Trabalho Final
Arquitetura para Aplicações Móveis

Xamarin

Pablo de Souza Nogueira

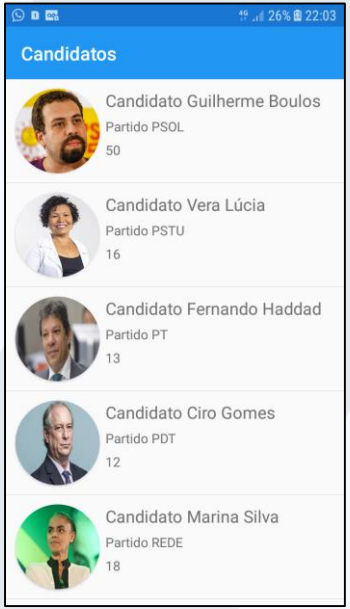
Aplicativo Presidenciáveis

- Aplicativo que exibe a lista dos candidatos a presidência do Brasil em 2018, dando uma breve descrição sobre cada um deles.

- Aplicativo composto por duas telas:
  - CandidatosPage: Lista de candidatos
  - DescricaoCandidatoPage: Descrição do candidato

- Os dados são acessados via API (Json Server) hospedado na plataforma Heroku:

https://votocerto.herokuapp.com/Candidatos/



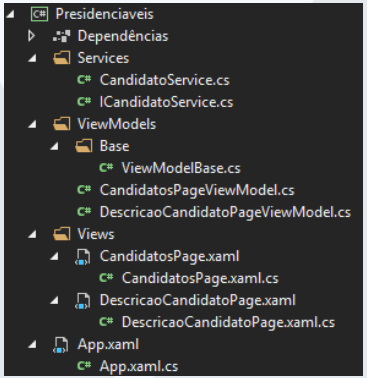
Estrutura da Solução

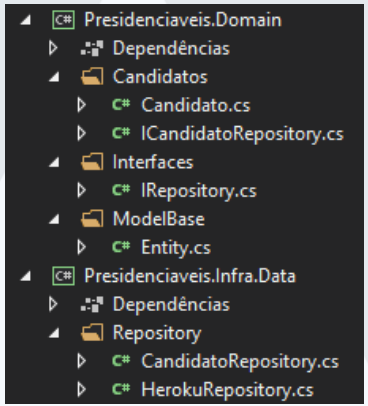
- Além do dos projetos nativos de cada plataforma (Android e iOS), a solução possui três outros projetos:

- Presidenciaveis: Projeto principal que contem a camada de visualização (Páginas e ViewModel) e a classe de serviço que tem a função de acessar os dados por meio do padrão repository.

- Presidenciaveis.Domain: Responsável por conter as models do aplicativo, no caso, apenas a model Candidato.

- Presidenciaveis.Infra.Data: Responsável por conter o repositório que da acesso a API hospedada no Heroku.





Arquitetura MVVM

Para que as páginas do aplicativo possam atualizar e serem atualizadas pela model, foram criadas uma classe .CS para cada uma delas (CandidatosPageViewModel e DescricaoCandidatoPageViewModel).

Exemplificando a CandidatosPageViewModel, ela vai herdar da ViewModelBase os métodos q ue atualiza a valor das propriedades da classe disparando um evento de atualização da propriedade.

Esta classe possui a ICandidatoService para poder acessar a camada de persistência e buscar os candidatos da API.

Esta classe também possui uma variável do tipo *ObservableCollection<Candidato>* para preencher a lista de candidatos da página CandidatosPage.

Esta classe também possui uma variável para identificar quando a lista de candidatos esta sendo ou não atualizada pela API.

```
3 referências
public class CandidatosPageViewModel : ViewModelBase
{
    private readonly ICandidatoService CandidatoService;
    4 referências
    public ObservableCollection<Candidato> Candidatos { get; set; }
    1 referência
    public Command AtualizarDados { get; }

    0 referências
    public CandidatosPageViewModel(ICandidatoService candidatoService)
    {
        CandidatoService = candidatoService;
        Candidatos = new ObservableCollection<Candidato>();
        GetCandidatos();
        AtualizarDados = new Command(ExecuteAtualizarDados);
    }

    private bool atualizando;
    2 referências
    public bool Atualizando
    {
        get { return atualizando; }
        set { SetProperty(ref atualizando, value); }
    }

    1 referência
    private async void ExecuteAtualizarDados()
    {
        Atualizando = true;
        await GetCandidatos();
        Atualizando = false;
    }

    2 referências
    private async Task GetCandidatos()
    {
        var candidatos = await CandidatoService.GetAll();

        if (Candidatos.Count > 0)
            Candidatos.Clear();

        foreach (var folheto in candidatos)
        {
            Candidatos.Add(folheto);
        }
    }

    private Candidato candidato;
    1 referência
    public Candidato Candidato
    {
        get { return candidato; }
        set { SetProperty(ref candidato, value); }
    }
}
```

Arquitetura MVVM

A página CandidatosPage é composta por um componente ListView que receve via Binding uma lista de candidatos e cada item da lista é formado por um objeto candidato exibindo o nome, partido, número e foto do mesmo.

A lista possui também dois comandos relacionados a atualização da lista, um para atualizar e outro para identificar se a lista esta em atualização.

Todos estas propriedades são referências da classe CandidatosPageViewModel que são atualizados via Bindings, fazendo o Two-way DataBinding, atualizando o valor na view e na model simultaneamente.

Para identificar o candidato selecionado a página CandidatosPage possui um método (CandidatoSelected) que irá chamar a página DescricaoCandidatoPage passando um objeto do candidato selecionado.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Presidenciaveis.Views.CandidatosPage"
             Title="Candidatos">
    <ContentPage.Content>
        <ListView ItemsSource="{Binding Candidatos}"
                  IsPullToRefreshEnabled="True"
                  RefreshCommand="{Binding AtualizarDados}"
                  IsRefreshing="{Binding Atualizando}"
                  ItemSelected="CandidatoSelected"
                  SelectedItem="{Binding Candidato}"
                  HasUnevenRows="True">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Padding="10" Orientation="Horizontal">
                            <StackLayout>
                                <Frame HeightRequest="90"  WidthRequest="90" CornerRadius="45"
                                       Margin="0" Padding="0" IsClippedToBounds="True">
                                    <Image Aspect = "Fill" Source="{Binding foto}"/>
                                </Frame>
                            </StackLayout>
                            <StackLayout>
                                <Label Text="{Binding nome, StringFormat='Candidato {0}'}"  FontSize="Medium" />
                                <Label Text="{Binding partido,  StringFormat='Partido {0}'}" FontSize="Small"/>
                                <Label Text="{Binding numero,  StringFormat='{0}'}" FontSize="Small"/>
                            </StackLayout>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </ContentPage.Content>
</ContentPage>
```

Padrão Repository

- A classe HerokuRepository implementa os métodos da interface Irepository fazendo a requisição ao servidor do Heroku por meio da biblioteca HttpClient. A resposta obtida é desserializada em JSON, por meio da biblioteca Newtonsoft – Json.

- A classe CandidatoRepository herda da classe HerokuRepository para poder ter acesso aos métodos de requisição a API, e implementa a interface ICandidatoRepository que vinculo cada objeto do domínio (no caso o Candidato) ao repositor.

```
public class CandidatoRepository : HerokuRepository<Candidato>, ICandidatoRepository
{
    1 referência
    public CandidatoRepository(string entidade) : base(entidade)
    {
    }
}
```

```
public class HerokuRepository<TEntity> : IRepository<TEntity> where TEntity : Entity
{
    readonly string RoutePrefix = "https://votocerto.herokuapp.com/";

    1 referência
    public HerokuRepository(string entidade)
    {
        RoutePrefix += (entidade + "/");
    }

    2 referências
    public Task<IEnumerable<TEntity>> GetAll()
    {
        return RequestGetAllAsync();
    }

    1 referência
    public async Task<IEnumerable<TEntity>> RequestGetAllAsync()
    {
        using (var httpClient = new HttpClient())
        {
            string resultJson = await httpClient.GetStringAsync(RoutePrefix);
            var data = JsonConvert.DeserializeObject<IEnumerable<TEntity>>(resultJson);

            return data;
        }
    }
```