Soccer App

Contents

[1 Classes 3](#_Toc481675985)

[1.1 FilesHelper 3](#_Toc481675986)

[1.2 Response 3](#_Toc481675987)

[1.3 TokenResponse 3](#_Toc481675988)

[1.4 UserRequest 4](#_Toc481675989)

[2 Controls 4](#_Toc481675990)

[2.1 BindablePicker 4](#_Toc481675991)

[3 Data 9](#_Toc481675992)

[3.1 DataAccess 9](#_Toc481675993)

[4 Infrastructure 11](#_Toc481675994)

[4.1 InstanceLocator 11](#_Toc481675995)

[5 Interfaces 11](#_Toc481675996)

[5.1 IConfig 11](#_Toc481675997)

[6 Models 11](#_Toc481675998)

[6.1 Date 11](#_Toc481675999)

[6.2 Group 12](#_Toc481676000)

[6.3 Match 12](#_Toc481676001)

[6.4 Parameter 13](#_Toc481676002)

[6.5 Team 13](#_Toc481676003)

[6.6 Tournament 14](#_Toc481676004)

[6.7 User 14](#_Toc481676005)

[6.8 UserType 15](#_Toc481676006)

[7 Pages 16](#_Toc481676007)

[7.1 EditPredictionPage 16](#_Toc481676008)

[7.2 HomePage 17](#_Toc481676009)

[7.3 LoginPage 18](#_Toc481676010)

[7.4 MasterPage 20](#_Toc481676011)

[7.5 MenuPage 21](#_Toc481676012)

[7.6 NewUserPage 22](#_Toc481676013)

[7.7 SelectGroupPage 26](#_Toc481676014)

[7.8 SelectMatchPage 26](#_Toc481676015)

[7.9 SelectTournamentPage 28](#_Toc481676016)

[8 Services 30](#_Toc481676017)

[8.1 ApiService 30](#_Toc481676018)

[8.2 DataService 33](#_Toc481676019)

[8.3 DialogService 35](#_Toc481676020)

[8.4 NavigationService 36](#_Toc481676021)

[9 ViewModels 37](#_Toc481676022)

[9.1 EditPredictionViewModel 37](#_Toc481676023)

[9.2 GroupItemViewModel 38](#_Toc481676024)

[9.3 LoginViewModel 38](#_Toc481676025)

[9.4 MainViewModel 42](#_Toc481676026)

[9.5 MatchItemViewModel 44](#_Toc481676027)

[9.6 MenuItemViewModel 44](#_Toc481676028)

[9.7 NewUserViewModel 45](#_Toc481676029)

[9.8 SelectGroupViewModel 50](#_Toc481676030)

[9.9 SelectMatchViewModel 51](#_Toc481676031)

[9.10 SelectTournamentViewModel 53](#_Toc481676032)

[9.11 TournamentItemViewModel 55](#_Toc481676033)

[10 Root 55](#_Toc481676034)

[10.1 App 55](#_Toc481676035)

[11 Android 57](#_Toc481676036)

[11.1 Config 57](#_Toc481676037)

[12 iOS 58](#_Toc481676038)

[12.1 Config 58](#_Toc481676039)

[13 API 58](#_Toc481676040)

[13.1 GetMatchesToPredict 58](#_Toc481676041)

[14 BackEnd 60](#_Toc481676042)

[14.1 CloseMatch 60](#_Toc481676043)

[14.1.1 CloseMatch Post 60](#_Toc481676044)

[14.1.2 GetStatus 62](#_Toc481676045)

[14.1.3 CloseMatch Get 62](#_Toc481676046)

[14.1.4 View 63](#_Toc481676047)

[14.2 PostUser 64](#_Toc481676048)

[14.2.1 UserRequest 64](#_Toc481676049)

[14.2.2 FilesHelper 64](#_Toc481676050)

[14.2.3 PostUser 65](#_Toc481676051)

# Classes

## FilesHelper

using System.IO;

namespace Soccer.Classes

{

public class FilesHelper

{

public static byte[] ReadFully(Stream input)

{

using (MemoryStream ms = new MemoryStream())

{

input.CopyTo(ms);

return ms.ToArray();

}

}

}

}

## 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; }

}

}

# Controls

## BindablePicker

using System;

using System.Collections;

using System.Collections.Specialized;

using System.Reflection;

using Xamarin.Forms;

namespace Soccer.Controls

{

public class BindablePicker : Picker

{

bool \_disableNestedCalls;

public static readonly BindableProperty ItemsSourceProperty =

BindableProperty.Create("ItemsSource", typeof(IEnumerable), typeof(BindablePicker),

null, propertyChanged: OnItemsSourceChanged);

public static readonly BindableProperty SelectedItemProperty =

BindableProperty.Create("SelectedItem", typeof(object), typeof(BindablePicker),

null, BindingMode.TwoWay, propertyChanged: OnSelectedItemChanged);

public static readonly BindableProperty SelectedValueProperty =

BindableProperty.Create("SelectedValue", typeof(object), typeof(BindablePicker),

null, BindingMode.TwoWay, propertyChanged: OnSelectedValueChanged);

public string DisplayMemberPath { get; set; }

public IEnumerable ItemsSource

{

get { return (IEnumerable)GetValue(ItemsSourceProperty); }

set { SetValue(ItemsSourceProperty, value); }

}

public object SelectedItem

{

get { return GetValue(SelectedItemProperty); }

set

{

if (this.SelectedItem != value)

{

SetValue(SelectedItemProperty, value);

InternalSelectedItemChanged();

}

}

}

public object SelectedValue

{

get { return GetValue(SelectedValueProperty); }

set

{

SetValue(SelectedValueProperty, value);

InternalSelectedValueChanged();

}

}

public string SelectedValuePath { get; set; }

public BindablePicker()

{

this.SelectedIndexChanged += OnSelectedIndexChanged;

}

public event EventHandler<SelectedItemChangedEventArgs> ItemSelected;

void InstanceOnItemsSourceChanged(object oldValue, object newValue)

{

\_disableNestedCalls = true;

this.Items.Clear();

var oldCollectionINotifyCollectionChanged = oldValue as INotifyCollectionChanged;

if (oldCollectionINotifyCollectionChanged != null)

{

oldCollectionINotifyCollectionChanged.CollectionChanged -= ItemsSource\_CollectionChanged;

}

var newCollectionINotifyCollectionChanged = newValue as INotifyCollectionChanged;

if (newCollectionINotifyCollectionChanged != null)

{

newCollectionINotifyCollectionChanged.CollectionChanged += ItemsSource\_CollectionChanged;

}

if (!Equals(newValue, null))

{

var hasDisplayMemberPath = !string.IsNullOrWhiteSpace(this.DisplayMemberPath);

foreach (var item in (IEnumerable)newValue)

{

if (hasDisplayMemberPath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.DisplayMemberPath);

this.Items.Add(prop.GetValue(item).ToString());

}

else

{

this.Items.Add(item.ToString());

}

}

this.SelectedIndex = -1;

this.\_disableNestedCalls = false;

if (this.SelectedItem != null)

{

this.InternalSelectedItemChanged();

}

else if (hasDisplayMemberPath && this.SelectedValue != null)

{

this.InternalSelectedValueChanged();

}

}

else

{

\_disableNestedCalls = true;

this.SelectedIndex = -1;

this.SelectedItem = null;

this.SelectedValue = null;

\_disableNestedCalls = false;

}

}

void InternalSelectedItemChanged()

{

if (\_disableNestedCalls)

{

return;

}

var selectedIndex = -1;

object selectedValue = null;

if (this.ItemsSource != null)

{

var index = 0;

var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);

foreach (var item in this.ItemsSource)

{

if (item != null && item.Equals(this.SelectedItem))

{

selectedIndex = index;

if (hasSelectedValuePath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.SelectedValuePath);

selectedValue = prop.GetValue(item);

}

break;

}

index++;

}

}

\_disableNestedCalls = true;

this.SelectedValue = selectedValue;

this.SelectedIndex = selectedIndex;

\_disableNestedCalls = false;

}

void InternalSelectedValueChanged()

{

if (\_disableNestedCalls)

{

return;

}

if (string.IsNullOrWhiteSpace(this.SelectedValuePath))

{

return;

}

var selectedIndex = -1;

object selectedItem = null;

var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);

if (this.ItemsSource != null && hasSelectedValuePath)

{

var index = 0;

foreach (var item in this.ItemsSource)

{

if (item != null)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.SelectedValuePath);

if (object.Equals(prop.GetValue(item), this.SelectedValue))

{

selectedIndex = index;

selectedItem = item;

break;

}

}

index++;

}

}

\_disableNestedCalls = true;

this.SelectedItem = selectedItem;

this.SelectedIndex = selectedIndex;

\_disableNestedCalls = false;

}

void ItemsSource\_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)

{

var hasDisplayMemberPath = !string.IsNullOrWhiteSpace(this.DisplayMemberPath);

if (e.Action == NotifyCollectionChangedAction.Add)

{

foreach (var item in e.NewItems)

{

if (hasDisplayMemberPath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.DisplayMemberPath);

this.Items.Add(prop.GetValue(item).ToString());

}

else

{

this.Items.Add(item.ToString());

}

}

}

else if (e.Action == NotifyCollectionChangedAction.Remove)

{

foreach (var item in e.NewItems)

{

if (hasDisplayMemberPath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.DisplayMemberPath);

this.Items.Remove(prop.GetValue(item).ToString());

}

else

{

this.Items.Remove(item.ToString());

}

}

}

else if (e.Action == NotifyCollectionChangedAction.Replace)

{

foreach (var item in e.NewItems)

{

if (hasDisplayMemberPath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.DisplayMemberPath);

this.Items.Remove(prop.GetValue(item).ToString());

}

else

{

var index = this.Items.IndexOf(item.ToString());

if (index > -1)

{

this.Items[index] = item.ToString();

}

}

}

}

else if (e.Action == NotifyCollectionChangedAction.Reset)

{

this.Items.Clear();

if (e.NewItems != null)

{

foreach (var item in e.NewItems)

{

if (hasDisplayMemberPath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.DisplayMemberPath);

this.Items.Remove(prop.GetValue(item).ToString());

}

else

{

var index = this.Items.IndexOf(item.ToString());

if (index > -1)

{

this.Items[index] = item.ToString();

}

}

}

}

else

{

\_disableNestedCalls = true;

this.SelectedItem = null;

this.SelectedIndex = -1;

this.SelectedValue = null;

\_disableNestedCalls = false;

}

}

}

static void OnItemsSourceChanged(BindableObject bindable, object oldValue, object newValue)

{

if (Equals(newValue, null) && Equals(oldValue, null))

{

return;

}

var picker = (BindablePicker)bindable;

picker.InstanceOnItemsSourceChanged(oldValue, newValue);

}

void OnSelectedIndexChanged(object sender, EventArgs e)

{

if (\_disableNestedCalls)

{

return;

}

if (this.SelectedIndex < 0 || this.ItemsSource == null || !this.ItemsSource.GetEnumerator().MoveNext())

{

\_disableNestedCalls = true;

if (this.SelectedIndex != -1)

{

this.SelectedIndex = -1;

}

this.SelectedItem = null;

this.SelectedValue = null;

\_disableNestedCalls = false;

return;

}

\_disableNestedCalls = true;

var index = 0;

var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);

foreach (var item in this.ItemsSource)

{

if (index == this.SelectedIndex)

{

this.SelectedItem = item;

if (hasSelectedValuePath)

{

var type = item.GetType();

var prop = type.GetRuntimeProperty(this.SelectedValuePath);

this.SelectedValue = prop.GetValue(item);

}

break;

}

index++;

}

\_disableNestedCalls = false;

}

static void OnSelectedItemChanged(BindableObject bindable, object oldValue, object newValue)

{

var boundPicker = (BindablePicker)bindable;

boundPicker.ItemSelected?.Invoke(boundPicker, new SelectedItemChangedEventArgs(newValue));

boundPicker.InternalSelectedItemChanged();

}

static void OnSelectedValueChanged(BindableObject bindable, object oldValue, object newValue)

{

var boundPicker = (BindablePicker)bindable;

boundPicker.InternalSelectedValueChanged();

}

}

}

# 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

namespace Soccer.Models

{

public class Match

{

public int MatchId { get; set; }

public int DateId { get; set; }

public string 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 bool WasPredicted { get; set; }

public Team Local { get; set; }

public Team Visitor { get; set; }

public int? LocalGoals2

{

get

{

if (WasPredicted)

{

return LocalGoals;

}

return null;

}

}

public int? VisitorGoals2

{

get

{

if (WasPredicted)

{

return VisitorGoals;

}

return null;

}

}

}

}

## 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

## EditPredictionPage

<?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.EditPredictionPage"

BackgroundColor="{StaticResource AccentColor1}"

Title="Edit Prediction"

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

<StackLayout

BindingContext="{Binding EditPrediction}"

Padding="8">

<StackLayout

Orientation="Horizontal">

<StackLayout

VerticalOptions="Center"

HorizontalOptions="Start">

<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.Initials}">

</Label>

</StackLayout>

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center">

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center"

Orientation="Horizontal">

<Entry

FontAttributes="Bold"

Keyboard="Numeric"

FontSize="40"

VerticalOptions="Center"

HorizontalOptions="FillAndExpand"

TextColor="Black"

BackgroundColor="White"

Text="{Binding LocalGoals2, Mode=TwoWay}">

</Entry>

<Label

FontAttributes="Bold"

FontSize="Medium"

VerticalOptions="Center"

TextColor="Navy"

Text=" Vs. ">

</Label>

<Entry

FontAttributes="Bold"

Keyboard="Numeric"

FontSize="40"

VerticalOptions="Center"

HorizontalOptions="FillAndExpand"

TextColor="Black"

BackgroundColor="White"

Text="{Binding VisitorGoals2, Mode=TwoWay}">

</Entry>

</StackLayout>

<Label

TextColor="Black"

HorizontalOptions="Center"

VerticalOptions="Center"

Text="{Binding DateTime, StringFormat='{0:yyyy/MM/dd HH:mm}'}}">

</Label>

</StackLayout>

<StackLayout

VerticalOptions="Center"

HorizontalOptions="End">

<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.Initials}">

</Label>

</StackLayout>

</StackLayout>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<Button

Command="{Binding SaveCommand}"

IsEnabled="{Binding IsEnabled}"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="Orange"

TextColor="White"

HeightRequest="50"

BorderRadius="25"

Text="Save">

</Button>

</StackLayout>

</ContentPage>

## 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>

## NewUserPage

<?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:control="clr-namespace:Soccer.Controls"

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

x:Class="Soccer.Pages.NewUserPage"

BackgroundColor="{StaticResource AccentColor1}"

Title="New User"

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

<ScrollView

BindingContext="{Binding NewUser}">

<StackLayout

Padding="8">

<StackLayout

Orientation="Horizontal">

<Label

FontSize="Large"

FontAttributes="Bold"

TextColor="Black"

HorizontalOptions="FillAndExpand"

Text="Soccer New User">

</Label>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<Button

Command="{Binding CancelCommand}"

IsEnabled="{Binding IsEnabled}"

BackgroundColor="Green"

TextColor="White"

Text="Cancel">

</Button>

</StackLayout>

<Grid>

<Grid Grid.Row="0">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="First name"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding FirstName, Mode=TwoWay}"

Placeholder="Enter the first name..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

<Grid Grid.Row="1">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Last name"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding LastName, Mode=TwoWay}"

Placeholder="Enter the last name..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

</Grid>

<StackLayout

Orientation="Horizontal">

<Image

WidthRequest="50"

HeightRequest="50"

Grid.Column="0"

VerticalOptions="Start"

Source="ic\_action\_add\_a\_photo.png">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<controls:CircleImage

Grid.Column="1"

Source="{Binding ImageSource, Mode=TwoWay}"

Aspect="AspectFill"

WidthRequest="200"

HeightRequest="200">

</controls:CircleImage>

</StackLayout>

<Grid>

<Grid Grid.Row="0">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Password"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding Password, Mode=TwoWay}"

IsPassword="True"

Placeholder="Enter the password..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

<Grid Grid.Row="1">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Password confirm"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding PasswordConfirm, Mode=TwoWay}"

IsPassword="True"

Placeholder="Enter the password confirm..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

<Grid Grid.Row="2">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Email"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding Email, Mode=TwoWay}"

Keyboard="Email"

Placeholder="Enter the Email..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

<Grid Grid.Row="3">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Nick name"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<Entry

Grid.Column="1"

Text="{Binding NickName, Mode=TwoWay}"

Placeholder="Enter the nick name..."

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

BackgroundColor="White">

</Entry>

</Grid>

<Grid Grid.Row="4">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Favorite league team"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<control:BindablePicker

Grid.Column="1"

Title="Select a league..."

DisplayMemberPath="Name"

SelectedValuePath="LeagueId"

ItemsSource="{Binding Leagues}"

BackgroundColor="White"

SelectedValue ="{Binding Path=FavoriteLeagueId, Mode=TwoWay}"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center">

</control:BindablePicker>

</Grid>

<Grid Grid.Row="5">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="Favorite team"

FontAttributes="Bold"

HorizontalOptions="Start"

VerticalOptions="Center"

TextColor="Black">

</Label>

<control:BindablePicker

Grid.Column="1"

Title="Select a team..."

DisplayMemberPath="Name"

SelectedValuePath="TeamId"

ItemsSource="{Binding Teams}"

BackgroundColor="White"

SelectedValue ="{Binding Path=FavoriteTeamId, Mode=TwoWay}"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center">

</control:BindablePicker>

</Grid>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<Button

Command="{Binding SaveCommand}"

HeightRequest="40"

BorderRadius="20"

HorizontalOptions="FillAndExpand"

TextColor="White"

BackgroundColor="Orange"

Text="Save">

</Button>

</StackLayout>

</ScrollView>

</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.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="90"></ColumnDefinition>

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

<ColumnDefinition Width="90"></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.Initials}">

</Label>

</StackLayout>

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center"

Grid.Column="1">

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="Center"

Orientation="Horizontal">

<Label

FontAttributes="Bold"

FontSize="40"

VerticalOptions="Center"

TextColor="Black"

BackgroundColor="White"

Text="{Binding LocalGoals2}">

</Label>

<Label

FontAttributes="Bold"

FontSize="Medium"

VerticalOptions="Center"

TextColor="Navy"

Text=" Vs. ">

</Label>

<Label

FontAttributes="Bold"

FontSize="40"

VerticalOptions="Center"

TextColor="Black"

BackgroundColor="White"

Text="{Binding VisitorGoals2}">

</Label>

</StackLayout>

<Label

TextColor="Black"

HorizontalOptions="Center"

VerticalOptions="Center"

Text="{Binding DateTime, StringFormat='{0:yyyy/MM/dd HH:mm}'}}">

</Label>

</StackLayout>

<StackLayout

VerticalOptions="Center"

HorizontalOptions="End"

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.Initials}">

</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

## EditPredictionViewModel

using Soccer.Models;

using System.ComponentModel;

namespace Soccer.ViewModels

{

public class EditPredictionViewModel : Match, INotifyPropertyChanged

{

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Attributes

private Match match;

private bool isRunning;

private bool isEnabled;

#endregion

#region Properties

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;

}

}

#endregion

#region Constructor

public EditPredictionViewModel(Match match)

{

this.match = match;

DateId = match.DateId;

DateTime = match.DateTime;

Local = match.Local;

LocalGoals = match.LocalGoals;

LocalId = match.LocalId;

MatchId = match.MatchId;

StatusId = match.StatusId;

TournamentGroupId = match.TournamentGroupId;

Visitor = match.Visitor;

VisitorGoals = match.VisitorGoals;

VisitorId = match.VisitorId;

WasPredicted = match.WasPredicted;

IsEnabled = true;

}

#endregion

}

}

## 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

}

}

## NewUserViewModel

using GalaSoft.MvvmLight.Command;

using Plugin.Connectivity;

using Plugin.Media;

using Plugin.Media.Abstractions;

using Soccer.Models;

using Soccer.Services;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Linq;

using System.Windows.Input;

using Xamarin.Forms;

using Soccer.Classes;

namespace Soccer.ViewModels

{

public class NewUserViewModel : User, INotifyPropertyChanged

{

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Attributes

private ApiService apiService;

private DialogService dialogService;

private NavigationService navigationService;

private DataService dataService;

private bool isRunning;

private bool isEnabled;

private int favoriteLeagueId;

private List<League> leagues;

private ImageSource imageSource;

private MediaFile file;

#endregion

#region Properties

public string PasswordConfirm { get; set; }

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 int FavoriteLeagueId

{

set

{

if (favoriteLeagueId != value)

{

favoriteLeagueId = value;

ReloadTeams(favoriteLeagueId);

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

}

}

get

{

return favoriteLeagueId;

}

}

public ImageSource ImageSource

{

set

{

if (imageSource != value)

{

imageSource = value;

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

}

}

get

{

return imageSource;

}

}

public ObservableCollection<LeagueItemViewModel> Leagues { get; set; }

public ObservableCollection<TeamItemViewModel> Teams { get; set; }

#endregion

#region Constructor

public NewUserViewModel()

{

apiService = new ApiService();

dialogService = new DialogService();

navigationService = new NavigationService();

dataService = new DataService();

Leagues = new ObservableCollection<LeagueItemViewModel>();

Teams = new ObservableCollection<TeamItemViewModel>();

Picture = "avatar\_user.png";

IsEnabled = true;

LoadLeagues();

}

#endregion

#region Methods

private async void LoadLeagues()

{

if (!CrossConnectivity.Current.IsConnected)

{

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

return;

}

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

if (!isReachable)

{

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

return;

}

IsRunning = true;

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

var response = await apiService.Get<League>(parameters.URLBase, "/api", "/Leagues");

IsRunning = false;

if (!response.IsSuccess)

{

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

return;

}

leagues = (List<League>)response.Result;

ReloadLeagues(leagues);

}

private void ReloadLeagues(List<League> leagues)

{

Leagues.Clear();

foreach (var league in leagues.OrderBy(l => l.Name))

{

Leagues.Add(new LeagueItemViewModel

{

LeagueId = league.LeagueId,

Logo = league.Logo,

Name = league.Name,

Teams = league.Teams,

});

}

}

private void ReloadTeams(int favoriteLeagueId)

{

var teams = leagues.Where(l => l.LeagueId == favoriteLeagueId).FirstOrDefault().Teams;

Teams.Clear();

foreach (var team in teams.OrderBy(t => t.Name))

{

Teams.Add(new TeamItemViewModel

{

Fans = team.Fans,

Initials = team.Initials,

LeagueId = team.LeagueId,

Logo = team.Logo,

Name = team.Name,

TeamId = team.TeamId,

});

}

}

#endregion

#region Commands

public ICommand SaveCommand { get { return new RelayCommand(Save); } }

private async void Save()

{

if (string.IsNullOrEmpty(FirstName))

{

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

return;

}

if (string.IsNullOrEmpty(LastName))

{

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

return;

}

if (string.IsNullOrEmpty(Password))

{

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

return;

}

if (Password.Length < 6)

{

await dialogService.ShowMessage("Error", "The password must have at least 6 characters.");

return;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

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

return;

}

if (Password != PasswordConfirm)

{

await dialogService.ShowMessage("Error", "The password and confirm does not match.");

return;

}

if (string.IsNullOrEmpty(Email))

{

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

return;

}

if (string.IsNullOrEmpty(NickName))

{

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

return;

}

if (FavoriteTeamId == 0)

{

await dialogService.ShowMessage("Error", "You must select a favorite team.");

return;

}

if (!CrossConnectivity.Current.IsConnected)

{

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

return;

}

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

if (!isReachable)

{

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

return;

}

IsRunning = true;

IsEnabled = false;

var imageArray = FilesHelper.ReadFully(file.GetStream());

file.Dispose();

var user = new User

{

Email = Email,

FavoriteTeamId = FavoriteTeamId,

FirstName = FirstName,

ImageArray = imageArray,

LastName = LastName,

NickName = NickName,

Password = Password,

UserTypeId = 1,

};

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

var response = await apiService.Post(parameters.URLBase2, "/api", "/Users", user);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

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

return;

}

await dialogService.ShowMessage("Confirmation", "The user was created, please login.");

await navigationService.Back();

}

public ICommand CancelCommand { get { return new RelayCommand(Cancel); } }

private void Cancel()

{

navigationService.SetMainPage("LoginPage");

}

public ICommand TakePictureCommand { get { return new RelayCommand(TakePicture); } }

private async void TakePicture()

{

await CrossMedia.Current.Initialize();

if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)

{

await dialogService.ShowMessage("No Camera", ":( No camera available.");

return;

}

IsRunning = true;

file = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

});

if (file != null)

{

ImageSource = ImageSource.FromStream(() =>

{

var stream = file.GetStream();

return stream;

});

}

IsRunning = false;

}

#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;

}

}

}

}

# API

## GetMatchesToPredict

public class MatchResponse

{

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 bool WasPredicted { get; set; }

public Team Local { get; set; }

public Team Visitor { get; set; }

}

[RoutePrefix("api/Tournaments")]

public class TournamentsController : ApiController

{

private DataContext db = new DataContext();

[Route("GetMatchesToPredict/{tournamentId}/{userId}")]

public async Task<IHttpActionResult> GetMatchesToPredict(int tournamentId, int userId)

{

var qry = await (from t in db.Tournaments

join d in db.Dates on t.TournamentId equals d.TournamentId

join m in db.Matches on d.DateId equals m.DateId

where t.TournamentId == tournamentId && m.StatusId != 3 && m.DateTime > DateTime.Now

select new { m }).ToListAsync();

var predictions = await db.Predictions.Where(p => p.UserId == userId).ToListAsync();

var matches = new List<MatchResponse>();

foreach (var item in qry)

{

var matchResponse = new MatchResponse

{

DateId = item.m.DateId,

DateTime = item.m.DateTime,

Local = item.m.Local,

LocalGoals = item.m.LocalGoals,

LocalId = item.m.LocalId,

MatchId = item.m.MatchId,

StatusId = item.m.StatusId,

TournamentGroupId = item.m.TournamentGroupId,

Visitor = item.m.Visitor,

VisitorGoals = item.m.VisitorGoals,

VisitorId = item.m.VisitorId,

};

var prediction = predictions.Where(p => p.MatchId == item.m.MatchId).FirstOrDefault();

if (prediction != null)

{

matchResponse.LocalGoals = prediction.LocalGoals;

matchResponse.VisitorGoals = prediction.VisitorGoals;

matchResponse.WasPredicted = true;

}

else

{

matchResponse.WasPredicted = false;

}

matches.Add(matchResponse);

}

return Ok(matches.OrderBy(m => m.DateTime));

}

# BackEnd

## CloseMatch

### CloseMatch Post

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize(Roles = "Admin")]

public async Task<ActionResult> CloseMatch(Match match)

{

using (var transacction = db.Database.BeginTransaction())

{

try

{

// Update match

var oldMatch = await db.Matches.FindAsync(match.MatchId);

oldMatch.LocalGoals = match.LocalGoals;

oldMatch.VisitorGoals = match.VisitorGoals;

oldMatch.StatusId = 3;

db.Entry(oldMatch).State = EntityState.Modified;

var statusMatch = GetStatus(match.LocalGoals.Value, match.VisitorGoals.Value);

// Update tournaments statatistics

var local = await db.TournamentTeams

.Where(tt => tt.TournamentGroupId == oldMatch.TournamentGroupId &&

tt.TeamId == oldMatch.LocalId)

.FirstOrDefaultAsync();

var visitor = await db.TournamentTeams

.Where(tt => tt.TournamentGroupId == oldMatch.TournamentGroupId &&

tt.TeamId == oldMatch.VisitorId)

.FirstOrDefaultAsync();

local.MatchesPlayed++;

local.FavorGoals += oldMatch.LocalGoals.Value;

local.AgainstGoals += oldMatch.VisitorGoals.Value;

visitor.MatchesPlayed++;

visitor.FavorGoals += oldMatch.VisitorGoals.Value;

visitor.AgainstGoals += oldMatch.LocalGoals.Value;

if (statusMatch == 1)

{

local.MatchesWon++;

local.Points += 3;

visitor.MatchesLost++;

}

else if (statusMatch == 2)

{

visitor.MatchesWon++;

visitor.Points += 3;

local.MatchesLost++;

}

else

{

local.MatchesTied++;

visitor.MatchesTied++;

local.Points++;

visitor.Points++;

}

db.Entry(local).State = EntityState.Modified;

db.Entry(visitor).State = EntityState.Modified;

await db.SaveChangesAsync();

// Update positions

var teams = await db.TournamentTeams

.Where(tt => tt.TournamentGroupId == oldMatch.TournamentGroupId)

.ToListAsync();

var i = 1;

foreach (var team in teams.OrderByDescending(t => t.Points)

.ThenByDescending(t => t.FavorGoals - t.AgainstGoals)

.ThenByDescending(t => t.FavorGoals))

{

team.Position = i;

db.Entry(team).State = EntityState.Modified;

i++;

}

// Update predictions

var predictions = await db.Predictions.Where(p => p.MatchId == oldMatch.MatchId).ToListAsync();

foreach (var prediction in predictions)

{

var points = 0;

if (prediction.LocalGoals == oldMatch.LocalGoals &&

prediction.VisitorGoals == oldMatch.VisitorGoals)

{

points = 3;

}

else

{

var statusPrediction = GetStatus(prediction.LocalGoals, prediction.VisitorGoals);

if (statusMatch == statusPrediction)

{

points = 1;

}

}

if (points != 0)

{

prediction.Points = points;

db.Entry(prediction).State = EntityState.Modified;

}

// Update user

var user = await db.Users.FindAsync(prediction.UserId);

user.Points += points;

db.Entry(user).State = EntityState.Modified;

// Update points in groups

var groupUsers = await db.GroupUsers.Where(gu => gu.UserId == user.UserId &&

gu.IsAccepted &&

!gu.IsBlocked)

.ToListAsync();

foreach (var groupUser in groupUsers)

{

groupUser.Points += points;

db.Entry(groupUser).State = EntityState.Modified;

}

}

await db.SaveChangesAsync();

transacction.Commit();

return RedirectToAction(string.Format("DetailsDate/{0}", oldMatch.DateId));

}

catch (Exception ex)

{

transacction.Rollback();

ModelState.AddModelError(string.Empty, ex.Message);

return View(match);

}

}

}

### GetStatus

private int GetStatus(int localGoals, int visitorGoals)

{

if (localGoals > visitorGoals)

{

return 1; // Local win

}

if (visitorGoals > localGoals)

{

return 2; // Visitor win

}

return 3; // Draw

}

### CloseMatch Get

public async Task<ActionResult> CloseMatch(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var match = await db.Matches.FindAsync(id);

if (match == null)

{

return HttpNotFound();

}

if (match.StatusId == 3)

{

return RedirectToAction(string.Format("DetailsDate/{0}", match.DateId));

}

return View(match);

}

### View

@model Domain.Match

@{

ViewBag.Title = "CloseMatch";

}

<div class="row">

<div class="col-md-6">

<h2>Close Match</h2>

<h3>Tournament: @Model.Date.Tournament.Name</h3>

<h4>Date: @Model.Date.Name</h4>

</div>

<div class="col-md-6">

<br />

<table>

<tr>

<td width="200px" align="center">

@if (!string.IsNullOrEmpty(Model.Local.Logo))

{

<img src="@Url.Content(Model.Local.Logo)" alt="Image" style="width:120px;height:120px;max-width: 100%; height: auto;" />

}

</td>

<td width="50px" align="center">

<h2>Vs.</h2>

</td>

<td width="200px" align="center">

@if (!string.IsNullOrEmpty(Model.Visitor.Logo))

{

<img src="@Url.Content(Model.Visitor.Logo)" alt="Image" style="width:120px;height:120px;max-width: 100%; height: auto;" />

}

</td>

</tr>

<tr>

<td align="center">

<h3>@Model.Local.Name</h3>

</td>

<td width="50px" align="center"></td>

<td align="center">

<h3>@Model.Visitor.Name</h3>

</td>

</tr>

</table>

</div>

</div>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.MatchId)

<div class="form-group">

@Html.LabelFor(model => model.LocalGoals, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.LocalGoals, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.LocalGoals, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.VisitorGoals, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.VisitorGoals, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.VisitorGoals, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Close Match" class="btn btn-primary" onclick = "return confirm('Are you sure to close the math?');" />

@Html.ActionLink("Back to Date", "DetailsDate", new { id = Model.DateId }, new { @class = "btn btn-success" })

</div>

</div>

</div>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

## PostUser

### UserRequest

using Domain;

using System.ComponentModel.DataAnnotations.Schema;

namespace Backend.Classes

{

[NotMapped]

public class UserRequest : User

{

public string Password { get; set; }

public byte[] ImageArray { get; set; }

}

}

### FilesHelper

using System.IO;

using System.Web;

namespace Backend.Classes

{

public class FilesHelper

{

public static string UploadPhoto(HttpPostedFileBase file, string folder)

{

var path = string.Empty;

var pic = string.Empty;

if (file != null)

{

pic = Path.GetFileName(file.FileName);

path = Path.Combine(HttpContext.Current.Server.MapPath(folder), pic);

file.SaveAs(path);

}

return pic;

}

public static bool UploadPhoto(MemoryStream stream, string folder, string name)

{

try

{

stream.Position = 0;

var path = Path.Combine(HttpContext.Current.Server.MapPath(folder), name);

File.WriteAllBytes(path, stream.ToArray());

}

catch

{

return false;

}

return true;

}

}

}

### PostUser

// POST: api/Users

[ResponseType(typeof(User))]

public async Task<IHttpActionResult> PostUser(UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (request.ImageArray != null && request.ImageArray.Length > 0)

{

var stream = new MemoryStream(request.ImageArray);

var guid = Guid.NewGuid().ToString();

var file = string.Format("{0}.jpg", guid);

var folder = "~/Content/Users";

var fullPath = string.Format("{0}/{1}", folder, file);

var response = FilesHelper.UploadPhoto(stream, folder, file);

if (response)

{

request.Picture = fullPath;

}

}

var user = ToUser(request);

db.Users.Add(user);

await db.SaveChangesAsync();

UsersHelper.CreateUserASP(request.Email, "User", request.Password);

return CreatedAtRoute("DefaultApi", new { id = user.UserId }, user);

}