Soccer App

Contents

[1 Classes 3](#_Toc479235821)

[1.1 Response 3](#_Toc479235822)

[1.2 TokenResponse 3](#_Toc479235823)

[1.3 UserRequest 3](#_Toc479235824)

[2 Data 3](#_Toc479235825)

[2.1 DataAccess 3](#_Toc479235826)

[3 Infrastructure 5](#_Toc479235827)

[3.1 InstanceLocator 5](#_Toc479235828)

[3.2 Interfaces 5](#_Toc479235829)

[4 Models 5](#_Toc479235830)

[4.1 Date 5](#_Toc479235831)

[4.2 Group 6](#_Toc479235832)

[4.3 Match 6](#_Toc479235833)

[4.4 Parameter 6](#_Toc479235834)

[4.5 Team 7](#_Toc479235835)

[4.6 Tournament 7](#_Toc479235836)

[4.7 User 8](#_Toc479235837)

[4.8 UserType 9](#_Toc479235838)

[5 Pages 9](#_Toc479235839)

[5.1 HomePage 9](#_Toc479235840)

[5.2 LoginPage 10](#_Toc479235841)

[5.3 MasterPage 12](#_Toc479235842)

[5.4 MenuPage 13](#_Toc479235843)

[5.5 SelectGroupPage 14](#_Toc479235844)

[5.6 SelectMatchPage 14](#_Toc479235845)

[5.7 SelectTournamentPage 16](#_Toc479235846)

[6 Services 17](#_Toc479235847)

[6.1 ApiService 17](#_Toc479235848)

[6.2 DataService 21](#_Toc479235849)

[6.3 DialogService 23](#_Toc479235850)

[6.4 NavigationService 23](#_Toc479235851)

[7 ViewModels 25](#_Toc479235852)

[7.1 GroupItemViewModel 25](#_Toc479235853)

[7.2 LoginViewModel 25](#_Toc479235854)

[7.3 MainViewModel 28](#_Toc479235855)

[7.4 MatchItemViewModel 30](#_Toc479235856)

[7.5 MenuItemViewModel 30](#_Toc479235857)

[7.6 SelectGroupViewModel 31](#_Toc479235858)

[7.7 SelectMatchViewModel 32](#_Toc479235859)

[7.8 SelectTournamentViewModel 34](#_Toc479235860)

[7.9 TournamentItemViewModel 36](#_Toc479235861)

[8 Root 36](#_Toc479235862)

[8.1 App 36](#_Toc479235863)

[9 Android 38](#_Toc479235864)

[9.1 Config 38](#_Toc479235865)

[10 iOS 39](#_Toc479235866)

[10.1 Config 39](#_Toc479235867)

# Classes

## Response

namespace Soccer.Classes

{

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

}

## TokenResponse

using Newtonsoft.Json;

using System;

namespace Soccer.Classes

{

public class TokenResponse

{

[JsonProperty(PropertyName = "access\_token")]

public string AccessToken { get; set; }

[JsonProperty(PropertyName = "token\_type")]

public string TokenType { get; set; }

[JsonProperty(PropertyName = "expires\_in")]

public int ExpiresIn { get; set; }

[JsonProperty(PropertyName = "userName")]

public string UserName { get; set; }

[JsonProperty(PropertyName = ".issued")]

public DateTime Issued { get; set; }

[JsonProperty(PropertyName = ".expires")]

public DateTime Expires { get; set; }

[JsonProperty(PropertyName = "error")]

public string Error { get; set; }

[JsonProperty(PropertyName = "error\_description")]

public string ErrorDescription { get; set; }

}

}

## UserRequest

namespace Soccer.Classes

{

public class UserRequest

{

public string Email { get; set; }

}

}

# Data

## DataAccess

using Soccer.Interfaces;

using Soccer.Models;

using SQLite.Net;

using SQLiteNetExtensions.Extensions;

using System;

using System.Collections.Generic;

using System.Linq;

using Xamarin.Forms;

namespace Soccer.Data

{

public class DataAccess : IDisposable

{

private SQLiteConnection connection;

public DataAccess()

{

var config = DependencyService.Get<IConfig>();

connection = new SQLiteConnection(config.Platform,

System.IO.Path.Combine(config.DirectoryDB, "Soccer.db3"));

connection.CreateTable<Parameter>();

connection.CreateTable<Team>();

connection.CreateTable<User>();

connection.CreateTable<UserType>();

}

public void Insert<T>(T model)

{

connection.Insert(model);

}

public void Update<T>(T model)

{

connection.Update(model);

}

public void Delete<T>(T model)

{

connection.Delete(model);

}

public T First<T>(bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().FirstOrDefault();

}

else

{

return connection.Table<T>().FirstOrDefault();

}

}

public List<T> GetList<T>(bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().ToList();

}

else

{

return connection.Table<T>().ToList();

}

}

public T Find<T>(int pk, bool WithChildren) where T : class

{

if (WithChildren)

{

return connection.GetAllWithChildren<T>().FirstOrDefault(m => m.GetHashCode() == pk);

}

else

{

return connection.Table<T>().FirstOrDefault(m => m.GetHashCode() == pk);

}

}

public void Dispose()

{

connection.Dispose();

}

}

}

# Infrastructure

## InstanceLocator

using Soccer.ViewModels;

namespace Soccer.Infrastructure

{

public class InstanceLocator

{

public MainViewModel Main { get; set; }

public InstanceLocator()

{

Main = new MainViewModel();

}

}

}

# Interfaces

## IConfig

using SQLite.Net.Interop;

namespace Soccer.Interfaces

{

public interface IConfig

{

string DirectoryDB { get; }

ISQLitePlatform Platform { get; }

}

}

# Models

## Date

namespace Soccer.Models

{

public class Date

{

public int DateId { get; set; }

public string Name { get; set; }

public int TournamentId { get; set; }

}

}

## Group

namespace Soccer.Models

{

public class Group

{

public int TournamentGroupId { get; set; }

public string Name { get; set; }

public int TournamentId { get; set; }

}

}

## Match

using System;

namespace Soccer.Models

{

public class Match

{

public int MatchId { get; set; }

public int DateId { get; set; }

public DateTime DateTime { get; set; }

public int LocalId { get; set; }

public int VisitorId { get; set; }

public int? LocalGoals { get; set; }

public int? VisitorGoals { get; set; }

public int StatusId { get; set; }

public int TournamentGroupId { get; set; }

public Team Local { get; set; }

public Team Visitor { get; set; }

}

}

## Parameter

using SQLite.Net.Attributes;

namespace Soccer.Models

{

public class Parameter

{

[PrimaryKey, AutoIncrement]

public int ParameterId { get; set; }

public string URLBase { get; set; }

public string Option { get; set; }

public override int GetHashCode()

{

return ParameterId;

}

}

}

## Team

using SQLite.Net.Attributes;

using SQLiteNetExtensions.Attributes;

using System.Collections.Generic;

namespace Soccer.Models

{

public class Team

{

[PrimaryKey]

public int TeamId { get; set; }

public string Name { get; set; }

public string Logo { get; set; }

public string Initials { get; set; }

public int LeagueId { get; set; }

[OneToMany(CascadeOperations = CascadeOperation.All)]

public List<User> Fans { get; set; }

public string FullLogo

{

get

{

if (string.IsNullOrEmpty(Logo))

{

return "avatar\_shield.png";

}

return string.Format("http://soccerbackend.azurewebsites.net{0}", Logo.Substring(1));

}

}

public override int GetHashCode()

{

return TeamId;

}

}

}

## Tournament

using System.Collections.Generic;

namespace Soccer.Models

{

public class Tournament

{

public int TournamentId { get; set; }

public string Name { get; set; }

public string Logo { get; set; }

public List<Group> Groups { get; set; }

public List<Date> Dates { get; set; }

public string FullLogo

{

get

{

if (string.IsNullOrEmpty(Logo))

{

return "avatar\_tournament.png";

}

return string.Format("http://soccerbackend.azurewebsites.net{0}", Logo.Substring(1));

}

}

}

}

## User

using SQLite.Net.Attributes;

using SQLiteNetExtensions.Attributes;

using System;

using System.Collections.Generic;

namespace Soccer.Models

{

public class User

{

[PrimaryKey]

public int UserId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public int UserTypeId { get; set; }

public string Picture { get; set; }

public string Email { get; set; }

public string NickName { get; set; }

public int FavoriteTeamId { get; set; }

public int Points { get; set; }

[ManyToOne]

public UserType UserType { get; set; }

[ManyToOne]

public Team FavoriteTeam { get; set; }

public string AccessToken { get; set; }

public string TokenType { get; set; }

public DateTime TokenExpires { get; set; }

public string Password { get; set; }

public bool IsRemembered { get; set; }

public string FullName { get { return string.Format("{0} {1}", FirstName, LastName); } }

public string FullPicture

{

get

{

if (string.IsNullOrEmpty(Picture))

{

return "avatar\_user.png";

}

return string.Format("http://soccerbackend.azurewebsites.net{0}", Picture.Substring(1));

}

}

public override int GetHashCode()

{

return UserId;

}

}

}

## UserType

using SQLite.Net.Attributes;

using SQLiteNetExtensions.Attributes;

using System.Collections.Generic;

namespace Soccer.Models

{

public class UserType

{

[PrimaryKey]

public int UserTypeId { get; set; }

public string Name { get; set; }

[OneToMany(CascadeOperations = CascadeOperation.All)]

public List<User> Users { get; set; }

public override int GetHashCode()

{

return UserTypeId;

}

}

}

# Pages

## HomePage

<?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:controls="clr-namespace:ImageCircle.Forms.Plugin.Abstractions;assembly=ImageCircle.Forms.Plugin.Abstractions"

x:Class="Soccer.Pages.HomePage"

BackgroundColor="{StaticResource AccentColor1}"

Title="Soccer"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<StackLayout

VerticalOptions="Center"

HorizontalOptions="Center">

<Label

Text="Welcome"

TextColor="Red"

FontAttributes="Bold"

FontSize="Medium"

VerticalOptions="Center"

HorizontalOptions="Center">

</Label>

<Label

Text="{Binding CurrentUser.FullName, Mode=TwoWay}"

FontSize="Large"

TextColor="Black"

FontAttributes="Bold"

VerticalOptions="Center"

HorizontalOptions="Center">

</Label>

<RelativeLayout>

<controls:CircleImage

Source="{Binding CurrentUser.FullPicture}"

Aspect="AspectFill"

WidthRequest="250"

HeightRequest="250">

</controls:CircleImage>

<Image

Source="{Binding CurrentUser.FavoriteTeam.FullLogo}"

WidthRequest="80"

RelativeLayout.XConstraint="{ConstraintExpression Type=Constant, Constant=160}"

RelativeLayout.YConstraint="{ConstraintExpression Type=Constant, Constant=160}">

</Image>

</RelativeLayout>

<Label

Text="{Binding CurrentUser.Email, Mode=TwoWay, StringFormat='Email: {0}'}"

FontSize="Medium"

TextColor="Black"

FontAttributes="Bold"

VerticalOptions="Center"

HorizontalOptions="Center">

</Label>

<Label

Text="{Binding CurrentUser.Points, Mode=TwoWay, StringFormat='Points: {0}'}"

FontSize="Large"

TextColor="Black"

FontAttributes="Bold"

VerticalOptions="Center"

HorizontalOptions="Center">

</Label>

</StackLayout>

</ContentPage>

## LoginPage

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Soccer.Pages.LoginPage"

Title="Soccer"

BackgroundColor="{StaticResource MainColor}"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<ScrollView

BindingContext="{Binding Login}">

<StackLayout

Spacing="10"

Padding="8">

<Image

Source="soccer\_logo.png">

</Image>

<Label

Text="Login"

FontSize="Large"

FontAttributes="Bold"

HorizontalOptions="Center"

TextColor="{StaticResource BackgroundColor}">

</Label>

<Grid

Padding="30,0,30,0">

<Grid Grid.Row="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="80"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Email"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="{StaticResource BackgroundColor}">

</Label>

<Entry

Grid.Column="1"

Text="{Binding Email}"

Placeholder="Enter an email..."

Keyboard="Email"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="{StaticResource BackgroundColor}">

</Entry>

</Grid>

<Grid Grid.Row="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="80"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Password"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="{StaticResource BackgroundColor}">

</Label>

<Entry

Grid.Column="1"

Text="{Binding Password}"

Placeholder="Enter a password..."

IsPassword="True"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="{StaticResource BackgroundColor}">

</Entry>

</Grid>

</Grid>

<StackLayout

Padding="30,0,30,0"

Orientation="Horizontal">

<Switch

HorizontalOptions="Start"

VerticalOptions="Center"

IsToggled="{Binding IsRemembered}">

</Switch>

<Label

Text="Rememberme"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="{StaticResource BackgroundColor}">

</Label>

</StackLayout>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<StackLayout

Padding="30,0,30,0"

Orientation="Vertical">

<Button

Command="{Binding LoginCommand}"

VerticalOptions="Center"

HorizontalOptions="FillAndExpand"

BackgroundColor="Orange"

TextColor="White"

Text="Login">

</Button>

<Button

Command="{Binding LoginFacebookCommand}"

VerticalOptions="Center"

HorizontalOptions="FillAndExpand"

BackgroundColor="Navy"

TextColor="White"

Text="Login With Facebook">

</Button>

<Button

Command="{Binding RegisterCommand}"

VerticalOptions="Center"

HorizontalOptions="FillAndExpand"

BackgroundColor="Purple"

TextColor="White"

Text="Register New User">

</Button>

<Label

Text="Forgot your password?"

FontAttributes="Bold"

HorizontalOptions="Center"

VerticalOptions="Center"

TextColor="{StaticResource BackgroundColor}">

<Label.GestureRecognizers>

<TapGestureRecognizer Command="{Binding ForgotPasswordCommand}"/>

</Label.GestureRecognizers>

</Label>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

## MasterPage

<?xml version="1.0" encoding="utf-8" ?>

<MasterDetailPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:pages="clr-namespace:Soccer.Pages;assembly=Soccer"

x:Class="Soccer.Pages.MasterPage">

<MasterDetailPage.Master>

<pages:MenuPage></pages:MenuPage>

</MasterDetailPage.Master>

<MasterDetailPage.Detail>

<NavigationPage

x:Name="Navigator">

<x:Arguments>

<pages:HomePage/>

</x:Arguments>

</NavigationPage>

</MasterDetailPage.Detail>

</MasterDetailPage>

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace Soccer.Pages

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class MasterPage : MasterDetailPage

{

public MasterPage()

{

InitializeComponent();

}

protected override void OnAppearing()

{

base.OnAppearing();

App.Master = this;

App.Navigator = Navigator;

}

}

}

## MenuPage

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Soccer.Pages.MenuPage"

BackgroundColor="{StaticResource MainColor}"

Title="Menu"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<StackLayout

Padding="8">

<Image

WidthRequest="250"

Source="soccer\_logo.png">

</Image>

<ListView

ItemsSource="{Binding Menu}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding NavigateCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="{Binding Icon}"

WidthRequest="50"

HeightRequest="50"/>

<Label

Grid.Column="1"

VerticalOptions="Center"

TextColor="{StaticResource AccentColor1}"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

## SelectGroupPage

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Soccer.Pages.SelectGroupPage"

BackgroundColor="{StaticResource AccentColor1}"

Title="Select Group"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<StackLayout

BindingContext="{Binding SelectGroup}"

Padding="8">

<ListView

ItemsSource="{Binding Groups}"

HasUnevenRows="True"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshCommand}"

IsRefreshing="{Binding IsRefreshing, Mode=TwoWay}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackLayout

VerticalOptions="Center">

<Label

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding Name}">

</Label>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

## SelectMatchPage

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Soccer.Pages.SelectMatchPage"

BackgroundColor="{StaticResource AccentColor1}"

Title="Select Match"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<StackLayout

BindingContext="{Binding SelectMatch}"

Padding="8">

<ListView

ItemsSource="{Binding Matches}"

HasUnevenRows="True"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshCommand}"

IsRefreshing="{Binding IsRefreshing, Mode=TwoWay}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

<ColumnDefinition Width="Auto"></ColumnDefinition>

</Grid.ColumnDefinitions>

<StackLayout

VerticalOptions="Center"

HorizontalOptions="Start"

Grid.Column="0">

<Image

Source="{Binding Local.FullLogo}"

VerticalOptions="Center"

HorizontalOptions="Center"

WidthRequest="50"

HeightRequest="50">

</Image>

<Label

FontAttributes="Bold"

VerticalOptions="Center"

HorizontalOptions="Center"

TextColor="Black"

Text="{Binding Local.Name}">

</Label>

</StackLayout>

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center"

Orientation="Horizontal"

Grid.Column="1">

<Label

FontAttributes="Bold"

FontSize="Large"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding LocalGoals}">

</Label>

<Label

FontAttributes="Bold"

FontSize="Large"

VerticalOptions="Center"

TextColor="Black"

Text="Vs.">

</Label>

<Label

FontAttributes="Bold"

FontSize="Large"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding VisitorGoals}">

</Label>

</StackLayout>

<StackLayout

VerticalOptions="Center"

HorizontalOptions="Start"

Grid.Column="2">

<Image

Source="{Binding Visitor.FullLogo}"

VerticalOptions="Center"

HorizontalOptions="Center"

WidthRequest="50"

HeightRequest="50">

</Image>

<Label

FontAttributes="Bold"

VerticalOptions="Center"

HorizontalOptions="Center"

TextColor="Black"

Text="{Binding Visitor.Name}">

</Label>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

## SelectTournamentPage

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Soccer.Pages.SelectTournamentPage"

BackgroundColor="{StaticResource AccentColor1}"

Title="Select Tournament"

BindingContext="{Binding Main, Source={StaticResource Locator}}">

<StackLayout

BindingContext="{Binding SelectTournament}"

Padding="8">

<ListView

ItemsSource="{Binding Tournaments}"

HasUnevenRows="True"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshCommand}"

IsRefreshing="{Binding IsRefreshing, Mode=TwoWay}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectTournamentCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="{Binding FullLogo}"

VerticalOptions="Start"

WidthRequest="50"

HeightRequest="50">

</Image>

<StackLayout

VerticalOptions="Center"

Grid.Column="1">

<Label

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="Black"

Text="{Binding Name}">

</Label>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

using Soccer.ViewModels;

using System;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace Soccer.Pages

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class SelectTournamentPage : ContentPage

{

public SelectTournamentPage()

{

InitializeComponent();

var instance = SelectTournamentViewModel.GetInstance();

Appearing += (object sender, EventArgs e) =>

{

instance.RefreshCommand.Execute(this);

};

}

}

}

# Services

## ApiService

using Newtonsoft.Json;

using Soccer.Classes;

using Soccer.Models;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace Soccer.Services

{

public class ApiService

{

public async Task<TokenResponse> GetToken(string urlBase, string username, string password)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var response = await client.PostAsync("Token",

new StringContent(string.Format("grant\_type=password&username={0}&password={1}", username, password),

Encoding.UTF8, "application/x-www-form-urlencoded"));

var resultJSON = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<TokenResponse>(resultJSON);

return result;

}

catch

{

return null;

}

}

public async Task<Response> GetUserByEmail(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, string email)

{

try

{

var userRequest = new UserRequest { Email = email, };

var request = JsonConvert.SerializeObject(userRequest);

var content = new StringContent(request, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

var response = await client.PostAsync(url, content);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

var result = await response.Content.ReadAsStringAsync();

var newRecord = JsonConvert.DeserializeObject<User>(result);

return new Response

{

IsSuccess = true,

Message = "Record added OK",

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Get<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken)

{

try

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

var response = await client.GetAsync(url);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

var result = await response.Content.ReadAsStringAsync();

var list = JsonConvert.DeserializeObject<List<T>>(result);

return new Response

{

IsSuccess = true,

Message = "Ok",

Result = list,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Post<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var request = JsonConvert.SerializeObject(model);

var content = new StringContent(request, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

var response = await client.PostAsync(url, content);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

var result = await response.Content.ReadAsStringAsync();

var newRecord = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Message = "Record added OK",

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Put<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var request = JsonConvert.SerializeObject(model);

var content = new StringContent(request, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}/{2}", servicePrefix, controller, model.GetHashCode());

var response = await client.PutAsync(url, content);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

var result = await response.Content.ReadAsStringAsync();

var newRecord = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Message = "Record updated OK",

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Delete<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

var url = string.Format("{0}{1}/{2}", servicePrefix, controller, model.GetHashCode());

var response = await client.DeleteAsync(url);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

return new Response

{

IsSuccess = true,

Message = "Record deleted OK",

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

}

}

## DataService

using Soccer.Data;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Soccer.Services

{

public class DataService

{

public T DeleteAllAndInsert<T>(T model) where T : class

{

try

{

using (var da = new DataAccess())

{

var oldRecords = da.GetList<T>(false);

foreach (var oldRecord in oldRecords)

{

da.Delete(oldRecord);

}

da.Insert(model);

return model;

}

}

catch (Exception ex)

{

ex.ToString();

return model;

}

}

public T InsertOrUpdate<T>(T model) where T : class

{

try

{

using (var da = new DataAccess())

{

var oldRecord = da.Find<T>(model.GetHashCode(), false);

if (oldRecord != null)

{

da.Update(model);

}

else

{

da.Insert(model);

}

return model;

}

}

catch (Exception ex)

{

ex.ToString();

return model;

}

}

public T Insert<T>(T model)

{

using (var da = new DataAccess())

{

da.Insert(model);

return model;

}

}

public T Find<T>(int pk, bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.Find<T>(pk, withChildren);

}

}

public T First<T>(bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.GetList<T>(withChildren).FirstOrDefault();

}

}

public List<T> Get<T>(bool withChildren) where T : class

{

using (var da = new DataAccess())

{

return da.GetList<T>(withChildren).ToList();

}

}

public void Update<T>(T model)

{

using (var da = new DataAccess())

{

da.Update(model);

}

}

public void Delete<T>(T model)

{

using (var da = new DataAccess())

{

da.Delete(model);

}

}

public void Save<T>(List<T> list) where T : class

{

using (var da = new DataAccess())

{

foreach (var record in list)

{

InsertOrUpdate(record);

}

}

}

}

}

## DialogService

using System.Threading.Tasks;

namespace Soccer.Services

{

public class DialogService

{

public async Task ShowMessage(string title, string message)

{

await App.Current.MainPage.DisplayAlert(title, message, "Accept");

}

public async Task<bool> ShowConfirm(string title, string message)

{

return await App.Current.MainPage.DisplayAlert(title, message, "Yes", "No");

}

}

}

## NavigationService

using Soccer.Models;

using Soccer.Pages;

using Soccer.ViewModels;

using System.Threading.Tasks;

namespace Soccer.Services

{

public class NavigationService

{

#region Attributes

private DataService dataService;

#endregion

#region Constructors

public NavigationService()

{

dataService = new DataService();

}

#endregion

#region Methods

public async Task Navigate(string pageName)

{

App.Master.IsPresented = false;

var mainViewModel = MainViewModel.GetInstance();

switch (pageName)

{

case "SelectGroupPage":

await App.Navigator.PushAsync(new SelectGroupPage());

break;

case "SelectTournamentPage":

mainViewModel.SelectTournament = new SelectTournamentViewModel();

await App.Navigator.PushAsync(new SelectTournamentPage());

break;

default:

break;

}

}

public void SetMainPage(string pageName)

{

switch (pageName)

{

case "MasterPage":

App.Current.MainPage = new MasterPage();

break;

case "LoginPage":

Logout();

App.Current.MainPage = new LoginPage();

break;

default:

break;

}

}

private void Logout()

{

var user = dataService.First<User>(false);

if (user != null)

{

user.IsRemembered = false;

dataService.Update(user);

}

}

public async Task Back()

{

await App.Navigator.PopAsync();

}

public async Task Clear()

{

await App.Navigator.PopToRootAsync();

}

#endregion

}

}

# ViewModels

## GroupItemViewModel

using Soccer.Models;

namespace Soccer.ViewModels

{

public class GroupItemViewModel : Group

{

}

}

## LoginViewModel

using GalaSoft.MvvmLight.Command;

using Plugin.Connectivity;

using Soccer.Models;

using Soccer.Services;

using System.ComponentModel;

using System.Windows.Input;

namespace Soccer.ViewModels

{

public class LoginViewModel : INotifyPropertyChanged

{

#region Attributes

private ApiService apiService;

private DialogService dialogService;

private DataService dataService;

private NavigationService navigationService;

private string email;

private string password;

private bool isRunning;

private bool isEnabled;

private bool isRemembered;

#endregion

#region Properties

public string Email

{

set

{

if (email != value)

{

email = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Email"));

}

}

get

{

return email;

}

}

public string Password

{

set

{

if (password != value)

{

password = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Password"));

}

}

get

{

return password;

}

}

public bool IsRunning

{

set

{

if (isRunning != value)

{

isRunning = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("IsRunning"));

}

}

get

{

return isRunning;

}

}

public bool IsEnabled

{

set

{

if (isEnabled != value)

{

isEnabled = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("IsEnabled"));

}

}

get

{

return isEnabled;

}

}

public bool IsRemembered

{

set

{

if (isRemembered != value)

{

isRemembered = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("IsRemembered"));

}

}

get

{

return isRemembered;

}

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Constructors

public LoginViewModel()

{

apiService = new ApiService();

dialogService = new DialogService();

dataService = new DataService();

navigationService = new NavigationService();

IsRemembered = true;

IsEnabled = true;

Email = null;

Password = null;

}

#endregion

#region Commands

public ICommand LoginCommand { get { return new RelayCommand(Login); } }

private async void Login()

{

if (string.IsNullOrEmpty(Email))

{

await dialogService.ShowMessage("Error", "You must enter the user email.");

return;

}

if (string.IsNullOrEmpty(Password))

{

await dialogService.ShowMessage("Error", "You must enter a password.");

return;

}

IsRunning = true;

IsEnabled = false;

if (!CrossConnectivity.Current.IsConnected)

{

IsRunning = false;

IsEnabled = true;

await dialogService.ShowMessage("Error", "Check you internet connection.");

return;

}

var isReachable = await CrossConnectivity.Current.IsRemoteReachable("google.com");

if (!isReachable)

{

IsRunning = false;

IsEnabled = true;

await dialogService.ShowMessage("Error", "Check you internet connection.");

return;

}

var parameters = dataService.First<Parameter>(false);

var token = await apiService.GetToken(parameters.URLBase, Email, Password);

if (token == null)

{

IsRunning = false;

IsEnabled = true;

await dialogService.ShowMessage("Error", "The user name or password in incorrect.");

Password = null;

return;

}

if (string.IsNullOrEmpty(token.AccessToken))

{

IsRunning = false;

IsEnabled = true;

await dialogService.ShowMessage("Error", token.ErrorDescription);

Password = null;

return;

}

var response = await apiService.GetUserByEmail(parameters.URLBase, "/api", "/Users/GetUserByEmail", token.TokenType, token.AccessToken, token.UserName);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await dialogService.ShowMessage("Error", "Problem ocurred retrieving user information, try latter.");

return;

}

var user = (User)response.Result;

user.AccessToken = token.AccessToken;

user.TokenType = token.TokenType;

user.TokenExpires = token.Expires;

user.IsRemembered = IsRemembered;

user.Password = Password;

dataService.DeleteAllAndInsert(user);

dataService.InsertOrUpdate(user.FavoriteTeam);

dataService.InsertOrUpdate(user.UserType);

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.SetCurrentUser(user);

Email = null;

Password = null;

IsRunning = false;

IsEnabled = true;

navigationService.SetMainPage("MasterPage");

}

#endregion

}

}

## MainViewModel

using Soccer.Models;

using System.Collections.ObjectModel;

using System.ComponentModel;

namespace Soccer.ViewModels

{

public class MainViewModel : INotifyPropertyChanged

{

#region Attributes

private User currentUser;

#endregion

#region Properties

public LoginViewModel Login { get; set; }

public SelectTournamentViewModel SelectTournament { get; set; }

public SelectGroupViewModel SelectGroup { get; set; }

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public User CurrentUser

{

set

{

if (currentUser != value)

{

currentUser = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("CurrentUser"));

}

}

get

{

return currentUser;

}

}

#endregion

#region Constructor

public MainViewModel()

{

instance = this;

Login = new LoginViewModel();

LoadMenu();

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Singleton

private static MainViewModel instance;

public static MainViewModel GetInstance()

{

if (instance == null)

{

instance = new MainViewModel();

}

return instance;

}

#endregion

#region Methods

private void LoadMenu()

{

Menu = new ObservableCollection<MenuItemViewModel>();

Menu.Add(new MenuItemViewModel

{

Icon = "predictions.png",

PageName = "SelectTournamentPage",

Title = "Predictions",

});

Menu.Add(new MenuItemViewModel

{

Icon = "groups.png",

PageName = "GroupsPage",

Title = "Groups",

});

Menu.Add(new MenuItemViewModel

{

Icon = "tournaments.png",

PageName = "TournamentsPage",

Title = "Tournaments",

});

Menu.Add(new MenuItemViewModel

{

Icon = "myresults.png",

PageName = "ResultsPage",

Title = "My Results",

});

Menu.Add(new MenuItemViewModel

{

Icon = "config.png",

PageName = "ConfigPage",

Title = "Config",

});

Menu.Add(new MenuItemViewModel

{

Icon = "logut.png",

PageName = "LoginPage",

Title = "Logut",

});

}

public void SetCurrentUser(User user)

{

CurrentUser = user;

}

#endregion

}

}

## MatchItemViewModel

using Soccer.Models;

namespace Soccer.ViewModels

{

public class MatchItemViewModel : Match

{

}

}

## MenuItemViewModel

using GalaSoft.MvvmLight.Command;

using Soccer.Models;

using Soccer.Services;

using System.Windows.Input;

namespace Soccer.ViewModels

{

public class MenuItemViewModel

{

#region Attributes

private NavigationService navigationService;

private DataService dataService;

#endregion

#region Properties

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

#endregion

#region Constructors

public MenuItemViewModel()

{

navigationService = new NavigationService();

dataService = new DataService();

}

#endregion

#region Commands

public ICommand NavigateCommand { get { return new RelayCommand(Navigate); } }

private async void Navigate()

{

if (PageName == "LoginPage")

{

navigationService.SetMainPage(PageName);

}

else

{

var parameter = dataService.First<Parameter>(false);

parameter.Option = Title;

dataService.Update(parameter);

await navigationService.Navigate(PageName);

}

}

#endregion

}

}

## SelectGroupViewModel

using Soccer.Models;

using Soccer.Services;

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace Soccer.ViewModels

{

public class SelectGroupViewModel

{

#region Attributes

private ApiService apiService;

private DataService dataService;

private DialogService dialogService;

private NavigationService navigationService;

private List<Group> groups;

#endregion

#region Properties

public ObservableCollection<GroupItemViewModel> Groups { get; set; }

#endregion

#region Constructor

public SelectGroupViewModel(List<Group> groups)

{

apiService = new ApiService();

dialogService = new DialogService();

navigationService = new NavigationService();

dataService = new DataService();

this.groups = groups;

Groups = new ObservableCollection<GroupItemViewModel>();

LoadGroups();

}

#endregion

#region Methods

private void LoadGroups()

{

Groups.Clear();

foreach (var group in groups)

{

Groups.Add(new GroupItemViewModel

{

Name = group.Name,

TournamentGroupId = group.TournamentGroupId,

TournamentId = group.TournamentId,

});

}

}

#endregion

}

}

## SelectMatchViewModel

using GalaSoft.MvvmLight.Command;

using Plugin.Connectivity;

using Soccer.Services;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Windows.Input;

namespace Soccer.ViewModels

{

public class SelectMatchViewModel : INotifyPropertyChanged

{

#region Attributes

private ApiService apiService;

private DataService dataService;

private DialogService dialogService;

private NavigationService navigationService;

private bool isRefreshing = false;

private int tournamentId;

#endregion

#region Properties

public ObservableCollection<MatchItemViewModel> Matches { get; set; }

public bool IsRefreshing

{

set

{

if (isRefreshing != value)

{

isRefreshing = value;

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs("IsRefreshing"));

}

}

}

get

{

return isRefreshing;

}

}

#endregion

#region Constructor

public SelectMatchViewModel(int tournamentId)

{

this.tournamentId = tournamentId;

instance = this;

apiService = new ApiService();

dialogService = new DialogService();

navigationService = new NavigationService();

dataService = new DataService();

Matches = new ObservableCollection<MatchItemViewModel>();

}

#endregion

#region Singleton

private static SelectMatchViewModel instance;

public static SelectMatchViewModel GetInstance()

{

return instance;

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Methods

private async void LoadMatches()

{

if (!CrossConnectivity.Current.IsConnected)

{

await dialogService.ShowMessage("Error", "Check you internet connection.");

await navigationService.Clear();

return;

}

var parameter = dataService.First<Parameter>(false);

var user = dataService.First<User>(false);

var response = await apiService.Get<Tournament>(parameter.URLBase, "/api", "/Tournaments", user.TokenType, user.AccessToken);

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

await navigationService.Clear();

return;

}

ReloadTournaments((List<Tournament>)response.Result);

}

private void ReloadTournaments(List<Tournament> tournaments)

{

Tournaments.Clear();

foreach (var tournament in tournaments)

{

Tournaments.Add(new TournamentItemViewModel

{

Dates = tournament.Dates,

Groups = tournament.Groups,

Logo = tournament.Logo,

Name = tournament.Name,

TournamentId = tournament.TournamentId,

});

}

}

#endregion

#region Commands

public ICommand RefreshCommand { get { return new RelayCommand(Refresh); } }

public void Refresh()

{

IsRefreshing = true;

LoadTournaments();

IsRefreshing = false;

}

#endregion

}

}

## SelectTournamentViewModel

using GalaSoft.MvvmLight.Command;

using Plugin.Connectivity;

using Soccer.Models;

using Soccer.Services;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Windows.Input;

namespace Soccer.ViewModels

{

public class SelectTournamentViewModel : INotifyPropertyChanged

{

#region Attributes

private ApiService apiService;

private DataService dataService;

private DialogService dialogService;

private NavigationService navigationService;

private bool isRefreshing = false;

#endregion

#region Properties

public ObservableCollection<TournamentItemViewModel> Tournaments { get; set; }

public bool IsRefreshing

{

set

{

if (isRefreshing != value)

{

isRefreshing = value;

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs("IsRefreshing"));

}

}

}

get

{

return isRefreshing;

}

}

#endregion

#region Constructor

public SelectTournamentViewModel()

{

instance = this;

apiService = new ApiService();

dialogService = new DialogService();

navigationService = new NavigationService();

dataService = new DataService();

Tournaments = new ObservableCollection<TournamentItemViewModel>();

}

#endregion

#region Singleton

private static SelectTournamentViewModel instance;

public static SelectTournamentViewModel GetInstance()

{

if (instance == null)

{

instance = new SelectTournamentViewModel();

}

return instance;

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Methods

private async void LoadTournaments()

{

if (!CrossConnectivity.Current.IsConnected)

{

await dialogService.ShowMessage("Error", "Check you internet connection.");

await navigationService.Clear();

return;

}

var parameter = dataService.First<Parameter>(false);

var user = dataService.First<User>(false);

var response = await apiService.Get<Tournament>(parameter.URLBase, "/api", "/Tournaments", user.TokenType, user.AccessToken);

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

await navigationService.Clear();

return;

}

ReloadTournaments((List<Tournament>)response.Result);

}

private void ReloadTournaments(List<Tournament> tournaments)

{

Tournaments.Clear();

foreach (var tournament in tournaments)

{

Tournaments.Add(new TournamentItemViewModel

{

Dates = tournament.Dates,

Groups = tournament.Groups,

Logo = tournament.Logo,

Name = tournament.Name,

TournamentId = tournament.TournamentId,

});

}

}

#endregion

#region Commands

public ICommand RefreshCommand { get { return new RelayCommand(Refresh); } }

public void Refresh()

{

IsRefreshing = true;

LoadTournaments();

IsRefreshing = false;

}

#endregion

}

}

## TournamentItemViewModel

using GalaSoft.MvvmLight.Command;

using Soccer.Models;

using Soccer.Services;

using System.Windows.Input;

namespace Soccer.ViewModels

{

public class TournamentItemViewModel : Tournament

{

private NavigationService navigationService;

public TournamentItemViewModel()

{

navigationService = new NavigationService();

}

public ICommand SelectTournamentCommand { get { return new RelayCommand(SelectTournament); } }

private async void SelectTournament()

{

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.SelectGroup = new SelectGroupViewModel(this.Groups);

await navigationService.Navigate("SelectGroupPage");

}

}

}

# Root

## App

<?xml version="1.0" encoding="utf-8" ?>

<Application

xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:infra="clr-namespace:Soccer.Infrastructure;assembly=Soccer"

x:Class="Soccer.App">

<Application.Resources>

<ResourceDictionary>

<!-- Locator -->

<infra:InstanceLocator x:Key="Locator"></infra:InstanceLocator>

<!-- Parameters -->

<x:String x:Key="URLBase">http://soccerapi.azurewebsites.net</x:String>

<!-- Colors -->

<Color x:Key="MainColor">#494E50</Color>

<Color x:Key="FontColor">#161605</Color>

<Color x:Key="AccentColor1">#DDE0C6</Color>

<Color x:Key="AccentColor2">#B5BE76</Color>

<Color x:Key="BackgroundColor">#839851</Color>

</ResourceDictionary>

</Application.Resources>

</Application>

using Soccer.Models;

using Soccer.Pages;

using Soccer.Services;

using Soccer.ViewModels;

using System;

using Xamarin.Forms;

namespace Soccer

{

public partial class App : Application

{

#region Attributes

private DataService dataService;

#endregion

#region Properties

public static NavigationPage Navigator { get; internal set; }

public static MasterPage Master { get; internal set; }

#endregion

#region Constructors

public App()

{

InitializeComponent();

dataService = new DataService();

LoadParameters();

var user = dataService.First<User>(false);

var favoriteTeam = dataService.Find<Team>(user.FavoriteTeamId, false);

user.FavoriteTeam = favoriteTeam;

if (user != null && user.IsRemembered && user.TokenExpires > DateTime.Now)

{

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.SetCurrentUser(user);

MainPage = new MasterPage();

}

else

{

MainPage = new LoginPage();

}

}

#endregion

#region Methods

private void LoadParameters()

{

var urlBase = Application.Current.Resources["URLBase"].ToString();

var parameter = dataService.First<Parameter>(false);

if (parameter == null)

{

parameter = new Parameter

{

URLBase = urlBase,

};

dataService.Insert(parameter);

}

else

{

parameter.URLBase = urlBase;

dataService.Update(parameter);

}

}

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

}

#endregion

}

}

# Android

## Config

using Soccer.Interfaces;

using SQLite.Net.Interop;

using Xamarin.Forms;

[assembly: Dependency(typeof(Soccer.Droid.Config))]

namespace Soccer.Droid

{

public class Config : IConfig

{

private string directoryDB;

private ISQLitePlatform platform;

public string DirectoryDB

{

get

{

if (string.IsNullOrEmpty(directoryDB))

{

directoryDB = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);

}

return directoryDB;

}

}

public ISQLitePlatform Platform

{

get

{

if (platform == null)

{

platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();

}

return platform;

}

}

}

}

# iOS

## Config

using System;

using Soccer.Interfaces;

using SQLite.Net.Interop;

[assembly: Dependency(typeof(Soccer.iOS.Config))]

namespace Soccer.iOS

{

public class Config : IConfig

{

private string directoryDB;

private ISQLitePlatform platform;

public string DirectoryDB

{

get

{

if (string.IsNullOrEmpty(directoryDB))

{

var directory = System.Environment.GetFolderPath(Environment.SpecialFolder.Personal);

directoryDB = System.IO.Path.Combine(directory, "..", "Library");

}

return directoryDB;

}

}

public ISQLitePlatform Platform

{

get

{

if (platform == null)

{

platform = new SQLite.Net.Platform.XamarinIOS.SQLitePlatformIOS();

}

return platform;

}

}

}

}