do Site Macoratti.net
- Super DVD .NET - A sua porta de entrada na plataforma .NET
- Super DVD Vídeo Aulas - Vídeo Aula sobre VB .NET, ASP .NET e C#
- Super DVD C# - Recursos de aprendizagens e vídeo aulas para C#
- Seção C# do site Macoratti.net
- Seção ASP .NET do site Macoratti .net
- Curso Básico VB .NET - Vídeo Aulas
- Curso C# Básico - Vídeo Aulas
- Curso Fundamentos da Programação Orientada a Objetos com VB .NET NEW
- **Macoratti .net | Facebook**
- **macoratti - YouTube**
- **Jose C Macoratti (@macorati) | Twitter**
- VB.NET 2005 - Controles - Macoratti.net
- Seção de Jogos do site Macoratti .net
- Xamarim - Desenvolvimento Multiplataforma com C# ... - Macoratti.net
- Xamarin - Apresentando Xamarin.Forms - Macoratti.net
- Xamarin.Forms - Olá Mundo - Criando sua primeira ... - Macoratti.net
- Xamarin.Forms - Olá Mundo - Anatomia da aplicação - Macoratti.net
- https://developer.xamarin.com/api/type/Android.App.AlertDialog/
- Xamarin Android - Tratando eventos de forma declarativa
- Seção Mobile/Xamarin do site Macoratti .net
- https://developer.android.com/reference/android/app/Activity.html
- https://developer.xamarin.com/api/type/Android.Widget.ProgressBar/

José Carlos Macoratti