**pett.NET Core**

**Xamarin Forms + Prism**

**Xamarin Classic + MVVM Cross**



**Veterinary Example**

By Zulu

Medellín 2019

**Index**

[**Veterinary example: functionality matrix**](#_egxkps3p84mo) **4**

[**Create the Solution**](#_rzx1qjmdkj4p) **5**

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

[Create the Database](#_3znysh7) 6

[Modify DB](#_2et92p0) 10

[Seed the DB with initial data](#_tyjcwt) 16

[Add User Identities](#_j2qk9hw9udeq) 22

[Implementing login and logout in Web](#_rvvzh1dny0x) 35

[Create personalized controllers](#_n8zywv46w5ma) 41

[Add API](#_rf0uhnkl8ptl) 88

[Add Tokens Generation](#_bk5k4jo96xeg) 94

[Publish on Azure](#_ofdhtv4fdwnx) 99

[**App Xamarin Forms First Part**](#_opym9lysl1d) **102**

[Login](#_cuw37vfwhxvh) 102

[Modify Response to generic Class](#_m9hxeoltmkl9) 113

[Check the internet connection](#_2whmgs8xsoyd) 117

[Navigate to another page and pass parameters](#_osfw3hz1dhmz) 118

[Fix the images on Android](#_8mbpyrimot2z) 136

[Add tabbed page](#_pht8wt3nspgs) 138

[Add SfBusyIndicator](#_f6iruwi9yg5t) 151

[GIT Workshop](#_66308cip4uk4) 153

[**Web Second Part**](#_nwdg1ugw215t) **154**

[Redirect Pages](#_xacmwy12tj9a) 154

[Self-registration of users](#_a8xsy9ylifaf) 156

[Modifying users](#_1ksv4uv) 161

[Confirm Email Registration](#_3whwml4) 167

[Password Recovery](#_49dsjrqnhgzx) 177

[GROUP2](#_gjeu0ngci26b) 183

[Improve Index View](#_sskla8juvpfq) 183

[Managers CRUD](#_36ee3rz0wepn) 207

[Pets CRUD](#_4n82mnsa4fyw) 222

[Look and feel for a veterinary system and users functionality](#_nyt9f2xo5hi3) 244

[GROUP3](#_urrwmupxrs5m) 247

[Agenda](#_iazxq4q44q45) 247

[Users functionality](#_w9cv79fc8ot1) 262

[Prepare the API for new functionality in App](#_6x09d4hxxp7x) 289

[Account](#_8cbwohrru61p) 289

[Pets](#_w05gyav52mhn) 297

[Agenda](#_tp958f4afw9y) 308

[**App Xamarin Forms Second Part**](#_ald5cft967rf) **315**

[Add persistent setting](#_lnhhjejq0mm4) 315

[Add a Master Detail](#_jru7x1wimlgw) 318

[Add Icon & Splash to Xamarin Forms For Android](#_ditq6h2uupkm) 325

[Adding Styles](#_byd2ngr6qq09) 327

[Multi Language in Xamarin Forms](#_3o7alnk) 332

[Register users from App](#_r7prm6btz1r8) 347

[Recover Password From App in Xamarin Forms](#_c7bwyq5zm41g) 368

[Remember Me functionality](#_brkvnq7d4oki) 374

[Modify User From App in Xamarin Forms](#_1lgx68toiptd) 376

[Modify Password From App in Xamarin Forms](#_qpyf5uxa0zxi) 388

[App Pet From App & Accessing Camera and Photo Library](#_9gfrccp34q8l) 398

[Show My Agenda](#_zcwobvx55czy) 436

[Show the map and move to current location](#_cgyol3lxmf51) 453

[Put pins in map](#_oxe37i3pus8z) 459

# Veterinary example: functionality matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Functionality** | **Web** | | **App** |
| **Admin** | **Customer** | **Customer** |
| Login | X | X | X |
| Register |  | X | X |
| Modify profile | X | X | X |
| Recover password | X | X | X |
| Admin managers | X |  |  |
| Admin Owners | X |  |  |
| Admin Pets | X | X | X |
| Admin pet types | X |  |  |
| Admin agenda | X |  |  |
| Assign / unassign appointments | X | X | X |
| Admin clinic history for pets | X |  |  |
| See clinic history | X | X | X |
| See veterinary on map |  |  | X |
| Service “Pet Love” |  |  | X |

# Create the Solution

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

Create the following solution:

MyVet.Common (.NET Standard)

MyVet.Web (.NET Core)

Front End Web (Admin)

API

MyVet.Prism

MyVet.Prism

.Android

MyVet.Prism

.iOS

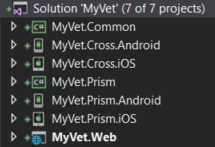
MyVet.Cross.Android

MyVet.Cross.iOS

Backend (Web)

Frontend (Mobile)

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>.

Ejemplo de comando: **Scaffold-DbContext "Data Source=bie-srv-db.database.windows.net;Initial Catalog=bie\_produccion;User ID=XXX;password=XXX" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Data**

1. Create the **Owner** class:

using System.ComponentModel.DataAnnotations;

namespace MyVet.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 MyVet.Web.Data.Entities;

namespace MyVet.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=MyVet;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:

Owner

Pet

Pet Type

History

Service Type

Agenda

1

\*

1

1

1

1

1

\*

\*

\*

\*

\*

1. Add the entity **PetType**:

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class PetType

{

public int Id { get; set; }

[Display(Name = "Pet 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 **ServiceType**:

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class ServiceType

{

public int Id { get; set; }

[Display(Name = "Service 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 **History**:

using System;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class History : IEntity

{

public int Id { get; set; }

public ServiceType ServiceType { get; set; }

[Display(Name = "Description\*")]

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

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

public string Description { get; set; }

[Display(Name = "Date\*")]

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

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

public DateTime Date { get; set; }

public string Remarks { get; set; }

public Pet Pet { get; set; }

[Display(Name = "Date\*")]

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

public DateTime DateLocal => Date.ToLocalTime();

}

}

1. Add the entity **Pet**:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class Pet

{

public int Id { get; set; }

[Display(Name = "Name")]

[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; }

[Display(Name = "Image")]

public string ImageUrl { get; set; }

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

public string Race { get; set; }

public Owner Owner { get; set; }

public PetType PetType { get; set; }

[Display(Name = "Born")]

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

[DataType(DataType.DateTime)]

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

public DateTime Born { get; set; }

public string Remarks { get; set; }

public ICollection<History> Histories { get; set; }

//TODO: replace the correct URL for the image

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? null

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

}

}

1. Add the entity **Agenda**:

using System;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class Agenda

{

public int Id { get; set; }

[Display(Name = "Date")]

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

[DataType(DataType.DateTime)]

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

public DateTime Date { get; set; }

public Owner Owner { get; set; }

public Pet Pet { get; set; }

public string Remarks { get; set; }

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

public bool IsAvailable { get; set; }

}

}

1. Update the **DataContext**:

using Microsoft.EntityFrameworkCore;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Data

{

public class DataContext : DbContext

{

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

{

}

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

public DbSet<PetType> PetTypes { get; set; }

public DbSet<Pet> Pets { get; set; }

public DbSet<ServiceType> ServiceTypes { get; set; }

public DbSet<History> Histories { get; set; }

public DbSet<Agenda> Agendas { 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 MyVet.Web.Data.Entities;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.Web.Data

{

public class SeedDb

{

private readonly DataContext \_context;

public SeedDb(DataContext context)

{

\_context = context;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckPetTypesAsync();

await CheckServiceTypesAsync();

await CheckOwnersAsync();

await CheckPetsAsync();

await CheckAgendasAsync();

}

private async Task CheckPetsAsync()

{

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

var petType = \_context.PetTypes.FirstOrDefault();

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

{

AddPet("Otto", owner, petType, "Shih tzu");

AddPet("Killer", owner, petType, "Dobermann");

await \_context.SaveChangesAsync();

}

}

private async Task CheckServiceTypesAsync()

{

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

{

\_context.ServiceTypes.Add(new ServiceType { Name = "Consulta" });

\_context.ServiceTypes.Add(new ServiceType { Name = "Urgencia" });

\_context.ServiceTypes.Add(new ServiceType { Name = "Vacunación" });

await \_context.SaveChangesAsync();

}

}

private async Task CheckPetTypesAsync()

{

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

{

\_context.PetTypes.Add(new PetType { Name = "Perro" });

\_context.PetTypes.Add(new PetType { Name = "Geto" });

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 AddPet(string name, Owner owner, PetType petType, string race)

{

\_context.Pets.Add(new Pet

{

Born = DateTime.Now.AddYears(-2),

Name = name,

Owner = owner,

PetType = petType,

Race = race

});

}

private async Task CheckAgendasAsync()

{

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

{

var initialDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day, 8, 0, 0);

var finalDate = initialDate.AddYears(1);

while (initialDate < finalDate)

{

if (initialDate.DayOfWeek != DayOfWeek.Sunday)

{

var finalDate2 = initialDate.AddHours(10);

while (initialDate < finalDate2)

{

\_context.Agendas.Add(new Agenda

{

Date = initialDate.ToUniversalTime(),

IsAvailable = true

});

initialDate = initialDate.AddMinutes(30);

}

initialDate = initialDate.AddHours(14);

}

else

{

initialDate = initialDate.AddDays(1);

}

}

await \_context.SaveChangesAsync();

}

}

}

}

1. Create the **Program** class:

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.DependencyInjection;

using MyVet.Web.Data;

namespace MyVet.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:

Owner

Pet

Pet Type

History

Service Type

Agenda

1

\*

1

1

1

1

1

\*

\*

\*

\*

\*

User

UserRole

Role

Manager

1

1

1

1

1

1

\*

\*

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

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Identity;

namespace MyVet.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; }

[Display(Name = "Full Name")]

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

[Display(Name = "Full Name")]

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

}

}

1. Modify the entity **Owner**:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class Owner

{

public int Id { get; set; }

public User User { get; set; }

public ICollection<Pet> Pets { get; set; }

public ICollection<Agenda> Agendas { get; set; }

}

}

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

namespace MyVet.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 MyVet.Web.Data.Entities;

namespace MyVet.Web.Data

{

public class DataContext : IdentityDbContext<User>

{

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

{

}

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

public DbSet<PetType> PetTypes { get; set; }

public DbSet<Pet> Pets { get; set; }

public DbSet<ServiceType> ServiceTypes { get; set; }

public DbSet<History> Histories { get; set; }

public DbSet<Agenda> Agendas { get; set; }

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

}

}

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

using Microsoft.AspNetCore.Identity;

using MyVet.Web.Data.Entities;

using System.Threading.Tasks;

namespace MyVet.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 MyVet.Web.Data.Entities;

using System.Threading.Tasks;

namespace MyVet.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 MyVet.Web.Data.Entities;

using MyVet.Web.Helpers;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.Web.Data

{

public class SeedDb

{

private readonly DataContext \_dataContext;

private readonly IUserHelper \_userHelper;

public SeedDb(

DataContext context,

IUserHelper userHelper)

{

\_dataContext = context;

\_userHelper = userHelper;

}

public async Task SeedAsync()

{

await \_dataContext.Database.EnsureCreatedAsync();

await CheckRoles();

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

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

await CheckPetTypesAsync();

await CheckServiceTypesAsync();

await CheckOwnerAsync(customer);

await CheckManagerAsync(manager);

await CheckPetsAsync();

await CheckAgendasAsync();

}

private async Task CheckRoles()

{

await \_userHelper.CheckRoleAsync("Admin");

await \_userHelper.CheckRoleAsync("Customer");

}

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 CheckPetsAsync()

{

if (!\_dataContext.Pets.Any())

{

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

var petType = \_dataContext.PetTypes.FirstOrDefault();

AddPet("Otto", owner, petType, "Shih tzu");

AddPet("Killer", owner, petType, "Dobermann");

await \_dataContext.SaveChangesAsync();

}

}

private async Task CheckServiceTypesAsync()

{

if (!\_dataContext.ServiceTypes.Any())

{

\_dataContext.ServiceTypes.Add(new ServiceType { Name = "Consulta" });

\_dataContext.ServiceTypes.Add(new ServiceType { Name = "Urgencia" });

\_dataContext.ServiceTypes.Add(new ServiceType { Name = "Vacunación" });

await \_dataContext.SaveChangesAsync();

}

}

private async Task CheckPetTypesAsync()

{

if (!\_dataContext.PetTypes.Any())

{

\_dataContext.PetTypes.Add(new PetType { Name = "Perro" });

\_dataContext.PetTypes.Add(new PetType { Name = "Gato" });

await \_dataContext.SaveChangesAsync();

}

}

private async Task CheckOwnerAsync(User user)

{

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

{

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

await \_dataContext.SaveChangesAsync();

}

}

private async Task CheckManagerAsync(User user)

{

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

{

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

await \_dataContext.SaveChangesAsync();

}

}

private void AddPet(string name, Owner owner, PetType petType, string race)

{

\_dataContext.Pets.Add(new Pet

{

Born = DateTime.Now.AddYears(-2),

Name = name,

Owner = owner,

PetType = petType,

Race = race

});

}

private async Task CheckAgendasAsync()

{

if (!\_dataContext.Agendas.Any())

{

var initialDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day, 8, 0, 0);

var finalDate = initialDate.AddYears(1);

while (initialDate < finalDate)

{

if (initialDate.DayOfWeek != DayOfWeek.Sunday)

{

var finalDate2 = initialDate.AddHours(10);

while (initialDate < finalDate2)

{

\_dataContext.Agendas.Add(new Agenda

{

Date = initialDate,

IsAvailable = true

});

initialDate = initialDate.AddMinutes(30);

}

initialDate = initialDate.AddHours(14);

}

else

{

initialDate = initialDate.AddDays(1);

}

}

}

await \_dataContext.SaveChangesAsync();

}

}

}

1. Delete and create the **OwnersController** and create the controller for **Manager** entity.
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 MyVet.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 MyVet.Web.Helpers;

using MyVet.Web.Models;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.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 MyVet.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 annotation authorize to your own controllers, except **AccountController**:

[Authorize(Roles = "Admin")]

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("Admin"))

{

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

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

<li><a asp-area="" asp-controller="Agenda" asp-action="Index">Agenda</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 **Owner** entity:

using System.Collections.Generic;

namespace MyVet.Web.Data.Entities

{

public class Owner : IEntity

{

public int Id { get; set; }

public User User { get; set; }

public ICollection<Pet> Pets { get; set; }

}

}

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

public IActionResult Index()

{

return View(\_dataContext.Owners

.Include(o => o.User)

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

}

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

@model IEnumerable<MyVet.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>

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" 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.Pets)

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

if (owner == null)

{

return NotFound();

}

return View(owner);

}

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

@model MyVet.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>

Pets

</dt>

<dd>

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

</dd>

</dl>

</div>

<div>

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

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

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

</div>

<h4>Pets</h4>

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

{

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

}

else

{

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Pets.FirstOrDefault().Name)

</th>

<th>

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

</th>

<th>

@Html.DisplayNameFor(model => model.Pets.FirstOrDefault().Race)

</th>

<th>

@Html.DisplayNameFor(model => model.Pets.FirstOrDefault().Born)

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Pets)

{

<tr>

<td>

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

</td>

<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>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

<a asp-action="DeletePet" 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 MyVet.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 MyVet.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

{

Pets = new List<Pet>(),

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, "Customer");

return newUser;

}

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

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Models

{

public class PetViewModel : Pet

{

public int OwnerId { get; set; }

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

[Display(Name = "Pet Type")]

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

public int PetTypeId { get; set; }

[Display(Name = "Image")]

public IFormFile ImageFile { get; set; }

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

}

}

1. Add the **ICombosHelper**:

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc.Rendering;

namespace MyVet.Web.Helpers

{

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboPetTypes();

}

}

1. Add the **CombosHelper**:

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyVet.Web.Data;

namespace MyVet.Web.Helpers

{

public class CombosHelper : ICombosHelper

{

private readonly DataContext \_dataContext;

public CombosHelper(DataContext dataContext)

{

\_dataContext = dataContext;

}

public IEnumerable<SelectListItem> GetComboPetTypes()

{

var list = \_dataContext.PetTypes.Select(pt => new SelectListItem

{

Text = pt.Name,

Value = $"{pt.Id}"

})

.OrderBy(pt => pt.Text)

.ToList();

list.Insert(0, new SelectListItem

{

Text = "[Select a pet type...]",

Value = "0"

});

return list;

}

}

}

1. Add the **IConverterHelper**:

using System.Threading.Tasks;

using MyVet.Web.Data.Entities;

using MyVet.Web.Models;

namespace MyVet.Web.Helpers

{

public interface IConverterHelper

{

Task<Pet> ToPetAsync(PetViewModel model, string path, bool isNew);

PetViewModel ToPetViewModel(Pet pet);

}

}

1. Add the **ConverterHelper**:

using System.Threading.Tasks;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

using MyVet.Web.Models;

namespace MyVet.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<Pet> ToPetAsync(PetViewModel model, string path, bool isNew)

{

var pet = new Pet

{

Agendas = model.Agendas,

Born = model.Born,

Histories = model.Histories,

Id = isNew ? 0 : model.Id,

ImageUrl = path,

Name = model.Name,

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

PetType = await \_dataContext.PetTypes.FindAsync(model.PetTypeId),

Race = model.Race,

Remarks = model.Remarks

};

return pet;

}

public PetViewModel ToPetViewModel(Pet pet)

{

return new PetViewModel

{

Agendas = pet.Agendas,

Born = pet.Born,

Histories = pet.Histories,

ImageUrl = pet.ImageUrl,

Name = pet.Name,

Owner = pet.Owner,

PetType = pet.PetType,

Race = pet.Race,

Remarks = pet.Remarks,

Id = pet.Id,

OwnerId = pet.Owner.Id,

PetTypeId = pet.PetType.Id,

PetTypes = \_combosHelper.GetComboPetTypes()

};

}

}

}

1. Add the **IImageHelper**:

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

namespace MyVet.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 MyVet.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\\Pets",

file);

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

{

await imageFile.CopyToAsync(stream);

}

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

}

}

}

1. Setup the new injections:

services.AddScoped<ICombosHelper, CombosHelper>();

services.AddScoped<IConverterHelper, ConverterHelper>();

services.AddScoped<IImageHelper, ImageHelper>();

1. Modify the **OwnersController**:

private readonly ICombosHelper \_combosHelper;

private readonly IConverterHelper \_converterHelper;

private readonly IImageHelper \_imageHelper;

…

public OwnersController(

DataContext context,

IUserHelper userHelper,

ICombosHelper combosHelper,

IConverterHelper converterHelper,

IImageHelper imageHelper)

{

\_context = context;

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_converterHelper = converterHelper;

\_imageHelper = imageHelper;

}

…

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

{

if (id == null)

{

return NotFound();

}

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

if (owner == null)

{

return NotFound();

}

var model = new PetViewModel

{

Born = DateTime.Today,

OwnerId = owner.Id,

PetTypes = \_combosHelper.GetComboPetTypes()

};

return View(model);

}

[HttpPost]

public async Task<IActionResult> AddPet(PetViewModel model)

{

if (ModelState.IsValid)

{

var path = string.Empty;

if(model.ImageFile != null)

{

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

}

var pet = await \_converterHelper.ToPetAsync(model, path, true);

\_context.Pets.Add(pet);

await \_context.SaveChangesAsync();

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

}

return View(model);

}

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

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Pet</h4>

<hr />

<div class="row">

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

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

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

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

<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">

<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">

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

<select asp-for="PetTypeId" asp-items="Model.PetTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Born" 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">

<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> EditPet(int? id)

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.Pets

.Include(p => p.Owner)

.Include(p => p.PetType)

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

if (pet == null)

{

return NotFound();

}

return View(\_converterHelper.ToPetViewModel(pet));

}

[HttpPost]

public async Task<IActionResult> EditPet(PetViewModel model)

{

if (ModelState.IsValid)

{

var path = model.ImageUrl;

if (model.ImageFile != null)

{

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

}

var pet = await \_converterHelper.ToPetAsync(model, path, false);

\_context.Pets.Update(pet);

await \_context.SaveChangesAsync();

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

}

return View(model);

}

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

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Pet</h4>

<hr />

<div class="row">

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

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

<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="ImageUrl" />

<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">

<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">

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

<select asp-for="PetTypeId" asp-items="Model.PetTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Born" 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">

<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 class="col-md-4">

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:400px;height:400px;max-width: 100%; height: auto;" />

}

</div>

</div>

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

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.Histories)

.ThenInclude(h => h.ServiceType)

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

if (pet == null)

{

return NotFound();

}

return View(pet);

}

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

@model MyVet.Web.Data.Entities.Pet

@{

ViewData["Title"] = "Details";

}

<h2>Pet</h2>

<div>

<h4>Details</h4>

<hr />

<div class="row">

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

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

</dl>

</div>

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

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:300px;height:300px;max-height: 100%; width: auto;" />

}

</div>

</div>

</div>

<div>

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

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

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

</div>

<h4>History</h4>

<hr />

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

{

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

}

else

{

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Date)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().ServiceType.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Description)

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Histories)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

</td>

</tr>

}

</tbody>

</table>

}

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

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Models

{

public class HistoryViewModel : History

{

public int PetId { get; set; }

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

[Display(Name = "Service Type")]

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

public int ServiceTypeId { get; set; }

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

}

}

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

IEnumerable<SelectListItem> GetComboServiceTypes();

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

public IEnumerable<SelectListItem> GetComboServiceTypes()

{

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

{

Text = p.Name,

Value = p.Id.ToString()

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

list.Insert(0, new SelectListItem

{

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

Value = "0"

});

return list;

}

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

Task<History> ToHistoryAsync(HistoryViewModel model, bool isNew);

HistoryViewModel ToHistoryViewModel(History history);

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

public async Task<History> ToHistoryAsync(HistoryViewModel model, bool isNew)

{

return new History

{

Date = model.Date,

Description = model.Description,

Id = isNew ? 0 : model.Id,

Pet = await \_dataContext.Pets.FindAsync(model.PetId),

Remarks = model.Remarks,

ServiceType = await \_dataContext.ServiceTypes.FindAsync(model.ServiceTypeId)

};

}

public HistoryViewModel ToHistoryViewModel(History history)

{

return new HistoryViewModel

{

Date = history.Date,

Description = history.Description,

Id = history.Id,

PetId = history.Pet.Id,

Remarks = history.Remarks,

ServiceTypeId = history.ServiceType.Id,

ServiceTypes = \_combosHelper.GetComboServiceTypes()

};

}

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

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets.FindAsync(id.Value);

if (pet == null)

{

return NotFound();

}

var model = new HistoryViewModel

{

Date = DateTime.Now,

PetId = pet.Id,

ServiceTypes = \_combosHelper.GetComboServiceTypes(),

};

return View(model);

}

[HttpPost]

public async Task<IActionResult> AddHistory(HistoryViewModel model)

{

if (ModelState.IsValid)

{

var history = await \_converterHelper.ToHistoryAsync(model, true);

\_dataContext.Histories.Add(history);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsPet)}/{model.PetId}");

}

return View(model);

}

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

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>History</h4>

<hr />

<div class="row">

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

<form asp-action="AddHistory">

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

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

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<select asp-for="ServiceTypeId" asp-items="Model.ServiceTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Description" 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">

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

<a asp-action="DetailsPet" asp-route-id="@Model.PetId" class="btn btn-success">Back to Pet</a>

</div>

</form>

</div>

</div>

1. Test it.
2. Add the methods **EditHistory** to **OwnersController**:

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

{

if (id == null)

{

return NotFound();

}

var history = await \_dataContext.Histories

.Include(h => h.Pet)

.Include(h => h.ServiceType)

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

if (history == null)

{

return NotFound();

}

return View(\_converterHelper.ToHistoryViewModel(history));

}

[HttpPost]

public async Task<IActionResult> EditHistory(HistoryViewModel model)

{

if (ModelState.IsValid)

{

var history = await \_converterHelper.ToHistoryAsync(model, false);

\_dataContext.Histories.Update(history);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsPet)}/{model.PetId}");

}

return View(model);

}

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

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>History</h4>

<hr />

<div class="row">

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

<form asp-action="EditHistory">

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

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

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

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<select asp-for="ServiceTypeId" asp-items="Model.ServiceTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Description" 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">

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

<a asp-action="DetailsPet" asp-route-id="@Model.PetId" class="btn btn-success">Back to Pet</a>

</div>

</form>

</div>

</div>

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

<td>

<a asp-action="EditHistory" asp-route-id="@item.Id" class="btn btn-warning">Edit</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/DeleteHistory/' + item\_to\_delete;

});

});

</script>

}

1. Add the method **DeleteHistory** in **OwnersController**:

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

{

if (id == null)

{

return NotFound();

}

var history = await \_dataContext.Histories

.Include(h => h.Pet)

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

if (history == null)

{

return NotFound();

}

\_dataContext.Histories.Remove(history);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(DetailsPet)}/{history.Pet.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(o => o.Id == id.Value);

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 MyVet.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 MyVet.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.

## Add API

1. Add the **EmailRequest** class:

using System.ComponentModel.DataAnnotations;

namespace MyVet.Common.Models

{

public class EmailRequest

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

1. Add the **HistoryResponse** class:

using System;

namespace MyVet.Common.Models

{

public class HistoryResponse

{

public int Id { get; set; }

public string ServiceType { get; set; }

public string Description { get; set; }

public DateTime Date { get; set; }

public string Remarks { get; set; }

}

}

1. Add the **PetResponse** class:

using System;

using System.Collections.Generic;

namespace MyVet.Common.Models

{

public class PetResponse

{

public int Id { get; set; }

public string Name { get; set; }

public string ImageUrl { get; set; }

public string Race { get; set; }

public DateTime Born { get; set; }

public string Remarks { get; set; }

public string PetType { get; set; }

public ICollection<HistoryResponse> Histories { get; set; }

}

}

1. Add the **OwnerResponse** class:

using System.Collections.Generic;

namespace MyVet.Common.Models

{

public class OwnerResponse

{

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<PetResponse> Pets { get; set; }

}

}

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

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Common.Models;

using MyVet.Web.Data;

namespace MyVet.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> GetOwner(EmailRequest emailRequest)

{

var owner = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Pets)

.ThenInclude(p => p.PetType)

.Include(o => o.Pets)

.ThenInclude(p => p.Histories)

.ThenInclude(h => h.ServiceType)

.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(emailRequest.Email.ToLower()));

var response = new OwnerResponse

{

FirstName = owner.User.FirstName,

LastName = owner.User.LastName,

Address = owner.User.Address,

Document = owner.User.Document,

Email = owner.User.Email,

PhoneNumber = owner.User.PhoneNumber,

Pets = owner.Pets.Select(p => new PetResponse

{

Born = p.Born,

Id = p.Id,

ImageUrl = p.ImageFullPath,

Name = p.Name,

Race = p.Race,

Remarks = p.Remarks,

PetType = p.PetType.Name,

Histories = p.Histories.Select(h => new HistoryResponse

{

Date = h.Date,

Description = h.Description,

Id = h.Id,

Remarks = h.Remarks,

ServiceType = h.ServiceType.Name

}).ToList()

}).ToList()

};

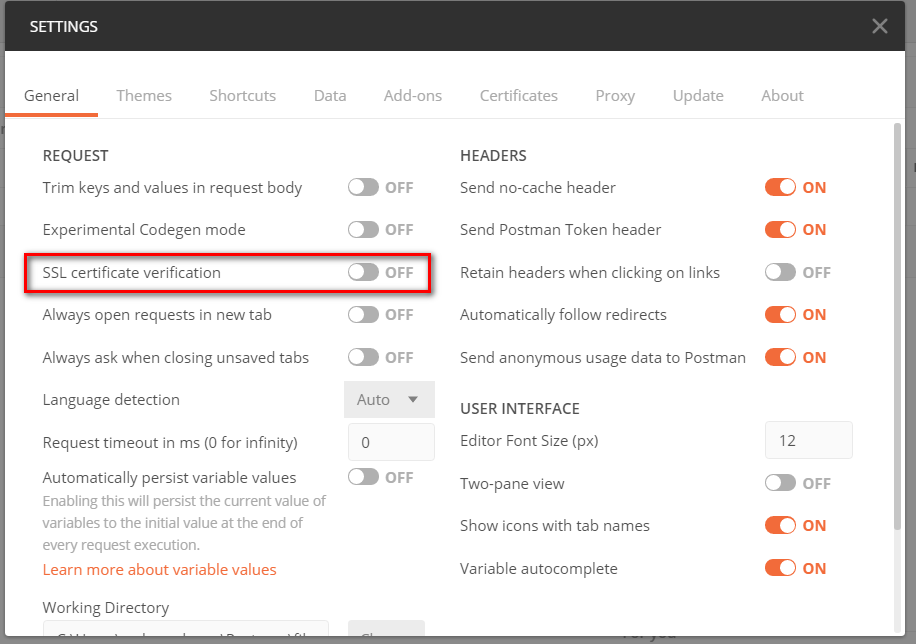
return Ok(response);

}

}

}

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=MyVet;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"]))

};

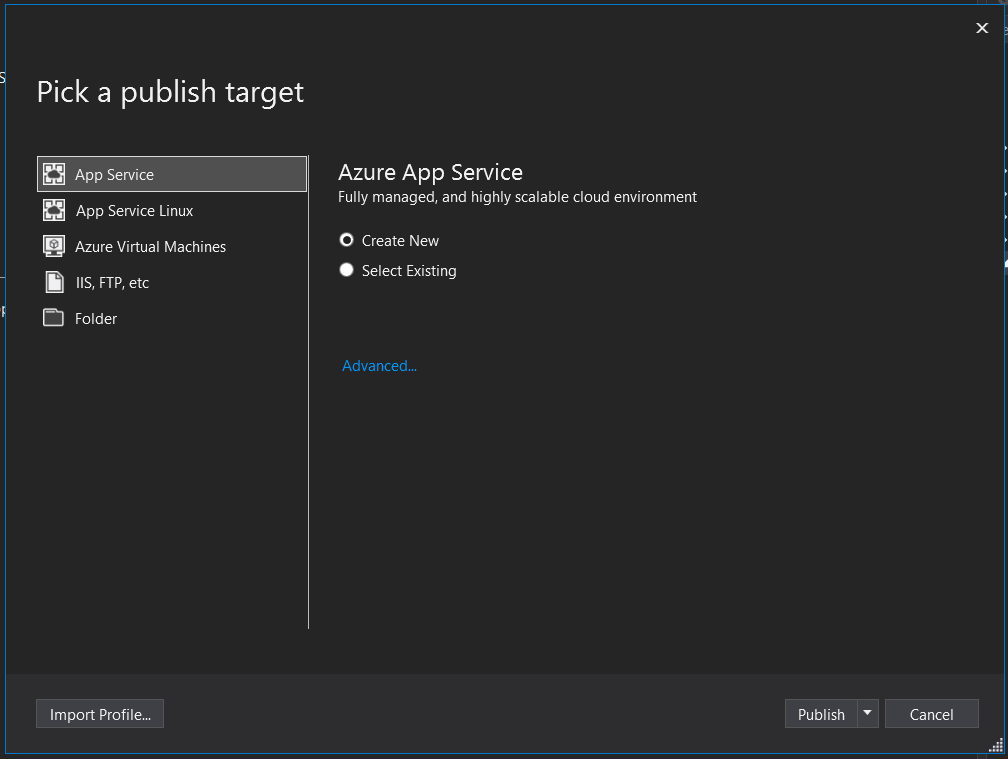
});

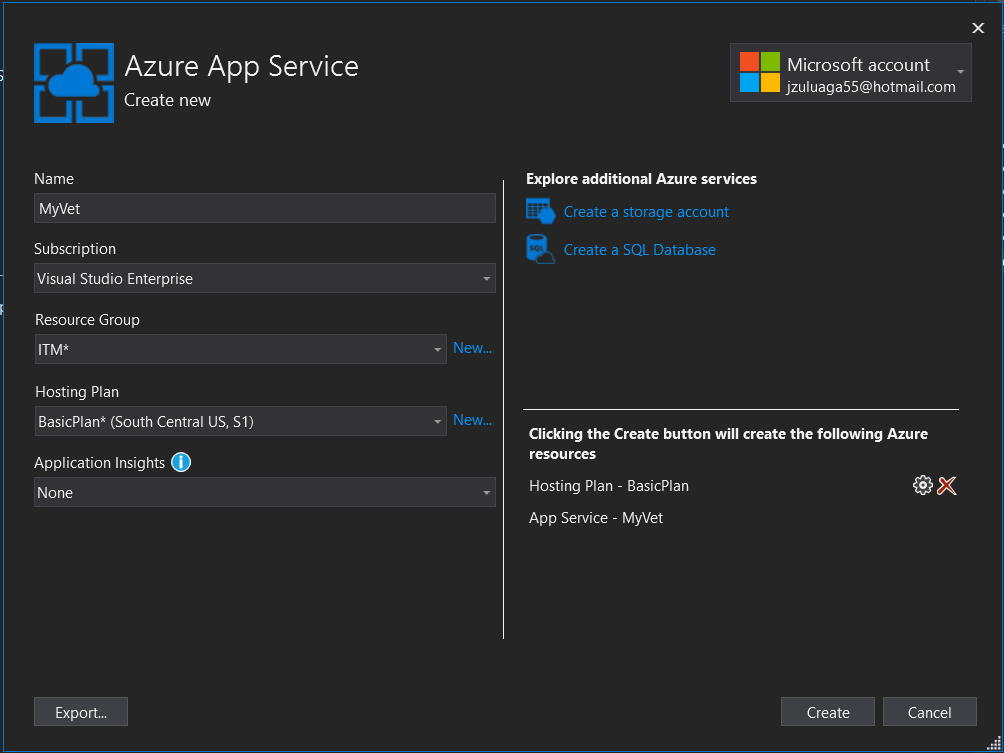
services.AddTransient<SeedDb>();

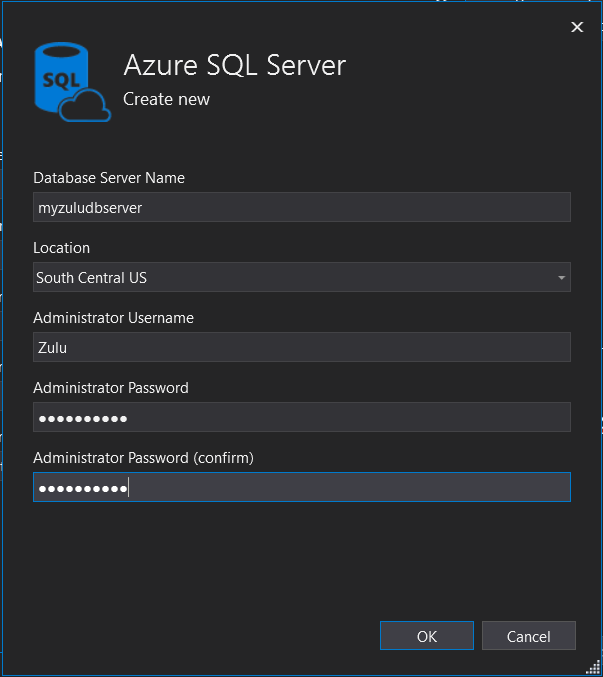
services.AddScoped<IUserHelper, UserHelper>();

1. Test it.

## Publish on Azure









Then, fix the correct route for images:

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? "https://myvet.azurewebsites.net/images/Pets/noimage.png"

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

And re-publish.

# App Xamarin Forms First Part

## Login

1. Delete the **MainPage** and **MainPageViewModel**.
2. Add the **Login** page, 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="MyVet.Prism.Views.Login"

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 **LoginViewModel**:

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class LoginViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private string \_password;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

public LoginViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "My Vet - 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 MyVet.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 MyVet.Common.Models

{

public class TokenRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

}

1. Add the **TokenResponse** class:

using System;

namespace MyVet.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 MyVet.Common.Models;

namespace MyVet.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 MyVet.Common.Models;

using Newtonsoft.Json;

namespace MyVet.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="MyVet.Prism.App">

<Application.Resources>

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">https://myvet.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/Login");

}

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<Login, LoginViewModel>();

}

1. Modify the method **Login** in **LoginViewMovil** class:

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;

}

var token = (TokenResponse)response.Result;

var response2 = await \_apiService.GetOwnerByEmail(

url,

"/api",

"/Owners/GetOwnerByEmail",

"bearer",

token.Token,

Email);

var owner = (OwnerResponse)response2.Result;

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 MyVet.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 **Pets** 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="MyVet.Prism.Views.Pets"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Label

HorizontalOptions="CenterAndExpand"

Text="{Binding Title}"

VerticalOptions="CenterAndExpand"/>

</StackLayout>

</ContentPage>

1. Modify the **PetsViewModel** class:

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetsViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

public PetsViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Pets";

}

}

}

1. Modify the **LoginViewModel** class:

IsEnabled = true;

IsRunning = false;

await \_navigationService.NavigateAsync("Pets");

}

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 parameters = new NavigationParameters

{

{ "token", token },

{ "owner", owner }

};

await \_navigationService.NavigateAsync("Pets", parameters);

1. Override the method **OnNavigatedTo** in **PetsViewModel** 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 **PetItemViewModel** class:

using MyVet.Common.Models;

namespace MyVet.Prism.ViewModels

{

public class PetItemViewModel : PetResponse

{

}

}

1. Modify the **Pets** 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:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.PetsPage"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ListView

BackgroundColor="Transparent"

SeparatorVisibility="None"

HasUnevenRows="True"

ItemsSource="{Binding Pets}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Frame

CornerRadius="10"

HasShadow="True"

Margin="0,0,0,5">

<Frame.GestureRecognizers>

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

</Frame.GestureRecognizers>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<ffimageloading:CachedImage

Grid.Column="0"

Source="{Binding ImageUrl}"

LoadingPlaceholder= "LoaderImage"

ErrorPlaceholder= "ErrorImage"

CacheDuration= "50"

RetryCount= "3"

RetryDelay= "600"

DownsampleToViewSize = "true"

WidthRequest="100"/>

<Grid

Grid.Column="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="Name"/>

<Label

Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

Text="{Binding Name}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="Race"/>

<Label

Grid.Column="1"

Grid.Row="1"

FontAttributes="Bold"

Text="{Binding Race}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="Born"/>

<Label

Grid.Column="1"

Grid.Row="2"

FontAttributes="Bold"

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

</Grid>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"/>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Modify the **PetsViewModel** class:

using System.Collections.ObjectModel;

using System.Linq;

using MyVet.Common.Models;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetsViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private OwnerResponse \_owner;

private TokenResponse \_token;

private ObservableCollection<PetItemViewModel> \_pets;

private bool \_isRefreshing;

public PetsViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Pets";

}

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

{ xmlns:ffimageloading ="clr-namespace" xmlns:ffimageloading ="clr-namespace"

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 **HistoryItemViewModel** class:

using MyVet.Common.Models;

namespace MyVet.Prism.ViewModels

{

public class HistoryItemViewModel : HistoryResponse

{

}

}

1. Modify the **PetItemViewModel** class:

using MyVet.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetItemViewModel : PetResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectPetCommand;

public PetItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectPetCommand => \_selectPetCommand ?? (\_selectPetCommand = new DelegateCommand(SelectPet));

private async void SelectPet()

{

var parameters = new NavigationParameters

{

{ "pet", this }

};

await \_navigationService.NavigateAsync("Pet", parameters);

}

}

}

1. Add the **Pet** 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="MyVet.Prism.Views.Pet"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Frame

CornerRadius="20"

HasShadow="True">

<StackLayout>

<StackLayout

Orientation="Horizontal">

<Image

WidthRequest="180"

Source="{Binding Pet.ImageUrl}"/>

<StackLayout>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="{Binding Pet.Name}"/>

<Label

Text="{Binding Pet.Race, StringFormat='Race: {0}'}"/>

<Label

Text="{Binding Pet.Born, StringFormat='Born: {0:yyyy/MM/dd}'}"/>

<Label

Text="{Binding Pet.PetType, StringFormat='Type: {0}'}"/>

</StackLayout>

</StackLayout>

<Label

BackgroundColor="White"

Text="{Binding Pet.Remarks}"/>

</StackLayout>

</Frame>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="History"

TextColor="Black"/>

<ListView

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Histories}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding Date, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding ServiceType}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Description}"

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 **PetViewModel** class:

using System.Collections.ObjectModel;

using System.Linq;

using MyVet.Common.Models;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PetResponse \_pet;

private ObservableCollection<HistoryItemViewModel> \_histories;

public PetViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

}

public PetResponse Pet

{

get => \_pet;

set => SetProperty(ref \_pet, value);

}

public ObservableCollection<HistoryItemViewModel> Histories

{

get => \_histories;

set => SetProperty(ref \_histories, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

Pet = parameters.GetValue<PetResponse>("pet");

Title = Pet.Name;

Histories = new ObservableCollection<HistoryItemViewModel>(Pet.Histories.Select(h => new HistoryItemViewModel

{

Date = h.Date,

Description = h.Description,

Id = h.Id,

Remarks = h.Remarks,

ServiceType = h.ServiceType

}).ToList());

}

}

}

}

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

using MyVet.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class HistoryItemViewModel : HistoryResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectHistoryCommand;

public HistoryItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectHistoryCommand => \_selectHistoryCommand ?? (\_selectHistoryCommand = new DelegateCommand(SelectHistory));

private async void SelectHistory()

{

var parameters = new NavigationParameters

{

{ "history", this }

};

await \_navigationService.NavigateAsync("History", parameters);

}

}

}

1. Add the **History** 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="MyVet.Prism.Views.History"

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"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

FontAttributes="Bold"

Text="Date"/>

<Label

Grid.Column="1"

Grid.Row="0"

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

<Label

Grid.Column="0"

Grid.Row="1"

FontAttributes="Bold"

Text="Type"/>

<Label

Grid.Column="1"

Grid.Row="1"

Text="{Binding History.ServiceType}"/>

<Label

Grid.Column="0"

Grid.Row="2"

FontAttributes="Bold"

Text="Description"/>

<Label

Grid.Column="1"

Grid.Row="2"

Text="{Binding History.Description}"/>

<Label

Grid.Column="0"

Grid.Row="3"

FontAttributes="Bold"

Text="Remarks"/>

<Label

Grid.Column="1"

Grid.Row="3"

Text="{Binding History.Remarks}"/>

</Grid>

</StackLayout>

</ContentPage>

1. Modify the **HistoryViewModel** class:

using MyVet.Common.Models;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class HistoryViewModel : ViewModelBase

{

private HistoryResponse \_history;

public HistoryViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "History";

}

public HistoryResponse History

{

get => \_history;

set => SetProperty(ref \_history, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("history"))

{

History = parameters.GetValue<HistoryResponse>("history");

}

}

}

}

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 tabbed page

1. Add the **HistoriesPage**:

<?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="MyVet.Prism.Views.HistoriesPage"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="48"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

FontAttributes="Bold"

Text="Date"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

FontAttributes="Bold"

Text="Service Type"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

FontAttributes="Bold"

Text="Description"

VerticalOptions="Center"/>

</Grid>

<ListView

HasUnevenRows="True"

ItemsSource="{Binding Histories}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

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

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding ServiceType}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Description}"

VerticalOptions="Center"/>

<Image

Grid.Column="3"

Source="ic\_chevron\_right"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Add the **HistoryItemViewModel**:

using MyVet.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class HistoryItemViewModel : HistoryResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectHistoryCommand;

public HistoryItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectHistoryCommand => \_selectHistoryCommand ?? (\_selectHistoryCommand = new DelegateCommand(SelectHistory));

private async void SelectHistory()

{

var parameters = new NavigationParameters

{

{ "history", this }

};

await \_navigationService.NavigateAsync("HistoryPage", parameters);

}

}

}

1. Modify the **HistoriesPageViewModel**:

using MyVet.Common.Models;

using Prism.Navigation;

using System.Collections.ObjectModel;

using System.Linq;

namespace MyVet.Prism.ViewModels

{

public class HistoriesPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private ObservableCollection<HistoryItemViewModel> \_histories;

private PetResponse \_pet;

public HistoriesPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Histories";

}

public PetResponse Pet

{

get => \_pet;

set => SetProperty(ref \_pet, value);

}

public ObservableCollection<HistoryItemViewModel> Histories

{

get => \_histories;

set => SetProperty(ref \_histories, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

Pet = parameters.GetValue<PetResponse>("pet");

Histories = new ObservableCollection<HistoryItemViewModel>(Pet.Histories.Select(h => new HistoryItemViewModel(\_navigationService)

{

Date = h.Date,

Description = h.Description,

Id = h.Id,

Remarks = h.Remarks,

ServiceType = h.ServiceType

}).ToList());

}

}

}

}

1. Add the **HistoryPage**:

<?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="MyVet.Prism.Views.HistoryPage"

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="Date"/>

<Label

Grid.Row="0"

Grid.Column="1"

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

<Label

Grid.Row="1"

Grid.Column="0"

FontAttributes="Bold"

Text="Service Type"/>

<Label

Grid.Row="1"

Grid.Column="1"

Text="{Binding History.ServiceType}"/>

<Label

Grid.Row="2"

Grid.Column="0"

FontAttributes="Bold"

Text="Description"/>

<Label

Grid.Row="2"

Grid.Column="1"

Text="{Binding History.Description}"/>

<Label

Grid.Row="4"

Grid.Column="0"

FontAttributes="Bold"

Text="Remarks"/>

<Label

Grid.Row="4"

Grid.Column="1"

Text="{Binding History.Remarks}"/>

</Grid>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modify the **HistoryPageViewModel**:

using MyVet.Common.Models;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class HistoryPageViewModel : ViewModelBase

{

private HistoryResponse \_history;

public HistoryPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "History";

}

public HistoryResponse History

{

get => \_history;

set => SetProperty(ref \_history, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("history"))

{

History = parameters.GetValue<HistoryResponse>("history");

}

}

}

}

1. Temporarily modify **PetItemViewModel**:

await \_navigationService.NavigateAsync("HistoriesPage", parameters);

1. Test it.
2. Add the **PetTabbedPage**:

<?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:MyVet.Prism.Views"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.PetTabbedPage"

Title="{Binding Title}">

<TabbedPage.Children>

<local:PetPage/>

<local:HistoriesPage />

</TabbedPage.Children>

</TabbedPage>

1. Modify the **PetTabbedPageViewModel**:

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetTabbedPageViewModel : ViewModelBase

{

public PetTabbedPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Pet";

}

}

}

1. Add the NuGet **Xam.Plugins.Settings** to all mobility projects and **Common**:
2. Add the **Settings** class:

using Plugin.Settings;

using Plugin.Settings.Abstractions;

namespace MyVet.Common.Helpers

{

public static class Settings

{

private const string \_pet = "Pet";

private static readonly string \_settingsDefault = string.Empty;

private static ISettings AppSettings => CrossSettings.Current;

public static string Pet

{

get => AppSettings.GetValueOrDefault(\_pet, \_settingsDefault);

set => AppSettings.AddOrUpdateValue(\_pet, value);

}

}

}

1. Modify the **PropertyItemViewModel**:

private async void SelectPet()

{

Settings.Pet = JsonConvert.SerializeObject(this);

await \_navigationService.NavigateAsync("PetTabbedPage");

}

1. Modify the **PropertyPageViewModel**:

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

Pet = JsonConvert.DeserializeObject<PetResponse>(Settings.Pet);

}

1. Modify the **HistoriesPageViewModel**:

…

Delete the method on natigated to

...

public HistoriesPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Histories";

Pet = JsonConvert.DeserializeObject<PetResponse>(Settings.Pet);

LoadHistories();

}

…

private void LoadHistories()

{

Histories = new ObservableCollection<HistoryItemViewModel>(Pet.Histories.Select(h => new HistoryItemViewModel(\_navigationService)

{

Date = h.Date,

Description = h.Description,

Id = h.Id,

Remarks = h.Remarks,

ServiceType = h.ServiceType

}).ToList());

}

1. Add icons to pages **HistoriesPage** and **PetPage**.
2. Test it.

## Add SfBusyIndicator

1. Get a Sync Fusion license: <https://help.syncfusion.com/common/essential-studio/licensing/license-key#xamarinforms>.
2. Add the NuGet **Syncfusion.Xamarin.SfBusyIndicator** 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**:

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 Home Controller 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 those methods to **AccountController**:

public IActionResult Register()

{

return View();

}

[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 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);

}

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, "Customer");

return newUser;

}

1. Add the view **Register** on **AccountController**:

@model MyVet.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="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. Test it.

## Modifying users

1. Add the **ChangePasswordViewModel** class:

using System.ComponentModel.DataAnnotations;

namespace MyVet.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 owner = await \_dataContext.Owners

.Include(o => o.User)

.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(User.Identity.Name.ToLower()));

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> ChangeUser(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("Index", "Home");

}

return View(view);

}

1. Add the view **ChangeUser** in **AccountController**:

@model MyVet.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Owner</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 MyVet.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 MyVet.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 MyVet.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. Let's take advantage and change the seed for pets:

private void AddPet(string name, Owner owner, PetType petType, string race)

{

var histories = new List<History>

{

new History

{

Date = DateTime.Now,

Description = "Consulta",

Remarks = "Fusce gravida convallis tortor, non lobortis massa. Duis hendrerit mauris et lectus dapibus finibus. Etiam dictum molestie tortor et tincidunt. Nam viverra nunc vitae leo porta, et dapibus dui ultrices.",

ServiceType = \_dataContext.ServiceTypes.FirstOrDefault()

},

new History

{

Date = DateTime.Now,

Description = "Consulta",

Remarks = "Maecenas quis molestie sem, at convallis magna. Vestibulum euismod augue eu erat fringilla tempus. Phasellus vel ante interdum, bibendum tortor quis, sodales ex.",

ServiceType = \_dataContext.ServiceTypes.FirstOrDefault()

},

new History

{

Date = DateTime.Now,

Description = "Consulta",

Remarks = "Quisque dapibus semper diam, vitae bibendum ex volutpat et. Proin eu posuere augue. Nulla at nisi purus. Proin a scelerisque orci. Ut sapien erat, tempor ac ligula sit amet, lobortis laoreet arcu.",

ServiceType = \_dataContext.ServiceTypes.FirstOrDefault()

}

};

\_dataContext.Pets.Add(new Pet

{

Born = DateTime.Now.AddYears(-2),

Name = name,

Owner = owner,

PetType = petType,

Race = race,

ImageUrl = $"~/images/Pets/{name}.png",

Remarks = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas non tempus velit. Vestibulum nec vehicula urna, quis tincidunt diam. In vitae ultricies ipsum.",

Histories = histories

});

}

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 MyVet.Web.Models

{

public class RecoverPasswordViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

}

1. Add the model **ResetPasswordViewModel**:

using System.ComponentModel.DataAnnotations;

namespace MyVet.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, "MyVet Password Reset", $"<h1>Shop 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 MyVet.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 MyVet.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<MyVet.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-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 = '/Owners/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Following the example, do the same for **Details** and **DetailsPet**.
3. **Homework**: according to owners example, modify the controller and views for: pet types and service types. Delete the controller and views for histories.
4. Homework solved:
5. Modify the **PetTypesController**.

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class PetTypesController : Controller

{

private readonly DataContext \_context;

public PetTypesController(DataContext context)

{

\_context = context;

}

public IActionResult Index()

{

return View(\_context.PetTypes);

}

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

{

if (id == null)

{

return NotFound();

}

var petType = await \_context.PetTypes.FirstOrDefaultAsync(m => m.Id == id);

if (petType == null)

{

return NotFound();

}

return View(petType);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(PetType petType)

{

if (ModelState.IsValid)

{

\_context.Add(petType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(petType);

}

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

{

if (id == null)

{

return NotFound();

}

var petType = await \_context.PetTypes.FindAsync(id);

if (petType == null)

{

return NotFound();

}

return View(petType);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(PetType petType)

{

if (ModelState.IsValid)

{

try

{

\_context.Update(petType);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!PetTypeExists(petType.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(petType);

}

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

{

if (id == null)

{

return NotFound();

}

var petType = await \_context.PetTypes

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

if (petType == null)

{

return NotFound();

}

\_context.PetTypes.Remove(petType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var petType = await \_context.PetTypes.FindAsync(id);

\_context.PetTypes.Remove(petType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool PetTypeExists(int id)

{

return \_context.PetTypes.Any(e => e.Id == id);

}

}

}

1. Modify the view **Index** on **PetTypesController**.

@model IEnumerable<MyVet.Web.Data.Entities.PetType>

@{

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">Pet 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 = '/PetTypes/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the view **Create** on **PetTypesController**.

@model MyVet.Web.Data.Entities.PetType

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Pet 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 **PetTypesController**.

@model MyVet.Web.Data.Entities.PetType

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>PetType</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. Modify the view **Details** on **PetTypesController**.

@model MyVet.Web.Data.Entities.PetType

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Pet Type</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

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

</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. Delete the view **Delete** on **PetTypesController**.
2. Test it.
3. Modify the **ServiceTypesController**.

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class ServiceTypesController : Controller

{

private readonly DataContext \_context;

public ServiceTypesController(DataContext context)

{

\_context = context;

}

public IActionResult Index()

{

return View(\_context.ServiceTypes);

}

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

{

if (id == null)

{

return NotFound();

}

var serviceType = await \_context.ServiceTypes

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

if (serviceType == null)

{

return NotFound();

}

return View(serviceType);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(ServiceType serviceType)

{

if (ModelState.IsValid)

{

\_context.Add(serviceType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(serviceType);

}

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

{

if (id == null)

{

return NotFound();

}

var serviceType = await \_context.ServiceTypes.FindAsync(id);

if (serviceType == null)

{

return NotFound();

}

return View(serviceType);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(ServiceType serviceType)

{

if (ModelState.IsValid)

{

try

{

\_context.Update(serviceType);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ServiceTypeExists(serviceType.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(serviceType);

}

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

{

if (id == null)

{

return NotFound();

}

var serviceType = await \_context.ServiceTypes

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

if (serviceType == null)

{

return NotFound();

}

\_context.ServiceTypes.Remove(serviceType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ServiceTypeExists(int id)

{

return \_context.ServiceTypes.Any(e => e.Id == id);

}

}

}

1. Modify the view **Index** on **ServiceTypesController**.

@model IEnumerable<MyVet.Web.Data.Entities.ServiceType>

@{

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">Service 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 = '/ServiceTypes/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Modify the view **Create** on **ServiceTypesController**.

@model MyVet.Web.Data.Entities.ServiceType

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Service 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 **ServiceTypesController**.

@model MyVet.Web.Data.Entities.ServiceType

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Service 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. Modify the view **Details** on **ServiceTypesController**.

@model MyVet.Web.Data.Entities.ServiceType

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Service Type</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

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

</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. Delete the view **Delete** on **ServiceTypesController**.
2. 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 MyVet.Web.Data;

using MyVet.Web.Data.Entities;

using MyVet.Web.Helpers;

using MyVet.Web.Models;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

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();

}

[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 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);

}

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, "Admin");

return newUser;

}

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 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);

}

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

{

if (id == null)

{

return NotFound();

}

var manager = await \_dataContext.Managers

.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));

}

private bool ManagerExists(int id)

{

return \_dataContext.Managers.Any(e => e.Id == id);

}

}

}

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

@model IEnumerable<MyVet.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>

<!--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 = '/Managers/Delete/' + item\_to\_delete;

});

});

</script>

}

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

@model MyVet.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 MyVet.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>

<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 **Edit** view for **ManagersController**:

@model MyVet.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" />

<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.

## Pets CRUD

1. Modify the **PetsController**:

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

using MyVet.Web.Helpers;

using MyVet.Web.Models;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class PetsController : Controller

{

private readonly ICombosHelper \_combosHelper;

private readonly DataContext \_dataContext;

public PetsController(

ICombosHelper combosHelper,

DataContext dataContext)

{

\_combosHelper = combosHelper;

\_dataContext = dataContext;

}

public IActionResult Index()

{

return View(\_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.PetType)

.Include(p => p.Histories));

}

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.Histories)

.ThenInclude(h => h.ServiceType)

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

if (pet == null)

{

return NotFound();

}

return View(pet);

}

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Owner)

.Include(p => p.PetType)

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

if (pet == null)

{

return NotFound();

}

var view = new PetViewModel

{

Born = pet.Born,

Id = pet.Id,

ImageUrl = pet.ImageUrl,

Name = pet.Name,

OwnerId = pet.Owner.Id,

PetTypeId = pet.PetType.Id,

PetTypes = \_combosHelper.GetComboPetTypes(),

Race = pet.Race,

Remarks = pet.Remarks

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(PetViewModel view)

{

if (ModelState.IsValid)

{

var path = view.ImageUrl;

if (view.ImageFile != null && view.ImageFile.Length > 0)

{

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

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

path = Path.Combine(

Directory.GetCurrentDirectory(),

"wwwroot\\images\\Pets",

file);

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

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Pets/{file}";

}

var pet = new Pet

{

Born = view.Born,

Id = view.Id,

ImageUrl = path,

Name = view.Name,

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

PetType = await \_dataContext.PetTypes.FindAsync(view.PetTypeId),

Race = view.Race,

Remarks = view.Remarks

};

\_dataContext.Pets.Update(pet);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(view);

}

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

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

if (pet == null)

{

return NotFound();

}

\_dataContext.Pets.Remove(pet);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

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

{

if (id == null)

{

return NotFound();

}

var history = await \_dataContext.Histories

.Include(h => h.Pet)

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

if (history == null)

{

return NotFound();

}

\_dataContext.Histories.Remove(history);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(Details)}/{history.Pet.Id}");

}

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

{

if (id == null)

{

return NotFound();

}

var history = await \_dataContext.Histories

.Include(h => h.Pet)

.Include(h => h.ServiceType)

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

if (history == null)

{

return NotFound();

}

var view = new HistoryViewModel

{

Date = history.Date,

Description = history.Description,

Id = history.Id,

PetId = history.Pet.Id,

Remarks = history.Remarks,

ServiceTypeId = history.ServiceType.Id,

ServiceTypes = \_combosHelper.GetComboServiceTypes()

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> EditHistory(HistoryViewModel view)

{

if (ModelState.IsValid)

{

var history = new History

{

Date = view.Date,

Description = view.Description,

Id = view.Id,

Pet = await \_dataContext.Pets.FindAsync(view.PetId),

Remarks = view.Remarks,

ServiceType = await \_dataContext.ServiceTypes.FindAsync(view.ServiceTypeId)

};

\_dataContext.Histories.Update(history);

await \_dataContext.SaveChangesAsync();

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

}

return View(view);

}

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets.FindAsync(id.Value);

if (pet == null)

{

return NotFound();

}

var view = new HistoryViewModel

{

Date = DateTime.Now,

PetId = pet.Id,

ServiceTypes = \_combosHelper.GetComboServiceTypes(),

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> AddHistory(HistoryViewModel view)

{

if (ModelState.IsValid)

{

var history = new History

{

Date = view.Date,

Description = view.Description,

Pet = await \_dataContext.Pets.FindAsync(view.PetId),

Remarks = view.Remarks,

ServiceType = await \_dataContext.ServiceTypes.FindAsync(view.ServiceTypeId)

};

\_dataContext.Histories.Add(history);

await \_dataContext.SaveChangesAsync();

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

}

return View(view);

}

}

}

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

@model IEnumerable<MyVet.Web.Data.Entities.Pet>

@{

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">Pets</h3>

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

Owner

</th>

<th>

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

</th>

<th>

@Html.DisplayNameFor(model => model.PetType.Name)

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

Histories

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:150px;height:150px;max-width: 100%; height: auto;" />

}

</td>

<td>

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

</td>

<td>

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

</td>

<td>

@Html.DisplayFor(modelItem => item.Histories.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>

<!--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 = '/Pets/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Delete the views **Create** and **Delete** for **PetsController**.
2. Modify the view **Edit** for **PetsController**:

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Pet</h4>

<hr />

<div class="row">

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

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

<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="ImageUrl" />

<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">

<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">

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

<select asp-for="PetTypeId" asp-items="Model.PetTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Born" 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">

<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 class="col-md-4">

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:300px;height:300px;max-width: 100%; height: auto;" />

}

</div>

</div>

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

@model MyVet.Web.Data.Entities.Pet

@{

ViewData["Title"] = "Details";

}

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

<h2>Pet</h2>

<div>

<h4>Details</h4>

<hr />

<div class="row">

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

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

</dl>

</div>

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

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:300px;height:300px;max-height: 100%; width: auto;" />

}

</div>

</div>

</div>

<div>

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

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

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

</div>

<hr />

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

{

<h4>Not histories added yet.</h4>

}

else

{

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

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

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Date)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().ServiceType.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Description)

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Histories)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

<a asp-action="EditHistory" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-pencil"></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 = '/Pets/DeleteHistory/' + item\_to\_delete;

});

});

</script>

}

1. Add the view **AddHistory** for **PetsController**:

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>History</h4>

<hr />

<div class="row">

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

<form asp-action="AddHistory">

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

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

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<select asp-for="ServiceTypeId" asp-items="Model.ServiceTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Description" 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">

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

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

</div>

</form>

</div>

</div>

1. Add the view **EditHistory** for **PetsController**:

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>History</h4>

<hr />

<div class="row">

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

<form asp-action="EditHistory">

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

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

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

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<select asp-for="ServiceTypeId" asp-items="Model.ServiceTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Description" 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">

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

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

</div>

</form>

</div>

</div>

1. Test it.

## Look and feel for a veterinary system and users functionality

1. Modify the **\_Layout.cshtml**:

...

<a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">My Vet</a>

…

<p>&copy; 2019 - My Vet 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/v1.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v2.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v3.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v4.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v5.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v6.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/v7.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/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-1">

<a asp-action="MyPets" class="btn btn-primary">My Pets</a>

</div>

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

<a asp-action="MyAgenda" class="btn btn-warning">My Agenda</a>

</div>

</div>

1. Test it.

## Agenda

1. Modify the view **Agenda** entity:

using System;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Web.Data.Entities

{

public class Agenda : IEntity

{

public int Id { get; set; }

[Display(Name = "Date")]

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

[DataType(DataType.DateTime)]

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

public DateTime Date { get; set; }

public Owner Owner { get; set; }

public Pet Pet { get; set; }

public string Remarks { get; set; }

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

public bool IsAvailable { get; set; }

[Display(Name = "Date")]

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

public DateTime DateLocal => Date.ToLocalTime();

}

}

**Nota**: add the ToLocalTime property to all date properties, I mean in History and Pet.

1. Add the **IAgendaHelper** interface:

using System.Threading.Tasks;

namespace MyVet.Web.Helpers

{

public interface IAgendaHelper

{

Task AddDays(int days);

}

}

1. Add the **AgendaHelper** class:

using System;

using System.Linq;

using System.Threading.Tasks;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Helpers

{

public class AgendaHelper : IAgendaHelper

{

private readonly DataContext \_dataContext;

public AgendaHelper(DataContext dataContext)

{

\_dataContext = dataContext;

}

public async Task AddDays(int days)

{

DateTime initialDate;

if (!\_dataContext.Agendas.Any())

{

initialDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day, 8, 0, 0);

}

else

{

var agenda = \_dataContext.Agendas.LastOrDefault();

initialDate = new DateTime(agenda.Date.Year, agenda.Date.Month, agenda.Date.AddDays(1).Day, 8, 0, 0);

}

var finalDate = initialDate.AddDays(days);

while (initialDate < finalDate)

{

if (initialDate.DayOfWeek != DayOfWeek.Sunday)

{

var finalDate2 = initialDate.AddHours(10);

while (initialDate < finalDate2)

{

\_dataContext.Agendas.Add(new Agenda

{

Date = initialDate.ToUniversalTime(),

IsAvailable = true

});

initialDate = initialDate.AddMinutes(30);

}

initialDate = initialDate.AddHours(14);

}

else

{

initialDate = initialDate.AddDays(1);

}

}

await \_dataContext.SaveChangesAsync();

}

}

}

1. Setup the injection for **AgendaHelper**:

services.AddScoped<IAgendaHelper, AgendaHelper>();

1. Modify the **AgendaController**:

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Web.Data;

using MyVet.Web.Helpers;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class AgendaController : Controller

{

private readonly DataContext \_dataContext;

private readonly IAgendaHelper \_agendaHelper;

public AgendaController(DataContext dataContext, IAgendaHelper agendaHelper)

{

\_dataContext = dataContext;

\_agendaHelper = agendaHelper;

}

public IActionResult Index()

{

return View(\_dataContext.Agendas

.Include(a => a.Owner)

.ThenInclude(o => o.User)

.Include(a => a.Pet)

.Where(a => a.Date >= DateTime.Today.ToUniversalTime()));

}

public async Task<IActionResult> AddDays()

{

await \_agendaHelper.AddDays(30);

return RedirectToAction(nameof(Index));

}

}

}

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

@model IEnumerable<MyVet.Web.Data.Entities.Agenda>

@{

ViewData["Title"] = "Index";

}

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

<br />

<p>

<a asp-action="AddDays" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Add Days</a>

</p>

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

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

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

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

</th>

<th>

Owner

</th>

<th>

Pet

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

@if (item.IsAvailable)

{

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

}

else

{

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

}

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<!--U-assign agenda-->

<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">Un-assign Agenda</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>Are you sure to un-assign this appoiment?</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">Un-Assign</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 = '/Agenda/Unassign/' + item\_to\_delete;

});

});

</script>

}

1. Delete the views **Create, Delete, Details, Edit** for **AgendaController**.
2. Test it.
3. Add those methods to **ICombosHelper** interface:

IEnumerable<SelectListItem> GetComboOwners();

IEnumerable<SelectListItem> GetComboPets(int ownerId);

1. Add those methods to **CombosHelper** class:

public IEnumerable<SelectListItem> GetComboOwners()

{

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

{

Text = p.User.FullNameWithDocument,

Value = p.Id.ToString()

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

list.Insert(0, new SelectListItem

{

Text = "(Select an owner...)",

Value = "0"

});

return list;

}

public IEnumerable<SelectListItem> GetComboPets(int ownerId)

{

var list = \_dataContext.Pets.Where(p => p.Owner.Id == ownerId).Select(p => new SelectListItem

{

Text = p.Name,

Value = p.Id.ToString()

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

list.Insert(0, new SelectListItem

{

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

Value = "0"

});

return list;

}

1. Add the **AgendaViewModel** class:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc.Rendering;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Models

{

public class AgendaViewModel : Agenda

{

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

[Display(Name = "Owner")]

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

public int OwnerId { get; set; }

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

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

[Display(Name = "Pet")]

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

public int PetId { get; set; }

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

}

}

1. Add those methods to **AgendaController** class:

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_dataContext.Agendas

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

if (agenda == null)

{

return NotFound();

}

var view = new AgendaViewModel

{

Id = agenda.Id,

Owners = \_combosHelper.GetComboOwners(),

Pets = \_combosHelper.GetComboPets(0)

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Assing(AgendaViewModel view)

{

if (ModelState.IsValid)

{

var agenda = await \_dataContext.Agendas.FindAsync(view.Id);

if (agenda != null)

{

agenda.IsAvailable = false;

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

agenda.Pet = await \_dataContext.Pets.FindAsync(view.PetId);

agenda.Remarks = view.Remarks;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

}

return View(view);

}

public async Task<JsonResult> GetPetsAsync(int ownerId)

{

var pets = await \_dataContext.Pets

.Where(p => p.Owner.Id == ownerId)

.OrderBy(p => p.Name)

.ToListAsync();

return Json(pets);

}

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_dataContext.Agendas

.Include(a => a.Owner)

.Include(a => a.Pet)

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

if (agenda == null)

{

return NotFound();

}

agenda.IsAvailable = true;

agenda.Pet = null;

agenda.Owner = null;

agenda.Remarks = null;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

1. Add the view **Assing** to **AgendaController** class:

@model MyVet.Web.Models.AgendaViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Assign</h2>

<h4>Agenda</h4>

<hr />

<div class="row">

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

<form asp-action="Assing" 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="OwnerId" class="control-label"></label>

<select asp-for="OwnerId" asp-items="Model.Owners" class="form-control"></select>

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

</div>

<div class="form-group">

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

<select asp-for="PetId" asp-items="Model.Pets" class="form-control"></select>

<span asp-validation-for="PetId" 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">

<input type="submit" value="Assign" 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");}

<script type="text/javascript">

$(document).ready(function () {

$("#OwnerId").change(function () {

debugger;

var x = $("#OwnerId").val();

$("#PetId").empty();

$.ajax({

type: 'POST',

url: '@Url.Action("GetPetsAsync")',

dataType: 'json',

data: { ownerId: $("#OwnerId").val() },

success: function (pets) {

$("#PetId").append('<option value="0">(Select a pet...)</option>');

$.each(pets, function (i, pet) {

$("#PetId").append('<option value="'

+ pet.id + '">'

+ pet.name + '</option>');

});

},

error: function (ex) {

alert('Failed to retrieve pets.' + ex.statusText);

}

});

return false;

})

});

</script>

}

1. Test it.

## Users functionality

1. Add the method **MyPets** for **HomeCotroller**:

[Authorize(Roles = "Customer")]

public IActionResult MyPets()

{

return View(\_dataContext.Pets

.Include(p => p.PetType)

.Include(p => p.Histories)

.Where(p => p.Owner.User.Email.ToLower().Equals(User.Identity.Name.ToLower())));

}

1. Add the view **MyPets** for **HomeCotroller**:

@model IEnumerable<MyVet.Web.Data.Entities.Pet>

@{

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">Pets</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>

@Html.DisplayNameFor(model => model.PetType.Name)

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

Histories

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:150px;height:150px;max-width: 100%; height: auto;" />

}

</td>

<td>

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

</td>

<td>

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

</td>

<td>

@Html.DisplayFor(modelItem => item.Histories.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>

<!--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 = '/Home/Delete/' + item\_to\_delete;

});

});

</script>

}

1. Test it.
2. Add those methods in **HomeCotroller**:

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Owner)

.Include(p => p.PetType)

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

if (pet == null)

{

return NotFound();

}

var view = new PetViewModel

{

Born = pet.Born,

Id = pet.Id,

ImageUrl = pet.ImageUrl,

Name = pet.Name,

OwnerId = pet.Owner.Id,

PetTypeId = pet.PetType.Id,

PetTypes = \_combosHelper.GetComboPetTypes(),

Race = pet.Race,

Remarks = pet.Remarks

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(PetViewModel view)

{

if (ModelState.IsValid)

{

var path = view.ImageUrl;

if (view.ImageFile != null && view.ImageFile.Length > 0)

{

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

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

path = Path.Combine(

Directory.GetCurrentDirectory(),

"wwwroot\\images\\Pets",

file);

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

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Pets/{file}";

}

var pet = new Pet

{

Born = view.Born,

Id = view.Id,

ImageUrl = path,

Name = view.Name,

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

PetType = await \_dataContext.PetTypes.FindAsync(view.PetTypeId),

Race = view.Race,

Remarks = view.Remarks

};

\_dataContext.Pets.Update(pet);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyPets));

}

return View(view);

}

1. Add the view **Edit** for **HomeCotroller**:

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Pet</h4>

<hr />

<div class="row">

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

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

<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="ImageUrl" />

<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">

<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">

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

<select asp-for="PetTypeId" asp-items="Model.PetTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Born" 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">

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

<a asp-action="MyPets" class="btn btn-success">Back to My Pets</a>

</div>

</form>

</div>

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

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:300px;height:300px;max-width: 100%; height: auto;" />

}

</div>

</div>

1. Test it.
2. Add the method **Details** for **HomeCotroller**:

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.Histories)

.ThenInclude(h => h.ServiceType)

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

if (pet == null)

{

return NotFound();

}

return View(pet);

}

1. Add the view **Details** for **HomeCotroller**:

@model MyVet.Web.Data.Entities.Pet

@{

ViewData["Title"] = "Details";

}

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

<h2>Pet</h2>

<div>

<h4>Details</h4>

<hr />

<div class="row">

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

<dl class="dl-horizontal">

<dt>

Owner

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

<dt>

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

</dt>

<dd>

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

</dd>

</dl>

</div>

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

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

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:300px;height:300px;max-height: 100%; width: auto;" />

}

</div>

</div>

</div>

<div>

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

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

</div>

<hr />

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

{

<h4>Not histories added yet.</h4>

}

else

{

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

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

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Date)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().ServiceType.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Histories.FirstOrDefault().Description)

</th>

<th>

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

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Histories)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</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 **Delete** for **HomeCotroller**:

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_dataContext.Pets

.Include(p => p.Histories)

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

if (pet == null)

{

return NotFound();

}

if (pet.Histories.Count > 0)

{

return RedirectToAction(nameof(MyPets));

}

\_dataContext.Pets.Remove(pet);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyPets));

}

1. Add those methods for **HomeCotroller**:

[Authorize(Roles = "Customer")]

public async Task<IActionResult> Create()

{

var owner = await \_dataContext.Owners

.FirstOrDefaultAsync(o => o.User.Email.ToLower().Equals(User.Identity.Name.ToLower()));

if (owner == null)

{

return NotFound();

}

var view = new PetViewModel

{

Born = DateTime.Now,

PetTypes = \_combosHelper.GetComboPetTypes(),

OwnerId = owner.Id

};

return View(view);

}

[HttpPost]

public async Task<IActionResult> Create(PetViewModel 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\\Pets",

file);

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

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Pets/{file}";

}

var pet = new Pet

{

Born = view.Born,

ImageUrl = path,

Name = view.Name,

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

PetType = await \_dataContext.PetTypes.FindAsync(view.PetTypeId),

Race = view.Race,

Remarks = view.Remarks

};

\_dataContext.Pets.Add(pet);

await \_dataContext.SaveChangesAsync();

return RedirectToAction($"{nameof(MyPets)}");

}

return View(view);

}

1. Add the view **Create** for **HomeCotroller**:

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Pet</h4>

<hr />

<div class="row">

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

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

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

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

<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">

<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">

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

<select asp-for="PetTypeId" asp-items="Model.PetTypes" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

<span asp-validation-for="Born" 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">

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

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

</div>

</form>

</div>

</div>

1. Test it.
2. Add those properties to **AgendaViewModel**:

public bool IsMine { get; set; }

public string Reserved => "Reserved";

1. Add the method **MyAgenda** to **HomeController**:

[Authorize(Roles = "Customer")]

public async Task<IActionResult> MyAgenda()

{

var agendas = await \_dataContext.Agendas

.Include(a => a.Owner)

.ThenInclude(o => o.User)

.Include(a => a.Pet)

.Where(a => a.Date >= DateTime.Today.ToUniversalTime()).ToListAsync();

var list = new List<AgendaViewModel>(agendas.Select(a => new AgendaViewModel

{

Date = a.Date,

Id = a.Id,

IsAvailable = a.IsAvailable,

Owner = a.Owner,

Pet = a.Pet,

Remarks = a.Remarks

}).ToList());

list.Where(a => a.Owner != null && a.Owner.User.UserName.ToLower().Equals(User.Identity.Name.ToLower()))

.All(a => { a.IsMine = true; return true; });

return View(list);

}

1. Add the view **MyAgenda** to **HomeController**:

@model IEnumerable<MyVet.Web.Models.AgendaViewModel>

@{

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">Agenda</h3>

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

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

</th>

<th>

Owner

</th>

<th>

Pet

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

@if (item.IsMine)

{

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

}

else if (!item.IsAvailable)

{

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

}

</td>

<td>

@if (item.IsMine)

{

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

}

</td>

<td>

@if (item.IsMine)

{

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

}

</td>

<td>

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

</td>

<td>

@if (item.IsAvailable)

{

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

}

else

{

@if (item.IsMine)

{

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

}

}

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<!-- Un-assign agenda-->

<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">Un-assign Agenda</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>Are you sure to un-assign this appoiment?</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">Un-Assign</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 = '/Home/Unassign/' + item\_to\_delete;

});

});

</script>

}

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

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_dataContext.Agendas

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

if (agenda == null)

{

return NotFound();

}

var owner = await \_dataContext.Owners.FirstOrDefaultAsync(o => o.User.UserName.ToLower().Equals(User.Identity.Name.ToLower()));

if (owner == null)

{

return NotFound();

}

var view = new AgendaViewModel

{

Id = agenda.Id,

OwnerId = owner.Id,

Pets = \_combosHelper.GetComboPets(owner.Id)

};

return View(view);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Assing(AgendaViewModel view)

{

if (ModelState.IsValid)

{

var agenda = await \_dataContext.Agendas.FindAsync(view.Id);

if (agenda != null)

{

agenda.IsAvailable = false;

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

agenda.Pet = await \_dataContext.Pets.FindAsync(view.PetId);

agenda.Remarks = view.Remarks;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyAgenda));

}

}

return View(view);

}

1. Add the view **Assing** to **HomeController**:

@model MyVet.Web.Models.AgendaViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Assign</h2>

<h4>Agenda</h4>

<hr />

<div class="row">

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

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

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

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

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

<div class="form-group">

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

<select asp-for="PetId" asp-items="Model.Pets" class="form-control"></select>

<span asp-validation-for="PetId" 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">

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

<a asp-action="MyAgenda" class="btn btn-success">Back to My Agenda</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Test it.
2. Add the method **Unassign** to **HomeController**:

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_dataContext.Agendas

.Include(a => a.Owner)

.Include(a => a.Pet)

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

if (agenda == null)

{

return NotFound();

}

agenda.IsAvailable = true;

agenda.Pet = null;

agenda.Owner = null;

agenda.Remarks = null;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return RedirectToAction(nameof(MyAgenda));

}

1. Test it.

## 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 pets, see the agenda and reserve and cancel appointments. To do that, first we need to modify the API.

### Account

1. Add the **UserRequest** model:

using System.ComponentModel.DataAnnotations;

namespace MyVet.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; }

}

}

1. Add the **AccountController** API:

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using MyVet.Common.Models;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

using MyVet.Web.Helpers;

namespace MyVet.Web.Controllers.API

{

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

public class AccountController : Controller

{

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

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user != null)

{

return BadRequest(new Response

{

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);

await \_userHelper.AddUserToRoleAsync(userNew, "Customer");

\_dataContext.Owners.Add(new Owner { 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

{

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 MyVet.Common.Models

{

public class EmailRequest

{

[Required]

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

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user == null)

{

return BadRequest(new Response

{

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

{

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 MyVet.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

{

IsSuccess = false,

Message = "Bad request"

});

}

var user = await \_userHelper.GetUserByEmailAsync(request.Email);

if (user == null)

{

return BadRequest(new Response

{

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

{

IsSuccess = false,

Message = result.Errors.FirstOrDefault().Description

});

}

return Ok(new Response

{

IsSuccess = true,

Message = "The password was changed successfully!"

});

}

1. Test it on postman.

### Pets

1. Add the **FilesHelper** class:

using System.IO;

namespace MyVet.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 **PetRequest** model:

using System;

using System.ComponentModel.DataAnnotations;

namespace MyVet.Common.Models

{

public class PetRequest

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

public string Race { get; set; }

public int OwnerId { get; set; }

public int PetTypeId { get; set; }

[Required]

public DateTime Born { get; set; }

public string Remarks { get; set; }

public byte[] ImageArray { get; set; }

}

}

1. Add the **PetsController** API:

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Controllers.API

{

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class PetsController : ControllerBase

{

private readonly DataContext \_dataContext;

public PetsController(DataContext dataContext)

{

\_dataContext = dataContext;

}

[HttpPost]

public async Task<IActionResult> PostPet([FromBody] PetRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var owner = await \_dataContext.Owners.FindAsync(request.OwnerId);

if (owner == null)

{

return BadRequest("Not valid owner.");

}

var petType = await \_dataContext.PetTypes.FindAsync(request.PetTypeId);

if (petType == null)

{

return BadRequest("Not valid pet type.");

}

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\\Pets";

var fullPath = $"~/images/Pets/{file}";

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

if (response)

{

imageUrl = fullPath;

}

}

var pet = new Pet

{

Born = request.Born,

ImageUrl = imageUrl,

Name = request.Name,

Owner = owner,

PetType = petType,

Race = request.Race,

Remarks = request.Remarks

};

\_dataContext.Pets.Add(pet);

await \_dataContext.SaveChangesAsync();

return Ok(pet);

}

[HttpPut("{id}")]

public async Task<IActionResult> PutPet([FromRoute] int id, [FromBody] PetRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != request.Id)

{

return BadRequest();

}

var oldPet = await \_dataContext.Pets.FindAsync(request.Id);

if (oldPet == null)

{

return BadRequest("Pet doesn't exists.");

}

var petType = await \_dataContext.PetTypes.FindAsync(request.PetTypeId);

if (petType == null)

{

return BadRequest("Not valid pet type.");

}

var imageUrl = oldPet.ImageUrl;

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\\Pets";

var fullPath = $"~/images/Pets/{file}";

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

if (response)

{

imageUrl = fullPath;

}

}

oldPet.Born = request.Born;

oldPet.ImageUrl = imageUrl;

oldPet.Name = request.Name;

oldPet.PetType = petType;

oldPet.Race = request.Race;

oldPet.Remarks = request.Remarks;

\_dataContext.Pets.Update(oldPet);

await \_dataContext.SaveChangesAsync();

return Ok(oldPet);

}

}

}

1. Test it on postman.
2. Add the **PetTypesController** API:

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

namespace MyVet.Web.Controllers.API

{

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class PetTypesController : Controller

{

private readonly DataContext \_dataContext;

public PetTypesController(DataContext dataContext)

{

\_dataContext = dataContext;

}

[HttpGet]

public IEnumerable<PetType> GetPetTypes()

{

return \_dataContext.PetTypes.OrderBy(pt => pt.Name);

}

}

}

1. Test it on postman.

### Agenda

1. Add the **AgendaResponse** class:

using System;

namespace MyVet.Common.Models

{

public class AgendaResponse

{

public int Id { get; set; }

public DateTime Date { get; set; }

public OwnerResponse Owner { get; set; }

public PetResponse Pet { get; set; }

public string Remarks { get; set; }

public bool IsAvailable { get; set; }

public DateTime DateLocal => Date.ToLocalTime();

}

}

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

PetResponse ToPetResponse(Pet pet);

OwnerResponse ToOwnerResposne(Owner owner);

And the implementation:

public PetResponse ToPetResponse(Pet pet)

{

if (pet == null)

{

return null;

}

return new PetResponse

{

Born = pet.Born,

Id = pet.Id,

ImageUrl = pet.ImageFullPath,

Name = pet.Name,

PetType = pet.PetType.Name,

Race = pet.Race,

Remarks = pet.Remarks

};

}

public OwnerResponse ToOwnerResposne(Owner owner)

{

if (owner == null)

{

return null;

}

return new OwnerResponse

{

Address = owner.User.Address,

Document = owner.User.Document,

Email = owner.User.Email,

FirstName = owner.User.FirstName,

LastName = owner.User.LastName,

PhoneNumber = owner.User.PhoneNumber

};

}

1. Add the **AgendaController** class:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyVet.Common.Models;

using MyVet.Web.Data;

using MyVet.Web.Helpers;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.Web.Controllers.API

{

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class AgendaController : Controller

{

private readonly DataContext \_dataContext;

private readonly IConverterHelper \_converterHelper;

public AgendaController(

DataContext dataContext,

IConverterHelper converterHelper)

{

\_dataContext = dataContext;

\_converterHelper = converterHelper;

}

[HttpPost]

[Route("GetAgendaForOwner")]

public async Task<IActionResult> GetAgendaForOwner(EmailRequest emailRequest)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var agendas = await \_dataContext.Agendas

.Include(a => a.Owner)

.ThenInclude(o => o.User)

.Include(a => a.Pet)

.ThenInclude(p => p.PetType)

.Where(a => a.Date >= DateTime.Today.ToUniversalTime())

.OrderBy(a => a.Date)

.ToListAsync();

var response = new List<AgendaResponse>();

foreach (var agenda in agendas)

{

var agendaRespose = new AgendaResponse

{

Date = agenda.Date,

Id = agenda.Id,

IsAvailable = agenda.IsAvailable

};

if (agenda.Owner != null)

{

if (agenda.Owner.User.Email.ToLower().Equals(emailRequest.Email.ToLower()))

{

agendaRespose.Owner = \_converterHelper.ToOwnerResposne(agenda.Owner);

agendaRespose.Pet = \_converterHelper.ToPetResponse(agenda.Pet);

agendaRespose.Remarks = agenda.Remarks;

}

else

{

agendaRespose.Owner = new OwnerResponse { FirstName = "Reserved" };

}

}

response.Add(agendaRespose);

}

return Ok(response);

}

}

}

1. Test it on postman.
2. Add the **AssignRequest** model:

using System.ComponentModel.DataAnnotations;

namespace MyVet.Common.Models

{

public class AssignRequest

{

[Required]

public int AgendaId { get; set; }

[Required]

public int OwnerId { get; set; }

[Required]

public int PetId { get; set; }

public string Remarks { get; set; }

}

}

1. Add the method **AssignAgenda** to **AgendaController** API:

[HttpPost]

[Route("AssignAgenda")]

public async Task<IActionResult> AssignAgenda(AssignRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var agenda = await \_dataContext.Agendas.FindAsync(request.AgendaId);

if (agenda == null)

{

return BadRequest("Agenda doesn't exists.");

}

if(!agenda.IsAvailable)

{

return BadRequest("Agenda is not available.");

}

var owner = await \_dataContext.Owners.FindAsync(request.OwnerId);

if (owner == null)

{

return BadRequest("Owner doesn't exists.");

}

var pet = await \_dataContext.Pets.FindAsync(request.PetId);

if (pet == null)

{

return BadRequest("Pet doesn't exists.");

}

agenda.IsAvailable = false;

agenda.Remarks = request.Remarks;

agenda.Owner = owner;

agenda.Pet = pet;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return Ok(agenda);

}

1. Test it on postman.
2. Add the **UnAssignRequest** model:

using System.ComponentModel.DataAnnotations;

namespace MyVet.Common.Models

{

public class UnAssignRequest

{

[Required]

public int AgendaId { get; set; }

}

}

1. Add the method **UnAssignAgenda** to **AgendaController** API:

[HttpPost]

[Route("UnAssignAgenda")]

public async Task<IActionResult> UnAssignAgenda(UnAssignRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var agenda = await \_dataContext.Agendas

.Include(a => a.Owner)

.Include(a => a.Pet)

.FirstOrDefaultAsync(a => a.Id == request.AgendaId);

if (agenda == null)

{

return BadRequest("Agenda doesn't exists.");

}

if (agenda.IsAvailable)

{

return BadRequest("Agenda is available.");

}

agenda.IsAvailable = true;

agenda.Remarks = null;

agenda.Owner = null;

agenda.Pet = null;

\_dataContext.Agendas.Update(agenda);

await \_dataContext.SaveChangesAsync();

return Ok(agenda);

}

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 MyVet.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 = (OwnerResponse)response2.Result;

Settings.Owner = JsonConvert.SerializeObject(owner);

Settings.Token = JsonConvert.SerializeObject(token);

IsEnabled = true;

IsRunning = false;

await \_navigationService.NavigateAsync("Pets");

1. Delete the method **OnNavigatedTo** and load the date on **PetsViewModel** constructor:

public PetsViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Pets";

LoadPets();

}

public bool IsRefreshing

{

get => \_isRefreshing;

set => SetProperty(ref \_isRefreshing, value);

}

private void LoadPets()

{

IsRefreshing = true;

\_token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

\_owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

Pets = new ObservableCollection<PetItemViewModel>(\_owner.Pets.Select(p => new PetItemViewModel(\_navigationService)

{

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.

## Add a Master Detail

1. Add the **Menu** model:

namespace MyVet.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 MyVet.Common.Models;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.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/Login");

return;

}

await \_navigationService.NavigateAsync($"/MyMasterDetail/NavigationPage/{PageName}");

}

}

}

1. Add the images for logo.
2. Add the **MasterDetailPage** called **VeterinaryMasterDetailPage**:

<?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="MyVet.Prism.Views.MyMasterDetail">

<MasterDetailPage.Master>

<ContentPage

BackgroundColor="Gray"

Title="Menu">

<StackLayout Padding="20">

<Image

HeightRequest="150"

Source="vet\_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. Modify the **VeterinaryMasterDetailPageViewModel**:

using MyVet.Common.Models;

using Prism.Navigation;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

namespace MyVet.Prism.ViewModels

{

public class VeterinaryMasterDetailPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

public VeterinaryMasterDetailPageViewModel(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\_pets\_menu",

PageName = "PetsPage",

Title = "My Pets"

},

new Menu

{

Icon = "ic\_list\_alt",

PageName = "AgendaPage",

Title = "My Agenda"

},

new Menu

{

Icon = "ic\_map",

PageName = "MapPage",

Title = "Map"

},

new Menu

{

Icon = "ic\_person",

PageName = "ProfilePage",

Title = "Modify Profile"

},

new Menu

{

Icon = "ic\_exit\_to\_app",

PageName = "LoginPage",

Title = "Logout"

}

};

Menus = new ObservableCollection<MenuItemViewModel>(

menus.Select(m => new MenuItemViewModel(\_navigationService)

{

Icon = m.Icon,

PageName = m.PageName,

Title = m.Title

}).ToList());

}

}

}

1. Add the images for menu icons.
2. Add the page **Agenda** 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="MyVet.Prism.Views.Agenda"

Title="{Binding Title}">

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="CenterAndExpand">

<Label Text="{Binding Title}"/>

</StackLayout>

</ContentPage>

1. Modify the **AgendaViewModel**:

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class AgendaViewModel : ViewModelBase

{

public AgendaViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Agenda";

}

}

}

1. Add the page **Map** 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="MyVet.Prism.Views.Map"

Title="{Binding Title}">

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="CenterAndExpand">

<Label Text="{Binding Title}"/>

</StackLayout>

</ContentPage>

1. Modify the **MapViewModel**:

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class MapViewModel : ViewModelBase

{

public MapViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Map";

}

}

}

1. Modify the **LoginViewModel**:

IsEnabled = true;

IsRunning = false;

await \_navigationService.NavigateAsync("/VeterinaryMasterDetailPage/NavigationPage/PetsPage");

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: **vet\_splash.png**.
2. Add this lines to **styles.xml**.

</style>

<style name="Theme.Splash" parent="android:Theme">

<item name="android:windowBackground">@drawable/vet\_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 MyVet.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 Vet",

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. 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 **Login.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"

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="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

BackgroundColor="{StaticResource colorPrimaryDark}"

BorderRadius="23"

Command="{Binding LoginCommand}"

HeightRequest="46"

IsEnabled="{Binding IsEnabled}"

Text="Login"

TextColor="{StaticResource colorFontInverse}"/>

</StackLayout>

</ScrollView>

</ContentPage>

1. Test it.
2. Add this style to dictionary:

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">https://myvet.azurewebsites.net</x:String>

<!-- Colors -->

<Color x:Key="colorPrimaryDark">#1976D2</Color>

<Color x:Key="colorFontInverse">#FFFFFF</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>

</ResourceDictionary>

1. Modify the login button on **Login.xaml**:

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Login"/>

1. Test it.
2. Add those values to dictionary:

<!-- 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>

<!-- 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>

1. Modify the **Login.xaml**:

<StackLayout

Orientation="Horizontal">

<Button

Command="{Binding LoginCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="Login"/>

<Button

Command="{Binding RegisterCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="Register"/>

</StackLayout>

1. Test it.
2. Modify the **Login.xaml**:

<Entry

IsPassword="True"

Placeholder="Enter your password..."

Text="{Binding Password}"/>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal">

<Label

Text="Rememberme in this device"/>

<Switch

IsToggled="{Binding IsRemember}"/>

</StackLayout>

<Label

HorizontalOptions="Center"

Text="Forgot your password?"

TextColor="Navy">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<StackLayout

1. Finally, add the property **IsRemember** to **LoginViewModel**:

…

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 MyVet.Prism.Interfaces

{

public interface ILocalize

{

CultureInfo GetCurrentCultureInfo();

void SetLocale(CultureInfo ci);

}

}

1. In the folder **Helpers** add the class **PlatformCulture**.

using System;

namespace MyVet.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 MyVet.Prism.Interfaces;

using MyVet.Prism.Resources;

using Xamarin.Forms;

namespace MyVet.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(MyVet.Prism.iOS.Implementations.Localize))]

namespace MyVet.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(MyVet.Prism.Droid.Implementations.Localize))]

namespace MyVet.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 MyVet.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

}

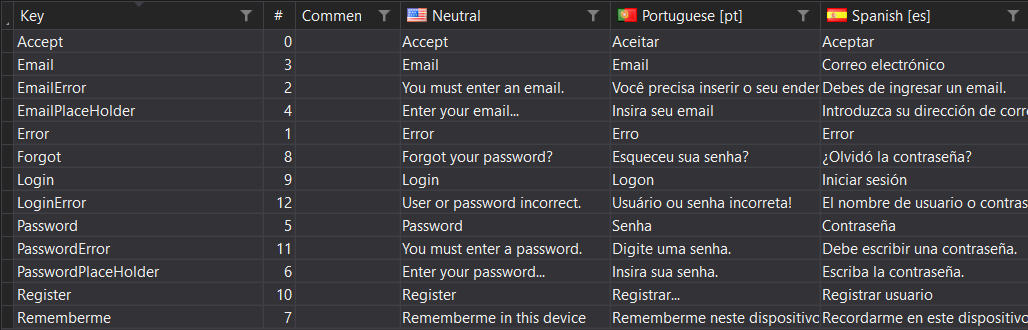
return translation;

}

}

}

1. Complete the literals:



1. And add the properties in **Languages** class:

using MyVet.Prism.Interfaces;

using MyVet.Prism.Resources;

using Xamarin.Forms;

namespace MyVet.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**:

<?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="MyVet.Prism.Views.RegisterPage"

BackgroundColor="{StaticResource colorBackgroud}"

Title="{Binding Title}">

</ContentPage>

1. Modify the **RegisterPageViewModel** class:

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class RegisterViewModel : ViewModelBase

{

public RegisterViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.Register;

}

}

}

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. Add those literals on **Resources**:

For English:

<data name="Document" xml:space="preserve">

<value>Document</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>First Name</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Last Name</value>

</data>

<data name="Address" xml:space="preserve">

<value>Address</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Phone</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Password Confirm</value>

</data>

<data name="DocumentPlaceHolder" xml:space="preserve">

<value>Enter your document.</value>

</data>

<data name="FirstNamePlaceHolder" xml:space="preserve">

<value>Enter your first name...</value>

</data>

<data name="LastNamePlaceHolder" xml:space="preserve">

<value>Enter your last name...</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Enter your address...</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Enter your phone number...</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Enter your password confirm...</value>

</data>

<data name="DocumentError" xml:space="preserve">

<value>You must enter a document.</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>You must enter the first name.</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>You must enter the last name.</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>You must enter a confirm.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>You must enter a phone number.</value>

</data>

<data name="AddressError" xml:space="preserve">

<value>You must enter the address.</value>

</data>

<data name="PasswordError2" xml:space="preserve">

<value>The password must be a least 6 characters length.</value>

</data>

<data name="PasswordError3" xml:space="preserve">

<value>The password and confirm does not match.</value>

</data>

For Spanish:

<data name="Document" xml:space="preserve">

<value>Documento</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>Nombre</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Apellidos</value>

</data>

<data name="Address" xml:space="preserve">

<value>Dirección</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Teléfono</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Confirmar contraseña</value>

</data>

<data name="DocumentPlaceHolder" xml:space="preserve">

<value>Ingrese su documento.</value>

</data>

<data name="FirstNamePlaceHolder" xml:space="preserve">

<value>Escribir nombre</value>

</data>

<data name="LastNamePlaceHolder" xml:space="preserve">

<value>Escribir apellidos</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Introduzca su dirección</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Escribir el número de teléfono</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Ingrese la confirmación de contraseña</value>

</data>

<data name="DocumentError" xml:space="preserve">

<value>Debe ingresar un documento.</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>Debe introducir un nombre</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>Debe introducir el apellido</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>Debe ingresar una confirmación de contraseña</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>Debes poner un número de teléfono</value>

</data>

<data name="AddressError" xml:space="preserve">

<value>Debes poner una dirección</value>

</data>

<data name="PasswordError2" xml:space="preserve">

<value>La contraseña debe tener al menos 6 caracteres de longitud.</value>

</data>

<data name="PasswordError3" xml:space="preserve">

<value>La nueva contraseña y la confirmación de contraseña no son iguales.</value>

</data>

For Portuguese:

<data name="Document" xml:space="preserve">

<value>Documento</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>Nome</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Sobrenome</value>

</data>

<data name="Address" xml:space="preserve">

<value>Endereço:</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Telefone</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Confirme a senha</value>

</data>

<data name="DocumentPlaceHolder" xml:space="preserve">

<value>Digite seu documento.</value>

</data>

<data name="FirstNamePlaceHolder" xml:space="preserve">

<value>Insira seu nome</value>

</data>

<data name="LastNamePlaceHolder" xml:space="preserve">

<value>Insira seu sobrenome</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Insira o seu endereço...</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Insira o seu número de telefone</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Digite sua senha para confirmar</value>

</data>

<data name="DocumentError" xml:space="preserve">

<value>Você deve inserir um documento.</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>Você deve digitar o nome</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>Você deve inserir o sobrenome</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>Você deve inserir uma confirmação.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>Você deve digitar um número de telefone.</value>

</data>

<data name="AddressError" xml:space="preserve">

<value>Você deve digitar um endereço</value>

</data>

<data name="PasswordError2" xml:space="preserve">

<value>A senha deve ter no mínimo 6 caracteres.</value>

</data>

<data name="PasswordError3" xml:space="preserve">

<value>A nova senha e senha de confirmação não coincidem.</value>

</data>

1. Modify **Languages** class:

using MyVet.Prism.Interfaces;

using MyVet.Prism.Resources;

using Xamarin.Forms;

namespace MyVet.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 Address => Resource.Address;

public static string AddressError => Resource.AddressError;

public static string AddressPlaceHolder => Resource.AddressPlaceHolder;

public static string Document => Resource.Document;

public static string DocumentError => Resource.DocumentError;

public static string DocumentPlaceHolder => Resource.DocumentPlaceHolder;

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 FirstName => Resource.FirstName;

public static string FirstNameError => Resource.FirstNameError;

public static string FirstNamePlaceHolder => Resource.FirstNamePlaceHolder;

public static string Forgot => Resource.Forgot;

public static string LastName => Resource.LastName;

public static string LastNameError => Resource.LastNameError;

public static string LastNamePlaceHolder => Resource.LastNamePlaceHolder;

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 PasswordError2 => Resource.PasswordError2;

public static string PasswordError3 => Resource.PasswordError3;

public static string PasswordPlaceHolder => Resource.PasswordPlaceHolder;

public static string PasswordConfirm => Resource.PasswordConfirm;

public static string PasswordConfirmError => Resource.PasswordConfirmError;

public static string PasswordConfirmPlaceHolder => Resource.PasswordConfirmPlaceHolder;

public static string Phone => Resource.Phone;

public static string PhoneError => Resource.PhoneError;

public static string PhonePlaceHolder => Resource.PhonePlaceHolder;

public static string Register => Resource.Register;

public static string Rememberme => Resource.Rememberme;

}

}

1. 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="MyVet.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..."

Text="{Binding Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="First Name"

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 phonenumber..."

Text="{Binding Phone}"/>

<Label

Grid.Row="6"

Grid.Column="0"

Text="Password"

VerticalOptions="Center"/>

<Entry

Grid.Row="6"

Grid.Column="1"

IsPassword="True"

Placeholder="Enter your password..."

Text="{Binding Password}"/>

<Label

Grid.Row="7"

Grid.Column="0"

Text="Password Confirm"

VerticalOptions="Center"/>

<Entry

Grid.Row="7"

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="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Add the **RegexHelper** class:

using System;

using System.Net.Mail;

namespace MyVet.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;

using System.Threading.Tasks;

using MyVet.Common.Helpers;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.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;

}

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 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 firsname.", "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 a valid email.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Phone))

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a phone.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Password) || Password.Length < 6)

{

await App.Current.MainPage.DisplayAlert("Error", "You must enter a password at least 6 character.", "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;

}

}

}

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. Add the literal for **Ok**.

1. Modify the **RegisterViewModel**:

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

};

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. 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="MyVet.Prism.Views.RememberPassword"

Title="{Binding Title}">

</ContentPage>

1. Modify the **RememberPasswordViewModel**:

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.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:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.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 **RememberPasswordViewModel**:

using System.Threading.Tasks;

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using MyVet.Common.Services;

using MyVet.Prism.Helpers;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class RememberPasswordViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_recoverCommand;

public RememberPasswordViewModel(

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 MyVet.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()

{

InitializeComponent();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

if (Settings.IsRemembered && token?.Expiration > DateTime.Now)

{

await NavigationService.NavigateAsync("/MyMasterDetail/NavigationPage/Pets");

}

else

{

await NavigationService.NavigateAsync("/NavigationPage/Login");

}

}

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**, **MyPets**, **MyAgenda**, **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="MyVet.Prism.Views.Profile"

Title="{Binding Title}">

</ContentPage>

1. Modify the **ProfileViewModel** class:

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.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\_pets",

PageName = "Pets",

Title = Languages.MyPets

},

new Menu

{

Icon = "ic\_calendar\_today",

PageName = "Agenda",

Title = Languages.MyAgenda

},

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 **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:i18n="clr-namespace:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.Profile"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

Text="{i18n:Translate Document}"

VerticalOptions="Center"/>

<Entry

Grid.Row="0"

Grid.Column="1"

Placeholder="{i18n:Translate DocumentPlaceHolder}"

Text="{Binding Owner.Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="{i18n:Translate FirstName}"

VerticalOptions="Center"/>

<Entry

Grid.Row="1"

Grid.Column="1"

Placeholder="{i18n:Translate FirstNamePlaceHolder}"

Text="{Binding Owner.FirstName}"/>

<Label

Grid.Row="2"

Grid.Column="0"

Text="{i18n:Translate LastName}"

VerticalOptions="Center"/>

<Entry

Grid.Row="2"

Grid.Column="1"

Placeholder="{i18n:Translate LastNamePlaceHolder}"

Text="{Binding Owner.LastName}"/>

<Label

Grid.Row="3"

Grid.Column="0"

Text="{i18n:Translate Address}"

VerticalOptions="Center"/>

<Entry

Grid.Row="3"

Grid.Column="1"

Placeholder="{i18n:Translate AddressPlaceHolder}"

Text="{Binding Owner.Address}"/>

<Label

Grid.Row="4"

Grid.Column="0"

Text="{i18n:Translate Phone}"

VerticalOptions="Center"/>

<Entry

Grid.Row="4"

Grid.Column="1"

Placeholder="{i18n:Translate PhonePlaceHolder}"

Text="{Binding Owner.PhoneNumber}"/>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<StackLayout

Orientation="Horizontal">

<Button

Command="{Binding SaveCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Save}"/>

<Button

Command="{Binding ChangePasswordCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="{i18n:Translate ChangePassword}"/>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modify the **ProfileViewModel**:

using System.Threading.Tasks;

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using MyVet.Prism.Helpers;

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class ProfileViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private OwnerResponse \_owner;

private DelegateCommand \_saveCommand;

public ProfileViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.MyProfile;

IsEnabled = true;

Owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

}

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(Save));

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 Save()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Owner.Document))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.DocumentError, Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(Owner.FirstName))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.FirstNameError, Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(Owner.LastName))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.LastNameError, Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(Owner.Address))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.AddressError, Languages.Accept);

return false;

}

return true;

}

}

}

1. Test it, that we do until this point.
2. Add the method **PutAsync** in **IApiService**:

Task<Response> 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> 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

{

IsSuccess = false,

Message = answer,

};

}

var obj = JsonConvert.DeserializeObject<T>(answer);

return new Response

{

IsSuccess = true,

Result = obj,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Add a literal for **UserUpdated**:

public static string UserUpdated => Resource.UserUpdated;

1. Modify the **ProfileViewModel**:

private async void Save()

{

var isValid = await ValidateData();

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(

Languages.Error,

response.Message,

Languages.Accept);

return;

}

Settings.Owner = JsonConvert.SerializeObject(Owner);

await App.Current.MainPage.DisplayAlert(

Languages.Ok,

Languages.UserUpdated,

Languages.Accept);

await \_navigationService.GoBackAsync();

}

1. Test it.

## Modify Password From App in Xamarin Forms

1. Add the method **ChangePasswordAsync** in **IApiService**:

Task<Response> ChangePasswordAsync(

string urlBase,

string servicePrefix,

string controller,

ChangePasswordRequest changePasswordRequest,

string tokenType,

string accessToken);

1. Add the method **ChangePasswordAsync** in **ApiService**:

public async Task<Response> 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>(answer);

return obj;

}

catch (Exception ex)

{

return new Response

{

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:i18n="clr-namespace:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.ChangePassword"

Title="{Binding Title}">

<ScrollView>

<StackLayout

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate CurrentPassword}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

IsPassword="True"

Placeholder="{i18n:Translate CurrentPasswordPlaceHolder}"

Text="{Binding CurrentPassword}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate NewPassword}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

IsPassword="True"

Placeholder="{i18n:Translate NewPasswordPlaceHolder}"

Text="{Binding NewPassword}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate ConfirmNewPassword}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="2"

IsPassword="True"

Placeholder="{i18n:Translate ConfirmNewPasswordPlaceHolder}"

Text="{Binding PasswordConfirm}"/>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<Button

Command="{Binding ChangePasswordCommand}"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate ChangePassword}"/>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modify the **ChangePasswordPageViewModel**:

using System.Threading.Tasks;

using MyVet.Common.Services;

using MyVet.Prism.Helpers;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.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 = Languages.ChangePassword;

}

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePassword));

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 ChangePassword()

{

var isValid = await ValidateData();

if (!isValid)

{

return;

}

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(CurrentPassword))

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.CurrentPasswordError,

Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(NewPassword) || NewPassword?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.NewPasswordError,

Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.ConfirmNewPasswordError,

Languages.Accept);

return false;

}

if (!NewPassword.Equals(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PasswordError3,

Languages.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 ChangePassword()

{

var isValid = await ValidateData();

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(

Languages.Error,

response.Message,

Languages.Accept);

return;

}

await App.Current.MainPage.DisplayAlert(

Languages.Ok,

response.Message,

Languages.Accept);

await \_navigationService.GoBackAsync();

}

1. Test it.

## App Pet From App & Accessing Camera and Photo Library

1. Add the icon for toolbar **ic\_action\_add\_circle**.
2. Modify the **Pets** 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="MyVet.Prism.Views.Pets"

BackgroundColor="Silver"

Title="{Binding Title}">

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add\_circle" Command="{Binding AddPetCommand}"/>

</ContentPage.ToolbarItems>

<StackLayout

Padding="10">

<ListView

1. Modify the **PetsViewModel** class:

private DelegateCommand \_addPetCommand;

…

public DelegateCommand AddPetCommand => \_addPetCommand ?? (\_addPetCommand = new DelegateCommand(AddPet));

…

private async void AddPet()

{

await \_navigationService.NavigateAsync("EditPet");

}

1. Add the **EditPet** 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="MyVet.Prism.Views.EditPet"

Title="{Binding Title}">

</ContentPage>

1. Add literal for: **NewPet**:

public static string NewPet => Resource.NewPet;

1. Modify the **EditPetViewModel** class:

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class EditPetViewModel : ViewModelBase

{

public EditPetViewModel(INavigationService navigationService) : base(navigationService)

{

Title = Languages.NewPet;

}

}

}

1. Test it, that we do until this point.
2. Add literals for: **Delete, EditPet, ChangeImage, Name, NameError, NamePlaceHolder, Race, RaceError, RacePlaceHolder, PetType, PetTypeError, PetTypePlaceHolder, Born** and **Remarks**.

public static string Delete => Resource.Delete;

public static string EditPet => Resource.EditPet;

public static string ChangeImage => Resource.ChangeImage;

public static string Name => Resource.Name;

public static string NameError => Resource.NameError;

public static string NamePlaceHolder => Resource.NamePlaceHolder;

public static string Race => Resource.Race;

public static string RaceError => Resource.RaceError;

public static string RacePlaceHolder => Resource.RacePlaceHolder;

public static string PetType => Resource.PetType;

public static string PetTypeError => Resource.PetTypeError;

public static string PetTypePlaceHolder => Resource.PetTypePlaceHolder;

public static string Born => Resource.Born;

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 **EditPet** 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:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.EditPet"

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"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate Name}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

Placeholder="{i18n:Translate NamePlaceHolder}"

Text="{Binding Pet.Name}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate Race}"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

Placeholder="{i18n:Translate RacePlaceHolder}"

Text="{Binding Pet.Race}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate PetType}"

VerticalOptions="Center"/>

<Picker

Grid.Column="1"

Grid.Row="2"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding PetTypes}"

SelectedItem="{Binding PetType}"

Title="{i18n:Translate PetTypePlaceHolder}"/>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Born}"

VerticalOptions="Center"/>

<DatePicker

Grid.Column="1"

Grid.Row="3"

Date="{Binding Pet.Born}"/>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate Remarks}"

VerticalOptions="Center"/>

<Editor

Grid.Column="1"

Grid.Row="4"

HeightRequest="80"

Text="{Binding Pet.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 **EditPetPageViewModel** class:

using System;

using MyVet.Common.Models;

using MyVet.Prism.Helpers;

using Prism.Navigation;

using Xamarin.Forms;

namespace MyVet.Prism.ViewModels

{

public class EditPetPageViewModel: ViewModelBase

{

private PetResponse \_pet;

private ImageSource \_imageSource;

private bool \_isRunning;

private bool \_isEnabled;

private bool \_isEdit;

public EditPetPageViewModel(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 PetResponse Pet

{

get => \_pet;

set => SetProperty(ref \_pet, value);

}

public ImageSource ImageSource

{

get => \_imageSource;

set => SetProperty(ref \_imageSource, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

Pet = parameters.GetValue<PetResponse>("pet");

ImageSource = Pet.ImageUrl;

IsEdit = true;

Title = Languages.EditPet;

}

else

{

Title = Languages.NewPet;

Pet = new PetResponse { Born = DateTime.Today };

ImageSource = "noimage";

IsEdit = false;

Title = Languages.NewPet;

}

}

}

}

1. Test it, that we do until this point.
2. Add the model **PetTypeResponse**:

namespace MyVet.Common.Models

{

public class PetTypeResponse

{

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 **EditPetViewModel**:

private readonly IApiService \_apiService;

…

private ObservableCollection<PetTypeResponse> \_petTypes;

private PetTypeResponse \_petType;

...

public EditPetViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

IsEnabled = true;

\_apiService = apiService;

}

…

public ObservableCollection<PetTypeResponse> PetTypes

{

get => \_petTypes;

set => SetProperty(ref \_petTypes, value);

}

public PetTypeResponse PetType

{

get => \_petType;

set => SetProperty(ref \_petType, value);

}

…

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

Pet = parameters.GetValue<PetResponse>("pet");

ImageSource = Pet.ImageUrl;

IsEdit = true;

Title = Languages.EditPet;

}

else

{

Title = Languages.NewPet;

Pet = new PetResponse { Born = DateTime.Today };

ImageSource = "noimage";

IsEdit = false;

Title = Languages.NewPet;

}

LoadPetTypes();

}

private async void LoadPetTypes()

{

IsRunning = true;

IsEnabled = false;

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.GetListAsync<PetTypeResponse>(url, "/api", "/PetTypes", "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var petTypes = (List<PetTypeResponse>)response.Result;

PetTypes = new ObservableCollection<PetTypeResponse>(petTypes);

if (!string.IsNullOrEmpty(Pet.PetType))

{

PetType = PetTypes.FirstOrDefault(pt => pt.Name == Pet.PetType);

}

}

1. Test it, that we do until this point.
2. Modify the **Pet** 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:MyVet.Prism.Helpers"

xmlns:prism="clr-namespace:Prism.Mvvm;assembly=Prism.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.Pet"

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Frame

CornerRadius="20"

HasShadow="True">

<StackLayout>

<StackLayout

Orientation="Horizontal">

<Image

WidthRequest="180"

Source="{Binding Pet.ImageUrl}"/>

<StackLayout>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="{Binding Pet.Name}"/>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

FontAttributes="Bold"

Text="{i18n:Translate Race}"

VerticalOptions="Center"/>

<Label

Grid.Row="0"

Grid.Column="1"

Text="{Binding Pet.Race}"

VerticalOptions="Center"/>

<Label

Grid.Row="1"

Grid.Column="0"

FontAttributes="Bold"

Text="{i18n:Translate Born}"

VerticalOptions="Center"/>

<Label

Grid.Row="1"

Grid.Column="1"

Text="{Binding Pet.Born, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Row="2"

Grid.Column="0"

FontAttributes="Bold"

Text="{i18n:Translate PetType}"

VerticalOptions="Center"/>

<Label

Grid.Row="2"

Grid.Column="1"

Text="{Binding Pet.PetType}"

VerticalOptions="Center"/>

</Grid>

</StackLayout>

</StackLayout>

<Label

BackgroundColor="White"

Text="{Binding Pet.Remarks}"/>

</StackLayout>

</Frame>

<Button

Command="{Binding EditPetCommand}"

Text="{i18n:Translate EditPet}"/>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="History"

TextColor="Black"/>

<ListView

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Histories}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Text="{Binding Date, StringFormat='{0:yyyy/MM/dd}'}"

VerticalOptions="Center"/>

<Label

Grid.Column="1"

Text="{Binding ServiceType}"

VerticalOptions="Center"/>

<Label

Grid.Column="2"

Text="{Binding Description}"

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 **PetViewModel** class:

private DelegateCommand \_editPetCommand;

…

public DelegateCommand EditPetCommand => \_editPetCommand ?? (\_editPetCommand = new DelegateCommand(EditPet));

…

private async void EditPet()

{

var parameters = new NavigationParameters

{

{ "pet", \_pet }

};

await \_navigationService.NavigateAsync("EditPet", parameters);

}

1. Test it, that we do until this point.
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 MyVet.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.myvet" 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 **EditPetViewModel**:

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.

## GROUP2

1. Fix the **Owners** API controller to send the owner Id:

var response = new OwnerResponse

{

Id = owner.Id,

FirstName = owner.User.FirstName,

1. Re-publish on Azure.
2. Add literals for **CreateEditPetConfirm**, **Created** and **Edited**:

public static string CreateEditPetConfirm => Resource.CreateEditPetConfirm;

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);

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,

};

}

}

1. Change the **EditPetPageViewModel**:

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);

byte[] imageArray = null;

if (\_file != null)

{

imageArray = FilesHelper.ReadFully(\_file.GetStream());

}

var petRequest = new PetRequest

{

Born = Pet.Born,

Id = Pet.Id,

ImageArray = imageArray,

Name = Pet.Name,

OwnerId = owner.Id,

PetTypeId = PetType.Id,

Race = Pet.Race,

Remarks = Pet.Remarks

};

Response response;

if (IsEdit)

{

response = await \_apiService.PutAsync(url, "/api", "/Pets", petRequest.Id, petRequest, "bearer", token.Token);

}

else

{

response = await \_apiService.PostAsync(url, "/api", "/Pets", petRequest, "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.Error,

string.Format(Languages.CreateEditPetConfirm, IsEdit ? Languages.Edited : Languages.Created),

Languages.Accept);

await \_navigationService.GoBackToRootAsync();

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Pet.Name))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.NameError, Languages.Accept);

return false;

}

if (string.IsNullOrEmpty(Pet.Race))

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.RaceError, Languages.Accept);

return false;

}

if (PetType == null)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.PetTypeError, Languages.Accept);

return false;

}

return true;

}

1. To fix the update, modify the **PetsViewModel**:

private static PetsViewModel \_instance;

....

public PetsViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_instance = this;

\_navigationService = navigationService;

\_apiService = apiService;

Title = "Pets";

LoadPets();

}

…

public static PetsViewModel 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;

LoadPets();

}

}

1. Modify the **EditPetViewModel**:

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);

byte[] imageArray = null;

if (\_file != null)

{

imageArray = FilesHelper.ReadFully(\_file.GetStream());

}

var petRequest = new PetRequest

{

Born = Pet.Born,

Id = Pet.Id,

ImageArray = imageArray,

Name = Pet.Name,

OwnerId = owner.Id,

PetTypeId = PetType.Id,

Race = Pet.Race,

Remarks = Pet.Remarks

};

Response response;

if (IsEdit)

{

response = await \_apiService.PutAsync(url, "/api", "/Pets", petRequest.Id, petRequest, "bearer", token.Token);

}

else

{

response = await \_apiService.PostAsync(url, "/api", "/Pets", petRequest, "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 App.Current.MainPage.DisplayAlert(

Languages.Ok,

string.Format(Languages.CreateEditPetConfirm, IsEdit ? Languages.Edited : Languages.Created),

Languages.Accept);

await \_navigationService.GoBackToRootAsync();

}

1. Test it, that we do until this point.
2. Add the method **DeletePet** to **Pets** API controller:

[HttpDelete("{id}")]

public async Task<IActionResult> DeletePet([FromRoute] int id)

{

if (!ModelState.IsValid)

{

return this.BadRequest(ModelState);

}

var pet = await \_dataContext.Pets

.Include(p => p.Histories)

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

if (pet == null)

{

return this.NotFound();

}

if (pet.Histories.Count > 0)

{

BadRequest("The pet can't be deleted because it has history.");

}

\_dataContext.Pets.Remove(pet);

await \_dataContext.SaveChangesAsync();

return Ok("Pet deleted");

}

1. Re-publish on Azure.
2. Add listerals for: **Confirm**, **QuestionToDeletePet**, **Yes** and **No**:

public static string Confirm => Resource.Confirm;

public static string QuestionToDeletePet => Resource.QuestionToDeletePet;

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 **EditPetViewModel**:

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. Finally, modify the **Pets** page:

<ListView

HasUnevenRows="True"

SeparatorVisibility="None"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshPetsCommand}"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Pets}">

1. And modify the **PetsViewModel**:

private DelegateCommand \_refreshPetsCommand;

....

public DelegateCommand RefreshPetsCommand => \_refreshPetsCommand ?? (\_refreshPetsCommand = new DelegateCommand(RefreshPets));

…

private async void RefreshPets()

{

IsRefreshing = true;

await UpdateOwner();

IsRefreshing = false;

}

1. Test it.

## Show My Agenda

1. Add the property **DateLocal** to **AgendaResponse**:

public DateTime DateLocal => Date.ToLocalTime();

1. Add the method **GetAgendaForOwner** to **IApiService**:

Task<Response> GetAgendaForOwner(

string urlBase,

string servicePrefix,

string controller,

string email,

string tokenType,

string accessToken);

1. Add the method **GetAgendaForOwner** to **ApiService**:

public async Task<Response> GetAgendaForOwner(

string urlBase,

string servicePrefix,

string controller,

string email,

string tokenType,

string accessToken)

{

try

{

var model = new EmailRequest { Email = email };

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,

};

}

var agenda = JsonConvert.DeserializeObject<List<AgendaResponse>>(answer);

return new Response

{

IsSuccess = true,

Result = agenda

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Add the class **AgendaItemViewModel**:

using MyVet.Common.Models;

namespace MyVet.Prism.ViewModels

{

public class AgendaItemViewModel : AgendaResponse

{

}

}

1. Add the **Agenda** 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"

BackgroundColor="Silver"

x:Class="MyVet.Prism.Views.Agenda"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ListView

HasUnevenRows="True"

SeparatorVisibility="None"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshPetsCommand}"

IsRefreshing="{Binding IsRefreshing}"

ItemsSource="{Binding Agenda}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Frame

CornerRadius="20"

HasShadow="True"

Margin="0,0,0,5">

<Frame.GestureRecognizers>

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

</Frame.GestureRecognizers>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

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

TextColor="Black">

</Label>

<Label

Grid.Column="1"

FontAttributes="Bold"

Text="{Binding Owner.FullName}"

TextColor="Black">

</Label>

<Label

Grid.Column="2"

Text="{Binding Pet.Name}"

TextColor="Black">

</Label>

<Image

Grid.Column="3"

Source="ic\_chevron\_right">

</Image>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Modify the **AgendaViewModel**:

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using MyVet.Common.Services;

using MyVet.Prism.Helpers;

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class AgendaViewModel : ViewModelBase

{

private readonly IApiService \_apiService;

private ObservableCollection<AgendaItemViewModel> \_agenda;

private bool \_isRefreshing;

private DelegateCommand \_refreshPetsCommand;

public AgendaViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

Title = "Agenda";

\_apiService = apiService;

LoadAgenda();

}

public DelegateCommand RefreshPetsCommand => \_refreshPetsCommand ?? (\_refreshPetsCommand = new DelegateCommand(LoadAgenda));

public ObservableCollection<AgendaItemViewModel> Agenda

{

get => \_agenda;

set => SetProperty(ref \_agenda, value);

}

public bool IsRefreshing

{

get => \_isRefreshing;

set => SetProperty(ref \_isRefreshing, value);

}

private async void LoadAgenda()

{

IsRefreshing = true;

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

var response = await \_apiService.GetAgendaForOwner(url, "/api", "/Agenda/GetAgendaForOwner", owner.Email, "bearer", token.Token);

if (!response.IsSuccess)

{

IsRefreshing = false;

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var myAgenda = (List<AgendaResponse>)response.Result;

Agenda = new ObservableCollection<AgendaItemViewModel>(myAgenda.Select(a => new AgendaItemViewModel

{

Date = a.Date,

Id = a.Id,

IsAvailable = a.IsAvailable,

Owner = a.Owner,

Pet = a.Pet,

Remarks = a.Remarks

}).ToList());

IsRefreshing = false;

}

}

}

1. Add literal for **AssignModifyAgenda**:

public static string AssignModifyAgenda => Resource.AssignModifyAgenda;

1. Add the **AssignModifyAgenda** 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="MyVet.Prism.Views.AssignModifyAgenda"

Title="{Binding Title}">

</ContentPage>

1. Modify the **AssignModifyAgendaViewModel**:

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class AssignModifyAgendaViewModel : ViewModelBase

{

public AssignModifyAgendaViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = Languages.AssignModifyAgenda;

}

}

}

1. Modify the **AgendaItemViewModel**:

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using Newtonsoft.Json;

using Prism.Commands;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class AgendaItemViewModel : AgendaResponse

{

private readonly INavigationService \_navigationService;

private DelegateCommand \_selectAgendaCommand;

public AgendaItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

public DelegateCommand SelectAgendaCommand => \_selectAgendaCommand ?? (\_selectAgendaCommand = new DelegateCommand(SelectAgenda));

private async void SelectAgenda()

{

var owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

if (!IsAvailable && Owner.Id != owner.Id)

{

return;

}

var parameters = new NavigationParameters

{

{ "Agenda", this }

};

await \_navigationService.NavigateAsync("AssignModifyAgenda", parameters);

}

}

}

1. Test it that we do until this moment.
2. Add literal for **Assign**:

public static string Assign => Resource.Assign;

1. Modify the **AssignModifyAgenda**:

<?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.AssignModifyAgenda"

Title="{Binding Title}">

<StackLayout

Padding="10">

<Label

Text="{i18n:Translate AgendaFor}"/>

<Label

FontAttributes="Bold"

FontSize="Medium"

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

<Label

Text="{i18n:Translate Pet}"/>

<Picker

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding Pets}"

SelectedItem="{Binding Pet}"

Title="{i18n:Translate PetPlaceHolder}"/>

<Label

Text="{i18n:Translate Remarks}"/>

<Editor

HeightRequest="70"

Text="{Binding Agenda.Remarks}"/>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<StackLayout

Orientation="Horizontal">

<Button

Command="{Binding AssignCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Assign}"/>

<Button

Command="{Binding CancelCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="{i18n:Translate Cancel}"/>

</StackLayout>

</StackLayout>

</ContentPage>

1. Modify the **AssignModifyAgendaViewModel**:

using System.Collections.ObjectModel;

using System.Linq;

using MyVet.Common.Models;

using MyVet.Prism.Helpers;

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class AssignModifyAgendaViewModel : ViewModelBase

{

private AgendaResponse \_agenda;

private PetResponse \_pet;

private ObservableCollection<PetResponse> \_pets;

private bool \_isRunning;

private bool \_isEnabled;

public AssignModifyAgendaViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = Languages.AssignModifyAgenda;

IsEnabled = true;

}

public AgendaResponse Agenda

{

get => \_agenda;

set => SetProperty(ref \_agenda, value);

}

public PetResponse Pet

{

get => \_pet;

set => SetProperty(ref \_pet, value);

}

public ObservableCollection<PetResponse> Pets

{

get => \_pets;

set => SetProperty(ref \_pets, value);

}

public bool IsRunning

{

get => \_isRunning;

set => SetProperty(ref \_isRunning, value);

}

public bool IsEnabled

{

get => \_isEnabled;

set => SetProperty(ref \_isEnabled, value);

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("Agenda"))

{

Agenda = parameters.GetValue<AgendaResponse>("Agenda");

LoadPets();

}

}

private void LoadPets()

{

var owner = JsonConvert.DeserializeObject<OwnerResponse>(Settings.Owner);

Pets = new ObservableCollection<PetResponse>(owner.Pets);

Pet = Pets.FirstOrDefault(p => p.Id == \_agenda.Pet.Id);

}

}

}

1. Test it that we do until this moment.
2. Add a literal for **CancelAgendaMessage**:

public static string CancelAgendaMessage => Resource.CancelAgendaMessage;

1. Modify the **AssignModifyAgendaViewModel**:

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private DelegateCommand \_assignCommand;

private DelegateCommand \_cancelCommand;

…

public AssignModifyAgendaViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

\_navigationService = navigationService;

\_apiService = apiService;

Title = Languages.AssignModifyAgenda;

IsEnabled = true;

}

…

public DelegateCommand AssignCommand => \_assignCommand ?? (\_assignCommand = new DelegateCommand(Assign));

public DelegateCommand CancelCommand => \_cancelCommand ?? (\_cancelCommand = new DelegateCommand(Cancel));

...

private async void Assign()

{

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 request = new AssignRequest

{

AgendaId = Agenda.Id,

OwnerId = owner.Id,

PetId = Pet.Id,

Remarks = Agenda.Remarks

};

var response = await \_apiService.PostAsync(url, "/api", "/Agenda/AssignAgenda", request, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var parameters = new NavigationParameters

{

{ "Refresh", true }

};

await \_navigationService.GoBackAsync(parameters);

}

private async Task<bool> ValidateData()

{

if (Pet == null)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, Languages.PetError, Languages.Accept);

return false;

}

return true;

}

private async void Cancel()

{

var answer = await App.Current.MainPage.DisplayAlert(

Languages.Confirm,

Languages.CancelAgendaMessage,

Languages.Yes,

Languages.No);

if(!answer)

{

return;

}

IsRunning = true;

IsEnabled = false;

var request = new UnAssignRequest { AgendaId = Agenda.Id };

var url = App.Current.Resources["UrlAPI"].ToString();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.PostAsync(url, "/api", "/Agenda/UnAssignAgenda", request, "bearer", token.Token);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(Languages.Error, response.Message, Languages.Accept);

return;

}

var parameters = new NavigationParameters

{

{ "Refresh", true }

};

await \_navigationService.GoBackAsync(parameters);

}

1. Modify the **AgendaViewModel**:

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("Refresh"))

{

LoadAgenda();

}

}

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.myvet" 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="MyVet.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 MyVet.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 MyVet.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 MyVet.Common.Services;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace MyVet.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 **User** to storage latitude and longitude:

[DisplayFormat(DataFormatString = "{0:N6}")]

public double Latitude { get; set; }

[DisplayFormat(DataFormatString = "{0:N6}")]

public double Longitude { 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 Index Owners view to show the new fields:

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

Pets

</th>

…

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

1. Modify the **EditUserViewModel** to add the new properties:

public double Latitude { get; set; }

public double Longitude { get; set; }

1. Modify the partial shared view **\_User** to add the new properties:

<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="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 **AddUser**:

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,

Latitude = view.Latitude,

Longitude = view.Longitude

};

1. Modify the method get to Edit User:

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,

Latitude = owner.User.Latitude,

Longitude = owner.User.Longitude

};

1. Modify the method pots to Edit User:

owner.User.PhoneNumber = view.PhoneNumber;

owner.User.Latitude = view.Latitude;

owner.User.Longitude = view.Longitude;

1. Test it, and set positions near to latitude 6 and longitude -75.
2. Add the properties to **OwnerResponse**:

public string Email { get; set; }

public double Latitude { get; set; }

public double Longitude { get; set; }

public ICollection<PetResponse> Pets { get; set; }

1. Add this method to **OwnersController** API:

[HttpGet]

public async Task<IActionResult> GetOwners()

{

var owners = await \_dataContext.Owners

.Include(o => o.User)

.Include(o => o.Pets)

.ThenInclude(p => p.PetType)

.ToListAsync();

var response = new List<OwnerResponse>(owners.Select(o => new OwnerResponse

{

Id = o.Id,

Latitude = o.User.Latitude,

Longitude = o.User.Longitude,

FirstName = o.User.FirstName,

LastName = o.User.LastName,

Address = o.User.Address,

Document = o.User.Document,

Email = o.User.Email,

PhoneNumber = o.User.PhoneNumber,

Pets = o.Pets.Select(p => new PetResponse

{

Born = p.Born,

Id = p.Id,

ImageUrl = p.ImageFullPath,

Name = p.Name,

Race = p.Race,

Remarks = p.Remarks,

PetType = p.PetType.Name

}).ToList()

}).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.

# Others

## Generate PDF from View

1. Create the folder **Rotativa** inside **wwwroot**.
2. Put in this folder the files: **wkhtmltoimage.exe** and **wkhtmltopdf.exe**. You can get it from: <https://github.com/webgio/Rotativa/tree/master/Rotativa.Demo/Rotativa>.
3. Install the NuGet package: **Rotativa.AspNetCore**.
4. Modify the **Startup** and add this line:

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

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

});

RotativaConfiguration.Setup(env);

1. Modify the controller, for example:

public IActionResult Index()

{

return new ViewAsPdf(\_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.PetType)

.Include(p => p.Histories));

}

1. Test it.
2. Add those lines to Index view:

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

<br />

<p>

<a asp-action="Print" class="btn btn-primary"><i class="glyphicon glyphicon-print"></i> Print</a>

</p>

<div class="row">

1. Add the method **Print**.

public IActionResult Print()

{

return new ViewAsPdf(\_dataContext.Pets

.Include(p => p.Owner)

.ThenInclude(o => o.User)

.Include(p => p.PetType)

.Include(p => p.Histories))

{ FileName = "Pets.pdf" };

}

1. Add the View **Print**.

@model IEnumerable<MyVet.Web.Data.Entities.Pet>

@{

Layout = null;

}

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css"

asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"

asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />

<h2>Pets</h2>

<div class="row" style="border: 1px solid #000;">

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

Name

</div>

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

Race

</div>

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

Born

</div>

</div>

<div>

@foreach (var item in Model)

{

<div class="row">

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

@item.Name

</div>

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

@item.Race

</div>

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

@item.Born

</div>

</div>

}

</div>

1. Test it.

## To Identify Errors In ViewModel

else

{

foreach (var item in ModelState.Values)

{

if (item.Errors.Count > 0)

{

var x = item.ToString();

}

}

}