**.NET Core**

**Xamarin Forms + Prism**

**Xamarin Classic + MVVM Cross**



**Leasing Example**

By Zulu

Medellín 2019

**Index**

[**Leasing example: functionality matrix**](#_2hu2uvvbf8) **4**

[**Create the Solution**](#_pia4kl3piujv) **5**

[**Web First Part**](#_ubhdk9v4j2i7) **6**

[Create the Database](#_3znysh7) 6

[Modify DB](#_2et92p0) 11

[Seed the DB with initial data](#_tyjcwt) 19

[Add User Identities](#_j2qk9hw9udeq) 26

[Implementing login and logout in Web](#_rvvzh1dny0x) 40

[Create personalized controllers](#_n8zywv46w5ma) 47

[Add API](#_2s8eyo1) 123

[Add Tokens Generation](#_sqsqxlje6ef2) 132

[Publish on Azure](#_ofdhtv4fdwnx) 137

[**App Xamarin Forms First Part**](#_opym9lysl1d) **140**

[Login](#_yotgj1amfdah) 140

[Modify Response to generic Class](#_m9hxeoltmkl9) 152

[Check the internet connection](#_2whmgs8xsoyd) 156

[GROUP1](#_o6a2cvl176yv) 158

[Navigate to another page and pass parameters](#_8sv7m46oywl) 158

[Fix the images on Android](#_8mbpyrimot2z) 180

[Add SfRotator](#_hfrp56ptc5uh) 182

[Improve SfRotator and fix for Android](#_nhtanb2er1uq) 188

[GROUP4](#_d1h8r2ku8rr0) 192

[Add tabbed page](#_i7367dwh88o8) 192

[Add SfBusyIndicator](#_f6iruwi9yg5t) 205

[GIT Workshop](#_g7gly1u2ougn) 207

[**Web Second Part**](#_7z2tyymtk9ia) **208**

[Redirect Pages](#_xacmwy12tj9a) 208

[Self-registration of users](#_a8xsy9ylifaf) 210

[Modifying users](#_1ksv4uv) 220

[Confirm Email Registration](#_3whwml4) 227

[Password Recovery](#_2bn6wsx) 236

[Improve Index View](#_8b6k2hep1w4n) 243

[Managers CRUD](#_36ee3rz0wepn) 286

[Look and feel for a leasing system and users functionality](#_nyt9f2xo5hi3) 302

[Users functionality](#_izblywp49i4m) 305

[Search Properties](#_se4nce8oefmr) 305

[My Properties](#_ujtacayo16tj) 314

[My Contracts](#_xzwr39tg7sp) 349

[Prepare the API for new functionality in App](#_6x09d4hxxp7x) 357

[Account](#_8cbwohrru61p) 357

[Owner](#_w05gyav52mhn) 366

[Properties](#_z81cd4yrea2n) 374

[**App Xamarin Forms Second Part**](#_ald5cft967rf) **383**

[Add persistent setting](#_lnhhjejq0mm4) 383

[Add a Master Detail](#_jru7x1wimlgw) 386

[Add Icon & Splash to Xamarin Forms For Android](#_ditq6h2uupkm) 394

[Adding Styles](#_k0swujas1epb) 396

[Multi Language in Xamarin Forms](#_y4iegndojh07) 402

[Register users from App](#_r7prm6btz1r8) 419

[Recover Password From App in Xamarin Forms](#_c7bwyq5zm41g) 444

[Remember Me functionality](#_brkvnq7d4oki) 451

[Modify User From App in Xamarin Forms](#_1lgx68toiptd) 454

[Modify Password From App in Xamarin Forms](#_qpyf5uxa0zxi) 467

[App Property From App & Accessing Camera and Photo Library](#_9gfrccp34q8l) 477

[Show the map and move to current location](#_cgyol3lxmf51) 544

[Put pins in map](#_oxe37i3pus8z) 550

# Leasing example: functionality matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Functionality** | **Web** | | | **App** | |
| **Admin** | **Owner** | **Lessee** | **Owner** | **Lessee** |
| Login | X | X | X | X | X |
| Register |  | X | X | X | X |
| Modify profile | X | X | X | X | X |
| Recover password | X | X | X | X | X |
| Admin managers | X |  |  |  |  |
| Admin property types | X |  |  |  |  |
| Admin owners | X |  |  |  |  |
| Admin leeses | X |  |  |  |  |
| Admin contracts | X |  |  |  |  |
| See my contracts |  | X | X | X | X |
| Admin properties | X | X |  | X |  |
| See Leasing offices on map |  |  |  | X | X |
| Search available properties |  |  |  |  | X |

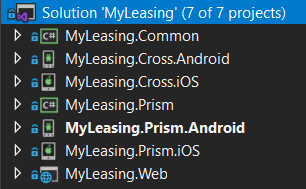
# Create the Solution

**Note:** all the code are in: <https://github.com/Zulu55/MyLeasing>

Create the following solution:



In Visual Studio, you must build something similar to:



# Web First Part

## Create the Database

**Note**: in this project we’ll work with entity framework code first, if you want to work with EF database first, I recommend this article: <https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

1. Create the **Owner** class:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class Owner

{

public int Id { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[Display(Name = "Fixed Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string FixedPhone { get; set; }

[Display(Name = "Cell Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string CellPhone { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

public string FullName => $"{FirstName} {LastName}";

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

}

1. Create the **DataContext** class:

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Owner> Owners { get; set; }

}

}

1. Add the connection string to the configuration json file: **appsettings.json** (see the SQL Server Object Explorer):

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=MyLeasing;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

}

**Note**: You must be sure of the servers names in your installation, you can check it out by clicking in SQL Server Object Explorer:



In this case, there are three available servers: (localdb)\MSSQLLocalDB, (localdb)\ProjectsV13 and (localdb)\v11.0. Or you can explore your server by clicking on “Add SQL Server” icon:



1. Add the database injection in **Startup** class:

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

1. Save changes and run those commands by command line in the same folder that is the web project:

PM> update-database

PM> add-migration InitialDb

PM> update-database

1. Add the **OwnersController**.
2. Add the Owners menu and test the DB connection.

<ul class="nav navbar-nav">

<li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a></li>

<li><a asp-area="" asp-controller="Owners" asp-action="Index">Owners</a></li>

</ul>

## Modify DB

We will complete the DB initially with these entities and relationships:



1. Add the entity **PropertyType**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class PropertyType

{

public int Id { get; set; }

[Display(Name = "Property Type")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Name { get; set; }

}

}

1. Add the entity **Lessee**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class Lessee

{

public int Id { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[Display(Name = "Fixed Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string FixedPhone { get; set; }

[Display(Name = "Cell Phone")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string CellPhone { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

public string FullName => $"{FirstName} {LastName}";

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

}

1. Add the entity **Property**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class Property

{

public int Id { get; set; }

[Display(Name = "Neighborhood")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Neighborhood { get; set; }

[Display(Name = "Address")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Address { get; set; }

[Display(Name = "Price")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:C2}", ApplyFormatInEditMode = false)]

public decimal Price { get; set; }

[Display(Name = "Square meters")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public int SquareMeters { get; set; }

[Display(Name = "Square meters")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public int Rooms { get; set; }

[Display(Name = "Stratum")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public int Stratum { get; set; }

[Display(Name = "Has Parking Lot?")]

public bool HasParkingLot { get; set; }

[Display(Name = "Is Available?")]

public bool IsAvailable { get; set; }

public string Remarks { get; set; }

public PropertyType PropertyType { get; set; }

public Owner Owner { get; set; }

}

}

1. Add the property **Properties** to **Owner** entity:

public ICollection<Property> Properties { get; set; }

1. Add the entity **PropertyImage**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class PropertyImage

{

public int Id { get; set; }

[Display(Name = "Image")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string ImageUrl { get; set; }

public Property Property { get; set; }

// TODO: Change the path when publish

public string ImageFullPath => $"https://TBD.azurewebsites.net{ImageUrl.Substring(1)}";

}

}

1. Add the property **PropertyImages** to **Property** entity:

public ICollection<PropertyImage> PropertyImages { get; set; }

1. Add the entity **Contract**:

using System;

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class Contract

{

public int Id { get; set; }

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Remarks { get; set; }

[Display(Name = "Price")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:C2}", ApplyFormatInEditMode = false)]

public decimal Price { get; set; }

[Display(Name = "Start Date")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd}", ApplyFormatInEditMode = true)]

public DateTime StartDate { get; set; }

[Display(Name = "End Date")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd}", ApplyFormatInEditMode = true)]

public DateTime EndDate { get; set; }

[Display(Name = "Is Active?")]

public bool IsActive { get; set; }

public Property Property { get; set; }

public Owner Owner { get; set; }

public Lessee Lessee { get; set; }

[Display(Name = "Start Date")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd}", ApplyFormatInEditMode = true)]

public DateTime StartDateLocal => StartDate.ToLocalTime();

[Display(Name = "End Date")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd}", ApplyFormatInEditMode = true)]

public DateTime EndDateLocal => EndDate.ToLocalTime();

}

}

1. Add the property **Contracts** to **Property**, **Owner** and **Lessee** entities:

public ICollection<Contract> Contracts { get; set; }

1. Update the **DataContext**:

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Data

{

public class DataContext : DbContext

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Contract> Contracts { get; set; }

public DbSet<Lessee> Lessees { get; set; }

public DbSet<Owner> Owners { get; set; }

public DbSet<Property> Properties { get; set; }

public DbSet<PropertyImage> PropertyImages { get; set; }

public DbSet<PropertyType> PropertyTypes { get; set; }

}

}

1. Save all and run this commands:

PM> add-migration CompleteDB

PM> update-database

1. Test it.

## Seed the DB with initial data

1. Create the **SeedDb** class, with your population data logic:

using MyLeasing.Web.Data.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MyLeasing.Web.Data

{

public class SeedDb

{

private readonly DataContext \_context;

public SeedDb(DataContext context)

{

\_context = context;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckPropertyTypesAsync();

await CheckOwnersAsync();

await CheckLesseesAsync();

await CheckPropertiesAsync();

}

private async Task CheckPropertyTypesAsync()

{

if (!\_context.PropertyTypes.Any())

{

\_context.PropertyTypes.Add(new Entities.PropertyType { Name = "Apartamento" });

\_context.PropertyTypes.Add(new Entities.PropertyType { Name = "Casa" });

\_context.PropertyTypes.Add(new Entities.PropertyType { Name = "Negocio" });

await \_context.SaveChangesAsync();

}

}

private async Task CheckLesseesAsync()

{

if (!\_context.Lessees.Any())

{

AddLessee("876543", "Ramon", "Gamboa", "234 3232", "310 322 3221", "Calle Luna Calle Sol");

AddLessee("654565", "Julian", "Martinez", "343 3226", "300 322 3221", "Calle 77 #22 21");

AddLessee("214231", "Carmenza", "Ruis", "450 4332", "350 322 3221", "Carrera 56 #22 21");

await \_context.SaveChangesAsync();

}

}

private void AddLessee(string document, string firstName, string lastName, string fixedPhone, string cellPhone, string address)

{

\_context.Lessees.Add(new Lessee

{

Address = address,

CellPhone = cellPhone,

Document = document,

FirstName = firstName,

FixedPhone = fixedPhone,

LastName = lastName

});

}

private async Task CheckPropertiesAsync()

{

var owner = \_context.Owners.FirstOrDefault();

var propertyType = \_context.PropertyTypes.FirstOrDefault();

if (!\_context.Properties.Any())

{

AddProperty("Calle 43 #23 32", "Poblado", owner, propertyType, 800000M, 2, 72, 4);

AddProperty("Calle 12 Sur #2 34", "Envigado", owner, propertyType, 950000M, 3, 81, 3);

await \_context.SaveChangesAsync();

}

}

private async Task CheckOwnersAsync()

{

if (!\_context.Owners.Any())

{

AddOwner("8989898", "Juan", "Zuluaga", "234 3232", "310 322 3221", "Calle Luna Calle Sol");

AddOwner("7655544", "Jose", "Cardona", "343 3226", "300 322 3221", "Calle 77 #22 21");

AddOwner("6565555", "Maria", "López", "450 4332", "350 322 3221", "Carrera 56 #22 21");

await \_context.SaveChangesAsync();

}

}

private void AddOwner(

string document,

string firstName,

string lastName,

string fixedPhone,

string cellPhone,

string address)

{

\_context.Owners.Add(new Owner

{

Address = address,

CellPhone = cellPhone,

Document = document,

FirstName = firstName,

FixedPhone = fixedPhone,

LastName = lastName

});

}

private void AddProperty(

string address,

string neighborhood,

Owner owner,

PropertyType propertyType,

decimal price,

int rooms,

int squareMeters,

int stratum)

{

\_context.Properties.Add(new Property

{

Address = address,

HasParkingLot = true,

IsAvailable = true,

Neighborhood = neighborhood,

Owner = owner,

Price = price,

PropertyType = propertyType,

Rooms = rooms,

SquareMeters = squareMeters,

Stratum = stratum

});

}

}

}

1. Modify the **Program** class:

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.DependencyInjection;

using MyLeasing.Web.Data;

namespace MyLeasing.Web

{

public class Program

{

public static void Main(string[] args)

{

var host = CreateWebHostBuilder(args).Build();

RunSeeding(host);

host.Run();

}

private static void RunSeeding(IWebHost host)

{

var scopeFactory = host.Services.GetService<IServiceScopeFactory>();

using (var scope = scopeFactory.CreateScope())

{

var seeder = scope.ServiceProvider.GetService<SeedDb>();

seeder.SeedAsync().Wait();

}

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args)

{

return WebHost.CreateDefaultBuilder(args).UseStartup<Startup>();

}

}

}

1. Add the injection for the seeder in **Startup**:

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddTransient<SeedDb>();

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

1. Test it.

## Add User Identities

We’ll complete out database with this entities:



1. Create your own **User** class inherit from **IdentityUser** class:

using Microsoft.AspNetCore.Identity;

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class User : IdentityUser

{

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

public string FullName => $"{FirstName} {LastName}";

public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";

}

}

1. Modify the entity **Owner**:

using System.Collections.Generic;

namespace MyLeasing.Web.Data.Entities

{

public class Owner

{

public int Id { get; set; }

public User User { get; set; }

public ICollection<Property> Properties { get; set; }

public ICollection<Contract> Contracts { get; set; }

}

}

1. Modify the entity **Lessee**:

using System.Collections.Generic;

namespace MyLeasing.Web.Data.Entities

{

public class Lessee

{

public int Id { get; set; }

public User User { get; set; }

public ICollection<Contract> Contracts { get; set; }

}

}

1. Add the new entity **Manager**:

namespace MyLeasing.Web.Data.Entities

{

public class Manager

{

public int Id { get; set; }

public User User { get; set; }

}

}

1. Modify the data context class:

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Data

{

public class DataContext : IdentityDbContext<User>

{

public DataContext(DbContextOptions<DataContext> options) : base(options)

{

}

public DbSet<Contract> Contracts { get; set; }

public DbSet<Lessee> Lessees { get; set; }

public DbSet<Manager> Managers { get; set; }

public DbSet<Owner> Owners { get; set; }

public DbSet<Property> Properties { get; set; }

public DbSet<PropertyImage> PropertyImages { get; set; }

public DbSet<PropertyType> PropertyTypes { get; set; }

}

}

1. Create the folder **Helpers** and inside it add the interface **IUserHelper**:

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Helpers

{

public interface IUserHelper

{

Task<User> GetUserByEmailAsync(string email);

Task<IdentityResult> AddUserAsync(User user, string password);

Task CheckRoleAsync(string roleName);

Task AddUserToRoleAsync(User user, string roleName);

Task<bool> IsUserInRoleAsync(User user, string roleName);

}

}

1. In the same folder add the implementation (**UserHelper**):

using Microsoft.AspNetCore.Identity;

using MyLeasing.Web.Data.Entities;

using System.Threading.Tasks;

namespace MyLeasing.Web.Helpers

{

public class UserHelper : IUserHelper

{

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UserHelper(

UserManager<User> userManager,

RoleManager<IdentityRole> roleManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

}

public async Task<IdentityResult> AddUserAsync(User user, string password)

{

return await \_userManager.CreateAsync(user, password);

}

public async Task AddUserToRoleAsync(User user, string roleName)

{

await \_userManager.AddToRoleAsync(user, roleName);

}

public async Task CheckRoleAsync(string roleName)

{

var roleExists = await \_roleManager.RoleExistsAsync(roleName);

if (!roleExists)

{

await \_roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

public async Task<User> GetUserByEmailAsync(string email)

{

var user = await \_userManager.FindByEmailAsync(email);

return user;

}

public async Task<bool> IsUserInRoleAsync(User user, string roleName)

{

return await \_userManager.IsInRoleAsync(user, roleName);

}

}

}

1. Add the injection in **Startup**:

services.AddScoped<IUserHelper, UserHelper>();

1. Modify the **SeedDb**:

using System;

using System.Linq;

using System.Threading.Tasks;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Helpers;

namespace MyLeasing.Web.Data

{

public class SeedDb

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

public SeedDb(

DataContext context,

IUserHelper userHelper)

{

\_context = context;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckRoles();

var manager = await CheckUserAsync("1010", "Juan", "Zuluaga", "jzuluaga55@gmail.com", "350 634 2747", "Calle Luna Calle Sol", "Manager");

var owner = await CheckUserAsync("2020", "Juan", "Zuluaga", "jzuluaga55@hotmail.com", "350 634 2747", "Calle Luna Calle Sol", "Owner");

var lessee = await CheckUserAsync("2020", "Juan", "Zuluaga", "carlos.zuluaga@globant.com", "350 634 2747", "Calle Luna Calle Sol", "Lessee");

await CheckPropertyTypesAsync();

await CheckManagerAsync(manager);

await CheckOwnersAsync(owner);

await CheckLesseesAsync(lessee);

await CheckPropertiesAsync();

await CheckContractsAsync();

}

private async Task CheckContractsAsync()

{

var owner = \_context.Owners.FirstOrDefault();

var lessee = \_context.Lessees.FirstOrDefault();

var property = \_context.Properties.FirstOrDefault();

if (!\_context.Contracts.Any())

{

\_context.Contracts.Add(new Contract

{

StartDate = DateTime.Today,

EndDate = DateTime.Today.AddYears(1),

IsActive = true,

Lessee = lessee,

Owner = owner,

Price = 800000M,

Property = property,

Remarks = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris nec iaculis ex. Nullam gravida nunc eleifend, placerat tellus a, eleifend metus. Phasellus id suscipit magna. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam volutpat ultrices ex, sed cursus sem tincidunt ut. Nullam metus lorem, convallis quis dignissim quis, porttitor quis leo. In hac habitasse platea dictumst. Duis pharetra sed arcu ac viverra. Proin dapibus lobortis commodo. Vivamus non commodo est, ac vehicula augue. Nam enim felis, rutrum in tortor sit amet, efficitur hendrerit augue. Cras pellentesque nisl eu maximus tempor. Curabitur eu efficitur metus. Sed ultricies urna et auctor commodo."

});

await \_context.SaveChangesAsync();

}

}

private async Task CheckManagerAsync(User user)

{

if (!\_context.Managers.Any())

{

\_context.Managers.Add(new Manager { User = user });

await \_context.SaveChangesAsync();

}

}

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string phone, string address, string role)

{

var user = await \_userHelper.GetUserByEmailAsync(email);

if (user == null)

{

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, role);

}

return user;

}

private async Task CheckRoles()

{

await \_userHelper.CheckRoleAsync("Manager");

await \_userHelper.CheckRoleAsync("Owner");

await \_userHelper.CheckRoleAsync("Lessee");

}

private async Task CheckLesseesAsync(User user)

{

if (!\_context.Lessees.Any())

{

\_context.Lessees.Add(new Lessee { User = user });

await \_context.SaveChangesAsync();

}

}

private async Task CheckPropertiesAsync()

{

var owner = \_context.Owners.FirstOrDefault();

var propertyType = \_context.PropertyTypes.FirstOrDefault();

if (!\_context.Properties.Any())

{

AddProperty("Calle 43 #23 32", "Poblado", owner, propertyType, 800000M, 2, 72, 4);

AddProperty("Calle 12 Sur #2 34", "Envigado", owner, propertyType, 950000M, 3, 81, 3);

await \_context.SaveChangesAsync();

}

}

private async Task CheckPropertyTypesAsync()

{

if (!\_context.PropertyTypes.Any())

{

\_context.PropertyTypes.Add(new PropertyType { Name = "Apartamento" });

\_context.PropertyTypes.Add(new PropertyType { Name = "Casa" });

\_context.PropertyTypes.Add(new PropertyType { Name = "Negocio" });

await \_context.SaveChangesAsync();

}

}

private async Task CheckOwnersAsync(User user)

{

if (!\_context.Owners.Any())

{

\_context.Owners.Add(new Owner { User = user });

await \_context.SaveChangesAsync();

}

}

private void AddProperty(string address, string neighborhood, Owner owner, PropertyType propertyType, decimal price, int rooms, int squareMeters, int stratum)

{

\_context.Properties.Add(new Property

{

Address = address,

HasParkingLot = true,

IsAvailable = true,

Neighborhood = neighborhood,

Owner = owner,

Price = price,

PropertyType = propertyType,

Rooms = rooms,

SquareMeters = squareMeters,

Stratum = stratum

});

}

}

}

1. Delete and create the **OwnersController**.
2. Drop the database and add the new migrations with those commands:

PM> drop-database

PM> add-migration Users

PM> update-database

1. Modify the configuration to setup the new functionality:

public void ConfigureServices(IServiceCollection services)

{

services.Configure<CookiePolicyOptions>(options =>

{

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddIdentity<User, IdentityRole>(cfg =>

{

cfg.User.RequireUniqueEmail = true;

cfg.Password.RequireDigit = false;

cfg.Password.RequiredUniqueChars = 0;

cfg.Password.RequireLowercase = false;

cfg.Password.RequireNonAlphanumeric = false;

cfg.Password.RequireUppercase = false;

}).AddEntityFrameworkStores<DataContext>();

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>();

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthentication();

app.UseCookiePolicy();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

## Implementing login and logout in Web

1. Create the **LoginViewModel**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class LoginViewModel

{

[Required]

[EmailAddress]

public string Username { get; set; }

[Required]

[MinLength(6)]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

}

1. Add those methods to interface **IUserHelper**:

Task<SignInResult> LoginAsync(LoginViewModel model);

Task LogoutAsync();

And Implementation **UserHelper**:

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly SignInManager<User> \_signInManager;

public UserHelper(

UserManager<User> userManager,

RoleManager<IdentityRole> roleManager,

SignInManager<User> signInManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

\_signInManager = signInManager;

}

public async Task<SignInResult> LoginAsync(LoginViewModel model)

{

return await \_signInManager.PasswordSignInAsync(

model.Username,

model.Password,

model.RememberMe,

false);

}

public async Task LogoutAsync()

{

await \_signInManager.SignOutAsync();

}

1. Create the **AccountController**:

using Microsoft.AspNetCore.Mvc;

using MyLeasing.Web.Helpers;

using MyLeasing.Web.Models;

using System.Linq;

using System.Threading.Tasks;

namespace MyLeasing.Web.Controllers

{

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

public AccountController(IUserHelper userHelper)

{

\_userHelper = userHelper;

}

public IActionResult Login()

{

if (User.Identity.IsAuthenticated)

{

return RedirectToAction("Index", "Home");

}

return View();

}

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await \_userHelper.LoginAsync(model);

if (result.Succeeded)

{

if (Request.Query.Keys.Contains("ReturnUrl"))

{

return Redirect(Request.Query["ReturnUrl"].First());

}

return RedirectToAction("Index", "Home");

}

}

ModelState.AddModelError(string.Empty, "Failed to login.");

return View(model);

}

public async Task<IActionResult> Logout()

{

await \_userHelper.LogoutAsync();

return RedirectToAction("Index", "Home");

}

}

}

1. Create the view for login:

@model MyLeasing.Web.Models.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<h2>Login</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="Username">Username</label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-warning"></span>

</div>

<script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>

<div class="form-group">

<label asp-for="Password">Password</label>

<input asp-for="Password" type="password" class="form-control" />

<span asp-validation-for="Password" class="text-warning"></span>

</div>

<div class="form-group">

<div class="form-check">

<input asp-for="RememberMe" type="checkbox" class="form-check-input" />

<label asp-for="RememberMe" class="form-check-label">Remember Me?</label>

</div>

<span asp-validation-for="RememberMe" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a asp-action="Register" class="btn btn-primary">Register New User</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Add the controller for the other entities (except **PropertyImage**):
2. Add the annotation authorize to your own controllers, except **AccountController**:

[Authorize(Roles = "Manager")]

1. Modify the menu:

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a></li>

@if (User.Identity.IsAuthenticated && User.IsInRole("Manager"))

{

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown">File<b class="caret"></b></a>

<ul class="dropdown-menu">

<li><a asp-area="" asp-controller="Managers" asp-action="Index">Managers</a></li>

<li><a asp-area="" asp-controller="PropertyTypes" asp-action="Index">Property Types</a></li>

</ul>

</li>

<li><a asp-area="" asp-controller="Owners" asp-action="Index">Owners</a></li>

<li><a asp-area="" asp-controller="Lessees" asp-action="Index">Lessees</a></li>

<li><a asp-area="" asp-controller="Contracts" asp-action="Index">Contracts</a></li>

<li><a asp-area="" asp-controller="Properties" asp-action="Index">Properties</a></li>

}

</ul>

<ul class="nav navbar-nav navbar-right">

@if (User.Identity.IsAuthenticated)

{

<li><a asp-area="" asp-controller="Account" asp-action="ChangeUser">@User.Identity.Name</a></li>

<li><a asp-area="" asp-controller="Account" asp-action="Logout">Logout</a></li>

}

else

{

<li><a asp-area="" asp-controller="Account" asp-action="Login">Login</a></li>

}

</ul>

</div>

1. Test it.

## Create personalized controllers

1. Modify the method **Index** in **OwnersController**:

public IActionResult Index()

{

return View(\_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Properties)

.Include(o => o.Contracts));

}

1. Modify the **Index** view for **OwnersController**:

@model IEnumerable<MyLeasing.Web.Data.Entities.Owner>

@{

ViewData["Title"] = "Index";

}

<h2>Owners</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Document)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.FirstName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.LastName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.PhoneNumber)

</th>

<th>

Properties

</th>

<th>

Contracts

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.User.Document)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.FirstName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.LastName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.PhoneNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Properties.Count)

</td>

<td>

@Html.DisplayFor(modelItem => item.Contracts.Count)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>

</td>

</tr>

}

</tbody>

</table>

1. Test it.
2. Modify the **Details** method on **OwnersController**:

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyType)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyImages)

.Include(o => o.Contracts)

.FirstOrDefaultAsync(m => m.Id == id);

if (owner == null)

{

return NotFound();

}

return View(owner);

}

1. Modify the **Details** view for **OwnersController**:

@model MyLeasing.Web.Data.Entities.Owner

@{

ViewData["Title"] = "Details";

}

<h2>Owner</h2>

<div>

<h4>Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.User.PhoneNumber)

</dd>

<dt>

Properties

</dt>

<dd>

@Html.DisplayFor(model => model.Properties.Count)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="AddProperty" asp-route-id="@Model.Id" class="btn btn-primary">Add Property</a>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

<h4>Properties</h4>

@if (Model.Properties.Count == 0)

{

<h5>Not properties added yet.</h5>

}

else

{

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().PropertyType.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().SquareMeters)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Rooms)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Stratum)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().HasParkingLot)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().IsAvailable)

</th>

<th>

Images

</th>

<th>

Contracts

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Properties)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.SquareMeters)

</td>

<td>

@Html.DisplayFor(modelItem => item.Rooms)

</td>

<td>

@Html.DisplayFor(modelItem => item.Stratum)

</td>

<td>

@Html.DisplayFor(modelItem => item.HasParkingLot)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsAvailable)

</td>

<td>

@Html.DisplayFor(modelItem => item.PropertyImages.Count)

</td>

<td>

@Html.DisplayFor(modelItem => item.Contracts.Count)

</td>

<td>

<a asp-action="EditProperty" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="DetailsProperty" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<a asp-action="DeleteProperty" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>

</td>

</tr>

}

</tbody>

</table>

}

1. Test it.
2. Modify the **AddUserViewModel** class:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class AddUserViewModel

{

[Display(Name = "Email")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[EmailAddress]

public string Username { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

[Display(Name = "Phone Number")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string PhoneNumber { get; set; }

[Display(Name = "Password")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

public string Password { get; set; }

[Display(Name = "Password Confirm")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

[Compare("Password")]

public string PasswordConfirm { get; set; }

}

}

1. Modify the **Create** view for **OwnersController**:

@model MyLeasing.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Owner</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="Username" class="control-label"></label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Document" class="control-label"></label>

<input asp-for="Document" class="form-control" />

<span asp-validation-for="Document" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="FirstName" class="control-label"></label>

<input asp-for="FirstName" class="form-control" />

<span asp-validation-for="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="LastName" class="control-label"></label>

<input asp-for="LastName" class="form-control" />

<span asp-validation-for="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber" class="control-label"></label>

<input asp-for="PhoneNumber" class="form-control" />

<span asp-validation-for="PhoneNumber" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PasswordConfirm" class="control-label"></label>

<input asp-for="PasswordConfirm" class="form-control" />

<span asp-validation-for="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Modify the **Create** post method for **OwnersController**:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await AddUser(view);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

var owner = new Owner

{

Properties = new List<Property>(),

Contracts = new List<Contract>(),

User = user,

};

\_dataContext.Owners.Add(owner);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(view);

}

private async Task<User> AddUser(AddUserViewModel view)

{

var user = new User

{

Address = view.Address,

Document = view.Document,

Email = view.Username,

FirstName = view.FirstName,

LastName = view.LastName,

PhoneNumber = view.PhoneNumber,

UserName = view.Username

};

var result = await \_userHelper.AddUserAsync(user, view.Password);

if (result != IdentityResult.Success)

{

return null;

}

var newUser = await \_userHelper.GetUserByEmailAsync(view.Username);

await \_userHelper.AddUserToRoleAsync(newUser, "Owner");

return newUser;

}

1. Test it.
2. Add the model **PropertyViewModel**:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Models

{

public class PropertyViewModel : Property

{

public int OwnerId { get; set; }

[Required(ErrorMessage = "The field {0} is mandatory.")]

[Display(Name = "Property Type")]

[Range(1, int.MaxValue, ErrorMessage = "You must select a property type.")]

public int PropertyTypeId { get; set; }

public IEnumerable<SelectListItem> PropertyTypes { get; set; }

}

}

1. Add those methods to **OwnersController**:

public async Task<IActionResult> AddProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners.FindAsync(id.Value);

if (owner == null)

{

return NotFound();

}

var view = new PropertyViewModel

{

OwnerId = owner.Id,

PropertyTypes = GetComboPropertyTypes()

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> AddProperty(PropertyViewModel view)

{

if (ModelState.IsValid)

{

var property = await ToPropertyAsync(view);

\_dataContext.Properties.Add(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(Details)}/{view.OwnerId}");

}

return View(view);

}

private async Task<Property> ToPropertyAsync(PropertyViewModel view)

{

return new Property

{

Address = view.Address,

HasParkingLot = view.HasParkingLot,

IsAvailable = view.IsAvailable,

Neighborhood = view.Neighborhood,

Price = view.Price,

Rooms = view.Rooms,

SquareMeters = view.SquareMeters,

Stratum = view.Stratum,

Owner = await \_dataContext.Owners.FindAsync(view.OwnerId),

PropertyType = await \_dataContext.PropertyTypes.FindAsync(view.PropertyTypeId),

Remarks = view.Remarks

};

}

private IEnumerable<SelectListItem> GetComboPropertyTypes()

{

var list = \_dataContext.PropertyTypes.Select(p => new SelectListItem

{

Text = p.Name,

Value = p.Id.ToString()

}).OrderBy(p => p.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a property type...)",

Value = "0"

});

return list;

}

1. Add the View **AddProperty** to **OwnerController**:

@model MyLeasing.Web.Models.PropertyViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddProperty">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="OwnerId" />

<div class="form-group">

<label asp-for="PropertyTypeId" class="control-label"></label>

<select asp-for="PropertyTypeId" asp-items="Model.PropertyTypes" class="form-control"></select>

<span asp-validation-for="PropertyTypeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Neighborhood" class="control-label"></label>

<input asp-for="Neighborhood" class="form-control" />

<span asp-validation-for="Neighborhood" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="SquareMeters" class="control-label"></label>

<input asp-for="SquareMeters" class="form-control" />

<span asp-validation-for="SquareMeters" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Rooms" class="control-label"></label>

<input asp-for="Rooms" class="form-control" />

<span asp-validation-for="Rooms" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Stratum" class="control-label"></label>

<input asp-for="Stratum" class="form-control" />

<span asp-validation-for="Stratum" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="HasParkingLot" /> @Html.DisplayNameFor(model => model.HasParkingLot)

</label>

</div>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailable" /> @Html.DisplayNameFor(model => model.IsAvailable)

</label>

</div>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="Details" asp-route-id="@Model.OwnerId" class="btn btn-success">Back to Owner</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add those methods to **OwnerController**:

public async Task<IActionResult> EditProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(p => p.Owner)

.Include(p => p.PropertyType)

.FirstOrDefaultAsync(p => p.Id == id.Value);

if (property == null)

{

return NotFound();

}

var view = ToPropertyViewModel(property);

return View(view);

}

private PropertyViewModel ToPropertyViewModel(Property property)

{

return new PropertyViewModel

{

Address = property.Address,

HasParkingLot = property.HasParkingLot,

Id = property.Id,

IsAvailable = property.IsAvailable,

Neighborhood = property.Neighborhood,

Price = property.Price,

Rooms = property.Rooms,

SquareMeters = property.SquareMeters,

Stratum = property.Stratum,

Owner = property.Owner,

OwnerId = property.Owner.Id,

PropertyType = property.PropertyType,

PropertyTypeId = property.PropertyType.Id,

PropertyTypes = GetComboPropertyTypes(),

Remarks = property.Remarks,

};

}

[HttpPost]

public async Task<IActionResult> EditProperty(PropertyViewModel view)

{

if (ModelState.IsValid)

{

var property = await ToPropertyAsync(view);

\_dataContext.Properties.Update(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(Details)}/{view.OwnerId}");

}

return View(view);

}

1. Add the **EditProperty** View to **OwnersController**:

@model MyLeasing.Web.Models.PropertyViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="EditProperty">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="OwnerId" />

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="PropertyTypeId" class="control-label"></label>

<select asp-for="PropertyTypeId" asp-items="Model.PropertyTypes" class="form-control"></select>

<span asp-validation-for="PropertyTypeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Neighborhood" class="control-label"></label>

<input asp-for="Neighborhood" class="form-control" />

<span asp-validation-for="Neighborhood" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="SquareMeters" class="control-label"></label>

<input asp-for="SquareMeters" class="form-control" />

<span asp-validation-for="SquareMeters" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Rooms" class="control-label"></label>

<input asp-for="Rooms" class="form-control" />

<span asp-validation-for="Rooms" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Stratum" class="control-label"></label>

<input asp-for="Stratum" class="form-control" />

<span asp-validation-for="Stratum" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="HasParkingLot" /> @Html.DisplayNameFor(model => model.HasParkingLot)

</label>

</div>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailable" /> @Html.DisplayNameFor(model => model.IsAvailable)

</label>

</div>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Details" asp-route-id="@Model.OwnerId" class="btn btn-success">Back to Owner</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add the method **DetailsProperty** to **OwnerController**:

public async Task<IActionResult> DetailsProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(o => o.Owner)

.ThenInclude(o => o.User)

.Include(o => o.Contracts)

.ThenInclude(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(o => o.PropertyType)

.Include(p => p.PropertyImages)

.FirstOrDefaultAsync(m => m.Id == id);

if (property == null)

{

return NotFound();

}

return View(property);

}

1. Add the view **DetailsProperty** to **OwnersController**:

@model MyLeasing.Web.Data.Entities.Property

@{

ViewData["Title"] = "Details";

}

<h2>Property</h2>

<div class="row">

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

<div>

<h4>Owner</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.PhoneNumber)

</dd>

</dl>

</div>

</div>

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

<div>

<h4>Property Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.SquareMeters)

</dt>

<dd>

@Html.DisplayFor(model => model.SquareMeters)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Rooms)

</dt>

<dd>

@Html.DisplayFor(model => model.Rooms)

</dd>

<dt>

@Html.DisplayNameFor(model => model.HasParkingLot)

</dt>

<dd>

@Html.DisplayFor(model => model.HasParkingLot)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsAvailable)

</dt>

<dd>

@Html.DisplayFor(model => model.IsAvailable)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

</dl>

</div>

</div>

</div>

<div>

<a asp-action="EditProperty" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="AddImage" asp-route-id="@Model.Id" class="btn btn-primary">Add Image</a>

<a asp-action="AddContract" asp-route-id="@Model.Id" class="btn btn-default">Add Contract</a>

<a asp-action="Details" asp-route-id="@Model.Owner.Id" class="btn btn-success">Back to Owner</a>

</div>

<div class="row">

<div class="col-md-3">

<div>

<h4>Images</h4>

@if (Model.PropertyImages.Count == 0)

{

<h5>Not images added yet.</h5>

}

else

{

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.PropertyImages.FirstOrDefault().ImageUrl)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.PropertyImages)

{

<tr>

<td>

@if (!string.IsNullOrEmpty(item.ImageUrl))

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:200px;height:200px;max-width: 100%; height: auto;" />

}

</td>

<td>

<a asp-action="DeleteImage" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>

</td>

</tr>

}

</tbody>

</table>

}

</div>

</div>

<div class="col-md-9">

<div>

<h4>Contracts</h4>

@if (Model.Contracts.Count == 0)

{

<h5>Not contracts added yet.</h5>

}

else

{

<table class="table">

<thead>

<tr>

<th>

Lessee

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Remarks)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().StartDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().EndDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().IsActive)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Contracts)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Lessee.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Remarks)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsActive)

</td>

<td>

<a asp-action="EditContract" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="DetailsContract" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<a asp-action="DeleteContract" asp-route-id="@item.Id" class="btn btn-danger">Delete</a>

</td>

</tr>

}

</tbody>

</table>

}

</div>

</div>

</div>

1. Test it.
2. Modify the entity **PropertyImage**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Data.Entities

{

public class PropertyImage

{

public int Id { get; set; }

[Display(Name = "Image")]

public string ImageUrl { get; set; }

public Property Property { get; set; }

// TODO: Change the path when publish

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? "https://TBD.azurewebsites.net/images/Properties/noImage.png"

: $"https://TBD.azurewebsites.net{ImageUrl.Substring(1)}";

}

}

1. Add the view model **PropertyImageViewModel**:

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Http;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Models

{

public class PropertyImageViewModel : PropertyImage

{

[Display(Name = "Image")]

public IFormFile ImageFile { get; set; }

}

}

1. Add the folder **Properties** inside **wwwroot/images**.
2. Add the **IImageHelper**:

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

namespace MyLeasing.Web.Helpers

{

public interface IImageHelper

{

Task<string> UploadImageAsync(IFormFile imageFile);

}

}

1. Add the **ImageHelper**:

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

namespace MyLeasing.Web.Helpers

{

public class ImageHelper : IImageHelper

{

public async Task<string> UploadImageAsync(IFormFile imageFile)

{

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

var file = $"{guid}.jpg";

var path = Path.Combine(

Directory.GetCurrentDirectory(),

"wwwroot\\images\\Properties",

file);

using (var stream = new FileStream(path, FileMode.Create))

{

await imageFile.CopyToAsync(stream);

}

return $"~/images/Properties/{file}";

}

}

}

1. Modify the **OwnersController**:

public OwnersController(

DataContext dataContext,

IUserHelper userHelper,

ICombosHelper combosHelper,

IConverterHelper converterHelper,

IImageHelper imageHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_converterHelper = converterHelper;

\_imageHelper = imageHelper;

}

...

public async Task<IActionResult> AddImage(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties.FindAsync(id.Value);

if (property == null)

{

return NotFound();

}

var model = new PropertyImageViewModel

{

Id = property.Id

};

return View(model);

}

[HttpPost]

public async Task<IActionResult> AddImage(PropertyImageViewModel model)

{

if (ModelState.IsValid)

{

var path = string.Empty;

if (model.ImageFile != null)

{

path = await \_imageHelper.UploadImageAsync(model.ImageFile);

}

var propertyImage = new PropertyImage

{

ImageUrl = path,

Property = await \_dataContext.Properties.FindAsync(model.Id)

};

\_dataContext.PropertyImages.Add(propertyImage);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsProperty)}/{model.Id}");

}

return View(model);

}

1. Add the view **AddImage** to **OwnersController**:

@model MyLeasing.Web.Models.PropertyImageViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Add Image</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddImage" enctype="multipart/form-data">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="ImageFile" class="control-label"></label>

<input asp-for="ImageFile" class="form-control" type="file" />

<span asp-validation-for="ImageFile" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="DetailsProperty" asp-route-id="@Model.Id" class="btn btn-success">Back to Property</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add the view model **ContractViewModel**:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Models

{

public class ContractViewModel : Contract

{

public int OwnerId { get; set; }

public int PropertyId { get; set; }

[Required(ErrorMessage = "The field {0} is mandatory.")]

[Display(Name = "Lessee")]

[Range(1, int.MaxValue, ErrorMessage = "You must select a lessess.")]

public int LesseeId { get; set; }

public IEnumerable<SelectListItem> Lessees { get; set; }

}

}

1. Add the methods **AddContract** to **OwnersController**:

public async Task<IActionResult> AddContract(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(p => p.Owner)

.FirstOrDefaultAsync(p => p.Id == id.Value);

if (property == null)

{

return NotFound();

}

var view = new ContractViewModel

{

OwnerId = property.Owner.Id,

PropertyId = property.Id,

Lessees = GetComboLessees(),

Price = property.Price,

StartDate = DateTime.Today,

EndDate = DateTime.Today.AddYears(1)

};

return View(view);

}

private IEnumerable<SelectListItem> GetComboLessees()

{

var list = \_dataContext.Lessees.Include(l => l.User).Select(p => new SelectListItem

{

Text = p.User.FullNameWithDocument,

Value = p.Id.ToString()

}).OrderBy(p => p.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a lessee...)",

Value = "0"

});

return list;

}

[HttpPost]

public async Task<IActionResult> AddContract(ContractViewModel view)

{

if (ModelState.IsValid)

{

var contract = await ToContractAsync(view);

\_dataContext.Contracts.Add(contract);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsProperty)}/{view.OwnerId}");

}

return View(view);

}

private async Task<Contract> ToContractAsync(ContractViewModel view)

{

return new Contract

{

EndDate = view.EndDate,

IsActive = view.IsActive,

Lessee = await \_dataContext.Lessees.FindAsync(view.LesseeId),

Owner = await \_dataContext.Owners.FindAsync(view.OwnerId),

Price = view.Price,

Property = await \_dataContext.Properties.FindAsync(view.PropertyId),

Remarks = view.Remarks,

StartDate = view.StartDate,

Id = view.Id

};

}

1. Add the view **AddContract** to **OwnersController**:

@model MyLeasing.Web.Models.ContractViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Contract</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddContract">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="OwnerId" />

<input type="hidden" asp-for="PropertyId" />

<div class="form-group">

<label asp-for="LesseeId" class="control-label"></label>

<select asp-for="LesseeId" asp-items="Model.Lessees" class="form-control"></select>

<span asp-validation-for="LesseeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="StartDate" class="control-label"></label>

<input asp-for="StartDate" class="form-control" />

<span asp-validation-for="StartDate" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="EndDate" class="control-label"></label>

<input asp-for="EndDate" class="form-control" />

<span asp-validation-for="EndDate" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsActive" /> @Html.DisplayNameFor(model => model.IsActive)

</label>

</div>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="DetailsProperty" asp-route-id="@Model.PropertyId" class="btn btn-success">Back to Property</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add this method to **ICombosHelper**:

IEnumerable<SelectListItem> GetComboLessees();

1. Add this method to **CombosHelper**:

public IEnumerable<SelectListItem> GetComboLessees()

{

var list = \_dataContext.Lessees.Include(l => l.User).Select(p => new SelectListItem

{

Text = p.User.FullNameWithDocument,

Value = p.Id.ToString()

}).OrderBy(p => p.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a lessee...)",

Value = "0"

});

return list;

}

1. Add those methods to **IConverterHelper**:

Task<Contract> ToContractAsync(ContractViewModel model, bool isNew);

ContractViewModel ToContractViewModel(Contract contract);

1. Add those methods to **ConverterHelper**:

public async Task<Contract> ToContractAsync(ContractViewModel model, bool isNew)

{

return new Contract

{

EndDate = model.EndDate,

Id = isNew ? 0 : model.Id,

IsActive = model.IsActive,

Lessee = await \_dataContext.Lessees.FindAsync(model.LesseeId),

Owner = await \_dataContext.Owners.FindAsync(model.OwnerId),

Price = model.Price,

Property = await \_dataContext.Properties.FindAsync(model.PropertyId),

Remarks = model.Remarks,

StartDate = model.StartDate

};

}

public ContractViewModel ToContractViewModel(Contract contract)

{

return new ContractViewModel

{

EndDate = contract.EndDate,

IsActive = contract.IsActive,

LesseeId = contract.Lessee.Id,

OwnerId = contract.Owner.Id,

Price = contract.Price,

Remarks = contract.Remarks,

StartDate = contract.StartDate,

Id = contract.Id,

Lessees = \_combosHelper.GetComboLessees(),

PropertyId = contract.Property.Id

};

}

1. Add the methods **EditContract** to **OwnersController**:

public async Task<IActionResult> EditContract(int? id)

{

if (id == null)

{

return NotFound();

}

var contract = await \_dataContext.Contracts

.Include(p => p.Owner)

.Include(p => p.Lessee)

.Include(p => p.Property)

.FirstOrDefaultAsync(p => p.Id == id.Value);

if (contract == null)

{

return NotFound();

}

return View(\_converterHelper.ToContractViewModel(contract));

}

[HttpPost]

public async Task<IActionResult> EditContract(ContractViewModel view)

{

if (ModelState.IsValid)

{

var contract = await \_converterHelper.ToContractAsync(view, false);

\_dataContext.Contracts.Update(contract);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsProperty)}/{view.OwnerId}");

}

return View(view);

}

1. Add the view **EditContract** to **OwnersController**:

@model MyLeasing.Web.Models.ContractViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Contract</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="EditContract">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<input type="hidden" asp-for="OwnerId" />

<input type="hidden" asp-for="PropertyId" />

<div class="form-group">

<label asp-for="LesseeId" class="control-label"></label>

<select asp-for="LesseeId" asp-items="Model.Lessees" class="form-control"></select>

<span asp-validation-for="LesseeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="StartDate" class="control-label"></label>

<input asp-for="StartDate" class="form-control" />

<span asp-validation-for="StartDate" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="EndDate" class="control-label"></label>

<input asp-for="EndDate" class="form-control" />

<span asp-validation-for="EndDate" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsActive" /> @Html.DisplayNameFor(model => model.IsActive)

</label>

</div>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="DetailsProperty" asp-route-id="@Model.PropertyId" class="btn btn-success">Back to Property</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Modify the view **DetailsProperty** in **OwnersController**:

…

<td>

<button data-id="@item.Id" class="btn btn-danger deleteImage" data-toggle="modal" data-target="#deleteDialog">Delete</button>

</td>

…

<td>

<a asp-action="EditContract" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="DetailsContract" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<button data-id="@item.Id" class="btn btn-danger deleteContract" data-toggle="modal" data-target="#deleteDialog">Delete</button>

</td>

…

</div>

<!--Delete Item-->

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Do you want to delete the record?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script type="text/javascript">

$(document).ready(function () {

// Delete item

var item\_to\_delete;

var action\_to\_delete;

$('.deleteImage').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

action\_to\_delete = 1;

});

$('.deleteContract').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

action\_to\_delete = 2;

});

$("#btnYesDelete").click(function () {

if (action\_to\_delete == 1) {

window.location.href = '/Owners/DeleteImage/' + item\_to\_delete;

} else {

window.location.href = '/Owners/DeleteContract/' + item\_to\_delete;

}

});

});

</script>

}

1. Add those methods in **OwnersController**:

public async Task<IActionResult> DeleteImage(int? id)

{

if (id == null)

{

return NotFound();

}

var propertyImage = await \_dataContext.PropertyImages

.Include(pi => pi.Property)

.FirstOrDefaultAsync(pi => pi.Id == id.Value);

if (propertyImage == null)

{

return NotFound();

}

\_dataContext.PropertyImages.Remove(propertyImage);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsProperty)}/{propertyImage.Property.Id}");

}

public async Task<IActionResult> DeleteContract(int? id)

{

if (id == null)

{

return NotFound();

}

var contract = await \_dataContext.Contracts

.Include(c => c.Property)

.FirstOrDefaultAsync(c => c.Id == id.Value);

if (contract == null)

{

return NotFound();

}

\_dataContext.Contracts.Remove(contract);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsProperty)}/{contract.Property.Id}");

}

1. Test it.
2. Add the method **DeleteUserAsync** to **IUserHelper** interface:

Task<bool> DeleteUserAsync(string email);

1. Add the method **DeleteUserAsync** to **UserHelper** class:

public async Task<bool> DeleteUserAsync(string email)

{

var user = await GetUserByEmailAsync(email);

if (user == null)

{

return true;

}

var response = await \_userManager.DeleteAsync(user);

return response.Succeeded;

}

1. Modify the **Index** view in **OwnersController**:

<td>

<a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog">Delete</button>

</td>

</tr>

}

</tbody>

</table>

<!--Delete Item-->

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Do you want to delete the record?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script type="text/javascript">

$(document).ready(function () {

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Owners/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the **Delete** method in **OwnersController** and delete the HttpPost:

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners

.Include(o => o.User)

.FirstOrDefaultAsync(m => m.Id == id);

if (owner == null)

{

return NotFound();

}

\_dataContext.Owners.Remove(owner);

await \_dataContext.SaveChangesAsync();

await \_userHelper.DeleteUserAsync(owner.User.Email);

return RedirectToAction(nameof(Index));

}

1. Test it.
2. Add the method **UpdateUserAsync** to **IUserHelper** interface:

Task<IdentityResult> UpdateUserAsync(User user);

1. Add the method **UpdateUserAsync** to **UserHelper** class:

public async Task<IdentityResult> UpdateUserAsync(User user)

{

return await \_userManager.UpdateAsync(user);

}

1. Add the **EditUserViewModel** class:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class EditUserViewModel

{

public int Id { get; set; }

[Display(Name = "Document")]

[MaxLength(20, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string Document { get; set; }

[Display(Name = "First Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string FirstName { get; set; }

[Display(Name = "Last Name")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

public string LastName { get; set; }

[MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string Address { get; set; }

[Display(Name = "Phone Number")]

[MaxLength(50, ErrorMessage = "The {0} field can not have more than {1} characters.")]

public string PhoneNumber { get; set; }

}

}

1. Modify the **Edit** methods in **OwnersController**:

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.Id == id.Value);

if (owner == null)

{

return NotFound();

}

var view = new EditUserViewModel

{

Address = owner.User.Address,

Document = owner.User.Document,

FirstName = owner.User.FirstName,

Id = owner.Id,

LastName = owner.User.LastName,

PhoneNumber = owner.User.PhoneNumber

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditUserViewModel view)

{

if (ModelState.IsValid)

{

var owner = await \_dataContext.Owners

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.Id == view.Id);

owner.User.Document = view.Document;

owner.User.FirstName = view.FirstName;

owner.User.LastName = view.LastName;

owner.User.Address = view.Address;

owner.User.PhoneNumber = view.PhoneNumber;

await \_userHelper.UpdateUserAsync(owner.User);

return RedirectToAction(nameof(Index));

}

return View(view);

}

1. Modify the **Edit** view in **OwnersController**:

@model MyLeasing.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Owner</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Document" class="control-label"></label>

<input asp-for="Document" class="form-control" />

<span asp-validation-for="Document" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="FirstName" class="control-label"></label>

<input asp-for="FirstName" class="form-control" />

<span asp-validation-for="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="LastName" class="control-label"></label>

<input asp-for="LastName" class="form-control" />

<span asp-validation-for="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber" class="control-label"></label>

<input asp-for="PhoneNumber" class="form-control" />

<span asp-validation-for="PhoneNumber" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.
2. Add the method **DeleteProperty** in **OwnersController**:

public async Task<IActionResult> DeleteProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(p => p.Owner)

.FirstOrDefaultAsync(pi => pi.Id == id.Value);

if (property == null)

{

return NotFound();

}

\_dataContext.Properties.Remove(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(Details)}/{property.Owner.Id}");

}

1. Modify the **Details** view in **OwnersController**:

…

<td>

<a asp-action="EditProperty" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="DetailsProperty" asp-route-id="@item.Id" class="btn btn-info">Details</a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog">Delete</button>

</td>

…

<!--Delete Item-->

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Do you want to delete the record?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script type="text/javascript">

$(document).ready(function () {

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Owners/DeleteProperty/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Add the method **DetailsContract** in **OwnersController**:

public async Task<IActionResult> DetailsContract(int? id)

{

if (id == null)

{

return NotFound();

}

var contract = await \_dataContext.Contracts

.Include(c => c.Owner)

.ThenInclude(o => o.User)

.Include(c => c.Lessee)

.ThenInclude(o => o.User)

.Include(c => c.Property)

.ThenInclude(p => p.PropertyType)

.FirstOrDefaultAsync(pi => pi.Id == id.Value);

if (contract == null)

{

return NotFound();

}

return View(contract);

}

1. Add the **DetailsContract** view in **OwnersController**:

@model MyLeasing.Web.Data.Entities.Contract

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Contract</h4>

<hr />

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FullNameWithDocument)

</dd>

<dt>

Lessee

</dt>

<dd>

@Html.DisplayFor(model => model.Lessee.User.FullNameWithDocument)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.PropertyType.Name)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.PropertyType.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.StartDate)

</dt>

<dd>

@Html.DisplayFor(model => model.StartDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.EndDate)

</dt>

<dd>

@Html.DisplayFor(model => model.EndDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsActive)

</dt>

<dd>

@Html.DisplayFor(model => model.IsActive)

</dd>

</dl>

</div>

<div>

<a asp-action="EditContract" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="DetailsProperty" asp-route-id="@Model.Property.Id" class="btn btn-success">Back to Property</a>

</div>

1. Test it.
2. Modify the **ContractsController**.

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

namespace MyLeasing.Web.Controllers

{

[Authorize(Roles = "Manager")]

public class ContractsController : Controller

{

private readonly DataContext \_dataContext;

public ContractsController(DataContext dataContext)

{

\_dataContext = dataContext;

}

public IActionResult Index()

{

return View(\_dataContext.Contracts

.Include(c => c.Owner)

.ThenInclude(o => o.User)

.Include(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(c => c.Property)

.ThenInclude(p => p.PropertyType));

}

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var contract = await \_dataContext.Contracts

.Include(c => c.Owner)

.ThenInclude(o => o.User)

.Include(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(c => c.Property)

.ThenInclude(p => p.PropertyType)

.FirstOrDefaultAsync(m => m.Id == id);

if (contract == null)

{

return NotFound();

}

return View(contract);

}

}

}

1. Modify the view **Index** for **ContractsController**.

@model IEnumerable<MyLeasing.Web.Data.Entities.Contract>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Contracts</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

Owner

</th>

<th>

Lessee

</th>

<th>

Property Type

</th>

<th>

@Html.DisplayNameFor(model => model.Property.Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Property.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.StartDate)

</th>

<th>

@Html.DisplayNameFor(model => model.EndDate)

</th>

<th>

@Html.DisplayNameFor(model => model.IsActive)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Owner.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Lessee.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsActive)

</td>

<td>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Modify the view **Details** for **ContractsController**.

@model MyLeasing.Web.Data.Entities.Contract

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Contract</h4>

<hr />

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FullNameWithDocument)

</dd>

<dt>

Lessee

</dt>

<dd>

@Html.DisplayFor(model => model.Lessee.User.FullNameWithDocument)

</dd>

<dt>

Property Type

</dt>

<dd>

@Html.DisplayFor(model => model.Property.PropertyType.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.StartDate)

</dt>

<dd>

@Html.DisplayFor(model => model.StartDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.EndDate)

</dt>

<dd>

@Html.DisplayFor(model => model.EndDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsActive)

</dt>

<dd>

@Html.DisplayFor(model => model.IsActive)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

</dl>

</div>

<div>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

1. Delete the methods and views **Create, Edit** and **Delete** for **ContractsController**.
2. Test it.

## Add API

1. Add the **EmailRequest** class:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Common.Models

{

public class EmailRequest

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

1. Add the **LesseeResponse** class:

namespace MyLeasing.Common.Models

{

public class LesseeResponse

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Document { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

}

}

1. Add the **ContractResponse** class:

using System;

namespace MyLeasing.Common.Models

{

public class ContractResponse

{

public int Id { get; set; }

public string Remarks { get; set; }

public decimal Price { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public bool IsActive { get; set; }

public LesseeResponse Lessee { get; set; }

public DateTime StartDateLocal => StartDate.ToLocalTime();

public DateTime EndDateLocal => EndDate.ToLocalTime();

}

}

1. Add the **PropertyImageResponse** class:

namespace MyLeasing.Common.Models

{

public class PropertyImageResponse

{

public int Id { get; set; }

public string ImageUrl { get; set; }

}

}

1. Add the **PropertyResponse** class:

using System.Collections.Generic;

namespace MyLeasing.Common.Models

{

public class PropertyResponse

{

public int Id { get; set; }

public string Neighborhood { get; set; }

public string Address { get; set; }

public decimal Price { get; set; }

public int SquareMeters { get; set; }

public int Rooms { get; set; }

public int Stratum { get; set; }

public bool HasParkingLot { get; set; }

public bool IsAvailable { get; set; }

public string Remarks { get; set; }

public string PropertyType { get; set; }

public ICollection<PropertyImageResponse> PropertyImages { get; set; }

public ICollection<ContractResponse> Contracts { get; set; }

}

}

1. Add the **OwnerResponse** class:

using System.Collections.Generic;

namespace MyLeasing.Common.Models

{

public class OwnerResponse

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Document { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

public ICollection<PropertyResponse> Properties { get; set; }

}

}

1. Create the API controller **OwnersController**:

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Common.Models;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

public class OwnersController : ControllerBase

{

private readonly DataContext \_dataContext;

public OwnersController(DataContext dataContext)

{

\_dataContext = dataContext;

}

[HttpPost]

[Route("GetOwnerByEmail")]

public async Task<IActionResult> GetOwnerByEmailAsync(EmailRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest();

}

var owner = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyType)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyImages)

.Include(o => o.Contracts)

.ThenInclude(c => c.Lessee)

.ThenInclude(l => l.User)

.FirstOrDefaultAsync(o => o.User.Email.ToLower() == request.Email.ToLower());

if (owner == null)

{

return NotFound();

}

var response = new OwnerResponse

{

Id = owner.Id,

FirstName = owner.User.FirstName,

LastName = owner.User.LastName,

Address = owner.User.Address,

Document = owner.User.Document,

Email = owner.User.Email,

PhoneNumber = owner.User.PhoneNumber,

Properties = owner.Properties?.Select(p => new PropertyResponse

{

Address = p.Address,

Contracts = p.Contracts?.Select(c => new ContractResponse

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Lessee = ToLessesResponse(c.Lessee),

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList(),

HasParkingLot = p.HasParkingLot,

Id = p.Id,

IsAvailable = p.IsAvailable,

Neighborhood = p.Neighborhood,

Price = p.Price,

PropertyImages = p.PropertyImages?.Select(pi => new PropertyImageResponse

{

Id = pi.Id,

ImageUrl = pi.ImageFullPath

}).ToList(),

PropertyType = p.PropertyType.Name,

Remarks = p.Remarks,

Rooms = p.Rooms,

SquareMeters = p.SquareMeters,

Stratum = p.Stratum

}).ToList()

};

return Ok(response);

}

private LesseeResponse ToLessesResponse(Lessee lessee)

{

return new LesseeResponse

{

Id = lessee.Id,

Address = lessee.User.Address,

Document = lessee.User.Document,

Email = lessee.User.Email,

FirstName = lessee.User.FirstName,

LastName = lessee.User.LastName,

PhoneNumber = lessee.User.PhoneNumber

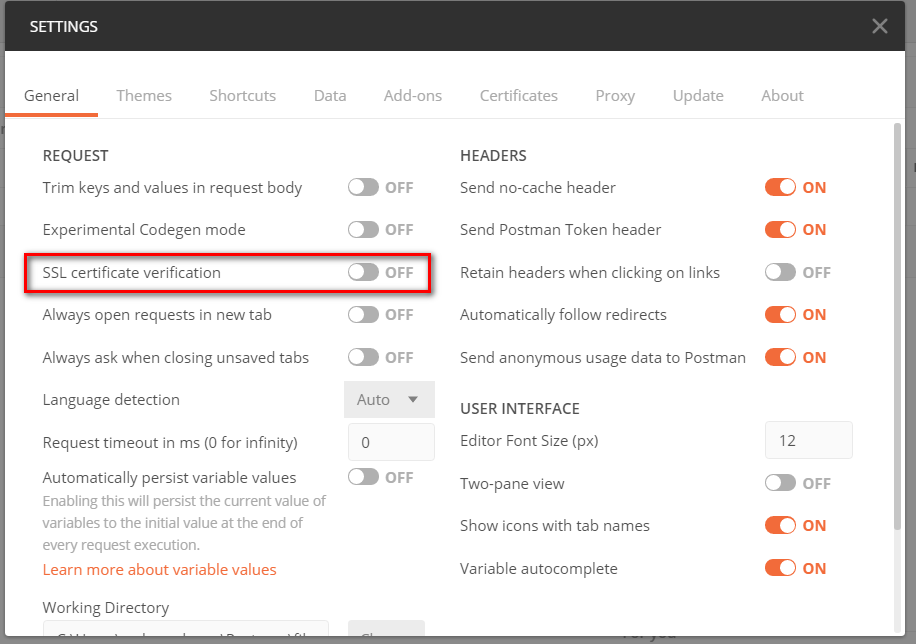
};

}

}

}

1. Ensure that this property on postmant configuration is off:



1. Test it on postmant.

## Add Tokens Generation

1. Add those values in json configuration file:

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=MyLeasing;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Tokens": {

"Key": "asdfghjikbnvcgfdsrtfyhgcvgfxdgc",

"Issuer": "localhost",

"Audience": "users"

}

}

1. Add this method to **IUserHelper**:

Task<SignInResult> ValidatePasswordAsync(User user, string password);

And the implementation:

public async Task<SignInResult> ValidatePasswordAsync(User user, string password)

{

return await \_signInManager.CheckPasswordSignInAsync(

user,

password,

false);

}

1. Modify the accounts controller constructor:

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

public AccountController(

IUserHelper userHelper,

IConfiguration configuration)

{

\_userHelper = userHelper;

\_configuration = configuration;

}

1. Add the method to generate the token in the account controller:

[HttpPost]

public async Task<IActionResult> CreateToken([FromBody] LoginViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(model.Username);

if (user != null)

{

var result = await \_userHelper.ValidatePasswordAsync(

user,

model.Password);

if (result.Succeeded)

{

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.Email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Tokens:Key"]));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

\_configuration["Tokens:Issuer"],

\_configuration["Tokens:Audience"],

claims,

expires: DateTime.UtcNow.AddDays(15),

signingCredentials: credentials);

var results = new

{

token = new JwtSecurityTokenHandler().WriteToken(token),

expiration = token.ValidTo

};

return Created(string.Empty, results);

}

}

}

return BadRequest();

}

1. Add the authorization annotation to API **OwnersController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

1. Add the new configuration for validate the tokens in **Startup** class:

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

});

services.AddAuthentication()

.AddCookie()

.AddJwtBearer(cfg =>

{

cfg.TokenValidationParameters = new TokenValidationParameters

{

ValidIssuer = Configuration["Tokens:Issuer"],

ValidAudience = Configuration["Tokens:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Tokens:Key"]))

};

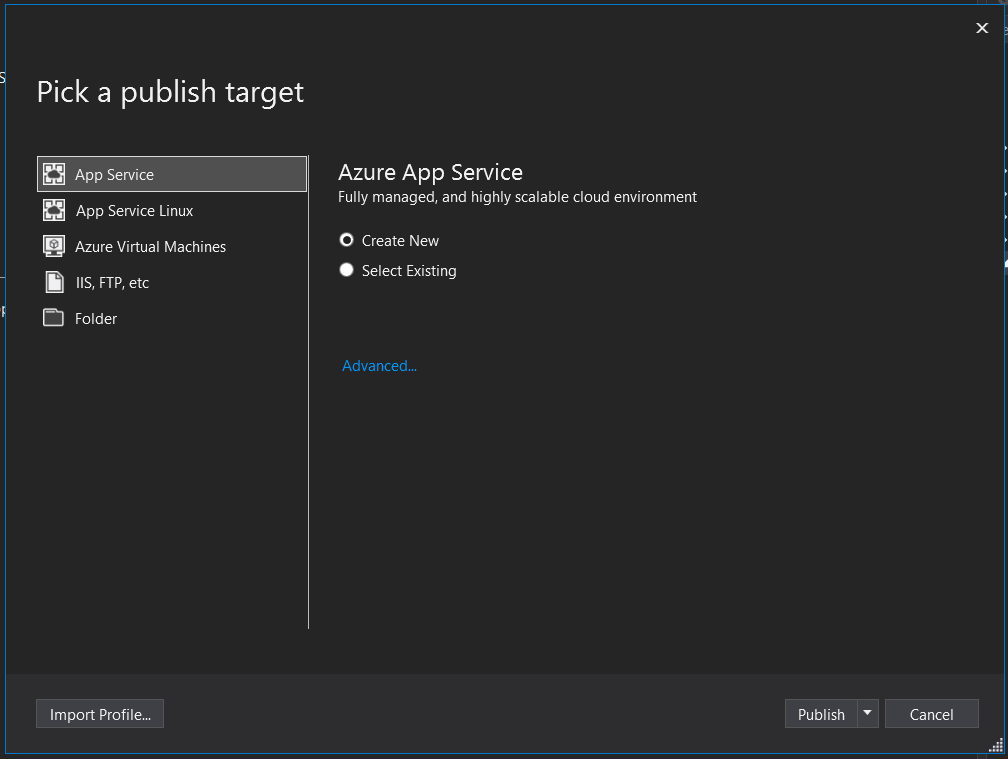
});

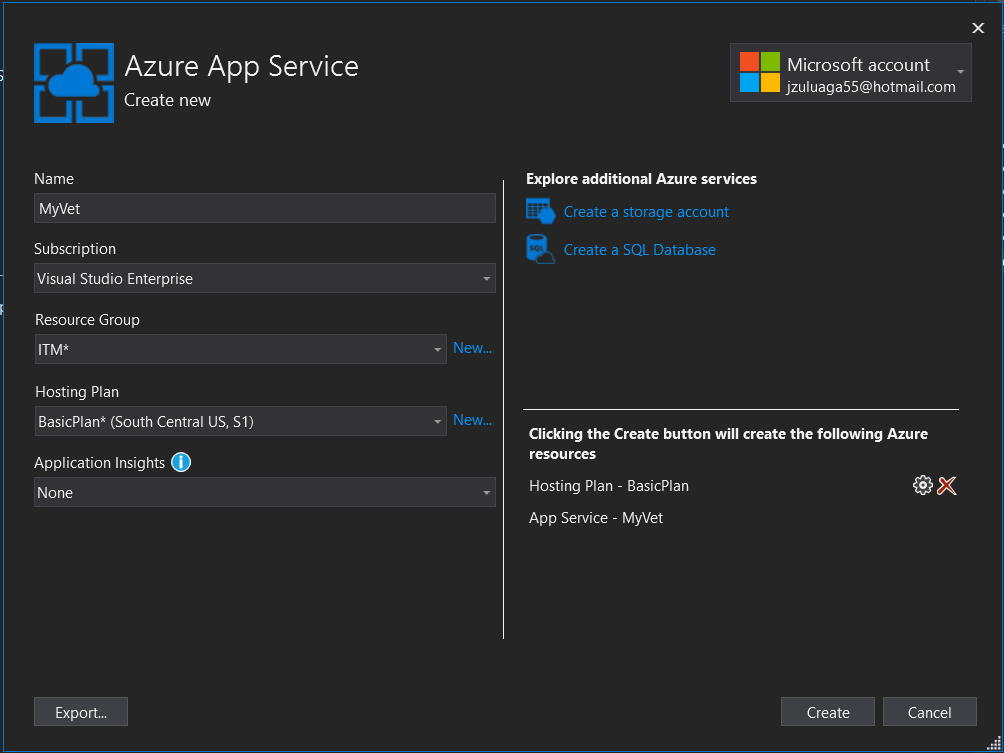
services.AddTransient<SeedDb>();

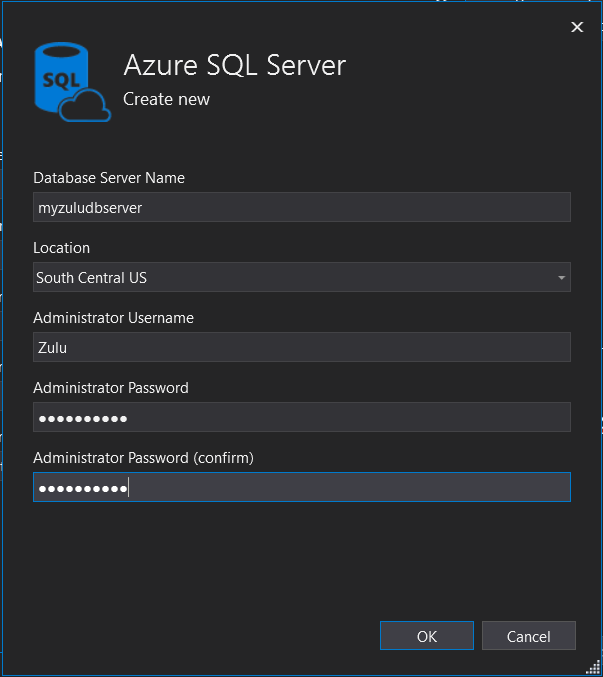
services.AddScoped<IUserHelper, UserHelper>();

1. Test it.

## Publish on Azure









Then, fix the correct route for images:

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? "https://myleasing.azurewebsites.net//images/Properties/noimage.png"

: $"https://myleasing.azurewebsites.net{ImageUrl.Substring(1)}";

And re-publish.

# App Xamarin Forms First Part

## Login

1. Delete the **MainPage** and **MainPageViewModel**.
2. Add the **LoginPage**, with this initial layout:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.LoginPage"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Label

Text="Email"/>

<Entry

Keyboard="Email"

Placeholder="Enter your email..."

Text="{Binding Email}"/>

<Label

Text="Password"/>

<Entry

IsPassword="True"

Placeholder="Enter your password..."

Text="{Binding Password}"/>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Login"/>

</StackLayout>

</ContentPage>

1. Modify the **LoginPageViewModel**:

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class LoginPageViewModel : ViewModelBase

{

private string \_password;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

public LoginPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Login";

IsEnabled = true;

}

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(Login));

public string Email { get; set; }

public string Password

{

get => \_password;

set => SetProperty(ref \_password, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void Login()

{

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter an email.", "Accept");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a password.", "Accept");

return;

}

await App.Current.MainPage.DisplayAlert("Ok", "We are making progress!", "Accept");

}

}

}

1. Test it.
2. Add the **Response** class:

namespace MyLeasing.Common.Models

{

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

}

1. Add the **TokenRequest** class:

namespace MyLeasing.Common.Models

{

public class TokenRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

}

1. Add the **TokenResponse** class:

using System;

namespace MyLeasing.Common.Models

{

public class TokenResponse

{

public string Token { get; set; }

public DateTime Expiration { get; set; }

public DateTime ExpirationLocal => Expiration.ToLocalTime();

}

}

1. Add the **IApiService** interface:

using System.Threading.Tasks;

using MyLeasing.Common.Models;

namespace MyLeasing.Common.Services

{

public interface IApiService

{

Task<Response> GetOwnerByEmail(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

string email);

Task<Response> GetTokenAsync(

string urlBase,

string servicePrefix,

string controller,

TokenRequest request);

}

}

1. Add the **ApiService** class:

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

using MyLeasing.Common.Models;

using Newtonsoft.Json;

namespace MyLeasing.Common.Services

{

public class ApiService : IApiService

{

public async Task<Response> GetTokenAsync(

string urlBase,

string servicePrefix,

string controller,

TokenRequest request)

{

try

{

var requestString = JsonConvert.SerializeObject(request);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

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

return new Response

{

IsSuccess = true,

Result = token

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

public async Task<Response> GetOwnerByEmail(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

string email)

{

try

{

var request = new EmailRequest { Email = email };

var requestString = JsonConvert.SerializeObject(request);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

var owner = JsonConvert.DeserializeObject<OwnerResponse>(result);

return new Response

{

IsSuccess = true,

Result = owner

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

}

}

1. Add a resource dictionary in **App.xaml**:

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

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

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

xmlns:prism="clr-namespace:Prism.DryIoc;assembly=Prism.DryIoc.Forms"

x:Class="MyLeasing.Prism.App">

<Application.Resources>

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">https://myleasing.azurewebsites.net/</x:String>

</ResourceDictionary>

</Application.Resources>

</prism:PrismApplication>

1. Modify the **App.xaml.cs** class:

protected override async void OnInitialized()

{

InitializeComponent();

await NavigationService.NavigateAsync("NavigationPage/LoginPage");

}

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<Login, LoginViewModel>();

}

1. Modify the method **Login** in **LoginPageViewMovil** class:

private readonly IApiService \_apiService;

…

public LoginPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_apiService = apiService;

Title = "Login";

IsEnabled = true;

}

...

private async void Login()

{

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter an email.", "Accept");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a password.", "Accept");

return;

}

IsRunning = true;

IsEnabled = false;

var request = new TokenRequest

{

Password = Password,

Username = Email

};

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.GetTokenAsync(url, "Account", "/CreateToken", request);

if (!response.IsSuccess)

{

IsEnabled = true;

IsRunning = false;

await App.Current.MainPage.DisplayAlert("Error", "User or password incorrect.", "Accept");

Password = string.Empty;

return;

}

IsEnabled = true;

IsRunning = false;

await App.Current.MainPage.DisplayAlert("Ok", "We are making progress!", "Accept");

}

1. Test it.

## Modify Response to generic Class

1. Modify the **Response** class:

namespace MyLeasing.Common.Models

{

public class Response<T> where T : class

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public T Result { get; set; }

}

}

1. Modify the **IApiService** interface:

Task<Response<OwnerResponse>> GetOwnerByEmailAsync(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

string email);

Task<Response<TokenResponse>> GetTokenAsync(

string urlBase,

string servicePrefix,

string controller,

TokenRequest request);

1. Modify the **ApiService** interface:

public async Task<Response<TokenResponse>> GetTokenAsync(

string urlBase,

string servicePrefix,

string controller,

TokenRequest request)

{

try

{

var requestString = JsonConvert.SerializeObject(request);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response<TokenResponse>

{

IsSuccess = false,

Message = result,

};

}

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

return new Response<TokenResponse>

{

IsSuccess = true,

Result = token

};

}

catch (Exception ex)

{

return new Response<TokenResponse>

{

IsSuccess = false,

Message = ex.Message

};

}

}

public async Task<Response<OwnerResponse>> GetOwnerByEmailAsync(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

string email)

{

try

{

var request = new EmailRequest { Email = email };

var requestString = JsonConvert.SerializeObject(request);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response<OwnerResponse>

{

IsSuccess = false,

Message = result,

};

}

var owner = JsonConvert.DeserializeObject<OwnerResponse>(result);

return new Response<OwnerResponse>

{

IsSuccess = true,

Result = owner

};

}

catch (Exception ex)

{

return new Response<OwnerResponse>

{

IsSuccess = false,

Message = ex.Message

};

}

}

1. Modify the **LoginPageViewModel** class:

…

var token = response.Result;

…

var owner = response2.Result;

...

1. Test it.

## Check the internet connection

1. Add the NuGet **Xam.Plugin.Connectivity** to all prism projects and **Common** too
2. Add the method to **IApiservice**:

Task<bool> CheckConnection(string url);

1. Add the method to **Apiservice**:

public async Task<bool> CheckConnection(string url)

{

if (!CrossConnectivity.Current.IsConnected)

{

return false;

}

return await CrossConnectivity.Current.IsRemoteReachable(url);

}

1. Modify the **LoginPageViewModel**:

IsRunning = true;

IsEnabled = false;

var url = App.Current.Resources["UrlAPI"].ToString();

var connection = await \_apiService.CheckConnection(url);

if (!connection)

{

IsEnabled = true;

IsRunning = false;

await App.Current.MainPage.DisplayAlert("Error", "Check the internet connection.", "Accept");

return;

}

var request = new TokenRequest

1. Test it.

## Navigate to another page and pass parameters

1. Create the **PropertiesPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Properties"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Label

HorizontalOptions="CenterAndExpand"

Text="{Binding Title}"

VerticalOptions="CenterAndExpand"/>

</StackLayout>

</ContentPage>

1. Modify the **PropertiesViewModel** class:

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class PropertiesViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

public PropertiesViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Properties";

}

}

}

1. Modify the **LoginViewModel** class:

IsEnabled = true;

IsRunning = false;

await \_navigationService.NavigateAsync("Properties");

}

1. To avoid entering email and password everytime, temporarily add those lines to **LoginViewModel** constructor:

public LoginViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Login";

IsEnabled = true;

//TODO: Delete those lines

Email = "jzuluaga55@hotmail.com";

Password = "123456";

}

1. Test it.
2. Add the property **FullName** to **OwnerResponse** class:

public string FullName => $"{FirstName} {LastName}";

1. Modify the method **Login** in **LoginViewModel** class:

var owner = (OwnerResponse)response2.Result;

var parameters = new NavigationParameters

{

{ "token", token },

{ "owner", owner }

};

IsEnabled = true;

IsRunning = false;

await \_navigationService.NavigateAsync("Properties", parameters);

1. Override the method **OnNavigatedTo** in **PropertiesViewModel** class:

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("token"))

{

\_token = parameters.GetValue<TokenResponse>("token");

}

if (parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner");

Title = \_owner.FullName;

}

}

1. Test it.
2. Create the **PropertyItemViewModel** class:

using MyLeasing.Common.Models;

namespace MyLeasing.Prism.ViewModels

{

public class PropertyItemViewModel : PropertyResponse

{

}

}

1. Add the property **FirstImage** to **PropertyResponse**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Properties"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ListView

HasUnevenRows="True"

SeparatorVisibility="None"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Properties}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Frame

CornerRadius="20"

HasShadow="True"

Margin="0,0,0,5">

<Frame.GestureRecognizers>

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

</Frame.GestureRecognizers>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding FirstImage}"

WidthRequest="100">

</Image>

<StackLayout

Grid.Column="1"

VerticalOptions="Center">

<Label

FontAttributes="Bold"

FontSize="Medium"

Text="{Binding Neighborhood}"

TextColor="Black">

</Label>

<Label

Text="{Binding Address}"

TextColor="Black">

</Label>

<Label

Text="{Binding Price, StringFormat='{0:C2}'}"

TextColor="Navy">

</Label>

</StackLayout>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"

WidthRequest="40">

</Image>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Modify the **PropertiesViewModel** class:

using System.Collections.ObjectModel;

using System.Linq;

using MyLeasing.Common.Models;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class PropertiesViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private OwnerResponse \_owner;

private TokenResponse \_token;

private ObservableCollection<PetItemViewModel> \_pets;

private bool \_isRefreshing;

public PropertiesViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Properties";

}

public ObservableCollection<PetItemViewModel> Pets

{

get => \_pets;

set => SetProperty(ref \_pets, value);

}

public bool IsRefreshing

{

get => \_isRefreshing;

set => SetProperty(ref \_isRefreshing, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

IsRefreshing = true;

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("token"))

{

\_token = parameters.GetValue<TokenResponse>("token");

}

if (parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner");

Pets = new ObservableCollection<PetItemViewModel>(\_owner.Pets.Select(p => new PetItemViewModel

{

Born = p.Born,

Histories = p.Histories,

Id = p.Id,

ImageUrl = p.ImageUrl,

Name = p.Name,

PetType = p.PetType,

Race = p.Race,

Remarks = p.Remarks

}).ToList());

}

IsRefreshing = false;

}

}

}

1. Test it.
2. Add the **ContractItemViewModel** class:

using MyLeasing.Common.Models;

namespace MyLeasing.Prism.ViewModels

{

public class ContractItemViewModel : ContractResponse

{

}

}

1. Modify the **PropertyItemViewModel** class:

using MyLeasing.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class PropertyItemViewModel : PropertyResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectPropertyCommand;

public PropertyItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectPropertyCommand => \_selectPropertyCommand ?? (\_selectPropertyCommand = new DelegateCommand(SelectProperty));

private async void SelectProperty()

{

var parameters = new NavigationParameters

{

{ "property", this }

};

await \_navigationService.NavigateAsync("Property", parameters);

}

}

}

1. Add the property **FullName** to **LesseeResponse** model:

public string FullName => $"{FirstName} {LastName}";

1. Add the **Property** page:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Property"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Frame

CornerRadius="20"

HasShadow="True">

<StackLayout>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

HeightRequest="20"

Source="ic\_chevron\_left"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Image

Grid.Column="1"

Source="{Binding Image}"

HeightRequest="300"

Aspect="AspectFill"/>

<Image

Grid.Column="2"

HeightRequest="20"

Source="ic\_chevron\_right"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Column="0" Grid.Row="0" Text="Property Type" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="0" Text="{Binding Property.PropertyType}"/>

<Label Grid.Column="0" Grid.Row="1" Text="Neighborhood" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="1" Text="{Binding Property.Neighborhood}"/>

<Label Grid.Column="0" Grid.Row="2" Text="Address" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="2" Text="{Binding Property.Address}"/>

<Label Grid.Column="0" Grid.Row="3" Text="Price" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="3" Text="{Binding Property.Price, StringFormat='{0:C2}'}"/>

<Label Grid.Column="0" Grid.Row="4" Text="Square Meters" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="4" Text="{Binding Property.SquareMeters, StringFormat='{0:N2}'}"/>

<Label Grid.Column="0" Grid.Row="5" Text="Rooms" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="5" Text="{Binding Property.Rooms}"/>

<Label Grid.Column="0" Grid.Row="6" Text="Remarks" FontAttributes="Bold"/>

<Label Grid.Column="1" Grid.Row="6" Text="{Binding Property.Remarks}"/>

</Grid>

</StackLayout>

</Frame>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="Contracts"

TextColor="Black"/>

<ListView

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Contracts}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding StartDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding EndDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Lessee.FullName}"

VerticalOptions="Center"/>

<Image

Grid.Column="3"

HeightRequest="20"

Margin="0,5"

Source="ic\_chevron\_right"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Modify the **PropertyViewModel** class:

using System.Collections.ObjectModel;

using System.Linq;

using MyLeasing.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class PropertyViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PropertyResponse \_property;

private ObservableCollection<ContractItemViewModel> \_contracts;

private string \_image;

private DelegateCommand \_previousImageCommand;

private DelegateCommand \_nextImageCommand;

private int \_positionImage;

public PropertyViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

\_positionImage = 0;

}

public DelegateCommand PreviousImageCommand => \_previousImageCommand ?? (\_previousImageCommand = new DelegateCommand(MovePreviousImage));

public DelegateCommand NextImageCommand => \_nextImageCommand ?? (\_nextImageCommand = new DelegateCommand(MoveNextImage));

public PropertyResponse Property

{

get => \_property;

set => SetProperty(ref \_property, value);

}

public string Image

{

get => \_image;

set => SetProperty(ref \_image, value);

}

public ObservableCollection<ContractItemViewModel> Contracts

{

get => \_contracts;

set => SetProperty(ref \_contracts, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

Title = Property.Neighborhood;

Contracts = new ObservableCollection<ContractItemViewModel>(Property.Contracts.Select(c => new ContractItemViewModel

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Lessee = c.Lessee,

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList());

Image = Property.FirstImage;

}

}

private void MoveNextImage()

{

MoveImage(1);

}

private void MovePreviousImage()

{

MoveImage(-1);

}

private void MoveImage(int delta)

{

if (Property.PropertyImages == null || Property.PropertyImages.Count <= 1)

{

return;

}

\_positionImage += delta;

if (\_positionImage < 0)

{

\_positionImage = Property.PropertyImages.Count - 1;

}

if (\_positionImage > Property.PropertyImages.Count - 1)

{

\_positionImage = 0;

}

Image = Property.PropertyImages.ToList()[\_positionImage].ImageUrl;

}

}

}

1. Test it.
2. Modify the **ContractItemViewModel** class:

using MyLeasing.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ContractItemViewModel : ContractResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectContractCommand;

public ContractItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectContractCommand => \_selectContractCommand ?? (\_selectContractCommand = new DelegateCommand(SelectContract));

private async void SelectContract()

{

var parameters = new NavigationParameters

{

{ "contract", this }

};

await \_navigationService.NavigateAsync("Contract", parameters);

}

}

}

1. Add the **Contract** page:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Contract"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="3\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

FontAttributes="Bold"

Text="Remarks"/>

<Label

Grid.Column="1"

Grid.Row="0"

Text="{Binding Contract.Remarks}"/>

<Label

Grid.Column="0"

Grid.Row="1"

FontAttributes="Bold"

Text="Start Date"/>

<Label

Grid.Column="1"

Grid.Row="1"

Text="{Binding Contract.StartDate, StringFormat='{0:yyyy/MM/dd}'}"/>

<Label

Grid.Column="0"

Grid.Row="2"

FontAttributes="Bold"

Text="End Date"/>

<Label

Grid.Column="1"

Grid.Row="2"

Text="{Binding Contract.EndDate, StringFormat='{0:yyyy/MM/dd}'}"/>

<Label

Grid.Column="0"

Grid.Row="3"

FontAttributes="Bold"

Text="Price"/>

<Label

Grid.Column="1"

Grid.Row="3"

Text="{Binding Contract.Price, StringFormat='{0:C2}'}"/>

<Label

Grid.Column="0"

Grid.Row="4"

FontAttributes="Bold"

Text="Lessee"/>

<Label

Grid.Column="1"

Grid.Row="4"

Text="{Binding Contract.Lessee.FullName}"/>

</Grid>

</StackLayout>

</ContentPage>

1. Modify the **ContractViewModel** class:

using MyLeasing.Common.Models;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ContractViewModel : ViewModelBase

{

private ContractResponse \_contract;

public ContractViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Contract";

}

public ContractResponse Contract

{

get => \_contract;

set => SetProperty(ref \_contract, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("contract"))

{

Contract = parameters.GetValue<ContractResponse>("contract");

}

}

}

}

1. Test it.

## Fix the images on Android

1. Update the NuGet **Xamarin.Forms** in all Prism projects.
2. Add the NuGet **Xamarin.FFImageLoading.Forms** to all prism projects.
3. Add this line on **MainActivity**:

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

LoadApplication(new App(new AndroidInitializer()));

1. Add this line on **AppDelegate**:

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

LoadApplication(new App(new iOSInitializer()));

1. Modify the **PetsPage**:

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

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

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.PetsPage"

BackgroundColor="Silver"

Title="{Binding Title}">

…

<ffimageloading:CachedImage

Grid.Column="0"

Source="{Binding ImageUrl}"

LoadingPlaceholder= "LoaderImage"

ErrorPlaceholder= "ErrorImage"

CacheDuration= "50"

RetryCount= "3"

RetryDelay= "600"

DownsampleToViewSize = "true"

WidthRequest="100"/>

1. Modify the **PetPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.PetPage"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ffimageloading:CachedImage

Source="{Binding Pet.ImageUrl}"

LoadingPlaceholder= "LoaderImage"

ErrorPlaceholder= "ErrorImage"

CacheDuration= "50"

RetryCount= "3"

RetryDelay= "600"

DownsampleToViewSize = "true"

WidthRequest="250"/>

<Grid>

1. Test it.

## Add SfRotator

1. Get a Sync Fusion license: <https://help.syncfusion.com/common/essential-studio/licensing/license-key#xamarinforms>.
2. Add the **NuGet Syncfusion.Xamarin.SfRotator** to all mobile projects.
3. Add your license in **App.xaml.cs**:

protected override async void OnInitialized()

{

Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("MTM3Njg0QDMxMzcyZTMyMmUzMGUvQlg3Tnk5ODRGQ01pbzNnWmEyWHdWcExaaUVOQ0FKODZGNDFpekRtd2M9");

InitializeComponent();

await NavigationService.NavigateAsync("NavigationPage/LoginPage");

}

1. Modify the **MainActivity**:

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfRotatorRenderer();

LoadApplication(new App(new AndroidInitializer()));

}

1. Add NuGet **Syncfusion.Xamarin.SfRotator.Android** on Android project.
2. Modify the **AppDelegate**:

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

new SfRotatorRenderer();

LoadApplication(new App(new iOSInitializer()));

return base.FinishedLaunching(app, options);

}

1. Add NuGet **Syncfusion.Xamarin.SfRotator.iOS** on iOS project.
2. Modify the **PropertyPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.PropertyPage"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<syncfusion:SfRotator

x:Name="rotator"

EnableAutoPlay="True"

EnableLooping="True"

NavigationDelay="5000"

HeightRequest="300"/>

<Grid>

1. Add the NuGet **Xam.Plugins.Settings** to Common project.
2. Add the **Settings** class:

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace MyLeasing.Common.Helpers

{

public static class Settings

{

private const string \_propertyImages = "PropertyImages";

private static readonly string \_settingsDefault = string.Empty;

private static ISettings AppSettings => CrossSettings.Current;

public static string PropertyImages

{

get => AppSettings.GetValueOrDefault(\_propertyImages, \_settingsDefault);

set => AppSettings.AddOrUpdateValue(\_propertyImages, value);

}

}

}

1. Add the **CustomData** class:

using Xamarin.Forms;

namespace MyLeasing.Prism.Models

{

public class CustomData : ContentPage

{

public CustomData()

{

}

public CustomData(string image)

{

Image = image;

}

public string Image { get; set;}

}

}

1. Modify the **PropertyItemViewModel**:

private async void SelectProperty()

{

Settings.PropertyImages = JsonConvert.SerializeObject(PropertyImages);

var parameters = new NavigationParameters

{

{ "property", this }

};

await \_navigationService.NavigateAsync("PropertyPage", parameters);

}

1. Modify the **PropertyPage.xaml.cs**:

using MyLeasing.Common.Helpers;

using MyLeasing.Common.Models;

using MyLeasing.Prism.Models;

using Newtonsoft.Json;

using System.Collections.Generic;

using Xamarin.Forms;

namespace MyLeasing.Prism.Views

{

public partial class PropertyPage : ContentPage

{

public PropertyPage()

{

InitializeComponent();

}

protected override void OnAppearing()

{

base.OnAppearing();

rotator.ItemsSource = GetDataSource();

var imageTemplate = new DataTemplate(() =>

{

var image = new Image();

image.SetBinding(Image.SourceProperty, "Image");

return image;

});

rotator.ItemTemplate = imageTemplate;

}

private IEnumerable<CustomData> GetDataSource()

{

List<CustomData> list = new List<CustomData>();

var propertyImages = JsonConvert.DeserializeObject<List<PropertyImageResponse>>(Settings.PropertyImages);

foreach (var itepropertyImage in propertyImages)

{

list.Add(new CustomData(itepropertyImage.ImageUrl));

}

return list;

}

}

}

1. Test it.

## Improve SfRotator and fix for Android

1. Modify the **PropertyPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

xmlns:syncfusion="clr-namespace:Syncfusion.SfRotator.XForms;assembly=Syncfusion.SfRotator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.PropertyPage"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<syncfusion:SfRotator

BackgroundColor="#ececec"

EnableAutoPlay="True"

EnableLooping="True"

HeightRequest="300"

ItemsSource="{Binding ImageCollection}"

NavigationDelay="5000"

NavigationDirection="Horizontal"

NavigationStripMode="Dots"

NavigationStripPosition="Bottom">

<syncfusion:SfRotator.ItemTemplate>

<DataTemplate>

<ffimageloading:CachedImage

Aspect="AspectFit"

CacheDuration= "50"

DownsampleToViewSize = "true"

ErrorPlaceholder= "ErrorImage"

HeightRequest="300"

LoadingPlaceholder= "LoaderImage"

RetryCount= "3"

RetryDelay= "600"

Source="{Binding Image}"/>

</DataTemplate>

</syncfusion:SfRotator.ItemTemplate>

</syncfusion:SfRotator>

<Grid>

1. Delete the **CustomData** class.
2. Add the **RotatorModel** class:

namespace MyLeasing.Common.Models

{

public class RotatorModel

{

public string Image { get; set; }

}

}

1. Modify the **PropertyPage.xaml.cs**:

using Xamarin.Forms;

namespace MyLeasing.Prism.Views

{

public partial class PropertyPage : ContentPage

{

public PropertyPage()

{

InitializeComponent();

}

}

}

1. Modify the **PropertyItemViewModel**:

private async void SelectProperty()

{

var parameters = new NavigationParameters

{

{ "property", this }

};

await \_navigationService.NavigateAsync("PropertyPage", parameters);

}

1. Modify the **PropertyPageViewModel**:

using MyLeasing.Common.Models;

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace MyLeasing.Prism.ViewModels

{

public class PropertyPageViewModel : ViewModelBase

{

private PropertyResponse \_property;

private ObservableCollection<RotatorModel> \_imageCollection;

public PropertyPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Property";

}

public ObservableCollection<RotatorModel> ImageCollection

{

get => \_imageCollection;

set => SetProperty(ref \_imageCollection, value);

}

public PropertyResponse Property

{

get => \_property;

set => SetProperty(ref \_property, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

Title = $"Property: {Property.Neighborhood}";

LoadImages();

}

}

private void LoadImages()

{

var list = new List<RotatorModel>();

foreach (var propertyImage in Property.PropertyImages)

{

list.Add(new RotatorModel { Image = propertyImage.ImageUrl });

}

ImageCollection = new ObservableCollection<RotatorModel>(list);

}

}

}

1. Test it

## Add tabbed page

1. Add the **ContractsPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.ContractsPage"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="10"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

FontAttributes="Bold"

Text="Start Date"/>

<Label

Grid.Column="1"

FontAttributes="Bold"

Text="End Date"/>

<Label

Grid.Column="2"

FontAttributes="Bold"

Text="Lessee"/>

</Grid>

<ListView

HasUnevenRows="True"

ItemsSource="{Binding Contracts}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding StartDateLocal, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding EndDateLocal, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Lessee.FullName}"

VerticalOptions="Center"/>

<Image

Grid.Column="3"

Source="ic\_more\_vert"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Add the **ContractItemViewModel**:

using MyLeasing.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ContractItemViewModel : ContractResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectContractCommand;

public ContractItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectContractCommand => \_selectContractCommand ?? (\_selectContractCommand = new DelegateCommand(SelectContract));

private async void SelectContract()

{

var parameters = new NavigationParameters

{

{ "contract", this }

};

await \_navigationService.NavigateAsync("ContractPage", parameters);

}

}

}

1. Modify the **ContractsPageViewModel**:

using MyLeasing.Common.Models;

using Prism.Navigation;

using System.Collections.ObjectModel;

using System.Linq;

namespace MyLeasing.Prism.ViewModels

{

public class ContractsPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PropertyResponse \_property;

private ObservableCollection<ContractItemViewModel> \_contracts;

public ContractsPageViewModel(

INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Contracts";

}

public PropertyResponse Property

{

get => \_property;

set => SetProperty(ref \_property, value);

}

public ObservableCollection<ContractItemViewModel> Contracts

{

get => \_contracts;

set => SetProperty(ref \_contracts, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

LoadContracts();

}

}

private void LoadContracts()

{

Title = $"Contracts: {Property.Neighborhood}";

Contracts = new ObservableCollection<ContractItemViewModel>(Property.Contracts.Select(c => new ContractItemViewModel(\_navigationService)

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Lessee = c.Lessee,

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList());

}

}

}

1. Add the **ContractPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.ContractPage"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

FontAttributes="Bold"

Text="Price"/>

<Label

Grid.Row="0"

Grid.Column="1"

Text="{Binding Contract.Price, StringFormat='{0:C2}'}"/>

<Label

Grid.Row="1"

Grid.Column="0"

FontAttributes="Bold"

Text="Start Date"/>

<Label

Grid.Row="1"

Grid.Column="1"

Text="{Binding Contract.StartDateLocal, StringFormat='{0:yyyy/MM/dd}'}"/>

<Label

Grid.Row="2"

Grid.Column="0"

FontAttributes="Bold"

Text="End Date"/>

<Label

Grid.Row="2"

Grid.Column="1"

Text="{Binding Contract.EndDateLocal, StringFormat='{0:yyyy/MM/dd}'}"/>

<Label

Grid.Row="3"

Grid.Column="0"

FontAttributes="Bold"

Text="Is Active?"

VerticalOptions="Center"/>

<CheckBox

Grid.Row="3"

Grid.Column="1"

IsChecked="{Binding Contract.IsActive}"/>

<Label

Grid.Row="4"

Grid.Column="0"

FontAttributes="Bold"

Text="Lessee"/>

<Label

Grid.Row="4"

Grid.Column="1"

Text="{Binding Contract.Lessee.FullName}"/>

<Label

Grid.Row="5"

Grid.Column="0"

FontAttributes="Bold"

Text="Remarks"/>

<Label

Grid.Row="5"

Grid.Column="1"

Text="{Binding Contract.Remarks}"/>

</Grid>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modify the **ContractPageViewModel**:

using MyLeasing.Common.Models;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ContractPageViewModel : ViewModelBase

{

private ContractResponse \_contract;

public ContractPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Contract";

}

public ContractResponse Contract

{

get => \_contract;

set => SetProperty(ref \_contract, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("contract"))

{

Contract = parameters.GetValue<ContractResponse>("contract");

Title = $"Contract to: {Contract.Lessee.FullName}";

}

}

}

}

1. Temporarily modify **PropertyItemViewModel**:

await \_navigationService.NavigateAsync("ContractsPage", parameters);

1. Test it.
2. Add the **PropertyTabbedPage**:

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

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

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

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:local="clr-namespace:MyLeasing.Prism.Views"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.PropertyTabbedPage"

Title="{Binding Title}">

<TabbedPage.Children>

<local:PropertyPage />

<local:ContractsPage />

</TabbedPage.Children>

</TabbedPage>

1. Modify the **PropertyTabbedPageViewModel**:

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class PropertyTabbedPageViewModel : ViewModelBase

{

public PropertyTabbedPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Property";

}

}

}

1. Modify the **Settings**:

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace MyLeasing.Common.Helpers

{

public static class Settings

{

private const string \_property = "Property";

private static readonly string \_settingsDefault = string.Empty;

private static ISettings AppSettings => CrossSettings.Current;

public static string Property

{

get => AppSettings.GetValueOrDefault(\_property, \_settingsDefault);

set => AppSettings.AddOrUpdateValue(\_property, value);

}

}

}

1. Modify the **PropertyItemViewModel**:

private async void SelectProperty()

{

Settings.Property = JsonConvert.SerializeObject(this);

await \_navigationService.NavigateAsync("PropertyTabbedPage");

}

1. Modify the **PropertyPageViewModel**:

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

Property = JsonConvert.DeserializeObject<PropertyResponse>(Settings.Property);

LoadImages();

}

1. Modify the **ContractsPageViewModel**:

…

Delete the method on natigated to

...

public ContractsPageViewModel(

INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Contracts";

Property = JsonConvert.DeserializeObject<PropertyResponse>(Settings.Property);

LoadContracts();

}

1. Add icons to pages **ContractsPage** and **PropertyPage**.
2. Test it.

## Add SfBusyIndicator

1. Add the NuGet **Syncfusion.Xamarin.SfBusyIndicator** to all mobile projects.
2. Modify the **MainActivity**:

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfBusyIndicatorRenderer();

LoadApplication(new App(new AndroidInitializer()));

1. Modify the **AppDelegate**:

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

new SfBusyIndicatorRenderer();

LoadApplication(new App(new iOSInitializer()));

1. Modify the **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"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.LoginPage"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Label

Text="Email"/>

…

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="Silver"

HorizontalOptions="Center"

TextColor="White"

IsVisible="{Binding IsRunning}"

Title="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

…

</StackLayout>

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Test it.

## GIT Workshop

<https://www.youtube.com/watch?v=AqocDsE_32c>

<https://www.youtube.com/watch?v=CDeG4S-mJts>

git checkout -b <nombre\_de\_la\_nueva\_rama> <hash\_del\_commit\_al\_que\_quiere\_volver>

git checkout -b old-state 0d1d7fc32

# Web Second Part

## Redirect Pages

1. Create **NotAuthorized** method on **AccountController**:

public IActionResult NotAuthorized()

{

return View();

}

1. Create correspondent view with this lines:

@{

ViewData["Title"] = "NotAuthorized";

}

<br />

<br />

<img src="~/images/gopher\_head-min.png" />

<h2>You are not authorized to perform this action!</h2>

1. Modify **Startup.cs** to configure the Application Cookie Options (after cookies lines):

services.ConfigureApplicationCookie(options =>

{

options.LoginPath = "/Account/NotAuthorized";

options.AccessDeniedPath = "/Account/NotAuthorized";

});

1. We add it to the pipeline inside **Startup.cs** with a wildcard as a parameter.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseStatusCodePagesWithReExecute("/error/{0}");

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthentication();

app.UseCookiePolicy();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

1. Inside the **HomeController** create the following action.

[Route("error/404")]

public IActionResult Error404()

{

return View();

}

1. Create the correspondent view.

@{

ViewData["Title"] = "Error404";

}

<br />

<br />

<img src="~/images/gopher\_head-min.png" />

<h2>Sorry, page not found</h2>

1. Test it!.

## Self-registration of users

1. Add this method to **IUserHelper**:

Task<User> AddUser(AddUserViewModel view, string role);

1. Add this method to **UserHelper**:

public async Task<User> AddUser(AddUserViewModel view, string role)

{

var user = new User

{

Address = view.Address,

Document = view.Document,

Email = view.Username,

FirstName = view.FirstName,

LastName = view.LastName,

PhoneNumber = view.PhoneNumber,

UserName = view.Username

};

var result = await AddUserAsync(user, view.Password);

if (result != IdentityResult.Success)

{

return null;

}

var newUser = await GetUserByEmailAsync(view.Username);

await AddUserToRoleAsync(newUser, role);

return newUser;

}

1. Modify the **OwnersController** to use the new method on **UserHelper**.
2. Add those properties to **AddUserViewModel**:

[Required(ErrorMessage = "The field {0} is mandatory.")]

[Display(Name = "Register as")]

[Range(1, int.MaxValue, ErrorMessage = "You must select a role.")]

public int RoleId { get; set; }

public IEnumerable<SelectListItem> Roles { get; set; }

1. Add the **ICombosHelper**:

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc.Rendering;

namespace MyLeasing.Web.Helpers

{

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboLessees();

IEnumerable<SelectListItem> GetComboPropertyTypes();

IEnumerable<SelectListItem> GetComboRoles();

}

}

1. Add the **CombosHelper**:

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

namespace MyLeasing.Web.Helpers

{

public class CombosHelper : ICombosHelper

{

private readonly DataContext \_dataContext;

public CombosHelper(DataContext dataContext)

{

\_dataContext = dataContext;

}

public IEnumerable<SelectListItem> GetComboRoles()

{

var list = new List<SelectListItem>

{

new SelectListItem { Value = "0", Text = "(Select a role...)" },

new SelectListItem { Value = "1", Text = "Lessee" },

new SelectListItem { Value = "2", Text = "Owner" }

};

return list;

}

public IEnumerable<SelectListItem> GetComboLessees()

{

var list = \_dataContext.Lessees.Include(l => l.User).Select(p => new SelectListItem

{

Text = p.User.FullNameWithDocument,

Value = p.Id.ToString()

}).OrderBy(p => p.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a lessee...)",

Value = "0"

});

return list;

}

public IEnumerable<SelectListItem> GetComboPropertyTypes()

{

var list = \_dataContext.PropertyTypes.Select(p => new SelectListItem

{

Text = p.Name,

Value = p.Id.ToString()

}).OrderBy(p => p.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a property type...)",

Value = "0"

});

return list;

}

}

}

1. Inject the new interface:

services.AddScoped<ICombosHelper, CombosHelper>();

1. Modify the **OwnersController** to use **CombosHelper**.
2. Add those methods to **AccountController**:

public IActionResult Register()

{

var view = new AddUserViewModel

{

Roles = \_combosHelper.GetComboRoles()

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var role = "Owner";

if (view.RoleId == 1)

{

role = "Lessee";

}

var user = await \_userHelper.AddUser(view, role);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

if (view.RoleId == 1)

{

var lessee = new Lessee

{

Contracts = new List<Contract>(),

User = user

};

\_dataContext.Lessees.Add(lessee);

await \_dataContext.SaveChangesAsync();

}

else

{

var owner = new Owner

{

Contracts = new List<Contract>(),

Properties = new List<Property>(),

User = user

};

\_dataContext.Owners.Add(owner);

await \_dataContext.SaveChangesAsync();

}

var loginViewModel = new LoginViewModel

{

Password = view.Password,

RememberMe = false,

Username = view.Username

};

var result2 = await \_userHelper.LoginAsync(loginViewModel);

if (result2.Succeeded)

{

return RedirectToAction("Index", "Home");

}

}

return View(view);

}

1. Add the view **Register** on **AccountController**:

@model MyLeasing.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Register</h2>

<h4>Owner</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Register">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="Username" class="control-label"></label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Document" class="control-label"></label>

<input asp-for="Document" class="form-control" />

<span asp-validation-for="Document" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="FirstName" class="control-label"></label>

<input asp-for="FirstName" class="form-control" />

<span asp-validation-for="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="LastName" class="control-label"></label>

<input asp-for="LastName" class="form-control" />

<span asp-validation-for="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber" class="control-label"></label>

<input asp-for="PhoneNumber" class="form-control" />

<span asp-validation-for="PhoneNumber" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="RoleId" class="control-label"></label>

<select asp-for="RoleId" asp-items="Model.Roles" class="form-control"></select>

<span asp-validation-for="RoleId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PasswordConfirm" class="control-label"></label>

<input asp-for="PasswordConfirm" class="form-control" />

<span asp-validation-for="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Register" class="btn btn-primary" />

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Modify the **Create** method on **OwnersController**:

public IActionResult Create()

{

var view = new AddUserViewModel { RoleId = 2 };

return View(view);

}

1. Modify the **Create** view on **OwnersController**:

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="RoleId" />

<div class="form-group">

1. Test it.

## Modifying users

1. Add the **ChangePasswordViewModel** class:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class ChangePasswordViewModel

{

[Display(Name = "Current password")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

public string OldPassword { get; set; }

[Display(Name = "New password")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

public string NewPassword { get; set; }

[Display(Name = "Password confirm")]

[Required(ErrorMessage = "The field {0} is mandatory.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

[Compare("NewPassword")]

public string Confirm { get; set; }

}

}

1. Add this method in **IUserHelper** interface:

Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword);

1. Add the implementation in **UserHelper** class:

public async Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword)

{

return await \_userManager.ChangePasswordAsync(user, oldPassword, newPassword);

}

1. Add those methods to **AccountController** class:

public async Task<IActionResult> ChangeUser()

{

var user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

if (user == null)

{

return NotFound();

}

var view = new EditUserViewModel

{

Address = user.Address,

Document = user.Document,

FirstName = user.FirstName,

LastName = user.LastName,

PhoneNumber = user.PhoneNumber

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangeUser(EditUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

user.Document = view.Document;

user.FirstName = view.FirstName;

user.LastName = view.LastName;

user.Address = view.Address;

user.PhoneNumber = view.PhoneNumber;

await \_userHelper.UpdateUserAsync(user);

return RedirectToAction("Index", "Home");

}

return View(view);

}

1. Add the view **ChangeUser** in **AccountController**:

@model MyLeasing.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>User</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="ChangeUser">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Document" class="control-label"></label>

<input asp-for="Document" class="form-control" />

<span asp-validation-for="Document" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="FirstName" class="control-label"></label>

<input asp-for="FirstName" class="form-control" />

<span asp-validation-for="FirstName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="LastName" class="control-label"></label>

<input asp-for="LastName" class="form-control" />

<span asp-validation-for="LastName" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber" class="control-label"></label>

<input asp-for="PhoneNumber" class="form-control" />

<span asp-validation-for="PhoneNumber" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="ChangePassword" class="btn btn-warning">Change Password</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.
2. Add those methods to **AccountController** class:

public IActionResult ChangePassword()

{

return View();

}

[HttpPost]

public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(User.Identity.Name);

if (user != null)

{

var result = await \_userHelper.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

return RedirectToAction("ChangeUser");

}

else

{

ModelState.AddModelError(string.Empty, result.Errors.FirstOrDefault().Description);

}

}

else

{

ModelState.AddModelError(string.Empty, "User no found.");

}

}

return View(model);

}

1. Add the view **ChangePassword** to **AccountController** class:

@model MyLeasing.Web.Models.ChangePasswordViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Change Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="OldPassword">Current password</label>

<input asp-for="OldPassword" type="password" class="form-control" />

<span asp-validation-for="OldPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="NewPassword">New password</label>

<input asp-for="NewPassword" type="password" class="form-control" />

<span asp-validation-for="NewPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Confirm">Confirm</label>

<input asp-for="Confirm" type="password" class="form-control" />

<span asp-validation-for="Confirm" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Change password" class="btn btn-primary" />

<a asp-action="ChangeUser" class="btn btn-success">Back to user</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.

## Confirm Email Registration

1. First, change the setup project:

services.AddIdentity<User, IdentityRole>(cfg =>

{

cfg.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;

cfg.SignIn.RequireConfirmedEmail = true;

cfg.User.RequireUniqueEmail = true;

cfg.Password.RequireDigit = false;

cfg.Password.RequiredUniqueChars = 0;

cfg.Password.RequireLowercase = false;

cfg.Password.RequireNonAlphanumeric = false;

cfg.Password.RequireUppercase = false;

})

.AddDefaultTokenProviders()

.AddEntityFrameworkStores<DataContext>();

1. Check if your email account is enabled to send email in: <https://myaccount.google.com/lesssecureapps>
2. Add this parameters to the configuration file:

"Mail": {

"From": "youremail@gmail.com",

"Smtp": "smtp.gmail.com",

"Port": 587,

"Password": "yourpassword"

}

1. Add the nuget “**Mailkit**”.
2. In **Helpers** folder add the interface **IMailHelper**:

namespace MyLeasing.Web.Helpers

{

public interface IMailHelper

{

void SendMail(string to, string subject, string body);

}

}

1. In **Helpers** folder add the implementation **MailHelper**:

using MailKit.Net.Smtp;

using Microsoft.Extensions.Configuration;

using MimeKit;

namespace MyLeasing.Web.Helpers

{

public class MailHelper : IMailHelper

{

private readonly IConfiguration \_configuration;

public MailHelper(IConfiguration configuration)

{

\_configuration = configuration;

}

public void SendMail(string to, string subject, string body)

{

var from = \_configuration["Mail:From"];

var smtp = \_configuration["Mail:Smtp"];

var port = \_configuration["Mail:Port"];

var password = \_configuration["Mail:Password"];

var message = new MimeMessage();

message.From.Add(new MailboxAddress(from));

message.To.Add(new MailboxAddress(to));

message.Subject = subject;

var bodyBuilder = new BodyBuilder

{

HtmlBody = body

};

message.Body = bodyBuilder.ToMessageBody();

using (var client = new SmtpClient())

{

client.Connect(smtp, int.Parse(port), false);

client.Authenticate(from, password);

client.Send(message);

client.Disconnect(true);

}

}

}

}

1. Configure the injection for the new interface:

services.AddScoped<IMailHelper, MailHelper>();

1. Add those methods to **IUserHelper**:

Task<string> GenerateEmailConfirmationTokenAsync(User user);

Task<IdentityResult> ConfirmEmailAsync(User user, string token);

Task<User> GetUserByIdAsync(string userId);

And the implementation:

public async Task<IdentityResult> ConfirmEmailAsync(User user, string token)

{

return await \_userManager.ConfirmEmailAsync(user, token);

}

public async Task<string> GenerateEmailConfirmationTokenAsync(User user)

{

return await \_userManager.GenerateEmailConfirmationTokenAsync(user);

}

public async Task<User> GetUserByIdAsync(string userId)

{

return await \_userManager.FindByIdAsync(userId);

}

1. Modify the register post method (first inject the **IMailHelper** in **AccountController**):

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await AddUser(view);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

var owner = new Owner

{

Pets = new List<Pet>(),

User = user,

};

\_dataContext.Owners.Add(owner);

await \_dataContext.SaveChangesAsync();

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(view.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

ViewBag.Message = "The instructions to allow your user has been sent to email.";

return View(view);

}

return View(view);

}

1. Add this to the register view ends:

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

1. Create the method confirm email in account controller:

public async Task<IActionResult> ConfirmEmail(string userId, string token)

{

if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(token))

{

return NotFound();

}

var user = await \_userHelper.GetUserByIdAsync(userId);

if (user == null)

{

return NotFound();

}

var result = await \_userHelper.ConfirmEmailAsync(user, token);

if (!result.Succeeded)

{

return NotFound();

}

return View();

}

1. Create the view:

@{

ViewData["Title"] = "Confirm email";

}

<h2>@ViewData["Title"]</h2>

<div>

<p>

Thank you for confirming your email. Now you can login into system.

</p>

</div>

1. Modify the **OwnersController** to send the email confirmation:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await AddUser(view);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

var owner = new Owner

{

Pets = new List<Pet>(),

User = user,

};

\_dataContext.Owners.Add(owner);

await \_dataContext.SaveChangesAsync();

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(view.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

return RedirectToAction(nameof(Index));

}

return View(view);

}

1. Drop the database (PM> drop-database) to ensure that all the users have a confirmed email.
2. Modify the seed class:

private async Task<User> CheckUserAsync(string document, string firstName, string lastName, string email, string phone, string address, string role)

{

var user = await \_userHelper.GetUserByEmailAsync(email);

if (user == null)

{

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, role);

var token = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

1. Test it.

## Password Recovery

1. Modify the login view:

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a asp-action="Register" class="btn btn-primary">Register New User</a>

<a asp-action="RecoverPassword" class="btn btn-link">Forgot your password?</a>

</div>

1. Add the model **RecoverPasswordViewModel**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class RecoverPasswordViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

1. Add the model **ResetPasswordViewModel**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Web.Models

{

public class ResetPasswordViewModel

{

[Required]

public string UserName { get; set; }

[Required]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between {2} and {1} characters.")]

[DataType(DataType.Password)]

[Compare("Password")]

public string ConfirmPassword { get; set; }

[Required]

public string Token { get; set; }

}

}

1. Add those methods to **IUserHelper**:

Task<string> GeneratePasswordResetTokenAsync(User user);

Task<IdentityResult> ResetPasswordAsync(User user, string token, string password);

And the implementation:

public async Task<string> GeneratePasswordResetTokenAsync(User user)

{

return await \_userManager.GeneratePasswordResetTokenAsync(user);

}

public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password)

{

return await \_userManager.ResetPasswordAsync(user, token, password);

}

1. Add this methods to account controller:

public IActionResult RecoverPassword()

{

return View();

}

[HttpPost]

public async Task<IActionResult> RecoverPassword(RecoverPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.GetUserByEmailAsync(model.Email);

if (user == null)

{

ModelState.AddModelError(string.Empty, "The email doesn't correspont to a registered user.");

return View(model);

}

var myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

var link = Url.Action(

"ResetPassword",

"Account",

new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(model.Email, "MyLeasing Password Reset", $"<h1>MyLeasing Password Reset</h1>" +

$"To reset the password click in this link:</br></br>" +

$"<a href = \"{link}\">Reset Password</a>");

ViewBag.Message = "The instructions to recover your password has been sent to email.";

return View();

}

return View(model);

}

public IActionResult ResetPassword(string token)

{

return View();

}

[HttpPost]

public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)

{

var user = await \_userHelper.GetUserByEmailAsync(model.UserName);

if (user != null)

{

var result = await \_userHelper.ResetPasswordAsync(user, model.Token, model.Password);

if (result.Succeeded)

{

ViewBag.Message = "Password reset successful.";

return View();

}

ViewBag.Message = "Error while resetting the password.";

return View(model);

}

ViewBag.Message = "User not found.";

return View(model);

}

1. Add the view:

@model MyLeasing.Web.Models.RecoverPasswordViewModel

@{

ViewData["Title"] = "Recover Password";

}

<h2>Recover Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="Email">Email</label>

<input asp-for="Email" class="form-control" />

<span asp-validation-for="Email" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Recover password" class="btn btn-primary" />

<a asp-action="Login" class="btn btn-success">Back to login</a>

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Add the view:

@model MyLeasing.Web.Models.ResetPasswordViewModel

@{

ViewData["Title"] = "Reset Password";

}

<h1>Reset Your Password</h1>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="All"></div>

<input type="hidden" asp-for="Token" />

<div class="form-group">

<label asp-for="UserName">Email</label>

<input asp-for="UserName" class="form-control" />

<span asp-validation-for="UserName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Password">New password</label>

<input asp-for="Password" type="password" class="form-control" />

<span asp-validation-for="Password" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="ConfirmPassword">Confirm</label>

<input asp-for="ConfirmPassword" type="password" class="form-control" />

<span asp-validation-for="ConfirmPassword" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Reset password" class="btn btn-primary" />

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.
2. Finally, delete all records in Azure DB and re-publish the solution.

## Improve Index View

1. Modify the **Index** view by for **OwnersController**:

@model IEnumerable<MyLeasing.Web.Data.Entities.Owner>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Create New</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Owners</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Document)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.FirstName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.LastName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.PhoneNumber)

</th>

<th>

Pets

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.User.Document)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.FirstName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.LastName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.PhoneNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Pets.Count)

</td>

<td>

<a asp-action="Edit" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<partial name="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Owners/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Following the example, do the same for **Details**, **DetailsProperty** and **DetailsContract**.
3. **Homework**: according to owners example, do the same for: property types, contracts and lessees.
4. Homework solved:
5. Modify the **PropertyTypesController**.

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Controllers

{

[Authorize(Roles = "Manager")]

public class PropertyTypesController : Controller

{

private readonly DataContext \_context;

public PropertyTypesController(DataContext context)

{

\_context = context;

}

public async Task<IActionResult> Index()

{

return View(await \_context.PropertyTypes.ToListAsync());

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(PropertyType propertyType)

{

if (ModelState.IsValid)

{

\_context.Add(propertyType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(propertyType);

}

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var propertyType = await \_context.PropertyTypes.FindAsync(id);

if (propertyType == null)

{

return NotFound();

}

return View(propertyType);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(PropertyType propertyType)

{

if (ModelState.IsValid)

{

try

{

\_context.Update(propertyType);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!PropertyTypeExists(propertyType.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(propertyType);

}

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var propertyType = await \_context.PropertyTypes

.Include(pt => pt.Properties)

.FirstOrDefaultAsync(m => m.Id == id);

if (propertyType == null)

{

return NotFound();

}

if (propertyType.Properties.Count > 0)

{

//TODO: message

return RedirectToAction(nameof(Index));

}

\_context.PropertyTypes.Remove(propertyType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool PropertyTypeExists(int id)

{

return \_context.PropertyTypes.Any(e => e.Id == id);

}

}

}

1. Modify the view **Index** on **PropertyTypesController**.

@model IEnumerable<MyLeasing.Web.Data.Entities.PropertyType>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Create New</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Property Types</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

<a asp-action="Edit" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<!--Delete Item-->

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Do you want to delete the record?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/PropertyTypes/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the view **Create** on **PropertyTypesController**.

@model MyLeasing.Web.Data.Entities.PropertyType

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Property Type</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="Name" class="control-label"></label>

<input asp-for="Name" class="form-control" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

1. Modify the view **Edit** on **PropertyTypesController**.

@model MyLeasing.Web.Data.Entities.PropertyType

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Property Type</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Name" class="control-label"></label>

<input asp-for="Name" class="form-control" />

<span asp-validation-for="Name" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

1. Delete the **Details** on **PropertyTypesController**.
2. Delete the view **Delete** on **PropertyTypesController**.
3. Test it.
4. Modify or add the **LesseesController**.

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Helpers;

using MyLeasing.Web.Models;

namespace MyLeasing.Web.Controllers

{

[Authorize(Roles = "Manager")]

public class LesseesController : Controller

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

public LesseesController(

DataContext dataContext,

IUserHelper userHelper,

IMailHelper mailHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

}

public IActionResult Index()

{

return View(\_dataContext.Lessees

.Include(o => o.User)

.Include(o => o.Contracts));

}

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var lessee = await \_dataContext.Lessees

.Include(l => l.User)

.Include(l => l.Contracts)

.FirstOrDefaultAsync(m => m.Id == id);

if (lessee == null)

{

return NotFound();

}

return View(lessee);

}

public IActionResult Create()

{

var view = new AddUserViewModel { RoleId = 1 };

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.AddUser(view, "Lessee");

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

var lessee = new Lessee

{

Contracts = new List<Contract>(),

User = user,

};

\_dataContext.Lessees.Add(lessee);

await \_dataContext.SaveChangesAsync();

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(view.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

return RedirectToAction(nameof(Index));

}

return View(view);

}

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var lessee = await \_dataContext.Lessees

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.Id == id.Value);

if (lessee == null)

{

return NotFound();

}

var view = new EditUserViewModel

{

Address = lessee.User.Address,

Document = lessee.User.Document,

FirstName = lessee.User.FirstName,

Id = lessee.Id,

LastName = lessee.User.LastName,

PhoneNumber = lessee.User.PhoneNumber

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditUserViewModel view)

{

if (ModelState.IsValid)

{

var lessee = await \_dataContext.Lessees

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.Id == view.Id);

lessee.User.Document = view.Document;

lessee.User.FirstName = view.FirstName;

lessee.User.LastName = view.LastName;

lessee.User.Address = view.Address;

lessee.User.PhoneNumber = view.PhoneNumber;

await \_userHelper.UpdateUserAsync(lessee.User);

return RedirectToAction(nameof(Index));

}

return View(view);

}

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var lessee = await \_dataContext.Lessees

.Include(o => o.User)

.FirstOrDefaultAsync(m => m.Id == id);

if (lessee == null)

{

return NotFound();

}

\_dataContext.Lessees.Remove(lessee);

await \_dataContext.SaveChangesAsync();

await \_userHelper.DeleteUserAsync(lessee.User.Email);

return RedirectToAction(nameof(Index));

}

}

}

1. Modify/Add the view **Index** on **LesseesController**.

@model IEnumerable<MyLeasing.Web.Data.Entities.Lessee>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Create New</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Lessees</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Document)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.FirstName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.LastName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.PhoneNumber)

</th>

<th>

Contracts

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.User.Document)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.FirstName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.LastName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.PhoneNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.Contracts.Count)

</td>

<td>

<a asp-action="Edit" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<partial name="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Lessees/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the view **Create** on **LesseesController**.

@model MyLeasing.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Lessee</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="RoleId" />

<div class="form-group">

<label asp-for="Username" class="control-label"></label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-danger"></span>

</div>

<partial name="\_User"/>

<div class="form-group">

<label asp-for="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PasswordConfirm" class="control-label"></label>

<input asp-for="PasswordConfirm" class="form-control" />

<span asp-validation-for="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Modify the view **Edit** on **LesseesController**.

@model MyLeasing.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Lessee</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<partial name="\_User" />

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Modify the view **Details** on **LesseesController**.

@model MyLeasing.Web.Data.Entities.Lessee

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Lessee</h2>

<div>

<h4>Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.User.PhoneNumber)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

1. Test it.
2. Modify/Add the **PropertiesController**.

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Controllers

{

[Authorize(Roles = "Manager")]

public class PropertiesController : Controller

{

private readonly DataContext \_dataContext;

public PropertiesController(DataContext dataContext)

{

\_dataContext = dataContext;

}

public IActionResult Index()

{

return View(\_dataContext.Properties

.Include(p => p.PropertyType)

.Include(p => p.PropertyImages)

.Include(p => p.Contracts)

.Include(p => p.Owner)

.ThenInclude(o => o.User));

}

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(o => o.Owner)

.ThenInclude(o => o.User)

.Include(o => o.Contracts)

.ThenInclude(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(o => o.PropertyType)

.Include(p => p.PropertyImages)

.FirstOrDefaultAsync(m => m.Id == id);

if (property == null)

{

return NotFound();

}

return View(property);

}

}

}

1. Modify/Add the view **Index** on **PropertiesController**.

@model IEnumerable<MyLeasing.Web.Data.Entities.Property>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Properties</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

Property Type

</th>

<th>

Owner

</th>

<th>

@Html.DisplayNameFor(model => model.Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.SquareMeters)

</th>

<th>

@Html.DisplayNameFor(model => model.IsAvailable)

</th>

<th>

Images

</th>

<th>

Contracts

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Owner.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.SquareMeters)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsAvailable)

</td>

<td>

@Html.DisplayFor(modelItem => item.Remarks)

</td>

<td>

@Html.DisplayFor(modelItem => item.PropertyImages.Count)

</td>

<td>

@Html.DisplayFor(modelItem => item.Contracts.Count)

</td>

<td>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Delete the view **Create** on **PropertiesController**.
2. Delete the view **Edit** on **PropertiesController**.
3. Modify the view **Details** on **PropertiesController**.

@model MyLeasing.Web.Data.Entities.Property

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Property</h2>

<div class="row">

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

<div>

<h4>Owner</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.PhoneNumber)

</dd>

</dl>

</div>

</div>

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

<div>

<h4>Property Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.SquareMeters)

</dt>

<dd>

@Html.DisplayFor(model => model.SquareMeters)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Rooms)

</dt>

<dd>

@Html.DisplayFor(model => model.Rooms)

</dd>

<dt>

@Html.DisplayNameFor(model => model.HasParkingLot)

</dt>

<dd>

@Html.DisplayFor(model => model.HasParkingLot)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsAvailable)

</dt>

<dd>

@Html.DisplayFor(model => model.IsAvailable)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

</dl>

</div>

</div>

</div>

<div>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

<hr />

<div class="row">

<div class="col-md-3">

<div>

@if (Model.PropertyImages.Count == 0)

{

<h5>Not images added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Images</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableImages">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.PropertyImages.FirstOrDefault().ImageUrl)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.PropertyImages)

{

<tr>

<td>

@if (!string.IsNullOrEmpty(item.ImageUrl))

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:200px;height:200px;max-width: 100%; height: auto;" />

}

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

}

</div>

</div>

<div class="col-md-9">

<div>

@if (Model.Contracts.Count == 0)

{

<h5>Not contracts added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Contracts</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableContracts">

<thead>

<tr>

<th>

Lessee

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Remarks)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().StartDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().EndDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().IsActive)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Contracts)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Lessee.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsActive)

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

}

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTableImages').DataTable();

$('#MyTableContracts').DataTable();

});

</script>

}

1. Test it.

## Managers CRUD

1. Modify the **ManagersController**:

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Helpers;

using MyLeasing.Web.Models;

namespace MyLeasing.Web.Controllers

{

[Authorize(Roles = "Manager")]

public class ManagersController : Controller

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

public ManagersController(

DataContext dataContext,

IUserHelper userHelper,

IMailHelper mailHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

}

public IActionResult Index()

{

return View(\_dataContext.Managers.Include(m => m.User));

}

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var manager = await \_dataContext.Managers

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.Id == id.Value);

if (manager == null)

{

return NotFound();

}

return View(manager);

}

public IActionResult Create()

{

return View(new AddUserViewModel { RoleId = 3 });

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel view)

{

if (ModelState.IsValid)

{

var user = await \_userHelper.AddUser(view, "Manager");

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

return View(view);

}

var manager = new Manager { User = user };

\_dataContext.Managers.Add(manager);

await \_dataContext.SaveChangesAsync();

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(view.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

return RedirectToAction(nameof(Index));

}

return View(view);

}

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var manager = await \_dataContext.Managers

.Include(m => m.User)

.FirstOrDefaultAsync(m => m.Id == id);

if (manager == null)

{

return NotFound();

}

var view = new EditUserViewModel

{

Address = manager.User.Address,

Document = manager.User.Document,

FirstName = manager.User.FirstName,

Id = manager.Id,

LastName = manager.User.LastName,

PhoneNumber = manager.User.PhoneNumber

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditUserViewModel view)

{

if (ModelState.IsValid)

{

var manager = await \_dataContext.Managers

.Include(m => m.User)

.FirstOrDefaultAsync(o => o.Id == view.Id);

manager.User.Document = view.Document;

manager.User.FirstName = view.FirstName;

manager.User.LastName = view.LastName;

manager.User.Address = view.Address;

manager.User.PhoneNumber = view.PhoneNumber;

await \_userHelper.UpdateUserAsync(manager.User);

return RedirectToAction(nameof(Index));

}

return View(view);

}

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var manager = await \_dataContext.Managers

.Include(m => m.User)

.FirstOrDefaultAsync(m => m.Id == id);

if (manager == null)

{

return NotFound();

}

\_dataContext.Managers.Remove(manager);

await \_dataContext.SaveChangesAsync();

await \_userHelper.DeleteUserAsync(manager.User.Email);

return RedirectToAction(nameof(Index));

}

}

}

1. Modify the **Index** view for **ManagersController**:

@model IEnumerable<MyLeasing.Web.Data.Entities.Manager>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Create New</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Managers</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Document)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.FirstName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.LastName)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.FirstOrDefault().User.PhoneNumber)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.User.Document)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.FirstName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.LastName)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.PhoneNumber)

</td>

<td>

<a asp-action="Edit" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<partial name="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Managers/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the **Details** view for **ManagersController**:

@model MyLeasing.Web.Data.Entities.Manager

@{

ViewData["Title"] = "Details";

}

<h2>Manager</h2>

<div>

<h4>Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.User.PhoneNumber)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

1. Modify the **Create** view for **ManagersController**:

@model MyLeasing.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Manager</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="RoleId" />

<div class="form-group">

<label asp-for="Username" class="control-label"></label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-danger"></span>

</div>

<partial name="\_User"/>

<div class="form-group">

<label asp-for="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PasswordConfirm" class="control-label"></label>

<input asp-for="PasswordConfirm" class="form-control" />

<span asp-validation-for="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Modify the **Edit** view for **ManagersController**:

@model MyLeasing.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Manager</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<partial name="\_User" />

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.

## Look and feel for a leasing system and users functionality

1. Modify the **\_Layout.cshtml**:

...

<a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">My Leasing</a>

…

<p>&copy; 2019 - My Leasing by Zulu</p>

…

1. Add some images related to a veterinary, I hardly recommend use 1200 x 400 pixels.
2. Modify the view **Index** for **HomeCotroller**:

@{

ViewData["Title"] = "Home Page";

}

<div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="6000">

<ol class="carousel-indicators">

<li data-target="#myCarousel" data-slide-to="0" class="active"></li>

<li data-target="#myCarousel" data-slide-to="1"></li>

<li data-target="#myCarousel" data-slide-to="2"></li>

<li data-target="#myCarousel" data-slide-to="3"></li>

<li data-target="#myCarousel" data-slide-to="4"></li>

<li data-target="#myCarousel" data-slide-to="5"></li>

<li data-target="#myCarousel" data-slide-to="6"></li>

<li data-target="#myCarousel" data-slide-to="7"></li>

</ol>

<div class="carousel-inner" role="listbox">

<div class="item active">

<img src="~/images/Home/v1.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v2.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v3.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v4.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v5.png" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v6.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v7.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/Home/v8.jpg" class="img-responsive" />

</div>

</div>

<a class="left carousel-control" href="#myCarousel" role="button" data-slide="prev">

<span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="right carousel-control" href="#myCarousel" role="button" data-slide="next">

<span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

<hr />

<div class="row">

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

<a asp-action="SearchProperties" class="btn btn-primary">Search Properties</a>

</div>

@if (User.IsInRole("Owner") || User.IsInRole("Lessee"))

{

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

<a asp-action="MyContracts" class="btn btn-warning">My Contracts</a>

</div>

}

@if (User.IsInRole("Owner"))

{

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

<a asp-action="MyProperties" class="btn btn-success">My Properties</a>

</div>

}

</div>

1. Test it.

## Users functionality

### Search Properties

1. Add the method **SearchProperties** to **HomeController**:

public IActionResult SearchProperties()

{

return View(\_dataContext.Properties

.Include(p => p.PropertyType)

.Include(p => p.PropertyImages)

.Where(p => p.IsAvailable));

}

1. Add the view **SearchProperties** to **HomeController**:

@model IEnumerable<MyLeasing.Web.Data.Entities.Property>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Properties</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

Image

</th>

<th>

Property Type

</th>

<th>

@Html.DisplayNameFor(model => model.Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.SquareMeters)

</th>

<th>

@Html.DisplayNameFor(model => model.Rooms)

</th>

<th>

@Html.DisplayNameFor(model => model.HasParkingLot)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@if (!string.IsNullOrEmpty(item.FirstImage))

{

<img src="@Url.Content(item.FirstImage)" alt="Image" style="width:300px;height:300px;max-width: 100%; height: auto;" />

}

</td>

<td>

@Html.DisplayFor(modelItem => item.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.SquareMeters)

</td>

<td>

@Html.DisplayFor(modelItem => item.Rooms)

</td>

<td>

@Html.DisplayFor(modelItem => item.HasParkingLot)

</td>

<td>

<a asp-action="DetailsProperty" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Test it.
2. Add the method **DetailsProperty** to **HomeController**:

public async Task<IActionResult> DetailsProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(o => o.PropertyType)

.Include(p => p.PropertyImages)

.FirstOrDefaultAsync(m => m.Id == id);

if (property == null)

{

return NotFound();

}

return View(property);

}

1. Add the view **DetailsProperty** to **HomeController**:

@model MyLeasing.Web.Data.Entities.Property

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Property</h2>

<div class="row">

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

<div>

<h4>Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

Property Type

</dt>

<dd>

@Html.DisplayFor(model => model.PropertyType.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.SquareMeters)

</dt>

<dd>

@Html.DisplayFor(model => model.SquareMeters)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Rooms)

</dt>

<dd>

@Html.DisplayFor(model => model.Rooms)

</dd>

<dt>

@Html.DisplayNameFor(model => model.HasParkingLot)

</dt>

<dd>

@Html.DisplayFor(model => model.HasParkingLot)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

</dl>

</div>

</div>

</div>

<div>

<a asp-action="SearchProperties" class="btn btn-success">Back to Properties</a>

</div>

<hr />

@if (Model.PropertyImages.Count == 0)

{

<h5>Not images added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-4">

@if (Model.PropertyImages.Count > 0)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[0].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 1)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[1].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 2)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[2].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

</div>

<div class="row">

<div class="col-md-4">

@if (Model.PropertyImages.Count > 3)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[3].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 4)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[4].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 5)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[5].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

</div>

<div class="row">

<div class="col-md-4">

@if (Model.PropertyImages.Count > 6)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[6].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 7)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[7].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

<div class="col-md-4">

@if (Model.PropertyImages.Count > 8)

{

<img src="@Url.Content(Model.PropertyImages.ToList()[8].ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

</div>

}

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Test it.

### My Properties

1. Add the method **MyProperties** for **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> MyProperties()

{

var owner = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Contracts)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyType)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyImages)

.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(User.Identity.Name.ToLower()));

if (owner == null)

{

return NotFound();

}

return View(owner);

}

1. Add the view **MyProperties** for **HomeCotroller**:

@model MyLeasing.Web.Data.Entities.Owner

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Owner</h2>

<div>

<h4>Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.User.PhoneNumber)

</dd>

<dt>

Properties

</dt>

<dd>

@Html.DisplayFor(model => model.Properties.Count)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

</dl>

</div>

<div>

<a asp-action="AddProperty" asp-route-id="@Model.Id" class="btn btn-primary">Add Property</a>

<a asp-action="Index" class="btn btn-success">Back to Home</a>

</div>

<hr />

@if (Model.Properties.Count == 0)

{

<h5>Not properties added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Properties</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().PropertyType.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().SquareMeters)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Rooms)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Stratum)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().IsAvailable)

</th>

<th>

Images

</th>

<th>

Contracts

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Properties)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.SquareMeters)

</td>

<td>

@Html.DisplayFor(modelItem => item.Rooms)

</td>

<td>

@Html.DisplayFor(modelItem => item.Stratum)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsAvailable)

</td>

<td>

@Html.DisplayFor(modelItem => item.PropertyImages.Count)

</td>

<td>

@Html.DisplayFor(modelItem => item.Contracts.Count)

</td>

<td>

<a asp-action="EditProperty" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a asp-action="DetailsPropertyOwner" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

}

<partial name="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteItem').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Home/DeleteProperty/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Add the **IConverterHelper**:

using System.Threading.Tasks;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Models;

namespace MyLeasing.Web.Helpers

{

public interface IConverterHelper

{

Task<Contract> ToContractAsync(ContractViewModel view);

ContractViewModel ToContractViewModel(Contract contract);

Task<Property> ToPropertyAsync(PropertyViewModel view);

PropertyViewModel ToPropertyViewModel(Property property);

}

}

1. Add the **ConverterHelper**:

using System.Threading.Tasks;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Models;

namespace MyLeasing.Web.Helpers

{

public class ConverterHelper : IConverterHelper

{

private readonly DataContext \_dataContext;

private readonly ICombosHelper \_combosHelper;

public ConverterHelper(

DataContext dataContext,

ICombosHelper combosHelper)

{

\_dataContext = dataContext;

\_combosHelper = combosHelper;

}

public async Task<Property> ToPropertyAsync(PropertyViewModel view)

{

return new Property

{

Address = view.Address,

HasParkingLot = view.HasParkingLot,

IsAvailable = view.IsAvailable,

Neighborhood = view.Neighborhood,

Price = view.Price,

Rooms = view.Rooms,

SquareMeters = view.SquareMeters,

Stratum = view.Stratum,

Owner = await \_dataContext.Owners.FindAsync(view.OwnerId),

PropertyType = await \_dataContext.PropertyTypes.FindAsync(view.PropertyTypeId),

Remarks = view.Remarks

};

}

public async Task<Contract> ToContractAsync(ContractViewModel view)

{

return new Contract

{

EndDate = view.EndDate,

IsActive = view.IsActive,

Lessee = await \_dataContext.Lessees.FindAsync(view.LesseeId),

Owner = await \_dataContext.Owners.FindAsync(view.OwnerId),

Price = view.Price,

Property = await \_dataContext.Properties.FindAsync(view.PropertyId),

Remarks = view.Remarks,

StartDate = view.StartDate,

Id = view.Id

};

}

public PropertyViewModel ToPropertyViewModel(Property property)

{

return new PropertyViewModel

{

Address = property.Address,

HasParkingLot = property.HasParkingLot,

Id = property.Id,

IsAvailable = property.IsAvailable,

Neighborhood = property.Neighborhood,

Price = property.Price,

Rooms = property.Rooms,

SquareMeters = property.SquareMeters,

Stratum = property.Stratum,

Owner = property.Owner,

OwnerId = property.Owner.Id,

PropertyType = property.PropertyType,

PropertyTypeId = property.PropertyType.Id,

PropertyTypes = \_combosHelper.GetComboPropertyTypes(),

Remarks = property.Remarks,

};

}

public ContractViewModel ToContractViewModel(Contract contract)

{

return new ContractViewModel

{

EndDate = contract.EndDate,

IsActive = contract.IsActive,

LesseeId = contract.Lessee.Id,

OwnerId = contract.Owner.Id,

Price = contract.Price,

Remarks = contract.Remarks,

StartDate = contract.StartDate,

Id = contract.Id,

Lessees = \_combosHelper.GetComboLessees(),

PropertyId = contract.Property.Id

};

}

}

}

1. Modify the **OwnersController** to use the new helper.
2. Add the methods **AddProperty** for **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> AddProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners.FindAsync(id.Value);

if (owner == null)

{

return NotFound();

}

var view = new PropertyViewModel

{

OwnerId = owner.Id,

PropertyTypes = \_combosHelper.GetComboPropertyTypes()

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> AddProperty(PropertyViewModel model)

{

if (ModelState.IsValid)

{

var property = await \_converterHelper.ToPropertyAsync(model, true);

\_dataContext.Properties.Add(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyProperties));

}

return View(model);

}

1. Add the view **AddProperty** for **HomeCotroller**:

@model MyLeasing.Web.Models.PropertyViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddProperty">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="OwnerId" />

<div class="form-group">

<label asp-for="PropertyTypeId" class="control-label"></label>

<select asp-for="PropertyTypeId" asp-items="Model.PropertyTypes" class="form-control"></select>

<span asp-validation-for="PropertyTypeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Neighborhood" class="control-label"></label>

<input asp-for="Neighborhood" class="form-control" />

<span asp-validation-for="Neighborhood" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="SquareMeters" class="control-label"></label>

<input asp-for="SquareMeters" class="form-control" />

<span asp-validation-for="SquareMeters" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Rooms" class="control-label"></label>

<input asp-for="Rooms" class="form-control" />

<span asp-validation-for="Rooms" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Stratum" class="control-label"></label>

<input asp-for="Stratum" class="form-control" />

<span asp-validation-for="Stratum" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="HasParkingLot" /> @Html.DisplayNameFor(model => model.HasParkingLot)

</label>

</div>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailable" /> @Html.DisplayNameFor(model => model.IsAvailable)

</label>

</div>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="MyProperties" class="btn btn-success">Back to My Properties</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add the methods **EditProperty** for **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> EditProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(p => p.Owner)

.Include(p => p.PropertyType)

.FirstOrDefaultAsync(p => p.Id == id.Value);

if (property == null)

{

return NotFound();

}

var view = \_converterHelper.ToPropertyViewModel(property);

return View(view);

}

[HttpPost]

public async Task<IActionResult> EditProperty(PropertyViewModel view)

{

if (ModelState.IsValid)

{

var property = await \_converterHelper.ToPropertyAsync(view, false);

\_dataContext.Properties.Update(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyProperties));

}

return View(view);

}

1. Add the view **EditProperty** for **HomeCotroller**:

@model MyLeasing.Web.Models.PropertyViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="EditProperty">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="OwnerId" />

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="PropertyTypeId" class="control-label"></label>

<select asp-for="PropertyTypeId" asp-items="Model.PropertyTypes" class="form-control"></select>

<span asp-validation-for="PropertyTypeId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Neighborhood" class="control-label"></label>

<input asp-for="Neighborhood" class="form-control" />

<span asp-validation-for="Neighborhood" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Address" class="control-label"></label>

<input asp-for="Address" class="form-control" />

<span asp-validation-for="Address" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Price" class="control-label"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="SquareMeters" class="control-label"></label>

<input asp-for="SquareMeters" class="form-control" />

<span asp-validation-for="SquareMeters" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Rooms" class="control-label"></label>

<input asp-for="Rooms" class="form-control" />

<span asp-validation-for="Rooms" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Stratum" class="control-label"></label>

<input asp-for="Stratum" class="form-control" />

<span asp-validation-for="Stratum" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="HasParkingLot" /> @Html.DisplayNameFor(model => model.HasParkingLot)

</label>

</div>

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailable" /> @Html.DisplayNameFor(model => model.IsAvailable)

</label>

</div>

</div>

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="MyProperties" class="btn btn-success">Back to My Properties</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add the method **DetailsPropertyOwner** for **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> DetailsPropertyOwner(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(o => o.Owner)

.ThenInclude(o => o.User)

.Include(o => o.Contracts)

.ThenInclude(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(o => o.PropertyType)

.Include(p => p.PropertyImages)

.FirstOrDefaultAsync(m => m.Id == id);

if (property == null)

{

return NotFound();

}

return View(property);

}

1. Add the view **DetailsPropertyOwner** for **HomeCotroller**:

@model MyLeasing.Web.Data.Entities.Property

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Property</h2>

<div class="row">

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

<div>

<h4>Owner</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.FirstName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FirstName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.LastName)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.LastName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Owner.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.PhoneNumber)

</dd>

</dl>

</div>

</div>

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

<div>

<h4>Property Details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.SquareMeters)

</dt>

<dd>

@Html.DisplayFor(model => model.SquareMeters)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Rooms)

</dt>

<dd>

@Html.DisplayFor(model => model.Rooms)

</dd>

<dt>

@Html.DisplayNameFor(model => model.HasParkingLot)

</dt>

<dd>

@Html.DisplayFor(model => model.HasParkingLot)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsAvailable)

</dt>

<dd>

@Html.DisplayFor(model => model.IsAvailable)

</dd>

<dt>

Contracts

</dt>

<dd>

@Html.DisplayFor(model => model.Contracts.Count)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

</dl>

</div>

</div>

</div>

<div>

<a asp-action="EditProperty" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="AddImage" asp-route-id="@Model.Id" class="btn btn-primary">Add Image</a>

<a asp-action="MyProperties" class="btn btn-success">Back to My Properties</a>

</div>

<hr />

<div class="row">

<div class="col-md-3">

<div>

@if (Model.PropertyImages.Count == 0)

{

<h5>Not images added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Images</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableImages">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.PropertyImages.FirstOrDefault().ImageUrl)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.PropertyImages)

{

<tr>

<td>

@if (!string.IsNullOrEmpty(item.ImageUrl))

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:200px;height:200px;max-width: 100%; height: auto;" />

}

</td>

<td>

<button data-id="@item.Id" class="btn btn-danger deleteImage" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

}

</div>

</div>

<div class="col-md-9">

<div>

@if (Model.Contracts.Count == 0)

{

<h5>Not contracts added yet.</h5>

}

else

{

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Contracts</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableContracts">

<thead>

<tr>

<th>

Lessee

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Remarks)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().StartDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().EndDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Contracts.FirstOrDefault().IsActive)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Contracts)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Lessee.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Remarks)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndDateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsActive)

</td>

<td>

<a asp-action="DetailsContract" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

}

</div>

</div>

</div>

<!--Delete Item-->

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">

<div class="modal-dialog" role="document">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>

<button type="button" class="close" data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<p>Do you want to delete the record?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTableImages').DataTable();

$('#MyTableContracts').DataTable();

// Delete item

var item\_to\_delete;

$('.deleteImage').click((e) => {

item\_to\_delete = e.currentTarget.dataset.id;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Home/DeleteImage/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Add the methods **AddImage** for **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> AddImage(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties.FindAsync(id.Value);

if (property == null)

{

return NotFound();

}

var view = new PropertyImageViewModel

{

Id = property.Id

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> AddImage(PropertyImageViewModel view)

{

if (ModelState.IsValid)

{

var path = string.Empty;

if (view.ImageFile != null && view.ImageFile.Length > 0)

{

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

var file = $"{guid}.jpg";

path = Path.Combine(

Directory.GetCurrentDirectory(),

"wwwroot\\images\\Properties",

file);

using (var stream = new FileStream(path, FileMode.Create))

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Properties/{file}";

}

var propertyImage = new PropertyImage

{

ImageUrl = path,

Property = await \_dataContext.Properties.FindAsync(view.Id)

};

\_dataContext.PropertyImages.Add(propertyImage);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsPropertyOwner)}/{view.Id}");

}

return View(view);

}

1. Add the view **AddImage** for **HomeCotroller**:

@model MyLeasing.Web.Models.PropertyImageViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Add Image</h2>

<h4>Property</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddImage" enctype="multipart/form-data">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="ImageFile" class="control-label"></label>

<input asp-for="ImageFile" class="form-control" type="file" />

<span asp-validation-for="ImageFile" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

<a asp-action="DetailsPropertyOwner" asp-route-id="@Model.Id" class="btn btn-success">Back to Property</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add those methods for **Delete** in **HomeCotroller**:

[Authorize(Roles = "Owner")]

public async Task<IActionResult> DeleteImage(int? id)

{

if (id == null)

{

return NotFound();

}

var propertyImage = await \_dataContext.PropertyImages

.Include(pi => pi.Property)

.FirstOrDefaultAsync(pi => pi.Id == id.Value);

if (propertyImage == null)

{

return NotFound();

}

\_dataContext.PropertyImages.Remove(propertyImage);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsPropertyOwner)}/{propertyImage.Property.Id}");

}

[Authorize(Roles = "Owner")]

public async Task<IActionResult> DeleteProperty(int? id)

{

if (id == null)

{

return NotFound();

}

var property = await \_dataContext.Properties

.Include(p => p.Owner)

.Include(p => p.Contracts)

.FirstOrDefaultAsync(pi => pi.Id == id.Value);

if (property == null)

{

return NotFound();

}

if (property.Contracts?.Count > 0)

{

return RedirectToAction(nameof(MyProperties));

}

\_dataContext.Properties.Remove(property);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyProperties));

}

1. Test it.

### My Contracts

1. Add the method **MyContracts** for **HomeCotroller**:

[Authorize(Roles = "Owner, Lessee")]

public IActionResult MyContracts()

{

return View(\_dataContext.Contracts

.Include(c => c.Owner)

.ThenInclude(o => o.User)

.Include(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(c => c.Property)

.ThenInclude(p => p.PropertyType)

.Where(c => c.Owner.User.UserName.ToLower().Equals(User.Identity.Name.ToLower()) ||

c.Lessee.User.UserName.ToLower().Equals(User.Identity.Name.ToLower())));

}

1. Add the view **MyContracts** for **HomeCotroller**:

@model IEnumerable<MyLeasing.Web.Data.Entities.Contract>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Contracts</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

Owner

</th>

<th>

Lessee

</th>

<th>

Property Type

</th>

<th>

@Html.DisplayNameFor(model => model.Property.Neighborhood)

</th>

<th>

@Html.DisplayNameFor(model => model.Property.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.StartDate)

</th>

<th>

@Html.DisplayNameFor(model => model.EndDate)

</th>

<th>

@Html.DisplayNameFor(model => model.IsActive)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Owner.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Lessee.User.FullNameWithDocument)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.PropertyType.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.Neighborhood)

</td>

<td>

@Html.DisplayFor(modelItem => item.Property.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.IsActive)

</td>

<td>

<a asp-action="DetailsContract" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list"> </i> </a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Test it.
2. Add the method **DetailsContract** for **HomeCotroller**:

[Authorize(Roles = "Owner, Lessee")]

public async Task<IActionResult> DetailsContract(int? id)

{

if (id == null)

{

return NotFound();

}

var contract = await \_dataContext.Contracts

.Include(c => c.Owner)

.ThenInclude(o => o.User)

.Include(c => c.Lessee)

.ThenInclude(l => l.User)

.Include(c => c.Property)

.ThenInclude(p => p.PropertyType)

.FirstOrDefaultAsync(m => m.Id == id);

if (contract == null)

{

return NotFound();

}

return View(contract);

}

1. Add the method **DetailsContract** for **HomeCotroller**:

@model MyLeasing.Web.Data.Entities.Contract

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Contract</h4>

<hr />

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

@Html.DisplayFor(model => model.Owner.User.FullNameWithDocument)

</dd>

<dt>

Lessee

</dt>

<dd>

@Html.DisplayFor(model => model.Lessee.User.FullNameWithDocument)

</dd>

<dt>

Property Type

</dt>

<dd>

@Html.DisplayFor(model => model.Property.PropertyType.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Neighborhood)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Neighborhood)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Property.Address)

</dt>

<dd>

@Html.DisplayFor(model => model.Property.Address)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.StartDate)

</dt>

<dd>

@Html.DisplayFor(model => model.StartDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.EndDate)

</dt>

<dd>

@Html.DisplayFor(model => model.EndDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsActive)

</dt>

<dd>

@Html.DisplayFor(model => model.IsActive)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

</dl>

</div>

<div>

<a asp-action="MyContracts" class="btn btn-success">Back to My Contracts</a>

</div>

1. Test it.

## GROUP4

## Prepare the API for new functionality in App

In the App we’re going to: register new owners, edit owner profile, chage and retrieve password, add and modify properties (for owners), search properties, see contracts. To do that, first we need to modify the API.

### Account

1. Add the **UserRequest** model:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Common.Models

{

public class UserRequest

{

[Required]

public string Document { get; set; }

[Required]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

[Required]

public string Address { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Phone { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string Password { get; set; }

[Required]

public int RoleId { get; set; } // 1: Owner, 2: Lessee

}

}

1. Add the **AccountController** API:

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using MyLeasing.Common.Models;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Helpers;

using System.Linq;

using System.Threading.Tasks;

namespace MyLeasing.Web.Controllers.API

{

[Route("api/[Controller]")]

public class AccountController : ControllerBase

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

public AccountController(

DataContext dataContext,

IUserHelper userHelper,

IMailHelper mailHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

}

[HttpPost]

public async Task<IActionResult> PostUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user != null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "This email is already registered."

});

}

user = new User

{

Address = request.Address,

Document = request.Document,

Email = request.Email,

FirstName = request.FirstName,

LastName = request.LastName,

PhoneNumber = request.Phone,

UserName = request.Email

};

var result = await \_userHelper.AddUserAsync(user, request.Password);

if (result != IdentityResult.Success)

{

return BadRequest(result.Errors.FirstOrDefault().Description);

}

var userNew = await \_userHelper.GetUserByEmailAsync(request.Email);

if (request.RoleId == 1)

{

await \_userHelper.AddUserToRoleAsync(userNew, "Owner");

\_dataContext.Owners.Add(new Owner { User = userNew });

}

else

{

await \_userHelper.AddUserToRoleAsync(userNew, "Lessee");

\_dataContext.Lessees.Add(new Lessee { User = userNew });

}

await \_dataContext.SaveChangesAsync();

var myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"please click on this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

return Ok(new Response<object>

{

IsSuccess = true,

Message = "A Confirmation email was sent. Please confirm your account and log into the App."

});

}

}

}

1. Test it on postman.
2. Modify the model **EmailRequest**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Common.Models

{

public class EmailRequest

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

1. Add the method **RecoverPassword** in **AccountController** API:

[HttpPost]

[Route("RecoverPassword")]

public async Task<IActionResult> RecoverPassword([FromBody] EmailRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user == null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "This email is not assigned to any user."

});

}

var myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

var link = Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Password Reset", $"<h1>Recover Password</h1>" +

$"To reset the password click in this link:</br></br>" +

$"<a href = \"{link}\">Reset Password</a>");

return Ok(new Response<object>

{

IsSuccess = true,

Message = "An email with instructions to change the password was sent."

});

}

1. Test it on postman.
2. Add the method **PutUser** in **AccountController** API:

[HttpPut]

public async Task<IActionResult> PutUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var userEntity = await \_userHelper.GetUserByEmailAsync(request.Email);

if (userEntity == null)

{

return BadRequest("User not found.");

}

userEntity.FirstName = request.FirstName;

userEntity.LastName = request.LastName;

userEntity.Address = request.Address;

userEntity.PhoneNumber = request.Phone;

userEntity.Document = request.Phone;

var respose = await \_userHelper.UpdateUserAsync(userEntity);

if (!respose.Succeeded)

{

return BadRequest(respose.Errors.FirstOrDefault().Description);

}

var updatedUser = await \_userHelper.GetUserByEmailAsync(request.Email);

return Ok(updatedUser);

}

1. Test it on postman.
2. Add the model **ChangePasswordRequest**:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Common.Models

{

public class ChangePasswordRequest

{

[Required]

[StringLength(20, MinimumLength = 6)]

public string OldPassword { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string NewPassword { get; set; }

[Required]

public string Email { get; set; }

}

}

1. Add the method **ChangePassword** in **AccountController** API:

[HttpPost]

[Route("ChangePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user == null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "This email is not assigned to any user."

});

}

var result = await \_userHelper.ChangePasswordAsync(user, request.OldPassword, request.NewPassword);

if (!result.Succeeded)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = result.Errors.FirstOrDefault().Description

});

}

return Ok(new Response<object>

{

IsSuccess = true,

Message = "The password was changed successfully!"

});

}

1. Test it on postman.

### Owner

1. Modify the **OwnerResponse** model:

using System.Collections.Generic;

namespace MyLeasing.Common.Models

{

public class OwnerResponse

{

public int RoleId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Document { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

public ICollection<PropertyResponse> Properties { get; set; }

public ICollection<ContractResponse> Contracts { get; set; }

public string FullName => $"{FirstName} {LastName}";

}

}

1. Modify the **OwnersController** API:

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyLeasing.Common.Models;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

using MyLeasing.Web.Helpers;

namespace MyLeasing.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class OwnersController : ControllerBase

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

public OwnersController(

DataContext dataContext,

IUserHelper userHelper)

{

\_dataContext = dataContext;

\_userHelper = userHelper;

}

[HttpPost]

[Route("GetOwnerByEmail")]

public async Task<IActionResult> GetOwner(EmailRequest emailRequest)

{

try

{

var user = await \_userHelper.GetUserByEmailAsync(emailRequest.Email);

if (user == null)

{

return BadRequest("User not found.");

}

if (await \_userHelper.IsUserInRoleAsync(user, "Owner"))

{

return await GetOwnerAsync(emailRequest);

}

else

{

return await GetLesseeAsync(emailRequest);

}

}

catch (Exception ex)

{

return BadRequest(ex);

}

}

private async Task<IActionResult> GetLesseeAsync(EmailRequest emailRequest)

{

var lessee = await \_dataContext.Lessees

.Include(o => o.User)

.Include(o => o.Contracts)

.ThenInclude(c => c.Owner)

.ThenInclude(o => o.User)

.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(emailRequest.Email.ToLower()));

var properties = await \_dataContext.Properties

.Include(p => p.PropertyType)

.Include(p => p.PropertyImages)

.Where(p => p.IsAvailable)

.ToListAsync();

var response = new OwnerResponse

{

RoleId = 2,

Id = lessee.Id,

FirstName = lessee.User.FirstName,

LastName = lessee.User.LastName,

Address = lessee.User.Address,

Document = lessee.User.Document,

Email = lessee.User.Email,

PhoneNumber = lessee.User.PhoneNumber,

Properties = properties?.Select(p => new PropertyResponse

{

Address = p.Address,

HasParkingLot = p.HasParkingLot,

Id = p.Id,

IsAvailable = p.IsAvailable,

Neighborhood = p.Neighborhood,

Price = p.Price,

PropertyImages = p.PropertyImages?.Select(pi => new PropertyImageResponse

{

Id = pi.Id,

ImageUrl = pi.ImageFullPath

}).ToList(),

PropertyType = p.PropertyType.Name,

Remarks = p.Remarks,

Rooms = p.Rooms,

SquareMeters = p.SquareMeters,

Stratum = p.Stratum

}).ToList(),

Contracts = lessee.Contracts?.Select(c => new ContractResponse

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList()

};

return Ok(response);

}

private async Task<IActionResult> GetOwnerAsync(EmailRequest emailRequest)

{

var owner = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyType)

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyImages)

.Include(o => o.Contracts)

.ThenInclude(c => c.Lessee)

.ThenInclude(l => l.User)

.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(emailRequest.Email.ToLower()));

var response = new OwnerResponse

{

RoleId = 1,

Id = owner.Id,

FirstName = owner.User.FirstName,

LastName = owner.User.LastName,

Address = owner.User.Address,

Document = owner.User.Document,

Email = owner.User.Email,

PhoneNumber = owner.User.PhoneNumber,

Properties = owner.Properties?.Select(p => new PropertyResponse

{

Address = p.Address,

Contracts = p.Contracts?.Select(c => new ContractResponse

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Lessee = ToLessesResponse(c.Lessee),

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList(),

HasParkingLot = p.HasParkingLot,

Id = p.Id,

IsAvailable = p.IsAvailable,

Neighborhood = p.Neighborhood,

Price = p.Price,

PropertyImages = p.PropertyImages?.Select(pi => new PropertyImageResponse

{

Id = pi.Id,

ImageUrl = pi.ImageFullPath

}).ToList(),

PropertyType = p.PropertyType.Name,

Remarks = p.Remarks,

Rooms = p.Rooms,

SquareMeters = p.SquareMeters,

Stratum = p.Stratum

}).ToList(),

Contracts = owner.Contracts?.Select(c => new ContractResponse

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList()

};

return Ok(response);

}

private LesseeResponse ToLessesResponse(Lessee lessee)

{

return new LesseeResponse

{

Address = lessee.User.Address,

Document = lessee.User.Document,

Email = lessee.User.Email,

FirstName = lessee.User.FirstName,

LastName = lessee.User.LastName,

PhoneNumber = lessee.User.PhoneNumber

};

}

}

}

1. Test it on postman.

### Properties

1. Add the **FilesHelper** class:

using System.IO;

namespace MyLeasing.Common.Helpers

{

public class FilesHelper

{

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

{

try

{

stream.Position = 0;

var path = Path.Combine(Directory.GetCurrentDirectory(), folder, name);

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

}

catch

{

return false;

}

return true;

}

}

}

1. Add the **PropertyRequest** model:

using System.ComponentModel.DataAnnotations;

namespace MyLeasing.Common.Models

{

public class PropertyRequest

{

public int Id { get; set; }

[Required]

public string Neighborhood { get; set; }

[Required]

public string Address { get; set; }

[Required]

public decimal Price { get; set; }

[Required]

public int SquareMeters { get; set; }

[Required]

public int Rooms { get; set; }

[Required]

public int Stratum { get; set; }

public bool HasParkingLot { get; set; }

public bool IsAvailable { get; set; }

public string Remarks { get; set; }

[Required]

public int PropertyTypeId { get; set; }

[Required]

public int OwnerId { get; set; }

}

}

1. Add the **ImageRequest** model:

namespace MyLeasing.Common.Models

{

public class ImageRequest

{

public int Id { get; set; }

public int PropertyId { get; set; }

public byte[] ImageArray { get; set; }

}

}

1. Add the **PropertiesController** API:

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MyLeasing.Common.Helpers;

using MyLeasing.Common.Models;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Controllers.API

{

[Route("api/[Controller]")]

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class PropertiesController : ControllerBase

{

private readonly DataContext \_dataContext;

public PropertiesController(DataContext dataContext)

{

\_dataContext = dataContext;

}

[HttpPost]

public async Task<IActionResult> PostProperty([FromBody] PropertyRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var owner = await \_dataContext.Owners.FindAsync(request.OwnerId);

if (owner == null)

{

return BadRequest("Not valid owner.");

}

var propertyType = await \_dataContext.PropertyTypes.FindAsync(request.PropertyTypeId);

if (propertyType == null)

{

return BadRequest("Not valid property type.");

}

var property = new Property

{

Address = request.Address,

HasParkingLot = request.HasParkingLot,

IsAvailable = request.IsAvailable,

Neighborhood = request.Neighborhood,

Owner = owner,

Price = request.Price,

PropertyType = propertyType,

Remarks = request.Remarks,

Rooms = request.Rooms,

SquareMeters = request.SquareMeters,

Stratum = request.Stratum

};

\_dataContext.Properties.Add(property);

await \_dataContext.SaveChangesAsync();

return Ok(property);

}

[HttpPost]

[Route("AddImageToProperty")]

public async Task<IActionResult> AddImageToProperty([FromBody] ImageRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var property = await \_dataContext.Properties.FindAsync(request.PropertyId);

if (property == null)

{

return BadRequest("Not valid property.");

}

var imageUrl = string.Empty;

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

{

var stream = new MemoryStream(request.ImageArray);

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

var file = $"{guid}.jpg";

var folder = "wwwroot\\images\\Properties";

var fullPath = $"~/images/Properties/{file}";

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

if (response)

{

imageUrl = fullPath;

}

}

var propertyImage = new PropertyImage

{

ImageUrl = imageUrl,

Property = property

};

\_dataContext.PropertyImages.Add(propertyImage);

await \_dataContext.SaveChangesAsync();

return Ok(propertyImage);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutProperty([FromRoute] int id, [FromBody] PropertyRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != request.Id)

{

return BadRequest();

}

var oldProperty = await \_dataContext.Properties.FindAsync(request.Id);

if (oldProperty == null)

{

return BadRequest("Property doesn't exists.");

}

var propertyType = await \_dataContext.PropertyTypes.FindAsync(request.PropertyTypeId);

if (propertyType == null)

{

return BadRequest("Not valid property type.");

}

oldProperty.Address = request.Address;

oldProperty.HasParkingLot = request.HasParkingLot;

oldProperty.IsAvailable = request.IsAvailable;

oldProperty.Neighborhood = request.Neighborhood;

oldProperty.Price = request.Price;

oldProperty.PropertyType = propertyType;

oldProperty.Remarks = request.Remarks;

oldProperty.Rooms = request.Rooms;

oldProperty.SquareMeters = request.SquareMeters;

oldProperty.Stratum = request.Stratum;

\_dataContext.Properties.Update(oldProperty);

await \_dataContext.SaveChangesAsync();

return Ok(oldProperty);

}

[HttpPost]

[Route("DeleteImageToProperty")]

public async Task<IActionResult> DeleteImageToProperty([FromBody] ImageRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var propertyImage = await \_dataContext.PropertyImages.FindAsync(request.Id);

if (propertyImage == null)

{

return BadRequest("Property image doesn't exist.");

}

\_dataContext.PropertyImages.Remove(propertyImage);

await \_dataContext.SaveChangesAsync();

return Ok(propertyImage);

}

}

}

1. Test it on postman.
2. Add the **PropertyTypesController** API:

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MyLeasing.Web.Data;

using MyLeasing.Web.Data.Entities;

namespace MyLeasing.Web.Controllers.API

{

[Route("api/[controller]")]

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class PropertyTypesController : Controller

{

private readonly DataContext \_dataContext;

public PropertyTypesController(DataContext dataContext)

{

\_dataContext = dataContext;

}

[HttpGet]

public IEnumerable<PropertyType> GetPropertyTypes()

{

return \_dataContext.PropertyTypes.OrderBy(pt => pt.Name);

}

}

}

1. Test it on postman.
2. Finally, publish on Azure.

# App Xamarin Forms Second Part

## Add persistent setting

1. Add the NuGet **Xam.Plugins.Settings** in **Common** project.
2. Add the **Settings** class:

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace MyLeasing.Common.Helpers

{

public static class Settings

{

private const string \_token = "token";

private const string \_owner = "owner";

private static readonly string \_stringDefault = string.Empty;

private static ISettings AppSettings => CrossSettings.Current;

public static string Token

{

get => AppSettings.GetValueOrDefault(\_token, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_token, value);

}

public static string Owner

{

get => AppSettings.GetValueOrDefault(\_owner, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_owner, value);

}

}

}

1. Modify the **LoginViewModel**:

var owner = response2.Result;

Settings.Owner = JsonConvert.SerializeObject(owner);

Settings.Token = JsonConvert.SerializeObject(token);

await \_navigationService.NavigateAsync("PropertiesPage");

IsRunning = false;

IsEnabled = true;

1. Delete the method **OnNavigatedTo** and load the date on **PropertiesViewModel** constructor:

public PropertiesViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Properties";

LoadProperties();

}

...

private void LoadProperties()

{

IsRefreshing = true;

\_token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

\_owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

Properties = new ObservableCollection<PropertyItemViewModel>(\_owner.Properties.Select(p => new PropertyItemViewModel(\_navigationService)

{

Address = p.Address,

Contracts = p.Contracts,

HasParkingLot = p.HasParkingLot,

Id = p.Id,

IsAvailable = p.IsAvailable,

Neighborhood = p.Neighborhood,

Price = p.Price,

PropertyImages = p.PropertyImages,

PropertyType = p.PropertyType,

Remarks = p.Remarks,

Rooms = p.Rooms,

SquareMeters = p.SquareMeters,

Stratum = p.Stratum

}).ToList());

IsRefreshing = false;

}

1. Test it.

## Add a Master Detail

1. Add the **Menu** model:

namespace MyLeasing.Common.Models

{

public class Menu

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

}

}

1. Add the **MenuItemViewModel** view model:

using MyLeasing.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class MenuItemViewModel : Menu

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectMenuCommand;

public MenuItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectMenuCommand => \_selectMenuCommand ?? (\_selectMenuCommand = new DelegateCommand(SelectMenu));

private async void SelectMenu()

{

if (PageName.Equals("Login"))

{

await \_navigationService.NavigateAsync("/NavigationPage/LoginPage");

return;

}

await \_navigationService.NavigateAsync($"/LeasingMasterDetailPage/NavigationPage/{PageName}");

}

}

}

1. Add the images for logo.
2. Add the **MasterDetailPage** called **LeasingMasterDetailPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.LeasingMasterDetailPage"

BackgroundColor="Gray">

<MasterDetailPage.Master>

<ContentPage Title="Menu">

<StackLayout Padding="20">

<Image

HeightRequest="150"

Source="logo"/>

<ListView

BackgroundColor="Transparent"

ItemsSource="{Binding Menus}"

HasUnevenRows="True"

SeparatorVisibility="None">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

HeightRequest="50"

Source="{Binding Icon}"

WidthRequest="50"/>

<Label

Grid.Column="1"

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="White"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

</MasterDetailPage.Master>

</MasterDetailPage>

1. Add the images for menu icons.
2. Modify the **LeasingMasterDetailPageViewModel**:

using MyLeasing.Common.Models;

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

namespace MyLeasing.Prism.ViewModels

{

public class LeasingMasterDetailPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

public LeasingMasterDetailPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

LoadMenus();

}

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

private void LoadMenus()

{

var menus = new List<Menu>

{

new Menu

{

Icon = "ic\_action\_home",

PageName = "PropertiesPage",

Title = "Propierties"

},

new Menu

{

Icon = "ic\_action\_list\_alt",

PageName = "ContractsPage",

Title = "Contracts"

},

new Menu

{

Icon = "ic\_action\_person",

PageName = "ModifyUserPage",

Title = "Modify User"

},

new Menu

{

Icon = "ic\_action\_map",

PageName = "MapPage",

Title = "Map"

},

new Menu

{

Icon = "ic\_action\_exit\_to\_app",

PageName = "LoginPage",

Title = "Log out"

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title

}).ToList());

}

}

}

1. Add the page **MapPage** initially with this layout:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.MapPage"

Title="{Binding Title}">

</ContentPage>

1. Modify the **MapPageViewModel**:

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class MapPageViewModel : ViewModelBase

{

public MapPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Map";

}

}

}

1. Add the page **ModifyUserPage** initially with this layout:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.ModifyUserPage"

Title="{Binding Title}">

</ContentPage>

1. Modify the **ModifyUserPageViewModel**:

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ModifyUserPageViewModel : ViewModelBase

{

public ModifyUserPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Modify User";

}

}

}

1. Modify the **LoginViewModel**:

var owner = response2.Result;

Settings.Owner = JsonConvert.SerializeObject(owner);

Settings.Token = JsonConvert.SerializeObject(token);

await \_navigationService.NavigateAsync("/LeasingMasterDetailPage/NavigationPage/PropertiesPage");

IsRunning = false;

IsEnabled = true;

1. Test it.

## Add Icon & Splash to Xamarin Forms For Android

1. Add a new image with the Splash in drawable, the dimensions are: 480 x 800 pixels. In the sample: **leasing\_splash.png**.
2. Add this lines to **styles.xml**.

</style>

<style name="Theme.Splash" parent="android:Theme">

<item name="android:windowBackground">@drawable/leasing\_splash</item>

<item name="android:windowNoTitle">true</item>

</style>

</resources>

1. In Xamarin Android root project, add the **SplashActivity**.

using Android.App;

using Android.OS;

namespace MyLeasing.Prism.Droid

{

[Activity(

Theme = "@style/Theme.Splash",

MainLauncher = true,

NoHistory = true)]

public class SplashActivity : Activity

{

protected override void OnCreate(Bundle bundle)

{

base.OnCreate(bundle);

System.Threading.Thread.Sleep(1800);

StartActivity(typeof(MainActivity));

}

}

}

1. Modify the **MainActivity** to change **MainLauncher** property to **false**.

[Activity(

Label = "My Leasing",

Icon = "@mipmap/ic\_launcher",

Theme = "@style/MainTheme",

MainLauncher = false,

ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity

1. Test it.
2. Now add the icon launcher. Go to <https://romannurik.github.io/AndroidAssetStudio/> and personalizate your own icon launcher. And add the image to Android and iOS projects.
3. And define the application name in Android Properties.
4. And define the application name in Android Properties.
5. For the iOS see the video it’s all graphical.
6. Test it.

## Adding Styles

1. Add those colors to dictionary:

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">https://MyLeasing.azurewebsites.net</x:String>

<!-- Colors -->

<Color x:Key="colorBackgroud">#F2F2F2</Color>

<Color x:Key="colorPrimary">#0468BF</Color>

<Color x:Key="colorSecondary">#067302</Color>

<Color x:Key="colorDanger">#F2055C</Color>

<Color x:Key="colorAccent">#BF4904</Color>

<Color x:Key="colorFont">#000000</Color>

<Color x:Key="colorFontInverse">#F2F2F2</Color>

</ResourceDictionary>

**Note**: I recommend <https://color.adobe.com/es/explore> to get valid color combinations. In my example I’ve used: <https://color.adobe.com/es/explore?page=2>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Danger | Primary | Secondary | Accent | Background |



1. Modify the **LoginPage.xaml**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.LoginPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Image

Margin="20"

Source="logo"

WidthRequest="230"/>

<Label

TextColor="{StaticResource colorFont}"

Text="Email"/>

<Entry

Keyboard="Email"

Placeholder="Enter your email..."

Text="{Binding Email}"/>

<Label

TextColor="{StaticResource colorFont}"

Text="Password"/>

<Entry

IsPassword="True"

Placeholder="Enter your password..."

Text="{Binding Password}"/>

<StackLayout

VerticalOptions="EndAndExpand">

<Button

BackgroundColor="{StaticResource colorPrimary}"

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Login"

TextColor="{StaticResource colorFontInverse}"/>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource colorSecondary}"

HorizontalOptions="Center"

TextColor="{StaticResource colorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Modify the colors in **styles.xml**:

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

<resources>

<style name="MainTheme" parent="MainTheme.Base">

</style>

<!-- Base theme applied no matter what API -->

<style name="MainTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar">

<!--If you are using revision 22.1 please use just windowNoTitle. Without android:-->

<item name="windowNoTitle">true</item>

<!--We will be using the toolbar so no need to show ActionBar-->

<item name="windowActionBar">false</item>

<!-- Set theme colors from http://www.google.com/design/spec/style/color.html#color-color-palette -->

<!-- colorPrimary is used for the default action bar background -->

<item name="colorPrimary">#0468BF</item>

<!-- colorPrimaryDark is used for the status bar -->

<item name="colorPrimaryDark">#0468BF</item>

<!-- colorAccent is used as the default value for colorControlActivated

which is used to tint widgets -->

<item name="colorAccent">#BF4904</item>

<!-- You can also set colorControlNormal, colorControlActivated

colorControlHighlight and colorSwitchThumbNormal. -->

<item name="windowActionModeOverlay">true</item>

<item name="android:datePickerDialogTheme">@style/AppCompatDialogStyle</item>

</style>

<style name="AppCompatDialogStyle" parent="Theme.AppCompat.Light.Dialog">

<item name="colorAccent">#BF4904</item>

</style>

<style name="Theme.Splash" parent="android:Theme">

<item name="android:windowBackground">@drawable/leasing\_splash</item>

<item name="android:windowNoTitle">true</item>

</style>

</resources>

1. Test it.
2. Add this style to dictionary:

<!-- Styles -->

<Style TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource colorPrimary}" />

<Setter Property="HorizontalOptions" Value="FillAndExpand" />

<Setter Property="TextColor" Value="{StaticResource colorFontInverse}" />

</Style>

<Style TargetType="Label">

<Setter Property="TextColor" Value="{StaticResource colorFont}" />

</Style>

1. Modify the login button on **LoginPage.xaml**:

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Login"/>

1. Test it.
2. Modify the colors in other pages.
3. Add this value to dictionary:

<Style x:Key="SecondaryButton" TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource colorSecondary}" />

</Style>

1. Modify the **LoginPage.xaml**:

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Login"/>

<Button

Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource secondaryButton}"

Text="Register"/>

</StackLayout>

1. Test it.
2. Modify the **LoginPage.xaml**:

<Entry

IsPassword="True"

Placeholder="Enter your password..."

Text="{Binding Password}"/>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal">

<Label

Text="Rememberme in this device"

VerticalOptions="Center"/>

<CheckBox

IsChecked="{Binding IsRemember}"/>

</StackLayout>

<Label

HorizontalOptions="Center"

FontAttributes="Bold"

Text="Forgot your password?"

TextColor="{StaticResource colorPrimary}">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

1. Finally, add the property **IsRemember** to **LoginPageViewModel**:

…

private bool \_isRemember;

…

public bool IsRemember

{

get => \_isRemember;

set => SetProperty(ref \_isRemember, value);

}

…

public LoginViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Login";

IsEnabled = true;

IsRemember = true;

//TODO: Delete those lines

Email = "jzuluaga55@hotmail.com";

Password = "123456";

}

1. Test it.

## Multi Language in Xamarin Forms

1. If you don’t have the ResX Manager Tool, install from: <https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager>
2. In **Prism** project add the folder **Resources** and inside it, add the resource call **Resource**, add some literals and translate with the ResX Manager tool. The default resource language must be Public, the others in no code generation.



1. In common project add the folder **Interfaces**, inside it, add the interface **ILocalize**.

using System.Globalization;

namespace MyLeasing.Prism.Interfaces

{

public interface ILocalize

{

CultureInfo GetCurrentCultureInfo();

void SetLocale(CultureInfo ci);

}

}

1. In the folder **Helpers** add the class **PlatformCulture**.

using System;

namespace MyLeasing.Prism.Helpers

{

public class PlatformCulture

{

public string PlatformString { get; private set; }

public string LanguageCode { get; private set; }

public string LocaleCode { get; private set; }

public PlatformCulture(string platformCultureString)

{

if (string.IsNullOrEmpty(platformCultureString))

{

throw new ArgumentException("Expected culture identifier", "platformCultureString"); // in C# 6 use nameof(platformCultureString)

}

PlatformString = platformCultureString.Replace("\_", "-"); // .NET expects dash, not underscore

var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);

if (dashIndex > 0)

{

var parts = PlatformString.Split('-');

LanguageCode = parts[0];

LocaleCode = parts[1];

}

else

{

LanguageCode = PlatformString;

LocaleCode = "";

}

}

public override string ToString()

{

return PlatformString;

}

}

}

1. In **Prism** project add folder **Helpers**  and add the class **Languages** with the literals.

using MyLeasing.Prism.Interfaces;

using MyLeasing.Prism.Resources;

using Xamarin.Forms;

namespace MyLeasing.Prism.Helpers

{

public static class Languages

{

static Languages()

{

var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();

Resource.Culture = ci;

DependencyService.Get<ILocalize>().SetLocale(ci);

}

public static string Accept => Resource.Accept;

public static string Error => Resource.Error;

public static string EmailError => Resource.EmailError;

}

}

1. Implement the interface in **iOS** in the folder **Implementations**.

using Foundation;

using MyVet.Prism.Helpers;

using MyVet.Prism.Interfaces;

using System.Globalization;

using System.Threading;

using Xamarin.Forms;

[assembly: Dependency(typeof(MyLeasing.Prism.iOS.Implementations.Localize))]

namespace MyLeasing.Prism.iOS.Implementations

{

public class Localize : ILocalize

{

public CultureInfo GetCurrentCultureInfo()

{

var netLanguage = "en";

if (NSLocale.PreferredLanguages.Length > 0)

{

var pref = NSLocale.PreferredLanguages[0];

netLanguage = iOSToDotnetLanguage(pref);

}

// this gets called a lot - try/catch can be expensive so consider caching or something

CultureInfo ci = null;

try

{

ci = new System.Globalization.CultureInfo(netLanguage);

}

catch (CultureNotFoundException)

{

// iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)

// fallback to first characters, in this case "en"

try

{

var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));

ci = new CultureInfo(fallback);

}

catch (CultureNotFoundException)

{

// iOS language not valid .NET culture, falling back to English

ci = new CultureInfo("en");

}

}

return ci;

}

public void SetLocale(CultureInfo ci)

{

Thread.CurrentThread.CurrentCulture = ci;

Thread.CurrentThread.CurrentUICulture = ci;

}

private string iOSToDotnetLanguage(string iOSLanguage)

{

var netLanguage = iOSLanguage;

//certain languages need to be converted to CultureInfo equivalent

switch (iOSLanguage)

{

case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture

case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture

netLanguage = "ms"; // closest supported

break;

case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET culture

netLanguage = "de-CH"; // closest supported

break;

// add more application-specific cases here (if required)

// ONLY use cultures that have been tested and known to work

}

return netLanguage;

}

private string ToDotnetFallbackLanguage(PlatformCulture platCulture)

{

var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);

switch (platCulture.LanguageCode)

{

case "pt":

netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)

break;

case "gsw":

netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app

break;

// add more application-specific cases here (if required)

// ONLY use cultures that have been tested and known to work

}

return netLanguage;

}

}

}

1. Add this lintes into the **info.plist**.

<key>CFBundleLocalizations</key>

<array>

<string>es</string>

<string>pt</string>

</array>

<key>CFBundleDevelopmentRegion</key>

<string>en</string>

1. Implement the interface in **Android** in the folder **Implementations**.

using System.Globalization;

using System.Threading;

using MyVet.Prism.Helpers;

using MyVet.Prism.Interfaces;

using Xamarin.Forms;

[assembly: Dependency(typeof(MyLeasing.Prism.Droid.Implementations.Localize))]

namespace MyLeasing.Prism.Droid.Implementations

{

public class Localize : ILocalize

{

public CultureInfo GetCurrentCultureInfo()

{

var netLanguage = "en";

var androidLocale = Java.Util.Locale.Default;

netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("\_", "-"));

// this gets called a lot - try/catch can be expensive so consider caching or something

CultureInfo ci = null;

try

{

ci = new CultureInfo(netLanguage);

}

catch (CultureNotFoundException)

{

// iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)

// fallback to first characters, in this case "en"

try

{

var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));

ci = new CultureInfo(fallback);

}

catch (CultureNotFoundException)

{

// iOS language not valid .NET culture, falling back to English

ci = new CultureInfo("en");

}

}

return ci;

}

public void SetLocale(CultureInfo ci)

{

Thread.CurrentThread.CurrentCulture = ci;

Thread.CurrentThread.CurrentUICulture = ci;

}

private string AndroidToDotnetLanguage(string androidLanguage)

{

var netLanguage = androidLanguage;

//certain languages need to be converted to CultureInfo equivalent

switch (androidLanguage)

{

case "ms-BN": // "Malaysian (Brunei)" not supported .NET culture

case "ms-MY": // "Malaysian (Malaysia)" not supported .NET culture

case "ms-SG": // "Malaysian (Singapore)" not supported .NET culture

netLanguage = "ms"; // closest supported

break;

case "in-ID": // "Indonesian (Indonesia)" has different code in .NET

netLanguage = "id-ID"; // correct code for .NET

break;

case "gsw-CH": // "Schwiizertüütsch (Swiss German)" not supported .NET culture

netLanguage = "de-CH"; // closest supported

break;

// add more application-specific cases here (if required)

// ONLY use cultures that have been tested and known to work

}

return netLanguage;

}

private string ToDotnetFallbackLanguage(PlatformCulture platCulture)

{

var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);

switch (platCulture.LanguageCode)

{

case "gsw":

netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app

break;

// add more application-specific cases here (if required)

// ONLY use cultures that have been tested and known to work

}

return netLanguage;

}

}

}

1. Modify the **LoginViewModel**:

if (string.IsNullOrEmpty(Email))

{

await Application.Current.MainPage.DisplayAlert(Languages.Error, Languages.EmailMessage, Languages.Accept);

return;

}

1. Test it.
2. Now to translate literals directly in the XAML add the class **TranslateExtension** in folder **Helpers**:

using System;

using System.Globalization;

using System.Reflection;

using System.Resources;

using MyVet.Prism.Interfaces;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace MyLeasing.Prism.Helpers

{

[ContentProperty("Text")]

public class TranslateExtension : IMarkupExtension

{

private readonly CultureInfo ci;

private const string ResourceId = "MyVet.Prism.Resources.Resource";

private static readonly Lazy<ResourceManager> ResMgr =

new Lazy<ResourceManager>(() => new ResourceManager(

ResourceId,

typeof(TranslateExtension).GetTypeInfo().Assembly));

public TranslateExtension()

{

ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();

}

public string Text { get; set; }

public object ProvideValue(IServiceProvider serviceProvider)

{

if (Text == null)

{

return "";

}

var translation = ResMgr.Value.GetString(Text, ci);

if (translation == null)

{

#if DEBUG

throw new ArgumentException(

string.Format(

"Key '{0}' was not found in resources '{1}' for culture '{2}'.",

Text, ResourceId, ci.Name), "Text");

#else

translation = Text; // returns the key, which GETS DISPLAYED TO THE USER

#endif

}

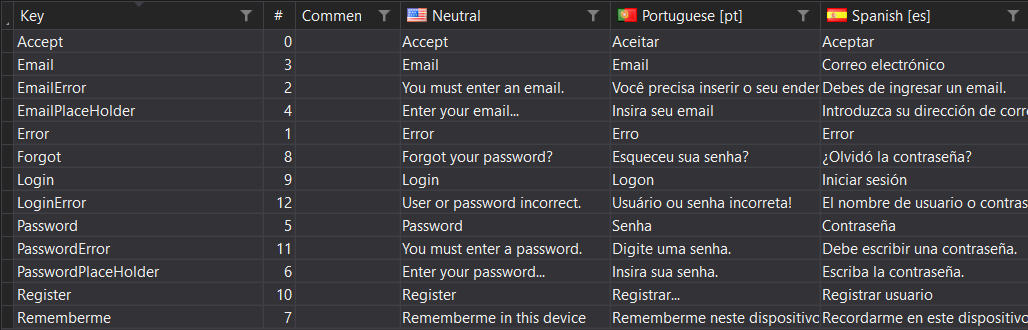
return translation;

}

}

}

1. Complete the literals:



1. And add the properties in **Languages** class:

using MyVet.Prism.Interfaces;

using MyVet.Prism.Resources;

using Xamarin.Forms;

namespace MyLeasing.Prism.Helpers

{

public static class Languages

{

static Languages()

{

var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();

Resource.Culture = ci;

DependencyService.Get<ILocalize>().SetLocale(ci);

}

public static string Accept => Resource.Accept;

public static string Email => Resource.Email;

public static string EmailError => Resource.EmailError;

public static string EmailPlaceHolder => Resource.EmailPlaceHolder;

public static string Error => Resource.Error;

public static string Forgot => Resource.Forgot;

public static string Login => Resource.Login;

public static string LoginError => Resource.LoginError;

public static string Password => Resource.Password;

public static string PasswordError => Resource.PasswordError;

public static string PasswordPlaceHolder => Resource.PasswordPlaceHolder;

public static string Register => Resource.Register;

public static string Rememberme => Resource.Rememberme;

}

}

1. Modify the **LoginVewModel** to complete the translations.

Title = Languages.Login;

...

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.EmailError, Languages.Accept);

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.PasswordError, Languages.Accept);

return;

}

…

if (!response.IsSuccess)

{

IsEnabled = true;

IsRunning = false;

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.LoginError, Languages.Accept);

Password = string.Empty;

return;

}

1. Modify the **LoginPage** for the translations in XAML.

<?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:i18n="clr-namespace:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.Login"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Image

WidthRequest="200"

Source="vet\_logo"/>

<Label

Text="{i18n:Translate Email}"/>

<Entry

Keyboard="Email"

Placeholder="{i18n:Translate EmailPlaceHolder}"

Text="{Binding Email}"/>

<Label

Text="{i18n:Translate Password}"/>

<Entry

IsPassword="True"

Placeholder="{i18n:Translate PasswordPlaceHolder}"

Text="{Binding Password}"/>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal">

<Label

Text="{i18n:Translate Rememberme}"/>

<Switch

IsToggled="{Binding IsRemember}"/>

</StackLayout>

<Label

HorizontalOptions="Center"

Text="{i18n:Translate Forgot}"

TextColor="Navy">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<StackLayout

Orientation="Horizontal">

<Button

Command="{Binding LoginCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Login}"/>

<Button

Command="{Binding RegisterCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="{i18n:Translate Register}"/>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

1. Test it.

## Register users from App

1. Add the **RegisterPage** page:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.RegisterPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

</ContentPage>

1. Modify the **RegisterPageViewModel** class:

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

public RegisterPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Register New Owner";

}

}

}

1. Modify the **LoginPageViewModel** class:

…

private DelegateCommand \_registerCommand;

…

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(Register));

…

private async void Register()

{

await \_navigationService.NavigateAsync("Register");

}

1. Test the navigation.
2. Modify the **RegisterPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.RegisterPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

Text="Document"

VerticalOptions="Center"/>

<Entry

Grid.Row="0"

Grid.Column="1"

Placeholder="Enter your document number..."

Text="{Binding Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="Firstname"

VerticalOptions="Center"/>

<Entry

Grid.Row="1"

Grid.Column="1"

Placeholder="Enter your firstname..."

Text="{Binding FirstName}"/>

<Label

Grid.Row="2"

Grid.Column="0"

Text="Lastname"

VerticalOptions="Center"/>

<Entry

Grid.Row="2"

Grid.Column="1"

Placeholder="Enter your last name..."

Text="{Binding LastName}"/>

<Label

Grid.Row="3"

Grid.Column="0"

Text="Address"

VerticalOptions="Center"/>

<Entry

Grid.Row="3"

Grid.Column="1"

Placeholder="Enter your address..."

Text="{Binding Address}"/>

<Label

Grid.Row="4"

Grid.Column="0"

Text="Email"

VerticalOptions="Center"/>

<Entry

Grid.Row="4"

Grid.Column="1"

Keyboard="Email"

Placeholder="Enter your email..."

Text="{Binding Email}"/>

<Label

Grid.Row="5"

Grid.Column="0"

Text="Phone"

VerticalOptions="Center"/>

<Entry

Grid.Row="5"

Grid.Column="1"

Placeholder="Enter your phone number..."

Text="{Binding Phone}"/>

<Label

Grid.Row="6"

Grid.Column="0"

Text="Register as"

VerticalOptions="Center"/>

<Picker

Grid.Row="6"

Grid.Column="1"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding Roles}"

SelectedItem="{Binding Role}"

Title="Select a role...">

</Picker>

<Label

Grid.Row="7"

Grid.Column="0"

Text="Password"

VerticalOptions="Center"/>

<Entry

Grid.Row="7"

Grid.Column="1"

IsPassword="True"

Placeholder="Enter your password"

Text="{Binding Password}"/>

<Label

Grid.Row="8"

Grid.Column="0"

Text="Password Confirm"

VerticalOptions="Center"/>

<Entry

Grid.Row="8"

Grid.Column="1"

IsPassword="True"

Placeholder="Enter the password confirm..."

Text="{Binding PasswordConfirm}"/>

</Grid>

<Button

Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Register"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource colorSecondary}"

HorizontalOptions="Center"

TextColor="{StaticResource colorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Registering..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Add the **RegexHelper** class:

using System;

using System.Net.Mail;

namespace MyLeasing.Common.Helpers

{

public static class RegexHelper

{

public static bool IsValidEmail(string emailaddress)

{

try

{

var mail = new MailAddress(emailaddress);

return true;

}

catch (FormatException)

{

return false;

}

}

}

}

1. Modify the **RegisterPageViewModel** class:

using System.Collections.ObjectModel;

using System.Threading.Tasks;

using MyLeasing.Common.Helpers;

using MyLeasing.Common.Models;

using MyLeasing.Prism.Helpers;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class RegisterPageViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

public RegisterPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Register new user";

IsEnabled = true;

LoadRoles();

}

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(Register));

public string Document { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Password { get; set; }

public string PasswordConfirm { get; set; }

public Role Role { get; set; }

public ObservableCollection<Role> Roles { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void Register()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Document))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a document.", "Accept");

return false;

}

if (string.IsNullOrEmpty(FirstName))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a firstname.", "Accept");

return false;

}

if (string.IsNullOrEmpty(LastName))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a lastname.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Address))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter an address.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Email) || !RegexHelper.IsValidEmail(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter an email.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Phone))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a phonenumber.", "Accept");

return false;

}

if (Role == null)

{

await App.Current.MainPage.DisplayAlert("Error", "You must seelct a role.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a password.", "Accept");

return false;

}

if (Password.Length < 6)

{

await App.Current.MainPage.DisplayAlert("Error", "The password must have at least 6 charactes length.", "Accept");

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a password confirm.", "Accept");

return false;

}

if (!Password.Equals(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert("Error", "The password and confirm does not match.", "Accept");

return false;

}

return true;

}

private void LoadRoles()

{

Roles = new ObservableCollection<Role>

{

new Role { Id = 2, Name = "Lessee" },

new Role { Id = 1, Name = "Owner" }

};

}

}

}

1. Test it.
2. Verify the API Controller.

1. Add the method **RegisterUserAsync** to interface **IApiService**:

Task<Response<object>> RegisterUserAsync(

string urlBase,

string servicePrefix,

string controller,

UserRequest userRequest);

1. Add the method **RegisterUserAsync** to class **ApiService**:

public async Task<Response<object>> RegisterUserAsync(

string urlBase,

string servicePrefix,

string controller,

UserRequest userRequest)

{

try

{

var request = JsonConvert.SerializeObject(userRequest);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

var url = $"{servicePrefix}{controller}";

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

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

var obj = JsonConvert.DeserializeObject<Response<object>>(answer);

return obj;

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Modify the **RegisterPageViewModel**:

private async void Register()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

var request = new UserRequest

{

Address = Address,

Document = Document,

Email = Email,

FirstName = FirstName,

LastName = LastName,

Password = Password,

Phone = Phone,

RoleId = Role.Id

};

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.RegisterUserAsync(

url,

"/api",

"/Account",

request);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert("Error", response.Message, "Accept");

return;

}

await App.Current.MainPage.DisplayAlert("Ok", response.Message, "Accept");

await \_navigationService.GoBackAsync();

}

1. Modify the **PropertiesPageViewModel**.

private void LoadProperties()

{

\_owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

if (\_owner.RoleId == 1)

{

Title = $"Properties of: {\_owner.FullName}";

}

else

{

Title = "Available Properties";

}

Properties = new ObservableCollection<PropertyItemViewModel>(\_owner.Properties.Select(p => new PropertyItemViewModel(\_navigationService)

1. Test it.

## Recover Password From App in Xamarin Forms

1. Add the litertal for **PasswordRecover**.
2. Add the **RememberPasswordPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.RememberPassword"

Title="{Binding Title}">

</ContentPage>

1. Modify the **RememberPasswordViewModel**:

using MyLeasing.Prism.Helpers;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class RememberPasswordViewModel : ViewModelBase

{

public RememberPasswordViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.PasswordRecover;

}

}

}

1. Modify the **LoginViewModel**:

…

private DelegateCommand \_forgotPasswordCommand;

…

public DelegateCommand ForgotPasswordCommand => \_forgotPasswordCommand ?? (\_forgotPasswordCommand = new DelegateCommand(ForgotPassword));

…

private async void ForgotPassword()

{

await \_navigationService.NavigateAsync("RememberPassword");

}

1. Test the navegation.
2. Modify the **RememberPassword** page:

<?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:i18n="clr-namespace:MyLeasing.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.RememberPassword"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Label

Text="{i18n:Translate Email}"/>

<Entry

Keyboard="Email"

Placeholder="{i18n:Translate EmailPlaceHolder}"

Text="{Binding Email}"/>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<Button

Command="{Binding RecoverCommand}"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate PasswordRecover}"/>

</StackLayout>

</ScrollView>

</ContentPage>

1. Add the method **RecoverPasswordAsync** to **IApiService**:

Task<Response> RecoverPasswordAsync(

string urlBase,

string servicePrefix,

string controller,

EmailRequest emailRequest);

1. Add the method **RecoverPasswordAsync** to **ApiService**:

public async Task<Response> RecoverPasswordAsync(

string urlBase,

string servicePrefix,

string controller,

EmailRequest emailRequest)

{

try

{

var request = JsonConvert.SerializeObject(emailRequest);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

var url = $"{servicePrefix}{controller}";

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

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

var obj = JsonConvert.DeserializeObject<Response>(answer);

return obj;

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Modify the **RememberPasswordPageViewModel**:

using System.Threading.Tasks;

using MyLeasing.Common.Helpers;

using MyLeasing.Common.Models;

using MyLeasing.Common.Services;

using MyLeasing.Prism.Helpers;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class RememberPasswordPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_recoverCommand;

public RememberPasswordPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = Languages.PasswordRecover;

IsEnabled = true;

}

public DelegateCommand RecoverCommand => \_recoverCommand ?? (\_recoverCommand = new DelegateCommand(Recover));

public string Email { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void Recover()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

var request = new EmailRequest

{

Email = Email

};

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.RecoverPasswordAsync(

url,

"/api",

"/Account/RecoverPassword",

request);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

response.Message,

Languages.Accept);

return;

}

await App.Current.MainPage.DisplayAlert(

Languages.Ok,

response.Message,

Languages.Accept);

await \_navigationService.GoBackAsync();

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Email) || !RegexHelper.IsValidEmail(Email))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.EmailError, Languages.Accept);

return false;

}

return true;

}

}

}

1. Test it.

## Remember Me functionality

1. Modify the **Settings** class:

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace MyLeasing.Common.Helpers

{

public static class Settings

{

private const string \_token = "Token";

private const string \_owner = "Owner";

private const string \_isRemembered = "IsRemembered";

private static readonly string \_stringDefault = string.Empty;

private static readonly bool \_boolDefault = false;

private static ISettings AppSettings => CrossSettings.Current;

public static string Token

{

get => AppSettings.GetValueOrDefault(\_token, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_token, value);

}

public static string Owner

{

get => AppSettings.GetValueOrDefault(\_owner, \_stringDefault);

set => AppSettings.AddOrUpdateValue(\_owner, value);

}

public static bool IsRemembered

{

get => AppSettings.GetValueOrDefault(\_isRemembered, \_boolDefault);

set => AppSettings.AddOrUpdateValue(\_isRemembered, value);

}

}

}

1. Modify the **LoginViewModel**:

Settings.Owner = JsonConvert.SerializeObject(owner);

Settings.Token = JsonConvert.SerializeObject(token);

Settings.IsRemembered = IsRemember;

1. Modify the **App.xaml.cs**:

protected override async void OnInitialized()

{

Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("MTU0OTIyQDMxMzcyZTMyMmUzMFR3RzVTejFPMTFkSmg3RTc2K2l3ZlBENkRKeTRlQXFEdmk3MnBLVWtYcUE9"); InitializeComponent();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

if (Settings.IsRemembered && token?.Expiration > DateTime.Now)

{

await NavigationService.NavigateAsync("/LeasingMasterDetailPage/NavigationPage/PropertiesPage");

}

else

{

await NavigationService.NavigateAsync("/NavigationPage/LoginPage");

}

}

1. Modify the **MenuItemViewModel**:

if (PageName.Equals("Login"))

{

Settings.IsRemembered = false;

await \_navigationService.NavigateAsync("/NavigationPage/Login");

return;

}

1. Test it.

## Modify User From App in Xamarin Forms

1. Add a new icon to modify user in menu.
2. Add literals for: **MyProfile**, **MyProperties**, **MyContracts**, **Map** and **Logout**.

public static string Logout => Resource.Logout;

public static string Map => Resource.Map;

public static string MyAgenda => Resource.MyAgenda;

public static string MyPets => Resource.MyPets;

public static string MyProfile => Resource.MyProfile;

1. Add the **Profile** page:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Profile"

Title="{Binding Title}">

</ContentPage>

1. Modify the **ProfileViewModel** class:

using MyLeasing.Prism.Helpers;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ProfileViewModel : ViewModelBase

{

public ProfileViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.MyProfile;

}

}

}

1. Modify the method **LoadMenus** in **MyMasterDetailViewModel**:

private void LoadMenus()

{

var menus = new List<Menu>

{

new Menu

{

Icon = "ic\_home",

PageName = "Properties",

Title = Languages.MyProperties

},

new Menu

{

Icon = "ic\_assignment\_turned\_in",

PageName = "Contrats",

Title = Languages.MyContracts

},

new Menu

{

Icon = "ic\_map",

PageName = "Map",

Title = Languages.Map

},

new Menu

{

Icon = "ic\_assignment\_ind",

PageName = "Profile",

Title = Languages.MyProfile

},

new Menu

{

Icon = "ic\_exit\_to\_app",

PageName = "Login",

Title = Languages.Logout

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title

}).ToList());

}

1. Test the navigation.
2. Add literals for **Save** and **ChangePassword**.

public static string Save => Resource.Save;

public static string ChangePassword => Resource.ChangePassword;

1. Modify the **ModifyUserPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.ModifyUserPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

Text="Document"

VerticalOptions="Center"/>

<Entry

Grid.Row="0"

Grid.Column="1"

Placeholder="Document..."

Text="{Binding Owner.Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="First Name"

VerticalOptions="Center"/>

<Entry

Grid.Row="1"

Grid.Column="1"

Placeholder="First Name..."

Text="{Binding Owner.FirstName}"/>

<Label

Grid.Row="2"

Grid.Column="0"

Text="Last Name"

VerticalOptions="Center"/>

<Entry

Grid.Row="2"

Grid.Column="1"

Placeholder="Last name..."

Text="{Binding Owner.LastName}"/>

<Label

Grid.Row="3"

Grid.Column="0"

Text="Address"

VerticalOptions="Center"/>

<Entry

Grid.Row="3"

Grid.Column="1"

Placeholder="Address..."

Text="{Binding Owner.Address}"/>

<Label

Grid.Row="4"

Grid.Column="0"

Text="Phone"

VerticalOptions="Center"/>

<Entry

Grid.Row="4"

Grid.Column="1"

Keyboard="Telephone"

Placeholder="Phonenumber..."

Text="{Binding Owner.PhoneNumber}"/>

</Grid>

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding SaveCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="Save"/>

<Button

Command="{Binding ChangePasswordCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource secondaryButton}"

Text="Change Password"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource colorDanger}"

HorizontalOptions="Center"

TextColor="{StaticResource colorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Saving..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Modify the **ModifyUserPageViewModel**:

using MyLeasing.Common.Helpers;

using MyLeasing.Common.Models;

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

using System.Threading.Tasks;

namespace MyLeasing.Prism.ViewModels

{

public class ModifyUserPageViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private OwnerResponse \_owner;

private DelegateCommand \_saveCommand;

public ModifyUserPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Modify User";

IsEnabled = true;

Owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

}

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(SaveAsync));

public OwnerResponse Owner

{

get => \_owner;

set => SetProperty(ref \_owner, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void SaveAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(Owner.Document))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must to enter a document.",

"Accept");

return false;

}

if (string.IsNullOrEmpty(Owner.FirstName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must to enter a first name.",

"Accept");

return false;

}

if (string.IsNullOrEmpty(Owner.LastName))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must to enter a last name.",

"Accept");

return false;

}

if (string.IsNullOrEmpty(Owner.Address))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must to enter an address.",

"Accept");

return false;

}

return true;

}

}

}

1. Test it, that we do until this point.
2. Add the method **PutAsync** in **IApiService**:

Task<Response<object>> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken);

1. Add the method **PutAsync** in **ApiService**:

public async Task<Response<object>> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(model);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response<object>

{

IsSuccess = false,

Message = answer,

};

}

var obj = JsonConvert.DeserializeObject<T>(answer);

return new Response<object>

{

IsSuccess = true,

Result = obj,

};

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Add a literal for **UserUpdated**:

public static string UserUpdated => Resource.UserUpdated;

1. Modify the **ProfileViewModel**:

private async void SaveAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

var userRequest = new UserRequest

{

Address = Owner.Address,

Document = Owner.Document,

Email = Owner.Email,

FirstName = Owner.FirstName,

LastName = Owner.LastName,

Password = "123456", // It doesn't matter what is sent here. It is only for the model to be valid

Phone = Owner.PhoneNumber

};

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.PutAsync(

url,

"/api",

"/Account",

userRequest,

"bearer",

token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

return;

}

Settings.Owner = JsonConvert.SerializeObject(Owner);

await App.Current.MainPage.DisplayAlert(

"Ok",

"User updated sucessfully.",

"Accept");

}

1. Test it.

## Modify Password From App in Xamarin Forms

1. Add the method **ChangePasswordAsync** in **IApiService**:

Task<Response<object>> ChangePasswordAsync(

string urlBase,

string servicePrefix,

string controller,

ChangePasswordRequest changePasswordRequest,

string tokenType,

string accessToken);

1. Add the method **ChangePasswordAsync** in **ApiService**:

public async Task<Response<object>> ChangePasswordAsync(

string urlBase,

string servicePrefix,

string controller,

ChangePasswordRequest changePasswordRequest,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(changePasswordRequest);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

var obj = JsonConvert.DeserializeObject<Response<object>>(answer);

return obj;

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Add literals for: **ConfirmNewPassword, ConfirmNewPasswordError, ConfirmNewPasswordPlaceHolder, CurrentPassword, CurrentPasswordError, CurrentPasswordPlaceHolder, NewPassword, NewPasswordError** and **NewPasswordPlaceHolder**.

public static string ConfirmNewPassword => Resource.ConfirmNewPassword;

public static string ConfirmNewPasswordError => Resource.ConfirmNewPasswordError;

public static string ConfirmNewPasswordPlaceHolder => Resource.ConfirmNewPasswordPlaceHolder;

public static string CurrentPassword => Resource.CurrentPassword;

public static string CurrentPasswordError => Resource.CurrentPasswordError;

public static string CurrentPasswordPlaceHolder => Resource.CurrentPasswordPlaceHolder;

public static string NewPassword => Resource.NewPassword;

public static string NewPasswordError => Resource.NewPasswordError;

public static string NewPasswordPlaceHolder => Resource.NewPasswordPlaceHolder;

1. Add the **ChangePasswordPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

x:Class="MyLeasing.Prism.Views.ChangePasswordPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="Current Password"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

IsPassword="True"

Placeholder="Current Password..."

Text="{Binding CurrentPassword}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="New Password"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

IsPassword="True"

Placeholder="New Password..."

Text="{Binding NewPassword}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="Confirm New Password"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="2"

IsPassword="True"

Placeholder="Confirm New Password..."

Text="{Binding PasswordConfirm}"/>

</Grid>

<Button

Command="{Binding ChangePasswordCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Change Password"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource colorDanger}"

HorizontalOptions="Center"

TextColor="{StaticResource colorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Saving..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Modify the **ChangePasswordViewModel**:

using System.Threading.Tasks;

using MyLeasing.Common.Services;

using Prism.Commands;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class ChangePasswordPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_changePasswordCommand;

public ChangePasswordPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

IsEnabled = true;

Title = "Change Password";

}

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePasswordAsync));

public string CurrentPassword { get; set; }

public string NewPassword { get; set; }

public string PasswordConfirm { get; set; }

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

private async void ChangePasswordAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateDataAsync()

{

if (string.IsNullOrEmpty(CurrentPassword))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must enter your current password.",

"Accept");

return false;

}

if (string.IsNullOrEmpty(NewPassword) || NewPassword?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must enter a new password at least 6 characters lenth.",

"Accept");

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"You must enter your a password confim.",

"Accept");

return false;

}

if (!NewPassword.Equals(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"The password and confirmation does not match.",

"Accept");

return false;

}

return true;

}

}

}

1. Modify the **ProfileViewModel**:

private DelegateCommand \_changePasswordCommand;

…

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePassword));

…

private async void ChangePassword()

{

await \_navigationService.NavigateAsync("ChangePassword");

}

1. Test it.
2. Modify the **ChangePasswordViewModel**:

private async void ChangePasswordAsync()

{

var isValid = await ValidateDataAsync();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

var owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var request = new ChangePasswordRequest

{

Email = owner.Email,

NewPassword = NewPassword,

OldPassword = CurrentPassword

};

var url = App.Current.Resources["UrlAPI"].ToString();

var response = await \_apiService.ChangePasswordAsync(

url,

"/api",

"/Account/ChangePassword",

request,

"bearer",

token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Accept");

await \_navigationService.GoBackAsync();

}

1. Test it.

## App Property From App & Accessing Camera and Photo Library

1. Add the icon for toolbar **ic\_action\_add\_circle**.
2. Modify the **Properties** page:

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

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

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Properties"

BackgroundColor="Silver"

Title="{Binding Title}">

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add\_circle" Command="{Binding AddPropertyCommand}"/>

</ContentPage.ToolbarItems>

<StackLayout

1. Add literal for **ErrorNoOwner**:

public static string ErrorNoOwner => Resource.ErrorNoOwner;

1. Modify the **PropertiesViewModel** class:

private DelegateCommand \_addPropertyCommand;

…

public DelegateCommand AddPropertyCommand => \_addPropertyCommand ?? (\_addPropertyCommand = new DelegateCommand(AddProperty));

…

private async void AddProperty()

{

if (\_owner.RoleId != 1)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.ErrorNoOwner, Languages.Accept);

return;

}

await \_navigationService.NavigateAsync("EditProperty");

}

1. Add the **EditPropertyPage**:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.EditProperty"

Title="{Binding Title}">

</ContentPage>

1. Add literal for **NewProperty**:

public static string NewProperty => Resource.NewProperty;

1. Modify the **EditPropertyViewModel** class:

using MyLeasing.Prism.Helpers;

using Prism.Navigation;

namespace MyLeasing.Prism.ViewModels

{

public class EditPropertyViewModel : ViewModelBase

{

public EditPropertyViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.NewProperty;

}

}

}

1. Test it, that we do until this point.
2. Add literals for: **Delete, EditProperty, ChangeImage, Neighborhood, NeighborhoodError, NeighborhoodPlaceHolder, Price, PriceError, PricePlaceHolder, SquareMeters, SquareMetersError, SquareMetersPlaceHolder, Rooms, RoomsError, RoomsPlaceHolder, Stratum, StratumError, StratumPlaceHolder, PropertyType, PropertyTypeError, PropertyTypePlaceHolder, HasParkingLot, IsAvailable** and **Remarks**.

public static string Delete => Resource.Delete;

public static string EditProperty => Resource.EditProperty;

public static string ChangeImage => Resource.ChangeImage;

public static string Neighborhood => Resource.Neighborhood;

public static string NeighborhoodError => Resource.NeighborhoodError;

public static string NeighborhoodPlaceHolder => Resource.NeighborhoodPlaceHolder;

public static string Price => Resource.Price;

public static string PriceError => Resource.PriceError;

public static string PricePlaceHolder => Resource.PricePlaceHolder;

public static string SquareMeters => Resource.SquareMeters;

public static string SquareMetersError => Resource.SquareMetersError;

public static string SquareMetersPlaceHolder => Resource.SquareMetersPlaceHolder;

public static string Rooms => Resource.Rooms;

public static string RoomsError => Resource.RoomsError;

public static string RoomsPlaceHolder => Resource.RoomsPlaceHolder;

public static string Stratum => Resource.Stratum;

public static string StratumError => Resource.StratumError;

public static string StratumPlaceHolder => Resource.StratumPlaceHolder;

public static string PropertyType => Resource.PropertyType;

public static string PropertyTypeError => Resource.PropertyTypeError;

public static string PropertyTypePlaceHolder => Resource.PropertyTypePlaceHolder;

public static string HasParkingLot => Resource.HasParkingLot;

public static string IsAvailable => Resource.IsAvailable;

public static string Remarks => Resource.Remarks;

1. Add an image for no image.
2. Add the style **DangerButton**:

<!-- Colors -->

<Color x:Key="ColorPrimary">#2196F3</Color>

<Color x:Key="ColorPrimaryDark">#1976D2</Color>

<Color x:Key="ColorSecondary">#8D07F6</Color>

<Color x:Key="ColorFontInverse">#FFFFFF</Color>

<Color x:Key="ColorFont">#000000</Color>

<Color x:Key="ColorDanger">#D9042B</Color>

<!-- Styles -->

<Style TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource ColorPrimaryDark}" />

<Setter Property="BorderRadius" Value="23" />

<Setter Property="HeightRequest" Value="46" />

<Setter Property="HorizontalOptions" Value="FillAndExpand" />

<Setter Property="TextColor" Value="{StaticResource ColorFontInverse}" />

</Style>

<Style x:Key="SecondaryButton" TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource ColorSecondary}" />

<Setter Property="TextColor" Value="{StaticResource ColorFontInverse}" />

</Style>

<Style x:Key="DangerButton" TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource ColorDanger}" />

<Setter Property="TextColor" Value="{StaticResource ColorFontInverse}" />

</Style>

1. Modify the **EditProperty** page:

<?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:i18n="clr-namespace:MyLeasing.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.EditProperty"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Image

HeightRequest="150"

Source="{Binding ImageSource}">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Label

FontSize="Micro"

HorizontalOptions="Center"

Text="{i18n:Translate ChangeImage}"/>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate Neighborhood}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

Placeholder="{i18n:Translate NeighborhoodPlaceHolder}"

Text="{Binding Property.Neighborhood}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate Address}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

Placeholder="{i18n:Translate AddressPlaceHolder}"

Text="{Binding Property.Address}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate Price}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="2"

Keyboard="Numeric"

Placeholder="{i18n:Translate PricePlaceHolder}"

Text="{Binding Property.Price}"/>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate SquareMeters}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="3"

Keyboard="Numeric"

Placeholder="{i18n:Translate SquareMetersPlaceHolder}"

Text="{Binding Property.SquareMeters}"/>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate Rooms}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="4"

Keyboard="Numeric"

Placeholder="{i18n:Translate RoomsPlaceHolder}"

Text="{Binding Property.Rooms}"/>

<Label

Grid.Column="0"

Grid.Row="5"

Text="{i18n:Translate Stratum}"

VerticalOptions="Center"/>

<Picker

Grid.Column="1"

Grid.Row="5"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding Stratums}"

SelectedItem="{Binding Stratum}"

Title="{i18n:Translate StratumPlaceHolder}"/>

<Label

Grid.Column="0"

Grid.Row="6"

Text="{i18n:Translate HasParkingLot}"

VerticalOptions="Center"/>

<Switch

Grid.Column="1"

Grid.Row="6"

IsToggled="{Binding Property.HasParkingLot}"/>

<Label

Grid.Column="0"

Grid.Row="7"

Text="{i18n:Translate IsAvailable}"

VerticalOptions="Center"/>

<Switch

Grid.Column="1"

Grid.Row="7"

IsToggled="{Binding Property.IsAvailable}"/>

<Label

Grid.Column="0"

Grid.Row="8"

Text="{i18n:Translate PropertyType}"

VerticalOptions="Center"/>

<Picker

Grid.Column="1"

Grid.Row="8"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding PropertyTypes}"

SelectedItem="{Binding PropertyType}"

Title="{i18n:Translate PropertyTypePlaceHolder}"/>

<Label

Grid.Column="0"

Grid.Row="9"

Text="{i18n:Translate Remarks}"

VerticalOptions="Center"/>

<Editor

Grid.Column="1"

Grid.Row="9"

HeightRequest="80"

Text="{Binding Property.Remarks}"/>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<StackLayout

Orientation="Horizontal">

<Button

Command="{Binding SaveCommand}"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Save}"/>

<Button

Command="{Binding DeleteCommand}"

IsEnabled="{Binding IsEnabled}"

IsVisible="{Binding IsEdit}"

Style="{StaticResource DangerButton}"

Text="{i18n:Translate Delete}"/>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modify the **EditPropertyPageViewModel** class:

using MyLeasing.Common.Models;

using MyLeasing.Prism.Helpers;

using Prism.Navigation;

using Xamarin.Forms;

namespace MyLeasing.Prism.ViewModels

{

public class EditPropertyPageViewModel : ViewModelBase

{

private PropertyResponse \_property;

private ImageSource \_imageSource;

private bool \_isRunning;

private bool \_isEnabled;

private bool \_isEdit;

public EditPropertyPageViewModel(INavigationService navigationService) : base(navigationService)

{

IsEnabled = true;

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEdit

{

get => \_isEdit;

set => SetProperty(ref \_isEdit, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public PropertyResponse Property

{

get => \_property;

set => SetProperty(ref \_property, value);

}

public ImageSource ImageSource

{

get => \_imageSource;

set => SetProperty(ref \_imageSource, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

ImageSource = Property.FirstImage;

IsEdit = true;

Title = Languages.EditProperty;

}

else

{

Title = Languages.NewProperty;

Property = new PropertyResponse { IsAvailable = true };

ImageSource = "noImage";

IsEdit = false;

Title = Languages.NewProperty;

}

}

}

}

1. Test it, that we do until this point.
2. Add those models:

namespace MyLeasing.Common.Models

{

public class PropertyTypeResponse

{

public int Id { get; set; }

public string Name { get; set; }

}

}

namespace MyLeasing.Common.Models

{

public class Stratum

{

public int Id { get; set; }

public string Name { get; set; }

}

}

1. Add the method **GetListAsync** to **IApiService**:

Task<Response> GetListAsync<T>(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken);

1. Add the method **GetListAsync** to **ApiService**:

public async Task<Response> GetListAsync<T>(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken)

{

try

{

var client = new HttpClient

{

BaseAddress = new Uri(urlBase),

};

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

var url = $"{servicePrefix}{controller}";

var response = await client.GetAsync(url);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

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

return new Response

{

IsSuccess = true,

Result = list

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

1. Modify the **EditPropertyViewModel**:

private readonly IApiService \_apiService;

…

private ObservableCollection<PropertyTypeResponse> \_propertyTypes;

private PropertyTypeResponse \_propertyType;

private ObservableCollection<Stratum> \_stratums;

private Stratum \_stratum;

...

public EditPropertyViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_apiService = apiService;

IsEnabled = true;

}

…

public ObservableCollection<PropertyTypeResponse> PropertyTypes

{

get => \_propertyTypes;

set => SetProperty(ref \_propertyTypes, value);

}

public PropertyTypeResponse PropertyType

{

get => \_propertyType;

set => SetProperty(ref \_propertyType, value);

}

public ObservableCollection<Stratum> Stratums

{

get => \_stratums;

set => SetProperty(ref \_stratums, value);

}

public Stratum Stratum

{

get => \_stratum;

set => SetProperty(ref \_stratum, value);

}

…

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

ImageSource = Property.FirstImage;

IsEdit = true;

Title = Languages.EditProperty;

}

else

{

Title = Languages.NewProperty;

Property = new PropertyResponse { IsAvailable = true };

ImageSource = "noImage";

IsEdit = false;

Title = Languages.NewProperty;

}

LoadPropertyTypes();

LoadStratums();

}

private void LoadStratums()

{

Stratums = new ObservableCollection<Stratum>();

for (int i = 1; i <= 6; i++)

{

Stratums.Add(new Stratum { Id = i, Name = $"{i}" });

}

Stratum = Stratums.FirstOrDefault(s => s.Id == Property.Stratum);

}

private async void LoadPropertyTypes()

{

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.GetListAsync<PropertyTypeResponse>(url, "/api", "/PropertyTypes", "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var propertyTypes = (List<PropertyTypeResponse>)response.Result;

PropertyTypes = new ObservableCollection<PropertyTypeResponse>(propertyTypes);

if (!string.IsNullOrEmpty(Property.PropertyType))

{

PropertyType = PropertyTypes.FirstOrDefault(pt => pt.Name == Property.PropertyType);

}

}

1. Test it, that we do until this point.
2. Add listeral for **Contracts**:

public static string Contracts => Resource.Contracts;

1. Modify the **Property** page:

<?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:i18n="clr-namespace:MyLeasing.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Property"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Frame

CornerRadius="20"

HasShadow="True">

<StackLayout>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

HeightRequest="20"

Source="ic\_chevron\_left"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Image

Grid.Column="1"

Source="{Binding Image}"

HeightRequest="300"

Aspect="AspectFill"/>

<Image

Grid.Column="2"

HeightRequest="20"

Source="ic\_chevron\_right"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate PropertyType}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="0"

Text="{Binding Property.PropertyType}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate Neighborhood}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="1"

Text="{Binding Property.Neighborhood}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate Address}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="2"

Text="{Binding Property.Address}"/>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Price}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="3"

Text="{Binding Property.Price, StringFormat='{0:C2}'}"/>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate SquareMeters}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="4"

Text="{Binding Property.SquareMeters, StringFormat='{0:N2}'}"/>

<Label

Grid.Column="0"

Grid.Row="5"

Text="{i18n:Translate Rooms}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="5"

Text="{Binding Property.Rooms}"/>

<Label

Grid.Column="0"

Grid.Row="6"

Text="{i18n:Translate Remarks}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="6"

Text="{Binding Property.Remarks}"/>

</Grid>

</StackLayout>

</Frame>

<Button

Command="{Binding EditPropertyCommand}"

Text="{i18n:Translate EditProperty}"/>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="{i18n:Translate Contracts}"

TextColor="Black"/>

<ListView

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Contracts}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding StartDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding EndDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Lessee.FullName}"

VerticalOptions="Center"/>

<Image

Grid.Column="3"

HeightRequest="20"

Margin="0,5"

Source="ic\_chevron\_right"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Modify the **PropertyViewModel** class:

private DelegateCommand \_editPropertyCommand;

…

public DelegateCommand EditPropertyCommand => \_editPropertyCommand ?? (\_editPropertyCommand = new DelegateCommand(EditProperty));

…

private async void EditProperty()

{

var parameters = new NavigationParameters

{

{ "property", \_property }

};

await \_navigationService.NavigateAsync("EditProperty", parameters);

}

1. Add literals for: **AddImage** and **DeleteImage**:

public static string AddImage => Resource.AddImage;

public static string DeleteImage => Resource.DeleteImage;

1. Modify the **EditProperty**:

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="ic\_chevron\_left"

VerticalOptions="Center"

WidthRequest="40"/>

<StackLayout

Grid.Column="1">

<Image

HeightRequest="150"

Source="{Binding ImageSource}">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</StackLayout>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"

VerticalOptions="Center"

WidthRequest="40"/>

</Grid>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal">

<Button

Command="{Binding AddImageCommand}"

IsVisible="{Binding IsEdit}"

Text="{i18n:Translate AddImage}"/>

<Button

Command="{Binding DeleteImageCommand}"

IsVisible="{Binding IsEdit}"

Style="{StaticResource DangerButton}"

Text="{i18n:Translate DeleteImage}"/>

</StackLayout>

1. Test it.
2. Make a Commit (to be ensured to roll back) and update all the packages in all Xamarin projects:
3. Run the project to check everything is ok.
4. Add the NuGet **Xam.Plugin.Media** in all Xamarin Forms projects:
5. Run the project again, to check everything is ok, and made a commit.
6. Modify the **MainActivity**:

using Android.App;

using Android.Content.PM;

using Android.OS;

using Android.Runtime;

using Plugin.CurrentActivity;

using Plugin.Permissions;

using Prism;

using Prism.Ioc;

namespace MyLeasing.Prism.Droid

{

[Activity(

Label = "My Vet",

Icon = "@mipmap/ic\_launcher",

Theme = "@style/MainTheme",

MainLauncher = false,

ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity

{

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

CrossCurrentActivity.Current.Init(this, bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

LoadApplication(new App(new AndroidInitializer()));

}

public override void OnRequestPermissionsResult(

int requestCode,

string[] permissions,

[GeneratedEnum] Permission[] grantResults)

{

PermissionsImplementation.Current.OnRequestPermissionsResult(

requestCode,

permissions,

grantResults);

}

}

public class AndroidInitializer : IPlatformInitializer

{

public void RegisterTypes(IContainerRegistry containerRegistry)

{

// Register any platform specific implementations

}

}

}

1. Modify the **AndroidManifest**:

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.zulusoftware.MyLeasing" android:installLocation="auto">

<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application android:label="My Vet" android:icon="@mipmap/ic\_launcher">

<provider android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.fileprovider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths"></meta-data>

</provider>

</application>

</manifest>

1. Add the folder **xml** inside **Resources** and inside it, add the **file\_paths.xml**:

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

<paths xmlns:android="http://schemas.android.com/apk/res/android">

<external-files-path name="my\_images" path="Pictures" />

<external-files-path name="my\_movies" path="Movies" />

</paths>

1. Modify the **info.plist**:

<key>NSCameraUsageDescription</key>

<string>This app needs access to the camera to take photos.</string>

<key>NSPhotoLibraryUsageDescription</key>

<string>This app needs access to photos.</string>

<key>NSMicrophoneUsageDescription</key>

<string>This app needs access to microphone.</string>

<key>NSPhotoLibraryAddUsageDescription</key>

<string>This app needs access to the photo gallery.</string>

1. Add those literals: **PictureSource, Cancel, FromCamera** and **FromGallery**:

public static string PictureSource => Resource.PictureSource;

public static string Cancel => Resource.Cancel;

public static string FromCamera => Resource.FromCamera;

public static string FromGallery => Resource.FromGallery;

1. Modify in **EditPropertyViewModel**:

private MediaFile \_file;

private DelegateCommand \_changeImageCommand;

…

public DelegateCommand ChangeImageCommand => \_changeImageCommand ?? (\_changeImageCommand = new DelegateCommand(ChangeImage));

…

private async void ChangeImage()

{

await CrossMedia.Current.Initialize();

var source = await Application.Current.MainPage.DisplayActionSheet(

Languages.PictureSource,

Languages.Cancel,

null,

Languages.FromGallery,

Languages.FromCamera);

if (source == Languages.Cancel)

{

\_file = null;

return;

}

if (source == Languages.FromCamera)

{

\_file = await CrossMedia.Current.TakePhotoAsync(

new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

}

);

}

else

{

\_file = await CrossMedia.Current.PickPhotoAsync();

}

if (\_file != null)

{

this.ImageSource = ImageSource.FromStream(() =>

{

var stream = \_file.GetStream();

return stream;

});

}

}

1. Test it, that we do until this point.
2. Fix the **Owners** API controller to send the owner Id:

var response = new OwnerResponse

{

Id = owner.Id,

FirstName = owner.User.FirstName,

1. Add the method **GetLastPropertyByOwnerId** to **Properties** API controller:

[HttpGet("GetLastPropertyByOwnerId/{id}")]

public async Task<IActionResult> GetLastPropertyByOwnerId([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var owner = await \_dataContext.Owners

.Include(o => o.Properties)

.ThenInclude(p => p.PropertyType)

.FirstOrDefaultAsync(o => o.Id == id);

if (owner == null)

{

return NotFound();

}

var property = owner.Properties.LastOrDefault();

var response = new PropertyResponse

{

Address = property.Address,

HasParkingLot = property.HasParkingLot,

Id = property.Id,

IsAvailable = property.IsAvailable,

Neighborhood = property.Neighborhood,

Price = property.Price,

PropertyType = property.PropertyType.Name,

Remarks = property.Remarks,

Rooms = property.Rooms,

SquareMeters = property.SquareMeters,

Stratum = property.Stratum

};

return Ok(response);

}

1. Re-publish on Azure.
2. Add literals for **CreateEditPropertyConfirm**, **Created** and **Edited**:

public static string CreateEditPropertyConfirm => Resource.CreateEditPropertyConfirm;

public static string Created => Resource.Created;

public static string Edited => Resource.Edited;

1. Add the method **ReadFully** to **FilesHelper**:

public static byte[] ReadFully(Stream input)

{

using (MemoryStream ms = new MemoryStream())

{

input.CopyTo(ms);

return ms.ToArray();

}

}

1. Add those methods to **IApiService**:

Task<Response> PostAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken);

Task<Response> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

int id,

T model,

string tokenType,

string accessToken);

Task<Response> GetLastPropertyByOwnerId(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

int ownerId);

1. Add those methods to **ApiService**:

public async Task<Response> PostAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(model);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

int id,

T model,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(model);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}/{id}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> GetLastPropertyByOwnerId(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken,

int ownerId)

{

try

{

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}/{ownerId}";

var response = await client.GetAsync(url);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

var property = JsonConvert.DeserializeObject<PropertyResponse>(result);

return new Response

{

IsSuccess = true,

Result = property

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message

};

}

}

1. Change the **EditPropertyViewModel**:

private DelegateCommand \_saveCommand;

…

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(Save));

…

private async void Save()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

IsRunning = true;

IsEnabled = false;

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

var propertyRequest = new PropertyRequest

{

Address = Property.Address,

HasParkingLot = Property.HasParkingLot,

Id = Property.Id,

IsAvailable = Property.IsAvailable,

Neighborhood = Property.Neighborhood,

OwnerId = owner.Id,

Price = Property.Price,

PropertyTypeId = PropertyType.Id,

Remarks = Property.Remarks,

Rooms = Property.Rooms,

SquareMeters = Property.SquareMeters,

Stratum = Stratum.Id

};

Response response;

if (IsEdit)

{

response = await \_apiService.PutAsync(url, "/api", "/Properties", propertyRequest.Id, propertyRequest, "bearer", token.Token);

}

else

{

response = await \_apiService.PostAsync(url, "/api", "/Properties", propertyRequest, "bearer", token.Token);

}

byte[] imageArray = null;

if (\_file != null)

{

imageArray = FilesHelper.ReadFully(\_file.GetStream());

if (Property.Id == 0)

{

var response2 = await \_apiService.GetLastPropertyByOwnerId(url, "/api", "/Properties/GetLastPropertyByOwnerId", "bearer", token.Token, owner.Id);

if (response2.IsSuccess)

{

var property = (PropertyResponse)response2.Result;

Property.Id = property.Id;

}

}

if (Property.Id != 0)

{

var imageRequest = new ImageRequest

{

PropertyId = Property.Id,

ImageArray = imageArray

};

var response3 = await \_apiService.PostAsync(url, "/api", "/Properties/AddImageToProperty", imageRequest, "bearer", token.Token);

if (!response3.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(Languages.Error, response3.Message, Languages.Accept);

}

}

}

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

await PropertiesViewModel.GetInstance().UpdateOwner();

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

Languages.Error,

string.Format(Languages.CreateEditPropertyConfirm, IsEdit ? Languages.Edited : Languages.Created),

Languages.Accept);

await \_navigationService.GoBackToRootAsync();

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Property.Neighborhood))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.NeighborhoodError, Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(Property.Address))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddressError, Languages.Accept);

return false;

}

if (Property.Price <= 0)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.PriceError, Languages.Accept);

return false;

}

if (Property.SquareMeters <= 0)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.SquareMetersError, Languages.Accept);

return false;

}

if (Property.Rooms <= 0)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.RoomsError, Languages.Accept);

return false;

}

if (Stratum == null)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.StratumError, Languages.Accept);

return false;

}

if (PropertyType == null)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.PropertyTypeError, Languages.Accept);

return false;

}

return true;

}

1. To fix the update, modify the **PetsViewModel**:

private static PropertiesViewModel \_instance;

....

public PropertiesViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_instance = this;

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Properties";

LoadProperties();

}

…

public static PropertiesViewModel GetInstance()

{

return \_instance;

}

public async Task UpdateOwner()

{

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.GetOwnerByEmail(

url,

"/api",

"/Owners/GetOwnerByEmail",

"bearer",

token.Token,

\_owner.Email);

if (response.IsSuccess)

{

var owner = (OwnerResponse)response.Result;

Settings.Owner = JsonConvert.SerializeObject(owner);

\_owner = owner;

LoadProperties();

}

}

1. Modify the **EditPropertyViewModel**:

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

await PropertiesViewModel.GetInstance().UpdateOwner();

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(

Languages.Error,

string.Format(Languages.CreateEditPropertyConfirm, IsEdit ? Languages.Edited : Languages.Created),

Languages.Accept);

await \_navigationService.GoBackToRootAsync();

1. Test it, that we do until this point.
2. Add the method **DeleteProperty** to **Properties** API controller:

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteProperty([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return this.BadRequest(ModelState);

}

var property = await \_dataContext.Properties

.Include(p => p.Contracts)

.Include(p => p.PropertyImages)

.FirstOrDefaultAsync(p => p.Id == id);

if (property == null)

{

return this.NotFound();

}

if (property.Contracts.Count > 0)

{

BadRequest("The property can't be deleted because it has contracts.");

}

\_dataContext.PropertyImages.RemoveRange(property.PropertyImages);

\_dataContext.Properties.Remove(property);

await \_dataContext.SaveChangesAsync();

return Ok("Property deleted");

}

1. Re-publish on Azure.
2. Add listerals for: **Confirm**, **QuestionToDeleteProperty**, **Yes** and **No**:

public static string Confirm => Resource.Confirm;

public static string QuestionToDeleteProperty => Resource.QuestionToDeleteProperty;

public static string Yes => Resource.Yes;

public static string No => Resource.No;

1. Add the method **DeleteAsync** to **IApiService**:

Task<Response> DeleteAsync(

string urlBase,

string servicePrefix,

string controller,

int id,

string tokenType,

string accessToken);

1. Add the method **DeleteAsync** to **ApiService**:

public async Task<Response> DeleteAsync(

string urlBase,

string servicePrefix,

string controller,

int id,

string tokenType,

string accessToken)

{

try

{

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}/{id}";

var response = await client.DeleteAsync(url);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

return new Response

{

IsSuccess = true

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Modify the **EditPropertyViewModel**:

private DelegateCommand \_deleteCommand;

…

public DelegateCommand DeleteCommand => \_deleteCommand ?? (\_deleteCommand = new DelegateCommand(Delete));

…

private async void Delete()

{

var answer = await App.Current.MainPage.DisplayAlert(

Languages.Confirm,

Languages.QuestionToDeletePet,

Languages.Yes,

Languages.No);

if (!answer)

{

return;

}

IsRunning = true;

IsEnabled = false;

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.DeleteAsync(url, "/api", "/Pets", Pet.Id, "bearer", token.Token);

if (!response.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

await PetsViewModel.GetInstance().UpdateOwner();

IsRunning = false;

IsEnabled = true;

await \_navigationService.GoBackToRootAsync();

}

1. Test it.
2. To add a refresh functionality, modify the **Properties** page:

<ListView

HasUnevenRows="True"

SeparatorVisibility="None"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshPropertiesCommand}"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Pets}">

1. And modify the **PropertiesViewModel**:

private DelegateCommand \_refreshPropertiesCommand;

....

public DelegateCommand RefreshPropertiesCommand => \_refreshPropertiesCommand ?? (\_refreshPropertiesCommand = new DelegateCommand(RefreshProperties));

…

private async void RefreshPets()

{

IsRefreshing = true;

await UpdateOwner();

IsRefreshing = false;

}

1. Test it.
2. To show the image a little bigger, modify the **Property** page:

<?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:i18n="clr-namespace:MyLeasing.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Property"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Frame

CornerRadius="20"

HasShadow="True">

<StackLayout>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Image

Grid.Column="0"

HeightRequest="40"

Source="ic\_chevron\_left"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Image

Grid.Column="1"

Source="{Binding Image}"

HeightRequest="150"

Aspect="AspectFill"/>

<Image

Grid.Column="2"

HeightRequest="40"

Source="ic\_chevron\_right"

VerticalOptions="Center">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate PropertyType}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="0"

Text="{Binding Property.PropertyType}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate Neighborhood}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="1"

Text="{Binding Property.Neighborhood}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate Address}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="2"

Text="{Binding Property.Address}"/>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Price}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="3"

Text="{Binding Property.Price, StringFormat='{0:C2}'}"/>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate SquareMeters}"

FontAttributes="Bold"/>

<Label

Grid.Column="1"

Grid.Row="4"

Text="{Binding Property.SquareMeters, StringFormat='{0:N2}'}"/>

</Grid>

</StackLayout>

</Frame>

<Button

Command="{Binding EditPropertyCommand}"

Text="{i18n:Translate EditProperty}"/>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="{i18n:Translate Contracts}"

TextColor="Black"/>

<ListView

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Contracts}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding StartDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding EndDate, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Lessee.FullName}"

VerticalOptions="Center"/>

<Image

Grid.Column="3"

HeightRequest="20"

Margin="0,5"

Source="ic\_chevron\_right"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Fix the method OnNavigatedTo on PropertyViewModel, to show the images for new properties:

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("property"))

{

Property = parameters.GetValue<PropertyResponse>("property");

Title = Property.Neighborhood;

Image = Property.FirstImage;

if (Property.Contracts != null)

{

Contracts = new ObservableCollection<ContractItemViewModel>(Property.Contracts.Select(c => new ContractItemViewModel(\_navigationService)

{

EndDate = c.EndDate,

Id = c.Id,

IsActive = c.IsActive,

Lessee = c.Lessee,

Price = c.Price,

Remarks = c.Remarks,

StartDate = c.StartDate

}).ToList());

}

}

}

1. Test it.
2. Add the literals: **AddImageError1**, **AddImageError2** and **AddImageConfirm**:

public static string AddImageError1 => Resource.AddImageError1;

public static string AddImageError2 => Resource.AddImageError2;

public static string AddImageConfirm => Resource.AddImageConfirm;

1. Modify the **EditPropertyViewModel**:

private DelegateCommand \_addImageCommand;

…

public DelegateCommand AddImageCommand => \_addImageCommand ?? (\_addImageCommand = new DelegateCommand(AddImage));

…

private async void AddImage()

{

if (\_file == null)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddImageError1, Languages.Accept);

return;

}

if (Property.Id == 0)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddImageError2, Languages.Accept);

return;

}

IsRunning = true;

IsEnabled = false;

var imageArray = FilesHelper.ReadFully(\_file.GetStream());

var imageRequest = new ImageRequest

{

PropertyId = Property.Id,

ImageArray = imageArray

};

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.PostAsync(url, "/api", "/Properties/AddImageToProperty", imageRequest, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

await App.Current.MainPage.DisplayAlert(Languages.Ok, Languages.AddImageConfirm, Languages.Accept);

\_file = null;

}

1. Test it.
2. Modify the EditProperty **page**:

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="ic\_chevron\_left"

VerticalOptions="Center"

WidthRequest="40">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<StackLayout

Grid.Column="1">

<Image

HeightRequest="150"

Source="{Binding ImageSource}">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</StackLayout>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"

VerticalOptions="Center"

WidthRequest="40">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</Grid>

1. Modify the **EditPropertyViewModel**:

private int \_positionImage;

private DelegateCommand \_previousImageCommand;

private DelegateCommand \_nextImageCommand;

…

public DelegateCommand PreviousImageCommand => \_previousImageCommand ?? (\_previousImageCommand = new DelegateCommand(MovePreviousImage));

public DelegateCommand NextImageCommand => \_nextImageCommand ?? (\_nextImageCommand = new DelegateCommand(MoveNextImage));

…

private void MoveNextImage()

{

MoveImage(1);

}

private void MovePreviousImage()

{

MoveImage(-1);

}

private void MoveImage(int delta)

{

if (Property.PropertyImages == null || Property.PropertyImages.Count <= 1)

{

return;

}

\_positionImage += delta;

if (\_positionImage < 0)

{

\_positionImage = Property.PropertyImages.Count - 1;

}

if (\_positionImage > Property.PropertyImages.Count - 1)

{

\_positionImage = 0;

}

ImageSource = Property.PropertyImages.ToList()[\_positionImage].ImageUrl;

}

1. Add literals for: **DeleteImageConfirmation** and **ImageDeletedMessage**:

public static string DeleteImageConfirmation => Resource.DeleteImageConfirmation;

public static string ImageDeletedMessage => Resource.ImageDeletedMessage;

1. Modify the **EditPropertyViewModel**:

private DelegateCommand \_deleteImageCommand;

…

public DelegateCommand DeleteImageCommand => \_deleteImageCommand ?? (\_deleteImageCommand = new DelegateCommand(DeleteImage));

…

private async void DeleteImage()

{

if (Property.PropertyImages.Count == 0)

{

return;

}

var answer = await App.Current.MainPage.DisplayAlert(Languages.Confirm, Languages.DeleteImageConfirmation, Languages.Yes, Languages.No);

if (!answer)

{

return;

}

IsRunning = true;

IsEnabled = false;

var imageRequest = new ImageRequest

{

Id = Property.PropertyImages.ToList()[\_positionImage].Id,

PropertyId = Property.Id,

};

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.PostAsync(url, "/api", "/Properties/DeleteImageToProperty", imageRequest, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

await App.Current.MainPage.DisplayAlert(Languages.Ok, Languages.ImageDeletedMessage, Languages.Accept);

await PropertiesViewModel.GetInstance().UpdateOwner();

await \_navigationService.GoBackToRootAsync();

}

1. Test it.

## Show the map and move to current location

1. Update the NuGet **Xamarin.Forms** packages in all mobility projects.
2. Add the NuGet **Xamarin.Forms.Maps** to all mobility projects.
3. Add the NuGet **Xam.Plugin.Geolocator** to all mobility projects and **Common** project.
4. Get a key for your maps in Google Service: https://developers.google.com/maps/?hl=es-419

1. Modify the **AndroidManifest.xml** (replace the key for your own map key):

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.zulusoftware.MyLeasing" android:installLocation="auto">

<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.ACCESS\_MOCK\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<application android:label="My Vet" android:icon="@mipmap/ic\_launcher">

<meta-data

android:name="com.google.android.maps.v2.API\_KEY"

android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

<provider android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.fileprovider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths"></meta-data>

</provider>

</application>

</manifest>

1. Modify the **info.plist**:

<key>NSLocationAlwaysUsageDescription</key>

<string>Can we use your location at all times?</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>Can we use your location when your app is being used?</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>Can we use your location at all times?</string>

1. Modify the **MainActivity**:

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

CrossCurrentActivity.Current.Init(this, bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

Xamarin.FormsMaps.Init(this, bundle);

LoadApplication(new App(new AndroidInitializer()));

}

1. Modify the **AppDelegate**:

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

Xamarin.FormsMaps.Init();

LoadApplication(new App(new iOSInitializer()));

return base.FinishedLaunching(app, options);

}

1. Modify the **Maps** page:

<?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:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyLeasing.Prism.Views.Map"

Title="{Binding Title}">

<StackLayout>

<maps:Map

x:Name="MyMap"

IsShowingUser="true"

MapType="Street"/>

</StackLayout>

</ContentPage>

1. Test it.
2. Add the **IGeolocatorService**:

using System.Threading.Tasks;

namespace MyLeasing.Common.Services

{

public interface IGeolocatorService

{

double Latitude { get; set; }

double Longitude { get; set; }

Task GetLocationAsync();

}

}

1. Add the **GeolocatorService**:

using System;

using System.Threading.Tasks;

using Plugin.Geolocator;

namespace MyLeasing.Common.Services

{

public class GeolocatorService : IGeolocatorService

{

public double Latitude { get; set; }

public double Longitude { get; set; }

public async Task GetLocationAsync()

{

try

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync();

Latitude = location.Latitude;

Longitude = location.Longitude;

}

catch (Exception ex)

{

ex.ToString();

}

}

}

}

1. Setup the injection for the new service on **App.xaml.cs**:

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.Register<IGeolocatorService, GeolocatorService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<Login, LoginViewModel>();

containerRegistry.RegisterForNavigation<Pets, PetsViewModel>();

containerRegistry.RegisterForNavigation<Pet, PetViewModel>();

containerRegistry.RegisterForNavigation<History, HistoryViewModel>();

containerRegistry.RegisterForNavigation<MyMasterDetail, MyMasterDetailViewModel>();

containerRegistry.RegisterForNavigation<Agenda, AgendaViewModel>();

containerRegistry.RegisterForNavigation<Map, MapViewModel>();

containerRegistry.RegisterForNavigation<Register, RegisterViewModel>();

containerRegistry.RegisterForNavigation<RememberPassword, RememberPasswordViewModel>();

containerRegistry.RegisterForNavigation<Profile, ProfileViewModel>();

containerRegistry.RegisterForNavigation<ChangePassword, ChangePasswordViewModel>();

containerRegistry.RegisterForNavigation<EditPet, EditPetViewModel>();

containerRegistry.RegisterForNavigation<AssignModifyAgenda, AssignModifyAgendaViewModel>();

}

1. Modify the **Map.xaml.cs**:

using MyLeasing.Common.Services;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace MyLeasing.Prism.Views

{

public partial class Map : ContentPage

{

private readonly IGeolocatorService \_geolocatorService;

public Map(IGeolocatorService geolocatorService)

{

InitializeComponent();

\_geolocatorService = geolocatorService;

MoveMapToCurrentPositionAsync();

}

private async void MoveMapToCurrentPositionAsync()

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

var position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(

position,

Distance.FromKilometers(.5)));

}

}

}

}

1. Test it.

## Put pins in map

1. Modify the entity **Property** to storage latitude and longitude:

public string Remarks { get; set; }

[DisplayFormat(DataFormatString = "{0:N6}")]

public double Latitude { get; set; }

[DisplayFormat(DataFormatString = "{0:N6}")]

public double Longitude { get; set; }

public PropertyType PropertyType { get; set; }

1. Save and run the command to add the new migration and update the database:

PM> add-migration AddLatLong

PM> update-database

1. Modify the **Details** Owners view to show the new fields:

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().IsAvailable)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Latitude)

</th>

<th>

@Html.DisplayNameFor(model => model.Properties.FirstOrDefault().Longitude)

</th>

<th>

Images

</th>

…

<td>

@Html.DisplayFor(modelItem => item.IsAvailable)

</td>

<td>

@Html.DisplayFor(modelItem => item.Latitude)

</td>

<td>

@Html.DisplayFor(modelItem => item.Longitude)

</td>

<td>

@Html.DisplayFor(modelItem => item.PropertyImages.Count)

</td>

1. Modify the partial shared view **\_Property** to add the new properties:

<div class="form-group">

<label asp-for="Remarks" class="control-label"></label>

<textarea asp-for="Remarks" class="form-control"></textarea>

<span asp-validation-for="Remarks" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Latitude" class="control-label"></label>

<input asp-for="Latitude" class="form-control" />

<span asp-validation-for="Latitude" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Longitude" class="control-label"></label>

<input asp-for="Longitude" class="form-control" />

<span asp-validation-for="Longitude" class="text-danger"></span>

</div>

1. Modify the method **ToPropertyAsync**:

public async Task<Property> ToPropertyAsync(PropertyViewModel model)

{

return new Property

{

Address = model.Address,

HasParkingLot = model.HasParkingLot,

IsAvailable = model.IsAvailable,

Neighborhood = model.Neighborhood,

Price = model.Price,

Rooms = model.Rooms,

SquareMeters = model.SquareMeters,

Stratum = model.Stratum,

Owner = await \_dataContext.Owners.FindAsync(model.OwnerId),

PropertyType = await \_dataContext.PropertyTypes.FindAsync(model.PropertyTypeId),

Remarks = model.Remarks,

Latitude = model.Latitude,

Longitude = model.Longitude

};

}

1. Modify the method **ToPropertyViewModel**:

public PropertyViewModel ToPropertyViewModel(Property property)

{

return new PropertyViewModel

{

Address = property.Address,

HasParkingLot = property.HasParkingLot,

Id = property.Id,

IsAvailable = property.IsAvailable,

Neighborhood = property.Neighborhood,

Price = property.Price,

Rooms = property.Rooms,

SquareMeters = property.SquareMeters,

Stratum = property.Stratum,

Owner = property.Owner,

OwnerId = property.Owner.Id,

PropertyType = property.PropertyType,

PropertyTypeId = property.PropertyType.Id,

PropertyTypes = \_combosHelper.GetComboPropertyTypes(),

Remarks = property.Remarks,

Latitude = property.Latitude,

Longitude = property.Longitude

};

}

1. Test it, and set positions near to latitude 6 and longitude -75.
2. Add the properties to **PropertyResponse**:

public string PropertyType { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public ICollection<PropertyImageResponse> PropertyImages { get; set; }

1. Add this method to **OwnersController** API:

[HttpGet]

[Route("GetAvailbleProperties")]

public async Task<IActionResult> GetAvailbleProperties()

{

var properties = await \_dataContext.Properties

.Include(p => p.PropertyType)

.Include(p => p.PropertyImages)

.Where(p => p.IsAvailable)

.ToListAsync();

var response = new List<PropertyResponse>(properties.Select(p => new PropertyResponse

{

Address = p.Address,

HasParkingLot = p.HasParkingLot,

Id = p.Id,

IsAvailable = p.IsAvailable,

Latitude = p.Latitude,

Longitude = p.Longitude,

Neighborhood = p.Neighborhood,

Price = p.Price,

PropertyImages = new List<PropertyImageResponse>(p.PropertyImages.Select(pi => new PropertyImageResponse

{

Id = pi.Id,

ImageUrl = pi.ImageFullPath

}).ToList()),

PropertyType = p.PropertyType.Name,

Remarks = p.Remarks,

Rooms = p.Rooms,

SquareMeters = p.SquareMeters,

Stratum = p.Stratum

}).ToList());

return Ok(response);

}

1. Publish the changes on Azure.
2. Modify the **Map.xaml.cs**:

private readonly IGeolocatorService \_geolocatorService;

…

public Map(

IGeolocatorService geolocatorService,

IApiService apiService)

{

InitializeComponent();

\_geolocatorService = geolocatorService;

\_apiService = apiService;

MoveMapToCurrentPositionAsync();

ShowOwners();

}

…

private async void ShowOwners()

{

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.GetListAsync<OwnerResponse>(url, "api", "/Owners", "bearer", token.Token);

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var owners = (List<OwnerResponse>)response.Result;

foreach (var owner in owners)

{

MyMap.Pins.Add(new Pin

{

Address = owner.Address,

Label = owner.FullName,

Position = new Position(owner.Latitude, owner.Longitude),

Type = PinType.Place

});

}

}

1. Test it.
2. Add literals for:

public static string NotAddressFound => Resource.NotAddressFound;

public static string NotLocationAvailable => Resource.NotLocationAvailable;

public static string SelectAnAdrress => Resource.SelectAnAdrress;

1. Modify the **RegisterViewModel**:

private readonly IGeolocatorService \_geolocatorService;

private Position \_position;

private string \_address;

…

public RegisterViewModel(

INavigationService navigationService,

IApiService apiService,

IGeolocatorService geolocatorService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

\_geolocatorService = geolocatorService;

Title = Languages.Register;

IsEnabled = true;

}

…

public string Address

{

get => \_address;

set => SetProperty(ref \_address, value);

}

…

var request = new UserRequest

{

Address = Address,

Document = Document,

Email = Email,

FirstName = FirstName,

LastName = LastName,

Password = Password,

Phone = Phone,

Latitude = \_position.Latitude,

Longitude = \_position.Longitude

};

…

if (string.IsNullOrEmpty(Address))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddressError, Languages.Accept);

return false;

}

var isValidAddress = await ValidateAddressAsync();

if (!isValidAddress)

{

return false;

}

…

private async Task<bool> ValidateAddressAsync()

{

var geoCoder = new Geocoder();

var locations = await geoCoder.GetPositionsForAddressAsync(Address);

var locationList = locations.ToList();

if (locationList.Count == 0)

{

var response = await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.NotAddressFound,

Languages.Yes,

Languages.No);

if (response)

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

\_position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

var list = await geoCoder.GetAddressesForPositionAsync(\_position);

Address = list.FirstOrDefault();

return true;

}

else

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.NotLocationAvailable,

Languages.Accept);

return false;

}

}

else

{

return false;

}

}

if (locationList.Count == 1)

{

\_position = locationList.FirstOrDefault();

return true;

}

if (locationList.Count > 1)

{

var addresses = new List<Address>();

var names = new List<string>();

foreach (var location in locationList)

{

var list = await geoCoder.GetAddressesForPositionAsync(location);

names.AddRange(list);

foreach (var item in list)

{

addresses.Add(new Address

{

Name = item,

Latitude = location.Latitude,

Longitude = location.Longitude

});

}

}

var source = await App.Current.MainPage.DisplayActionSheet(

Languages.SelectAnAdrress,

Languages.Cancel,

null,

names.ToArray());

if (source == Languages.Cancel)

{

return false;

}

Address = source;

var address = addresses.FirstOrDefault(a => a.Name == source);

\_position = new Position(address.Latitude, address.Longitude);

}

return true;

}

1. Test it.
2. Modify the **ProfileViewModel**:

private readonly IGeolocatorService \_geolocatorService;

private Position \_position;

…

public ProfileViewModel(

INavigationService navigationService,

IApiService apiService,

IGeolocatorService geolocatorService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

\_geolocatorService = geolocatorService;

Title = Languages.MyProfile;

IsEnabled = true;

Owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

}

…

var userRequest = new UserRequest

{

Address = Owner.Address,

Document = Owner.Document,

Email = Owner.Email,

FirstName = Owner.FirstName,

LastName = Owner.LastName,

Password = "123456", // It doesn't matter what is sent here. It is only for the model to be valid

Phone = Owner.PhoneNumber,

Latitude = \_position.Latitude,

Longitude = \_position.Longitude

};

…

if (string.IsNullOrEmpty(Owner.Address))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddressError, Languages.Accept);

return false;

}

var isValidAddress = await ValidateAddressAsync();

if (!isValidAddress)

{

return false;

}

…

private async Task<bool> ValidateAddressAsync()

{

var geoCoder = new Geocoder();

var locations = await geoCoder.GetPositionsForAddressAsync(Owner.Address);

var locationList = locations.ToList();

if (locationList.Count == 0)

{

var response = await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.NotAddressFound,

Languages.Yes,

Languages.No);

if (response)

{

await \_geolocatorService.GetLocationAsync();

if (\_geolocatorService.Latitude != 0 && \_geolocatorService.Longitude != 0)

{

\_position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

var list = await geoCoder.GetAddressesForPositionAsync(\_position);

Owner.Address = list.FirstOrDefault();

return true;

}

else

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.NotLocationAvailable,

Languages.Accept);

return false;

}

}

else

{

return false;

}

}

if (locationList.Count == 1)

{

\_position = locationList.FirstOrDefault();

return true;

}

if (locationList.Count > 1)

{

var addresses = new List<Address>();

var names = new List<string>();

foreach (var location in locationList)

{

var list = await geoCoder.GetAddressesForPositionAsync(location);

names.AddRange(list);

foreach (var item in list)

{

addresses.Add(new Address

{

Name = item,

Latitude = location.Latitude,

Longitude = location.Longitude

});

}

}

var source = await App.Current.MainPage.DisplayActionSheet(

Languages.SelectAnAdrress,

Languages.Cancel,

null,

names.ToArray());

if (source == Languages.Cancel)

{

return false;

}

Owner.Address = source;

var address = addresses.FirstOrDefault(a => a.Name == source);

\_position = new Position(address.Latitude, address.Longitude);

}

return true;

}

1. Test it.

## Integration with Facebook

1. Modify the entity **Property** to storage latitude and longitude: