FIAP

Disciplina	Desenvolvimento Mobile Cross-Platform (Xamarin)
Professor	Flávio Secchieri Mariotti

1. Introdução

Este documento contém instruções detalhadas de como criar um aplicativo com Xamarin.Forms e trabalhar com dados locais por meio do SQLite (https://github.com/praeclarum/sqlite-net).

2. Projeto

O projeto se concentra em apresentar como criar aplicativos que possam ler e escrever dados locais por meio do SQLite Database. O projeto consiste de duas páginas. A página Lista Alunos, lista todos os alunos registrados na tabela ALUNO existente na base de dados FIAPDB.DB3. A página Novo Aluno, permite o cadastrar novos alunos.

2.1. Passo 1 – Criar o Projeto

Para criar um novo projeto Xamarin.Forms, abra o Visual Studio 2017 e clique em **File, New** e **Project**. Será apresentado uma janela com todos os templates instalados, expanda a opção Templates, Visual C# e clique em Cross-Platform.

Na parte superior da janela, verifique se a versão do .NET é a .NET Framework 4.6. Assim, marque o tipo de projeto - Blank App (Xamarin.Forms Portable). Ainda na janela novo projeto, informe o nome do projeto, neste exemplo, usaremos o nome XF.LocalDB e informe a localização de sua preferência, em seguida, clique no botão OK para criar o projeto.

NOTA: Se a opção “Criar diretório para solução” estiver marcado, o Visual Studio criará um novo diretório para o projeto (csproj), separado da solução (sln).

Caso tenha seguido todas as instruções corretamente, a janela Novo Projeto estará conforme Figura 1.

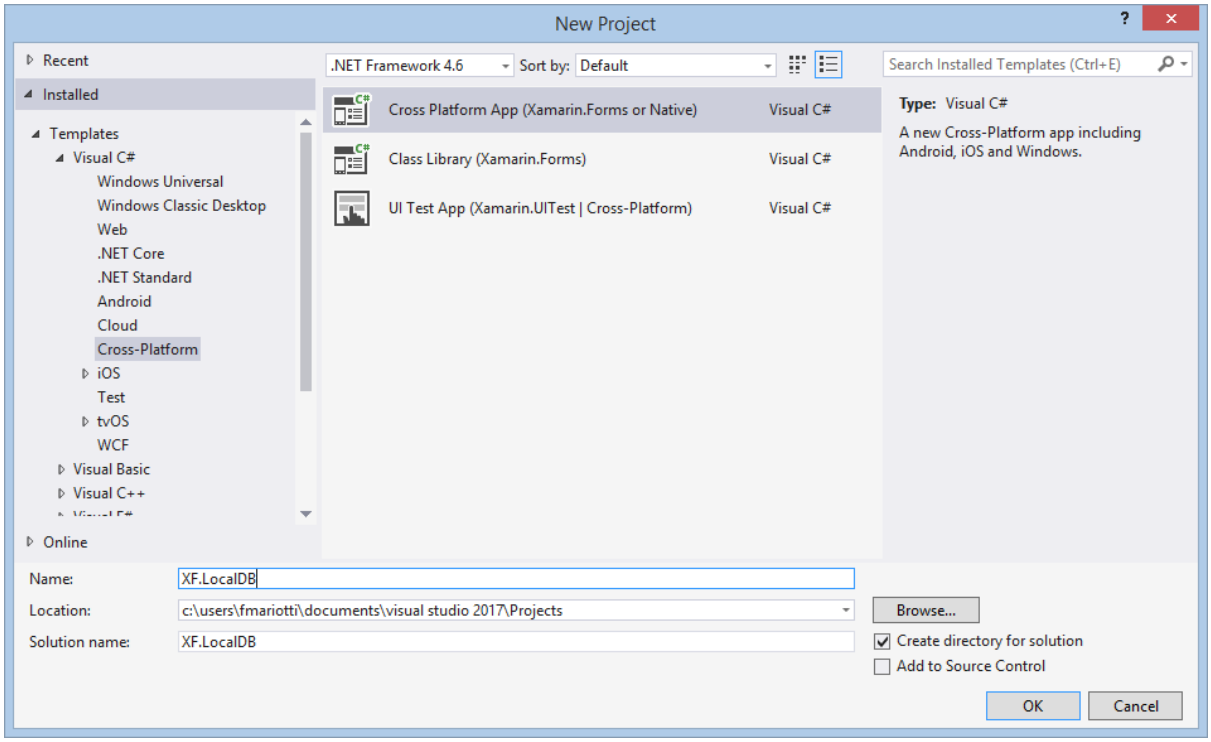


Figure 1. Janela Novo Projeto.

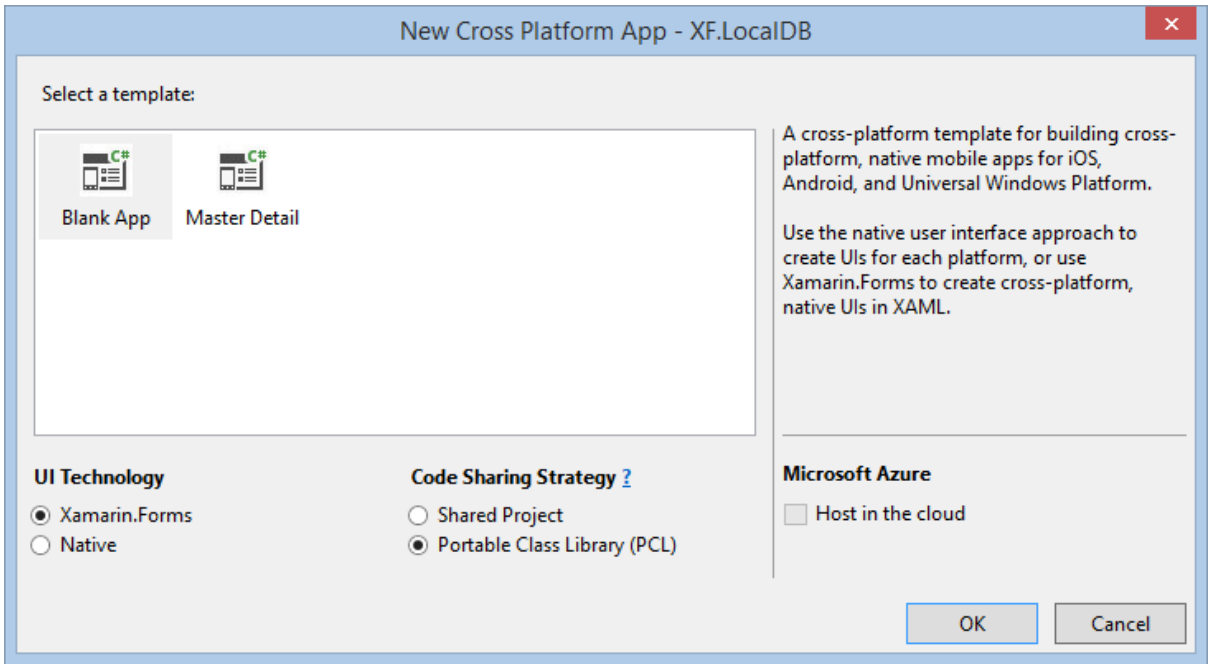


Figure 2. Janela Novo Projeto – Selecione o tipo de projeto.

NOTA: o projeto criado é do tipo Portable Class Library (PCL) o que permite utilizar bibliotecas de classes portáteis, tais como, SQLite, Json.NET ou ReactiveUI em todas as plataformas por meio de DI. Diferentemente do tipo Shared Asset Project (SAP) que permite anexar qualquer arquivo em um

único projeto e compartilhar automaticamente em todas as plataformas (código, imagens e qualquer outra mídia em iOS, Android e Windows). Saiba mais em: Introduction to Portable Class Libraries e Shared Projects.

Durante a criação do dos projetos, será apresentado a tela de configuração do Xamarin Mac Agent Instructions, o que permite conectar o PC com uma máquina MacOS, conforme Figura 3.

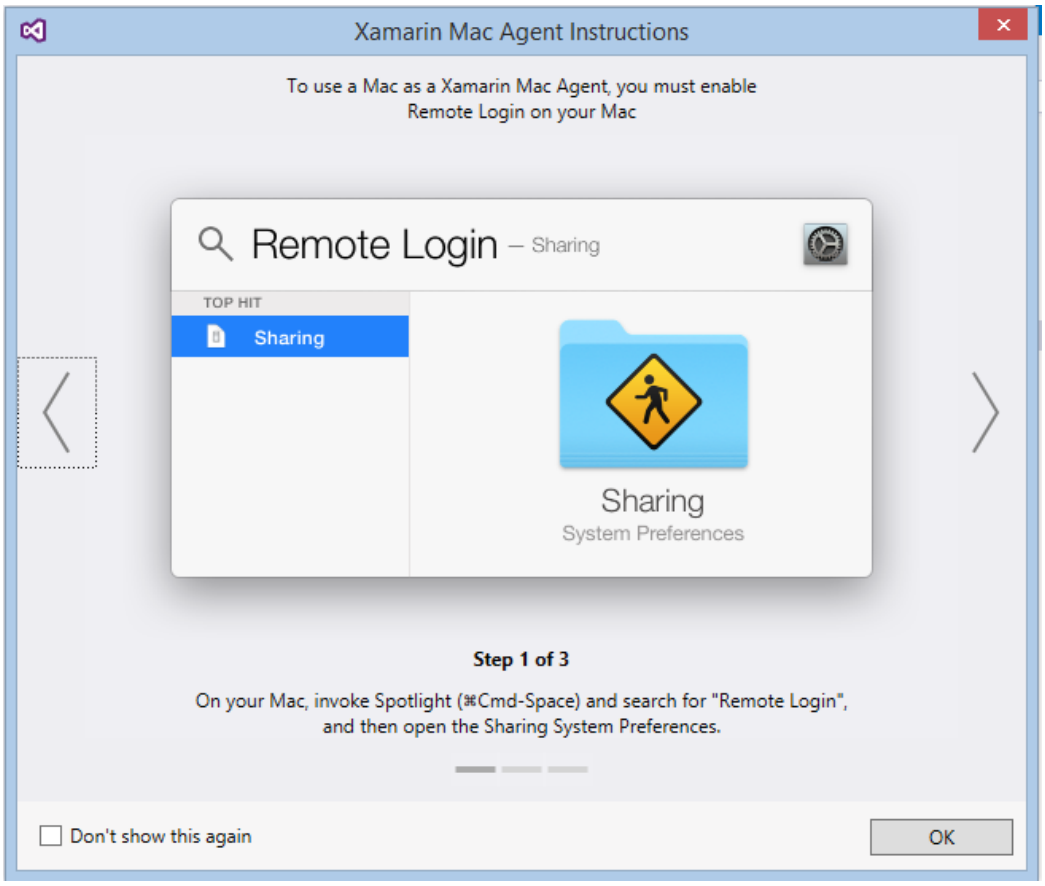


Figure 3. Xamarin Mac Agent Instructions.

Além disso, para criar projetos no Windows (UWP) será apresentado uma nova janela, solicitando a configuração referente versão mínimo e máximo do sistema operacional que deverá oferecer suporte à aplicação XF.LocalDB.UWP.

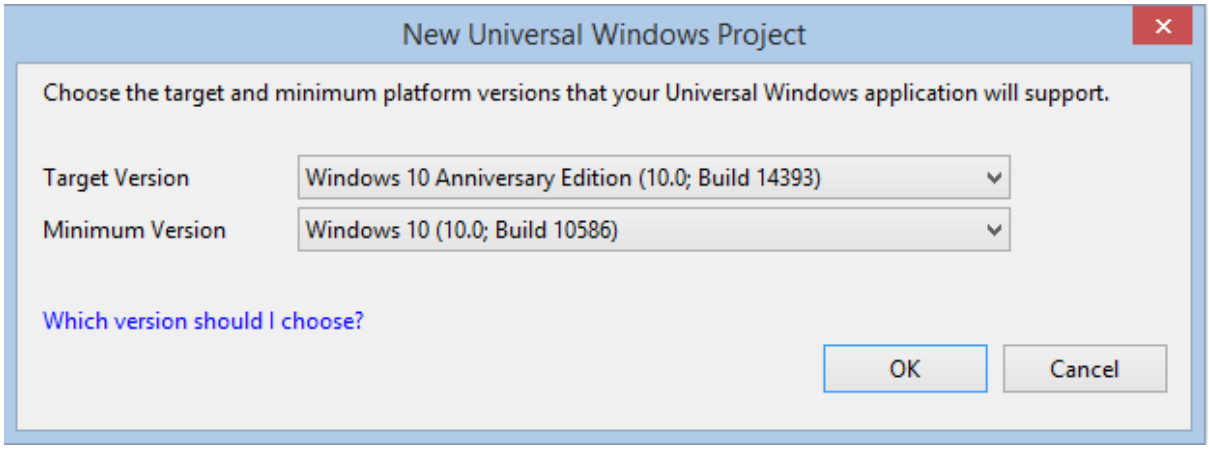
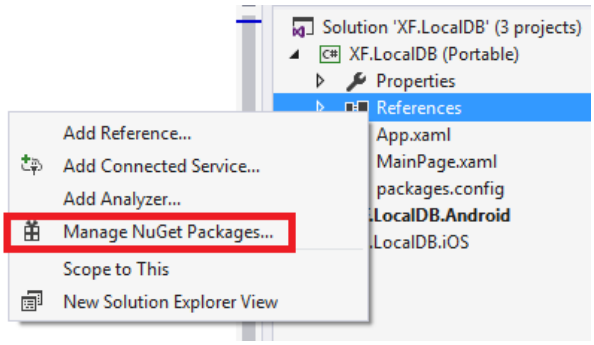


Figure 4. Selecionar a versão mínimo e máxima do Windows 10.

2.2. Passo 2 – Configurando o componente SQLite

Para instalar o componente SQLite em projetos PCL, clique com o botão direito do mouse em References e escolha a opção Manage NuGet Packages.



Na janela de gestão do NuGet, clique na guia Browse e pesquise pelo componente "sqlite-net-pcl". Antes de instalar, certifique-se que a versão selecionada é a mesma apresentada na Figura 5.

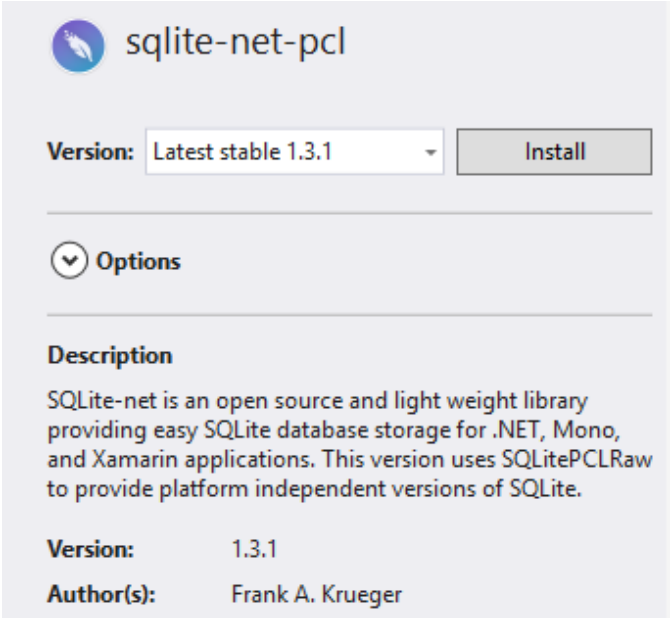


Figure 5. SQLite-net-pcl criada por Frank A. Krueger.

NOTA: existem várias versões similares do SQLite disponíveis no NuGet, logo, certifique-se que instalou a versão correta, caso contrário, o projeto poderá falhar com erros de compatibilidade.

Devido as diferenças na estrutura de organização dos diretórios e arquivos entre as plataformas, será necessário implementar separadamente as instruções e suas especificidades de criação e manipulação do arquivo (db3) por plataforma. Isto é, todas as plataformas deverão ter suporte ao componente SQLite. Para isso, execute o Passo 2 e instale o SQLite-net-pcl em todas as plataformas que irão trabalhar com SQLite.

2.3. Passo 3 – Criando o objeto de conexão

A fim de organizar os objetos relacionados a base de dados, crie no projeto XF.LocalDB uma pasta chamada “Data”. Dentro desta nova pasta, crie um novo objeto do tipo Interface chamada ISQLite, conforme instruções da Tabela 1.

Tabela 1. Interface ISQLite

`namespace XF.LocalDB.Data`
`{`
`  public interface ISQLite`
`  {`
`  SQLiteConnection GetConexao();`
`  }`
`}`
	- Através do padrão Dependency Injection, é possível escrever instruções especificas de cada plataforma em projetos do tipo Xamarin.Forms Portable. Para isso, usaremos a classe DependecyService disponível no Xamarin.Forms.
	
	- A interface ISQLite será substituída em tempo de execução pela classe concreta que contém as instruções necessárias para gestão do arquivo que representa nosso banco de dados.

2.3.1. Dependency Service

DependencyService permite que o desenvolvedor implemente instruções especificas de cada plataforma e invoque tais funcionalidades a partir do código compartilhado. Esse recurso é considerado extremamente poderoso, por permitir que qualquer funcionalidade nativa possa ser implementada por meio do Xamarin.Forms. DependencyService é responsável por resolver as dependências existente na aplicação. Na prática, uma interface é definida e o DependencyService localiza sua implementação, o que pode ser diferente em cada plataforma.

São necessários alguns passos para implementar o pattern Dependency Injection com Xamarin.Forms:

- Interface: a funcionalidade é definida por uma interface compartilhada.
- Implementação por Plataforma: as classes com o comportamento da Interface serão adicionadas dentro de cada projeto (Android, iOS e UWP).
- Registro: cada implementação requer o registro através da notação de atributo. O registro é responsável por permitir que a classe implementada seja localizada pelo DependencyService a partir da Interface.
- Invocar: no projeto compartilhado se faz necessário invocar o método usando explicitamente o objeto DependencyService.

Observe que cada implementação deve ser feita por plataforma, uma vez não implementado, a chamada deverá falhar e disparar uma exceção. A estrutura funcional do aplicativo pode ser melhor compreendida na Figura 6:

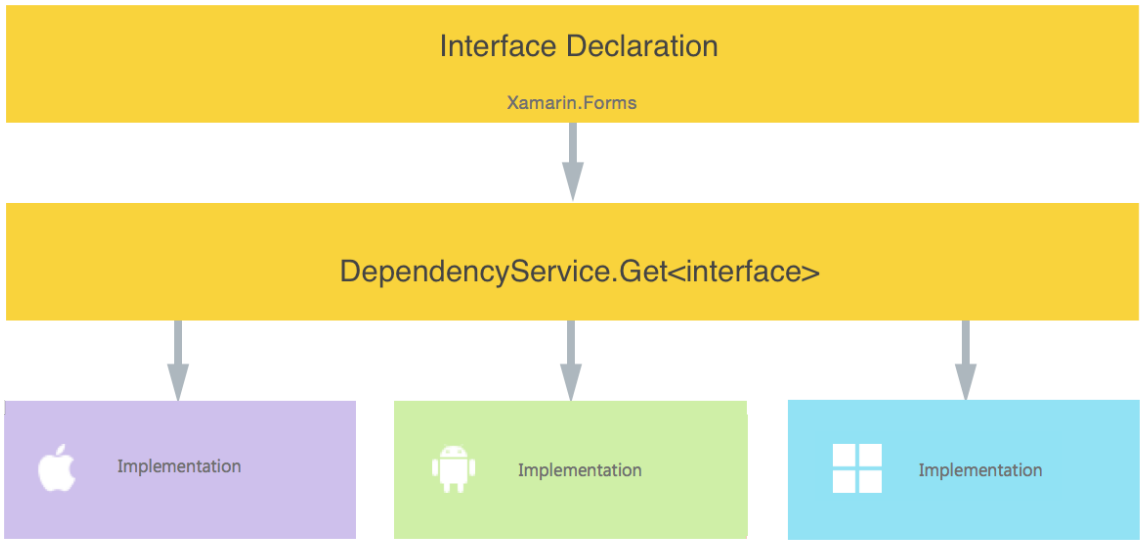


Figure 6. Representação lógica da classe DependecyService.

2.4. Passo 4 – Criando o objeto View

O objeto View é responsável por definir a aparência ou estrutura que o usuário vê na tela. O ideal é que o código fonte por trás da visualização (codebehind's view), contenha apenas a chamada ao método InitializeComponent. A View se conecta ao ViewModel, através da propriedade BindingContext que é implementada pela classe ViewModel correspondente à aquela View.

2.4.1. Página Lista Aluno

Para organização dos objetos criados dentro de um projeto Xamarin.Forms baseado em arquitetura MVVM, recomenda-se como boa prática de programação o uso da convenção que propõe a distribuição das classes nas respectivas pastas: Model, View e ViewModel.

Neste sentido, crie uma nova pasta no projeto XF.LocalDB com o nome View, para isso, clique com o botão direito do mouse no projeto e selecione a opção **Add**, **New Folder**. Em seguida, dentro da pasta View, crie uma nova pasta com o nome Aluno. Dentro da pasta Aluno, crie um novo objeto do tipo Page, relativo a página Lista Alunos, clique com o botão direito do mouse sobre a basta Aluno e crie novo objeto do tipo "*Forms Blank Content Page Xaml*", informe o nome MainPage.cs, uma vez que essa será a página inicial referente a alunos.

Neste exemplo, a implementação da página Lista Aluno está dividida em duas partes, visualização (xaml) e código-fonte (codebehind). Insira o código localizado na Tabela 2 no arquivo XAML e o código-fonte localizado na Tabela 3 no codebehind (.cs).

Tabela 2. Instruções XAML da página MainPage.xaml

`<ContentPage.Content>`
`<StackLayout Padding="20">`
`<StackLayout`
`Orientation="Horizontal"`
 	
`VerticalOptions="Center"`
`HorizontalOptions="Center">`
`<Button x:Name="btnNovo"`
`Text="Novo" Clicked="OnNovo" />`
`</StackLayout>`
`<Label Text="Alunos" Font="18" />`
`<ListView x:Name="lstAlunos"`
`ItemsSource="{Binding`
`Alunos}" ItemTapped="OnAlunoTapped">`
`<ListView.ItemTemplate>`
`<DataTemplate>`
`<ViewCell>`
`<StackLayout`
`Padding="5,0,5,0">`
`<Label Text="{Binding`
`Nome}" Font="14" />`
`<Label Text="{Binding`
`Email}" Font="10" />`
`</StackLayout>`
`</ViewCell>`
`</DataTemplate>`
`</ListView.ItemTemplate>`
	StackLayout – elemento de layout que permite organizar os objetos de visualização de modo horizontal ou vertical.
	
	Button – elemento de visualização que permite invocar um comando do tipo evento.
	
	Label – elemento de visualização que cria uma etiqueta de texto.
	
	ListView – elemento visual que permite mostrar um conjunto de dados no formado de lista.
	
	{Binding DataSource} – poderoso recurso suportado no XAML para ligação entre objeto concreto (fonte) e visualização (xaml).

`</ListView>`
`</StackLayout>`
`</ContentPage.Content>`	

Tabela 3. Instruções C# da página MainPage.cs

`public partial class MainPage : ContentPage`
`{`
`AlunoViewModel vmAluno;`
`public MainPage()`
`{`
`vmAluno = new AlunoViewModel();`
`BindingContext = vmAluno;`
`InitializeComponent();`
`}`
`protected override void OnAppearing()`
`{`
`vmAluno = new AlunoViewModel();`
`BindingContext = vmAluno;`
`base.OnAppearing();`
`}`
`private void OnNovo(object sender, EventArgs args)`
`{`
`Navigation.PushAsync(new NovoAlunoView());`
`}`
`private void OnAlunoTapped(object sender, ItemTappedEventArgs args)`
`{`
`var selecionado = args.Item as XF.LocalDB.Model.Aluno;`
`DisplayAlert("Aluno selecionado", "Aluno: " + selecionado.Id, "OK");`
`}`
`}`
	- Declaração da variável do tipo ViewModel, o que permite acionar eventos da classe que contém o comportamento de apresentação da página.
	
	- Criação da instância ViewModel e associação com a página através da propriedade BindingContext.
	
	- Eventos dos botões Novo Aluno e Seleção de um aluno da lista.

2.4.2. Página Novo Aluno

Repita as instruções citadas no item 2.4.1 e crie dentro da pasta Aluno uma nova classe do tipo "*Content Page*" e informe o nome NovoAlunoView.cs. A implementação da página Novo Aluno está dividida em duas partes, visualização (xaml) e código-fonte (codebehind). Insira o código localizado na Tabela 4 no arquivo XAML e o código-fonte localizado na Tabela 5 no codebehind (.cs).

Tabela 4. Instruções XAML da página NovoAlunoView.xaml

<ContentPage.Content> <StackLayout Padding="20"> <Entry x:Name="txtNome" Placeholder="Nome" /> <Entry x:Name="txtRM" Placeholder="RM" /> <Entry x:Name="txtEmail" Placeholder="E-mail" /> <StackLayout Orientation="Horizontal"> <Label Text="Aprovado?" /> <Switch x:Name="IsAprovado" /> </StackLayout> <StackLayout Orientation="Horizontal"> <Button x:Name="btnSalvar" Text="Salvar" Clicked="OnSalvar" /> <Button x:Name="btnCancelar" Text="Cancelar" Clicked="OnCancelar" /> </StackLayout> </StackLayout> </ContentPage.Content>	StackLayout – elemento de layout que permite organizar os objetos de visualização de modo horizontal ou vertical. Button – elemento de visualização que permite invocar um comando do tipo evento. Label – elemento de visualização que cria uma etiqueta de texto. Switch – elemento visual que permite escolher o valor entre 0 e 1 em um botão que simula o liga (on) e desliga (off).

Tabela 5. Instruções C# da página NovoAlunoView.cs

public partial class NovoAlunoView : ContentPage
{
private int alunoId = 0;
public NovoAlunoView()
{
InitializeComponent();
}
 	
public NovoAlunoView(int Id)
{
InitializeComponent();
 	
var aluno = App.AlunoModel.GetAluno(Id);
txtNome.Text = aluno.Nome;
txtRM.Text = aluno.RM;
txtEmail.Text = aluno.Email;
IsAprovado.IsToggled = aluno.Aprovado;
alunoId = aluno.Id;
}
 	
public void OnSalvar(object sender, EventArgs args)
{
XF.LocalDB.Model.Aluno aluno = new XF.LocalDB.Model.Aluno()
{
Nome = txtNome.Text,
RM = txtRM.Text,
Email = txtEmail.Text,
Aprovado = IsAprovado.IsToggled,
	- Declaração da variável do tipo inteiro que deverá receber e transacionar o código de identificação única do aluno (Id).
	
	- Implementação de uma sobrecarga do construtor, a qual permite passar a identificação de um aluno para edição ou visualização.
	
	- Eventos dos botões Salvar e Cancelar de um aluno da lista.
	
	- O evento Salvar invoca o método SalvarAluno implementado dentro do objeto Model.

Id = alunoId
};
Limpar();
 	
App.AlunoModel.SalvarAluno(aluno);
Navigation.PopAsync();
}
 	
public void OnCancelar(object sender, EventArgs args)
{
Limpar();
Navigation.PopAsync();
}
 	
private void Limpar()
{
txtNome.Text = txtRM.Text = txtEmail.Text = string.Empty;
IsAprovado.IsToggled = false;
}
}	

2.5. Passo 5 – Criando o objeto Model

Crie uma nova pasta no projeto XF.LocalDB com o nome Model, para isso, clique com o botão direito do mouse no projeto citado e selecione a opção **Add**, **New Folder**. Crie um objeto do tipo Class, relativo a classe de entidade de domínio do Aluno. Clique com o botão direito do mouse sobre a basta Model e crie novo objeto do tipo Class com o nome Aluno.cs, **Add**, **New Item**, **Code** e selecione o template Class.

Implemente a classe Aluno de acordo com o código-fonte localizado na Tabela 6.

Tabela 6. Instruções C# da classe Aluno.cs

public class Aluno
{
public Aluno()
{
database = DependencyService.Get<ISQLite>().GetConexao();
database.CreateTable<Aluno>();
}
 	
#region Propriedades
 	
[PrimaryKey, AutoIncrement]
public int Id { get; set; }
public string RM { get; set; }
public string Nome { get; set; }
	- Classe concreta referente da entidade aluno, a qual deverá conter as regras de negócio e acesso a base de dados.
	
	- Anotação de Atributo [PrimaryKey] e [AutoIncrement], respectivamente, responsável por marcar a propriedade como chave primária e assegurar que o campo não aceitará valor nulo, e incremento automático de valor + 1.

```
        public string Email { get; set; }
        public bool Aprovado { get; set; }

        #endregion

        #region Aluno Local Database
        private SQLiteConnection database;
        static object locker = new object();

        public int SalvarAluno(Aluno aluno)
        {
            lock (locker)
            {
                if (aluno.Id != 0)
                {
                    database.Update(aluno);
                    return aluno.Id;
                }
                else return
database.Insert(aluno);
            }
        }

        public IEnumerable<Aluno> GetAlunos()
        {
            lock (locker)
            {
                return (from c in
database.Table<Aluno>()
                        select c).ToList();
            }
        }

        public Aluno GetAluno(int Id)
        {
            lock (locker)
            {
                // return database.Query<
Aluno>("SELECT * FROM [Aluno] WHERE [Id] = " +
Id);
                return
database.Table<Aluno>().Where(c => c.Id ==
Id).FirstOrDefault();
            }
        }

        public int RemoverAluno(int Id)
        {
            lock (locker)
            {
                return
database.Delete<Aluno>(Id);
            }
        }
        #endregion
    }
```

- O .Net Framework implemente o recurso LINQ to SQL que permite trabalhar com arquivos do tipo SQLite. LINQ to SQL é um componente do .NET Framework versão 3.5 que fornece uma infraestrutura em tempo de execução para gerenciar dados relacionais como objetos.

2.6. Passo 6 – Criando o objeto ViewModel

Crie uma nova pasta no projeto XF.LocalDB com o nome AlunoViewModel, para isso, clique com o botão direito do mouse no projeto citado e selecione a opção **Add**, **New Folder**. Crie um novo objeto do tipo Class, relativo a classe que sincroniza o modelo com a visualização, contendo o comportamento lógico e visual da interface do usuário (página). Clique com o botão direito do mouse sobre a basta ViewModel e crie novo objeto do tipo Class com o nome AlunoViewModel.cs, **Add**, **New Item**, **Code** e selecione o template Class.

Implemente a classe AlunoViewModel segundo o código-fonte localizado na Tabela 7.

Tabela 7. Instruções C# da classe AlunoViewModel.cs

`public class AlunoViewModel`
`{`
`public AlunoViewModel() { }`
 	
`#region Propriedades`
`public string RM { get; set; }`
`public string Nome { get; set; }`
`public string Email { get; set; }`
 	
`public List<Aluno> Alunos`
`{`
`get`
`{`
`return App.AlunoModel.GetAlunos().ToList();`
`}`
`}`
`#endregion`
`}`
	- #region – diretriz de compilação utilizada para organizar, normalmente utilizado em arquivos extensos.
	
	- Campos RM, Nome e Email, os quais serão apresentados na página (view).
	
	- Propriedade Alunos, responsável por recuperar os dados da base de dados local, a partir do método GetAlunos implementado no objeto Model.

2.7. Passo 7 – Implementando a classe SQLite por plataforma

Para implementar a classe concreta que será responsável por criar e gerenciar o arquivo físico em cada plataforma, precisamos criar as instruções nos diferentes projetos.

2.7.1. Android

Crie uma nova classe no projeto XF.LocalDB.Android e implemente o código-fonte localizado na Tabela 8.

Tabela 8. Instruções C# da classe SQLite_Android.cs

[assembly:
Dependency(typeof(SQLite_Android))]
namespace XF.LocalDB.Android
{
public class SQLite_Android : ISQLite
{
public SQLite_Android()
{
 	
}
public SQLite.SQLiteConnection
GetConexao()
{
var arquivodb = "fiapdb.db3";
string caminho =
System.Environment.GetFolderPath
(System.Environment.SpecialFolder.Personal);
var local =
Path.Combine(caminho, arquivodb);
 	
var conexao = new
SQLite.SQLiteConnection(local);
return conexao;
}
}
}
	- Cria o arquivo físico no sistema operacional Android, respeitando a estrutura organizacional de pastas e arquivos.

2.7.2. iOS

Crie uma nova classe no projeto XF.LocalDB.iOS e implemente o código-fonte localizado na Tabela 9.

Tabela 9. Instruções C# da classe SQLite_iOS.cs

[assembly: Dependency(typeof(SQLite_iOS))]
namespace XF.LocalDB.iOS
{
public class SQLite_iOS : ISQLite
{
public SQLite_iOS()
{
 	
}
public SQLite.SQLiteConnection
GetConexao()
{
var arquivodb = "fiapdb.db3";
string caminho =
Environment.GetFolderPath(
Environment.SpecialFolder.Personal);
string bibliotecaPessoal =
Path.Combine(caminho, "..", "Library");
var local =
Path.Combine(bibliotecaPessoal, arquivodb);
	- Cria o arquivo físico no sistema operacional Apple iOS, respeitando a estrutura organizacional de pastas e arquivos.

`            var conexao = new SQLite.SQLiteConnection(local);`
`            return conexao;`
`        }`
`    }`
`}`	

2.7.3. Universal Windows Platform (UWP)

Crie uma nova classe no projeto XF.LocalDB.UWP e implemente o código-fonte localizado na Tabela 10.

Tabela 10. Instruções C# da classe SQLite_UWP.cs

`[assembly: Dependency(typeof(SQLite_UWP))]`
`namespace XF.LocalDB.UWP`
`{`
`    public class SQLite_UWP : ISQLite`
`    {`
`        public SQLite_UWP() { }`
`        public SQLite.SQLiteConnection GetConexao()`
`        {`
`            var arquivodb = "fiapdb.db3";`
`            string caminho = Path.Combine( ApplicationData.Current.LocalFolder.Path, arquivodb);`
 	
`            var conexao = new SQLite.SQLiteConnection(caminho);`
`            return conexao;`
`        }`
`    }`
`}`
	- Cria o arquivo físico no sistema operacional Windows 10, respeitando a estrutura organizacional de pastas e arquivos.

2.8. Passo 8 – Configurando a classe App

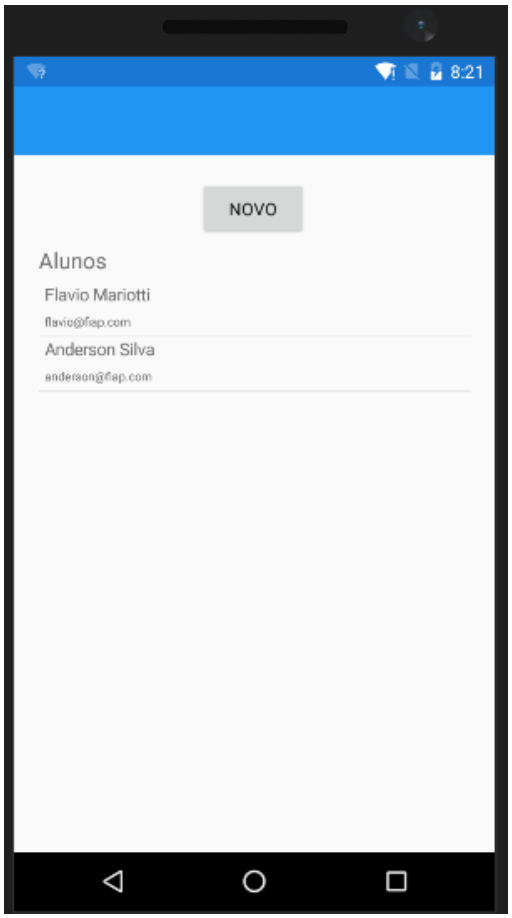
Para finalizarmos o aplicativo, precisamos configurar a página principal do projeto. Implemente a classe App segundo o código-fonte localizado na Tabela 13.

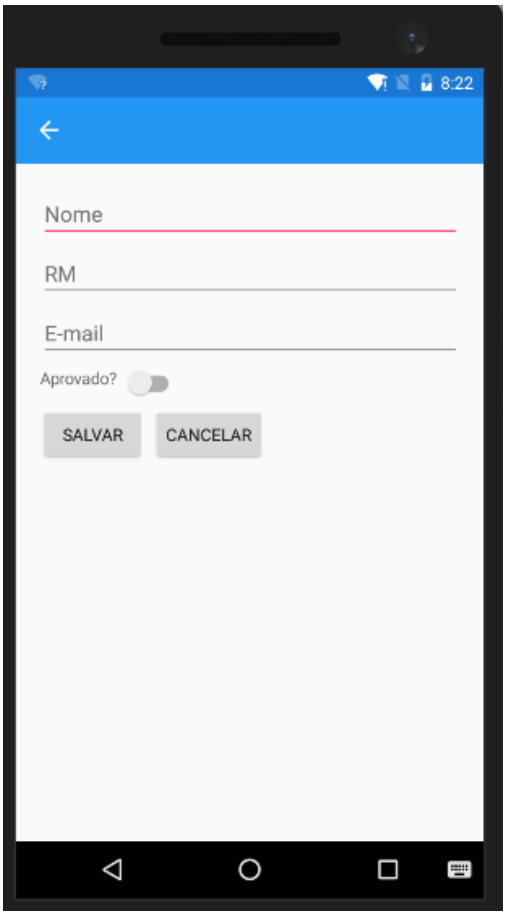
Tabela 13. Instruções C# da classe App.cs

`public App()`
`        {`
`            // The root page of your application`
`            MainPage = new NavigationPage(new View.Aluno.MainPage());`
`        }`
 	
`static Aluno alunoModel;`
`        public static Aluno AlunoModel`
`        {`
`            get`
	- Uso da classe NavigationPage para nagevar até a página MainPage através do padrão arquitetural page-based.

```
            {
                if (alunoModel == null)
                {
                    alunoModel = new
Aluno();
                }
                return alunoModel;
            }
        }
```

3. Aplicativo





4. Desafio

Com o objetivo de praticar os conhecimentos adquiridos em sala de aula, implemente os desafios propostos abaixo:

1. Implemente a funcionalidade que permite editar os dados dos alunos cadastrados;
2. Implemente a funcionalidade que permite remover um aluno selecionado na lista;
3. Implemente controle de autenticação de acesso, garantindo que somente usuários cadastrados tenham acesso aos dados dos alunos;
    a. Base de usuários: http://wopek.com/xml/usuarios.xml
4. Implemente os eventos de botões localizados no codebehind das Views diretamente no ViewModel correspondente; e
5. Agrupe a lista de alunos em ordem alfabética.