**APLICACIÓN VETERINARIA**

* Servicios y el modo administrativo en .**NET Core**
* Front end de la aplicación en **Xamarin Form** con ayuda de **Prism** es un framework que facilita el desarrollo permite ligar view con la viewModel
* Rescribir la aplicación en **Xamarin Classic**

|  |  |  |  |
| --- | --- | --- | --- |
| **Funcionality** | **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 appoinments | X | X | X |
| Admin clinic history for Pets | X |  |  |
| See clinic history | X | X | X |
| See veterinary on map |  |  | X |
| Service “Pet love” |  |  | X |

**Proyectos**

1. **MyVet.Common(librería .NET Estándar)\*:** Transversal a toda la solución es decir todo código reutilizable tanto en web como en app se coloca acá

**Backend (web)**

1. **MyVet.Web(librería .NET Core)\*:** Tiene dos funcionalidades implícitas
2. **Front End Web(Admin):**
3. **API:** Servicios de comunicaciones porque se quiere acceder además de web vía app.

**Frontend (Mobile)**

1. **MyVet.Prism\*:** Se tiene todo el código compartido con limitaciones en interface de usuario
2. **MiVet.Prism.Android\*:** Proyecto en android de xamarin form
3. **MiVet.Prism.Ios\*:** Proyecto en ios de xamarin form
4. **MyVet.Cross.Android\***
5. **MyVet.Cross.iOS\***

El 4 y 5 Comparten código de la librería .NET Estándar no se tiene limitante en la interface de usuario

**GITHUB**

1. Crear repositorio

**Visual Studio**

1. Clonar repositorio en una ruta corta
2. Crear una solución en blanco llamada **MyVet**
3. Crear el primer proyecto es .NET Estándar una class library llamado **MyVet.Common** tendrá todo el código que se comparte
4. Crear el 2do proyecto el cual es será de tipo ASP.NET Core Web Application y se llamará **MyVet.Web** en **Core 2.1** y se escoge la opción **web Application(Model-View-Controller)** y se le habilita HTTPS para que los servicios se puedan configurar de forma segura.
5. Creación del modelo Owner

public class Owner

{

public int Id { get; set; }

[Required]

[MaxLength(30)]

[Display(Name ="Documento")]

public string Document { get; set; }

[Required]

[MaxLength(50)]

[Display(Name = "Nombres")]

public string FirstName { get; set; }

[Required]

[MaxLength(50)]

[Display(Name = "Apellidos")]

public string LastName { get; set; }

[MaxLength(20)]

[Display(Name = "Teléfono fijo")]

public string FixedPhone { get; set; }

[Required]

[MaxLength(20)]

[Display(Name = "Teléfono celular")]

public string CellPhone { get; set; }

[MaxLength(100)]

[Display(Name = "Dirección")]

public string Address { get; set; }

[Display(Name = "Propietario")

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

[Display(Name = "Propietario")]

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

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

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

}

1. Creación de la clase DataContext

public class DataContext : DbContext

{

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

{

}

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

}

1. En .NET Core hay 3 archivos importantes en la configuración del proyecto

-**appsettings.json:** Se define el string de conexión, pero 1ro se mira las instancias que se tienen instaladas o si la conexión se va hacer a una maquina remota. Para el caso será local en este archivo se adiciona el string

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

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

}

}

- **Program.cs**

- **Starup.cs:** Se le dice al proyecto que use la bd.

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. Se corren comandos en la consola nuget package manage asegurarse que este seleccionado el proyecto web y correr los siguientes comandos

Update-database

add-migration nombreMigracion

Update-database

1. Se crea controlador para owner
2. Base de datos inicial



1. Los otros 5 modelos
2. Agenda

public class Agenda

{

public int Id { get; set; }

[Display(Name = "Fecha")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.DateTime)]

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

public DateTime Date { get; set; }

[Display(Name = "Comentarios")]

public string Remarks { get; set; }

[Display(Name = "Disponible?")]

public bool IsAvailable { get; set; }

[Display(Name = "Fecha:")]

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

public DateTime DateLocal => Date.ToLocalTime();

public Pet Pet { get; set; }

public Owner Owner { get; set; }

}

1. Pet

public class Pet

{

public int Id { get; set; }

[Display(Name = "Nombre")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Name { get; set; }

[Display(Name = "Imagen")]

public string ImageUrl { get; set; }

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

[Display(Name = "Raza")]

public string Race { get; set; }

[Display(Name = "Fecha de nacimiento")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.DateTime)]

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

public DateTime Born { get; set; }

[Display(Name = "Comentarios")]

public string Remarks { get; set; }

//TODO: replace the correct URL for the image

public string ImageFullPath => string.IsNullOrEmpty(ImageUrl)

? null

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

[Display(Name = "Fecha de nacimiento")]

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

public DateTime BornLocal => Born.ToLocalTime();

public PetType PetType { get; set; }

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

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

}

1. History

public class History

{

public int Id { get; set; }

[Display(Name = "Descripción")]

[MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Description { get; set; }

[Display(Name = "Fecha")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

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

public DateTime Date { get; set; }

[Display(Name = "Comentarios")]

public string Remarks { get; set; }

[Display(Name = "Fecha:")]

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

public DateTime DateLocal => Date.ToLocalTime();

public ServiceType ServiceType { get; set; }

}

1. Pettype

public class PetType

{

public int Id { get; set; }

[Display(Name = "Tipo de mascota")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Name { get; set; }

public Owner Owner { get; set; }

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

}

1. ServiceType

public class ServiceType

{

public int Id { get; set; }

[Display(Name = "Tipo de servicio")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Name { get; set; }

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

}

1. Añadir las otras tablas al datacontext

public class DataContext : DbContext

{

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

{

}

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

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

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

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

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

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

}

1. Agregar otra migración

Add-migration CompleteDB

Update-database

1. Alimentador de base de datos registros de prueba

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 = "Gato" });

await \_context.SaveChangesAsync();

}

}

private async Task CheckOwnersAsync()

{

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

{

AddOwner("8989898", "Carolina", "Russi", "234 3232", "310 322 3221", "Calle Luna Calle Sol");

AddOwner("7655544", "Alfonso", "Mercado", "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. Se hace un cambio en la clase program inicialmente esta de esta forma

public class Program

{

public static void Main(string[] args)

{

CreateWebHostBuilder(args).Build().Run();

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>();

}

1. Queda de esta forma

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. Inyectar el seeder en el archivo de configuración en el starup

services.AddDbContext<DataContext>(cfg =>

{

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

});

services.AddTransient<SeedDb>();

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

1. Crear controlador para agenda con respecto a la hora en la vista index en vez de mostrar el date se muestra el dateLocal
2. Adicionar al MER las tablas de usuario son del userEntity framework las 3 ms importantes son las mostradas y uma tabla manager que sirve para guardar los datos básicos de los administradores



1. Crear el modelo user el cual hereda del identityUser y se le adiciona los campos que se necesita que esta tabla no tiene para el caso de owner
2. En owner se reemplaza todo excepto la clave primaria y las relaciones por una relación con usuario

public class Owner

{

public int Id { get; set; }

public User User { get; set; }

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

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

}

1. Crea el modelo para manager que para integrarlo con el modelo de usuarios queda así

public class Manager

{

public int Id { get; set; }

public User User { get; set; }

}

1. En el datacontext ya no hereda del dbcontext así se le dice cual es la clase o modelo de usuario personalizado y tb se agrega la tabla manager queda así

public class DataContext : IdentityDbContext<User>

{

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

{

}

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

1. Las tablas adicionales que están en otro color no se accede a ellas directamente, si no que se acceden por medio del userManager y roleManager
2. Crear carpeta llamada helper las cuales tienen interfaces y clases con utilidades
3. Se crea dentro de esta carpeta una interface llamada UserHelper que inicialmente tendrá 4 métodos

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. La clase userHelper

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. Configurar la nueva inyección en el Starup

services.AddDbContext<DataContext>(cfg =>

{

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

})

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>(); services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

1. Modifica el seeder

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", "Diana", "Russi", "diana.russiposada@gmail.com", "350 634 2747", "Calle Luna Calle Sol", "Admin");

var customer = await CheckUserAsync("2020", "Diana", "Russi", "dianarussi@yahoo.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. Borrar el controlador de owner y sus vistas y volverlo a crear y crear controlador para lo manger
2. Crear la opciones en el menú
3. Se ejecutan los siguientes comandos

drop-database

add-migration Users

update-database

1. Configurar las condiciones del password en el Starup

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.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>();

1. También se le dice que la aplicación va a utiliza autentificación por eso se agrega tb en ese mismo archivo esta liea de código

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?}");

});

}

1. Ya se prueba ya maneja usuarios y la se puede hacer ogin y logout
2. Crear controlador MVC muchas veces el entity sirve de modelo en otras toca crearlo como en el login
3. Vamos al menú para hacer la parte de como se mostraría el login y logout

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

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

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

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

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

{

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

<li><a **asp-area**="" **asp-controller**="Managers" **asp-action**="Index">Administrador</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. Crear controlador accountController dentro del cual crear la acción login el get y la propiedad privada de solo lectura

public class AccountController : Controller

private readonly IUserHelper \_userHelper;

public AccountController(IUserHelper userHelper)

{

\_userHelper = userHelper;

}

public IActionResult Login()

{

return View();

}

{

public IActionResult Login()

{

return View();

}

}

1. Crear el modelo loginviewmodel

public class LoginViewModel

{

[Required]

[EmailAddress]

public string Username { get; set; }

[Required]

[MinLength(6)]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

1. la vista login

@model MyVet.Web.Models.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<h2>Inicio de sesión</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">Usuario</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">Contraseña</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">Recordarme?</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">Registrar nuevo usuario</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. En la interface de userHelper adicionar metodos para loguearse y desloguearse

Task<SignInResult> LoginAsync(LoginViewModel model);

Task LogoutAsync();

1. Implementar esos dos metodos en la clase de userHelper

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. Crear el post de la acción login

[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);

}

1. Acción logout en el controlador account

public async Task <IActionResult> Logout()

{

await \_userHelper.LogoutAsync();

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

}

1. Colocar en los controladores la anotación de authorice para que solo deje ingresar al usuario admin para el caso de owner, manager y agenda

[Authorize(Roles ="Admin")]

1. Controlador con izard para tipos de mascotas y de servicio
2. Modificaciones en la acción index de owner para que incluya usuarios y mascotas

public IActionResult Index()

{

return View( \_context.Owners

.Include(o=> o.User)

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

}

1. Hacer cambios en la vista referentes a ropa de botones y mostrar la tabla con los campos mas importantes de owner

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

@{

ViewData["Title"] = "Index";

}

<h2>Propietario</h2>

<p>

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

</p>

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

# de mascotas

</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.FullName)

</td>

<td>

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

</td>

<td>

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

</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">Editar</a>

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

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

</td>

</tr>

}

</tbody>

</table>

1. La acción details en el onercontroller

// GET: Owners/Details/5

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

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners

.Include(o => o.User)

.Include(o => o.Pets)

.ThenInclude(p=>p.PetType)

.Include(o => o.Pets)

.ThenInclude(p=>p.Histories)

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

if (owner == null)

{

return NotFound();

}

return View(owner);

}

1. La vista details

@model MyVet.Web.Data.Entities.Owner

@{

ViewData["Title"] = "Details";

}

<h2>Propietario</h2>

<div>

<h4>Detalles</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.Address)

</dt>

<dd>

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

</dd>

<dt>

Teléfono

</dt>

<dd>

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

</dd>

<dt>

# de mascotas

</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">Editar</a>

<a **asp-action**="AddPet" **asp-route-id**="@Model.Id" class="btn btn-primary">Agregar mascotas</a>

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</a>

</div>

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

{

<h4>No hay mascotas adicionadas todavia</h4>

}

else

{

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

# Historias clinica

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Pets)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

}

1. Crear el modelo para agregar usuarios

public class AddUserViewModel

{

[Display(Name = "Email")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres.")]

[EmailAddress]

public string Username { get; set; }

[Display(Name = "Documento")]

[MaxLength(20, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Document { get; set; }

[Display(Name = "Nombres")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string FirstName { get; set; }

[Display(Name = "Apellidos")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string LastName { get; set; }

[MaxLength(100, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres.")]

public string Address { get; set; }

[Display(Name = "Teléfono")]

[MaxLength(50, ErrorMessage = "El campo {0} debe tener máximo {1} caracteres")]

public string PhoneNumber { get; set; }

[Display(Name = "Contraseña")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "La {0} debe contener entre {2} y {1} caracteres.")]

public string Password { get; set; }

[Display(Name = "Confirmar contraseña")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "La {0} debe contener entre {2} y {1} caracteres.")]

[Compare("Password")]

public string PasswordConfirm { get; set; }

}

}

1. La acción créate en el controlador de owner la parte del get

// GET: Owners/Create

public IActionResult Create()

{

return View();

}

1. La vista créate

@model MyVet.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Nuevo</h2>

<h4>Propietario</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">Regresar a la lista</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Se debe de inyectar el userhelper en el ownerController para cuando se cree el propietario también se cree el usuario y se le aigne su respectivo tol habilitar el owner controler para que pueda administrar usuarios

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

public OwnersController(DataContext context, IUserHelper userHelper)

{

\_context = context;

\_userHelper = userHelper;

}

1. El modelo créate de owner la parte post

// POST: Owners/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel model)

{

if (ModelState.IsValid)

{

//crear usuario

var user = new User

{

Address = model.Address,

Document = model.Document,

Email = model.Username,

FirstName = model.FirstName,

LastName = model.LastName,

PhoneNumber = model.PhoneNumber,

UserName = model.Username

};

var response = await \_userHelper.AddUserAsync(user, model.Password);

if(response.Succeeded)

{

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

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

//crear owner

var owner = new Owner

{

Agendas = new List<Agenda>(),

Pets = new List<Pet>(),

User = userInDB

};

//mandar bd

\_context.Owners.Add(owner);

try

{

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

catch (Exception ex)

{

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

return View(model);

}

}

ModelState.AddModelError(string.Empty, response.Errors.FirstOrDefault().Description);

}

return View(model);

}

1. Para poder agregar mascotas a los propietarios es necesario crear un nuevo modelo

public class PetViewModel : Pet

{

public int OwnerId { get; set; }

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[Display(Name = "Tipo de mascota")]

[Range(1, int.MaxValue, ErrorMessage = "Debe selleccionar un tipo de mascota.")]

public int PetTypeId { get; set; }

[Display(Name = "Imagen")]

public IFormFile ImageFile { get; set; }

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

}

1. Crear interface IComboHelper y clase combosHelper para meter todo el código relacionado con los combos

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboPetTypes();

}

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. Configurar esta nueva inyccion en el Starup

services.AddTransient<SeedDb>();

services.AddScoped<IUserHelper, UserHelper>();

services.AddScoped<ICombosHelper, CombosHelper>(); services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

1. Agregar la inyección en el ownerController

public class OwnersController : Controller

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

public OwnersController(

DataContext context,

IUserHelper userHelper,

ICombosHelper combosHelper)

{

\_context = context;

\_userHelper = userHelper;

\_combosHelper = combosHelper;

}

1. En el controlador de owner adicionar la acción de addPet el get

// GET: Owners/CreatePet

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);

}

1. Crear la vista para addPet

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Agregar</h2>

<h4>Mascota</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="Crear" class="btn btn-primary" />

<a **asp-action**="Details" **asp-route-id**="@Model.OwnerId" class="btn btn-success">Regresar a propietario</a>

</div>

</form>

</div>

</div>

1. En el modelo de pet la fecha de nacimiento en vez de pedirla como datetime pedirla como string quitarle solo la anotación de datatype
2. Crear en la carpeta imágenes un nuevo folder llamado Pets
3. Crear interface IImagenHelper con su respectiva clase donde se colocara un método que verifique que el nombre de la imagen no se va a repetir

public interface IImageHelper

{

Task<string> UploadImageAsync(IFormFile imageFile);

}

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. Crear interface IConverterHelper para la conversión de objetos con su respetiva clase que inicialmente tendrá un método que convierta el objeto petVieModel en pet

public interface IConverterHelper

{

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

PetViewModel ToPetViewModel(Pet pet);

}

public class ConverterHelper : IConverterHelper

{

private readonly DataContext \_dataContext;

private readonly ICombosHelper \_combosHelper;

public ConverterHelper(

DataContext dataContext)

{

\_dataContext = dataContext;

}

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;

}

1. Hacer las inyecciones de los helper en el contreolador de owner

public class OwnersController : Controller

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

private readonly IImageHelper \_imageHelper;

private readonly IConverterHelper \_converterHelper;

public OwnersController(

DataContext context,

IUserHelper userHelper,

ICombosHelper combosHelper,

IImageHelper imageHelper,

IConverterHelper converterHelper)

{

\_context = context;

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_imageHelper = imageHelper;

\_converterHelper = converterHelper;

}

1. Para usarlo se debe configurar en la clase Starup

services.AddScoped<IUserHelper, UserHelper>();

services.AddScoped<ICombosHelper, CombosHelper>();

services.AddScoped<IImageHelper, ImageHelper>();

services.AddScoped<IConverterHelper, ConverterHelper>(); services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

1. El post en el controlador de owner de la acción addPet

// POST: Owners/CreatePet

[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}");

}

model.PetTypes = \_combosHelper.GetComboPetTypes();

return View(model);

}

1. En la vista details de owner cambiar para que muestre la foto de la mascota en la tabla agregar la nueva columna tanto titulo como lo que va a mostrar

<th>

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

</th>

<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>

1. En los converter hacer método que reciba un petl y devuelva un petviewmodel se debe inyectar el combohelper para que funcione

public ConverterHelper(

DataContext dataContext,

ICombosHelper combosHelper )

{

\_dataContext = dataContext;

\_combosHelper = combosHelper;

}

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. Colocar este método en la interface de converter helper para que quede disponible

PetViewModel ToPetViewModel(Pet pet)

1. Hacer la acción editpet en el controlador de owner la parte get

// GET: Owners/EditPet

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));

}

1. Vista editpet

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Mascota</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="Guardar" class="btn btn-primary" />

<a **asp-action**="Details" **asp-route-id**="@Model.OwnerId" class="btn btn-success">Regresar a propietario</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. La acción post para editpet

// POST: Owners/EditPet

[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}");

}

model.PetTypes = \_combosHelper.GetComboPetTypes();

return View(model);

}

1. Para no tener código duplicado en las vistas para el caso addpet y editpet se introduce concepto de vistas parciales
2. Crear vista parcial que se llama \_pet

* Se add y edit reciben el mismo modelo este se copia y se copia código que se repite en ambas vistas para el caso todo el formulario solamente
* En addpet y editpet se reemplaza todo ese código por la ste línea de código

<**partial** **name**="\_Pet" />

1. PENDIENTE UNIFICAR CÓDIGO DE LAS VISTAS CON UNA VISTA PARCIAL
2. En detalles de la mascota ver los detalles de esta y la historia clínica
3. Crear la acción en el controlador para detailsPet

// GET: Owners/DetailsPet

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.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. Crear la vista para detalles de mascota

@model MyVet.Web.Data.Entities.Pet

@{

ViewData["Title"] = "Details";

}

<h2>Mascota</h2>

<div>

<h4>Detalles</h4>

<hr />

<div class="row">

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

<dl class="dl-horizontal">

<dt>

Propietario

</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">Editar</a>

<a **asp-action**="AddHistory" **asp-route-id**="@Model.Id" class="btn btn-primary">Nueva historia</a>

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

</div>

<h4>Historia clinica</h4>

<hr />

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

{

<h5>No hay historias clinicas adicionadas aun.</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.DateLocal)

</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">Editar</a>

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

</td>

</tr>

}

</tbody>

</table>

}

1. Crear el historyViewModel

public class HistoryViewModel : History

{

public int PetId { get; set; }

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[Display(Name = "Tipo servicio")]

[Range(1, int.MaxValue, ErrorMessage = "Debe seleccionar un tipo de servicio.")]

public int ServiceTypeId { get; set; }

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

}

1. Adicionar el combo de servicetype en la interface icombohelper e implementarlo en la clase comboshelper

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboPetTypes();

IEnumerable<SelectListItem> GetComboServiceTypes();

}

public IEnumerable<SelectListItem> GetComboServiceTypes()

{

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

{

Text = st.Name,

Value = $"{st.Id}"

})

.OrderBy(pt => st.Text)

.ToList();

list.Insert(0, new SelectListItem

{

Text = "[Seleccione un tipo de servicio...]",

Value = "0"

});

return list;

}

1. En la interface IConverter helper se deben adicionar dos métodos mas la historyViewModel que se necesita convertir al objeto history y el objeto history convertirlo a historyViewModel y estos métodos se implementan en la clase converterHelper

public interface IConverterHelper

{

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

PetViewModel ToPetViewModel(Pet pet);

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

HistoryViewModel ToHistoryViewModel(History history);

}

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.ToUniversalTime(),

Description = history.Description,

Id = history.Id,

PetId = history.Pet.Id,

Remarks = history.Remarks,

ServiceTypeId = history.ServiceType.Id,

ServiceTypes = \_combosHelper.GetComboServiceTypes()

};

}

1. En el controlador de owner se debe adicionar la acción addHistory la parte get

// GET: Owners/AddHistory

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.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);

}

1. La vista de addHistory

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Agregar</h2>

<h4>Historia</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">Regresar a mascotas</a>

</div>

</form>

</div>

</div>

1. El post para la acción addHistory

// POST: Owners/AddHistory

[HttpPost]

public async Task<IActionResult> AddHistory(HistoryViewModel model)

{

if (ModelState.IsValid)

{

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

\_context.Histories.Add(history);

await \_context.SaveChangesAsync();

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

}

model.ServiceTypes = \_combosHelper.GetComboServiceTypes();

return View(model);

}

1. Se sigue con la acción editHistory en el controlador owner la parte get

// GET: Owners/EditHistory

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

{

if (id == null)

{

return NotFound();

}

var history = await \_context.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));

}

1. La vista de editHistory

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Editar";

}

<h2>Editar</h2>

<h4>Historia</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="Guardar" class="btn btn-primary" />

<a **asp-action**="DetailsPet" **asp-route-id**="@Model.PetId" class="btn btn-success">Regresar a mascota</a>

</div>

</form>

</div>

</div>

1. La parte post del editHistory

// POST: Owners/EditHistory

[HttpPost]

public async Task<IActionResult> EditHistory(HistoryViewModel model)

{

if (ModelState.IsValid)

{

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

\_context.Histories.Update(history);

await \_context.SaveChangesAsync();

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

}

model.ServiceTypes = \_combosHelper.GetComboServiceTypes();

return View(model);

}

1. La vista addhistory y editHistory tienen una parte igual que es la del formulario esa parte se guarda en una nueva vista parcial para evitar repetir código llamada \_Histories y en la vistas add y edit history se reemplaza por el llamado además de que en la vista parcial también se copia la primera línea que es el modelo que usa así se referenciaría en ambas vistas y en la prcial queda el form lo repetido

<**partial** **name**="\_Histories" />

1. En el owner controler en la vista detailsPet se va hacer un cambio con respecto al botón de borrar historia

Se cambia este

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

Por este

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

1. Se copia el código para la ventana modal después del cierre de la tabla

<!--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>

1. Se copia el script con la función para borrar

@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. En el owner controller hacer la acción para deleteHistory para el caso solo es get

// GET: Owners/DeleteHistory

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

{

if (id == null)

{

return NotFound();

}

var history = await \_context.Histories

.Include(h => h.Pet)

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

if (history == null)

{

return NotFound();

}

\_context.Histories.Remove(history);

await \_context.SaveChangesAsync();

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

}

1. En la vista details de propietario para hacer ahora el borrado de las mascotas se reemplazará el Action de deletePet por el botón quedando este código

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

1. Como lo referente a la ventana modal es un código que se va a repetir mucho se puede crear una vista parcial pero se haría en la carpeta shared llamada \_DeleteDialog no recibe modelo no se coloca y el código de la vista modal que anteriormente se copio en la vista detailsPet y que tb se va a necesitar en detail de owner se pasa para esa vista y se reemplazaria por el llamado de la vista parcial el cual iria en details y en detailsPet

<**partial** **name**="\_DeleteDialog" />

1. Se copia el script en su respectiva sección

@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/DeletePet/' + item\_to\_delete;

});

});

</script>

}

1. Seguiria la acción en el cotrolador de owner para deletePet solo get

// GET: Owners/DeletePet

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.Pets

.Include(p => p.Owner)

.Include(p => p.Histories)

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

if (pet == null)

{

return NotFound();

}

if(pet.Histories.Count >0)

{

ModelState.AddModelError(string.Empty, "La mascota no puede ser borrada porque tiene registros relacionados");

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

}

\_context.Pets.Remove(pet);

await \_context.SaveChangesAsync();

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

}

1. En pettypes y Servicetypes en la vista index hacer lo del delete reemplazar el link por botón y llamar a la vista parcial y se coloca el mismo script solo cambiándole la parte ultima que se refiere al controlador y a la acción que esta llamando que para pettypes seria controlador PetTypes y acción delete y para Service type seria cotrolador ServiceTypes y acción delete
2. Como original el delete tiene get y post solo se deja el get con algunos cambios quedando así en los respectivos controladores de PetTypes y ServiceTypes

// GET: PetTypes/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var petType = await \_context.PetTypes

.Include(pt => pt.Pets)

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

if (petType == null)

{

return NotFound();

}

if(petType.Pets.Count > 0)

{

ModelState.AddModelError(string.Empty, "El tipo de mascota no puede ser borrado.");

return RedirectToAction(nameof(Index));

}

\_context.PetTypes.Remove(petType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

// GET: ServiceTypes/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var serviceType = await \_context.ServiceTypes

.Include(st => st.Histories)

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

if (serviceType == null)

{

return NotFound();

}

if (serviceType.Histories.Count > 0)

{

ModelState.AddModelError(string.Empty, "El tipo de servicio no puede ser borrado.");

return RedirectToAction(nameof(Index));

}

\_context.ServiceTypes.Remove(serviceType);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

1. Método para borrar el propietario para el cual antes se debe borrar usuario para que no quede hueco de seguridad en la interface de userhelper crear el método para borrar usuario y en la clase implementarlo

Task<bool> DeleteUserAsync(string email);

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. En el index de owner cambiar el código de link por botón para borrar propietario

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

1. Llamar a la vista parcial que tiene el código de la ventana modal y escribir el código js

<**partial** **name**="\_DeleteDialog" />

@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. En el controlador de owner la acción delete original que tiene get y post cambiarla solo por un get con las modificaciones quedaría así

// GET: Owners/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners

.Include(o=>o.User)

.Include(o=>o.Pets)

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

if (owner == null)

{

return NotFound();

}

if(owner.Pets.Count > 0)

{

ModelState.AddModelError(string.Empty, "El propietario no puede ser borrado");

return RedirectToAction(nameof(Index));

}

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

\_context.Owners.Remove(owner);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

1. En la interface de IUserHelper agregar método para actualizar usuario e implementarlo en la clase

Task<IdentityResult> UpdateUserAsync(User user);

public async Task<IdentityResult> UpdateUserAsync(User user)

{

return await \_userManager.UpdateAsync(user);

}

1. Crear modelo editUserViewModel

public class EditUserViewModel

{

public int Id { get; set; }

[Display(Name = "Documento")]

[MaxLength(20, ErrorMessage = "El {0} puede tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string Document { get; set; }

[Display(Name = "Nombres")]

[MaxLength(50, ErrorMessage = "El {0} puede tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string FirstName { get; set; }

[Display(Name = "Apellidos")]

[MaxLength(50, ErrorMessage = "El {0} puede tener máximo {1} caracteres.")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

public string LastName { get; set; }

[Display(Name = "Dirección")]

[MaxLength(100, ErrorMessage = "El {0} puede tener máximo {1} caracteres.")]

public string Address { get; set; }

[Display(Name = "Teléfono")]

[MaxLength(50, ErrorMessage = "El {0} puede tener máximo {1} caracteres.")]

public string PhoneNumber { get; set; }

}

}

1. Como addUserVieModel es muy parecido para que no haya duplicidad de código addUserVieModel hereda de editUserViewModel y en el add se eliminan campos repetidos que el los señalara como advertencias
2. En el controlador de owner la acción edit get y post se cambian y quedarían de esta forma ya para cuando edite propietario edite los campos que se puedan del usuario también acción get

// GET: Owners/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners

.Include(o => o.User)

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

if (owner == null)

{

return NotFound();

}

var model = 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(model);

}

1. Modificaciones en la vista edit

@model MyVet.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Editar";

}

<h2>Editar</h2>

<h4>Propietario</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="Guardar" class="btn btn-primary" />

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Acción post para edit de owner

// POST: Owners/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditUserViewModel model)

{

if (ModelState.IsValid)

{

var owner = await \_context.Owners

.Include(o => o.User)

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

owner.User.Document = model.Document;

owner.User.FirstName = model.FirstName;

owner.User.LastName = model.LastName;

owner.User.Address = model.Address;

owner.User.PhoneNumber = model.PhoneNumber;

await \_userHelper.UpdateUserAsync(owner.User);

return RedirectToAction(nameof(Index));

}

return View(model);

}

1. La vista edit y créate son muy parecidas entonces se hace una vista parcial con los campos en común y se borran de ambas vistas y solo se llama la vista parcial que se hace en la carpeta compartida y se llamara \_Owner y el modelo que recibe es de la vista edit

**PENDIENTE ARREGLAR CRUD MANAGER**

API: Obtener datos de los diferentes modelos en formato json para poderlos consumir desde aplicaciones móviles

1. Crear controlador de api usando entity framework para pettypes
2. Para probar se hace en postman se debe entrar a la configuración de postman y deshabilitar el parámetro SSL certificate verification para que funcione

<https://localhost:44397/api/PetTypes>

1. Json es un estándar de comunicaciones
2. Del controlador pettypes se borran todas las acciones solo se deja getPetTypes
3. Crear controlador con un método llamado getOwner pasándole el email devuelva los datos del owner (datos del owner, cuantas mascotas tiene y si estas mascotas tienen historias)

using Microsoft.AspNetCore.Mvc;

using MyVet.Web.Data;

using System.Threading.Tasks;

namespace MyVet.Web.Controllers.API

{

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

[ApiController]

public class OwnersController : ControllerBase

{

private readonly DataContext \_context;

public OwnersController(DataContext context)

{

\_context = context;

}

[HttpPost]

[Route("GetOwnerByEmail")]

public async Task<IActionResult> GetOwner(EmailRequest emailRequest)

{

if(!ModelState.IsValid)

{

return BadRequest();

}

var owner = await \_context.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() == emailRequest.Email.ToLower());

var response = new OwnerResponse

{

Id = owner.Id,

FirstName = owner.User.FirstName,

LastName = owner.User.LastName,

Address = owner.User.Address,

Document = owner.User.Document,

Email = owner.User.Email,

PhoneNumber = owner.User.PhoneNumber,

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. En el proyecto common se crea una nueva carpeta llamada Models
2. Se debe crear la clase EmailRequest dentro de la carpeta que se acabo de crear para que quede en el proyecto compartido

public class EmailRequest

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

1. Crea clase llamada ownersResponse que tendra los atributos que se necesitan y también tiene la colección de petResponse por lo que se necesita esa clase también la cual tendrá una colección de historyResponse lo cual también se necesita esa clase
2. Clase HistoryResponse en el proyecto common en carpeta 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; }

public DateTime DateLocal => Date.ToLocalTime();

}

1. Clase petResponse

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. Clase ownerResponse

public class OwnerResponse

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Document { get; set; }

public string Address { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

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

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

}

1. PROBAR EN PSTMAN se debe correr el proyecto

<https://localhost:44397/api/Owners/GetOwnerByEmail> POSTMAN

1. En appsetting.json adicionar unas líneas de código para crear token de seguridad

{

"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. En la interface IUserHelper crear otro método Task<SignInResult> ValidatePasswordAsync(User user, string password);
2. e implementarlo en la clase respectiva el cual valida si la combinación de usuario y pass son validos

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

{

return await \_signInManager.CheckPasswordSignInAsync(

user,

password,

false);

}

1. En el accountController hacer la inyección del código se necesita leer los valores de la configuración inyecta iconfiguration eto sirve para acceder a as 3 claves key,issuer y audience (129)

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

public AccountController(

IUserHelper userHelper,

IConfiguration configuration)

{

\_userHelper = userHelper;

\_configuration = configuration;

}

1. En el account controller se debe crear un método post el cual servirá para loguearse en la aplicación llamado createToken

[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. Para que el api sea autenticado en el owner controller y pettypes controller pero del api agregar esta anotación

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class OwnersController : ControllerBase

1. En el Starup se debe decir que se van a utilizar los token

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. En el postman correr método para crear token porque ya los métodos no funcionan sin autentificación URL/PREFIJO/CONTROLADOR/ACCION para el caso no hay prefijo método POST

<https://localhost:44397/Account/CreateToken> BODY - RAW - JSON Pasando en formato json user y una pass valida se manda por el body para que viaje encriptado

{

"userName": "dianarussi@yahoo.com",

"password": "123456"

}

1. Eso genera un token con un tiempo de expiración , ese token se usa en los métodos de getownerbyemail post y el get de pettypes para que funconen en la parte de autentificación como tipo de token bearer y se copia el token que genero al darle enviar vuelve a funcionar
2. Para publicar en azure del proyecto web clic derecho publicar











Pass Diana2019



1. En el modelo pet cambiar la ruta de publicación en azure por la que se creo que seria para el caso esta

<https://myvirtualweb.azurewebsites.net>

1. Extensión al vs y nuget al proyecto se le debe agregar una extensión al VS llamada prism template pack
2. Agregar un nyevo proyecto a la solución en plantillas buscar Prism escoger Prism blank app(xamarin.forms) llamar MyVet.Prism crea 4 proyectos nuevos
3. Para que dispositivo android funcione toca habilitar depuración usb
4. Borrar en el proyecto compartido de Prism mainViewModel
5. En la carpeta view del proyecto Prism adicionar nuevo elemento Prism – xamarin form de tipo contentPage llamada LoginPage

**StackLayout:** Los elementos que se vayan colocando se apilan hacia abajo tiene propiedad **Padding:** Unidad que se calcula dependiendo de la densidad y tamaño del pantalla del telefono izq, arri, der, aba

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

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

Title="{Binding Title}">

<StackLayout

Padding="10">

<Label

Text="Email"/>

<Entry

Keyboard="Email"

Placeholder="Ingrese su correo..."

Text="{Binding Email}"/>

<Label

Text="Contraseña"/>

<Entry

IsPassword="True"

Placeholder="Ingrese su contraseña..."

Text="{Binding Password}"/>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<Button

BackgroundColor="Navy"

BorderRadius="23"

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

HeightRequest="46"

Text="Ingresar"

TextColor="White"/>

</StackLayout>

</ContentPage>

1. En el app.xaml se define el diccionario de recursos por la clase asociada es donde dice por donde arranca la aplicación que por defecto dice mainPage se cambia a LoginPage y en el método se debe borrar la relacionada al mainPage para que no saque error ahi se configura la inyección de dependencia
2. Cada binding debe tener relación en la LoginPageViewModel para este caso
3. En la loginPageViewModel y en todas las viewModel se cambia la herencia, por defecto viene BindableBase se debe cambiar por ViewModelBase
4. Despues de ese cambio el contructor saca error se le debe mandar una inyeccion que es importante en Prism INavigationService como parámetro y al contructor base se le manda la clase navigationService este es el constructor

public LoginPageViewModel(

INavigationService navigationService) : base(navigationService)

{

}

1. La navigationPage tiene la propiedad title por lo que dentro del anterior método se coloca en el contructor

Title = "Ingresar";

1. Orden como deben de ir por el mismo orden

* Atributos privados
* Constructor
* Propiedades publicas
* Métodos públicos
* Métodos privados

1. Para el binding de email se define propiedad
2. Las propiedades que no se vayan a cambiar desde la viewModel como el email se define así

public string Email { get; set; }

1. Las propiedades que se les quiera hacer cambio desde la viewModel y que ese cambio se vea reflejado en la interface de usuario para el caso seria password

* Se crea atributo privado (según orden antes mencionado)

private string \_password;

* Se crea la pripedad publica (nombre que se le da en el binding iniciando con mayuscula). Esto se hace con todas las propiedades que se desee que cuando se haga cambio en viewModel ese cambio se vea reflejado en la interface de usuario

public string Password

{ get => \_password;

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

}

1. Se continua con el siguiente binding IsRunning lo primero es preguntar si se hacen cambios desde la viewModel para este caso si porque cuando se empieza a procesar se le dice que se active y cuando termine que se desactive se le crea atributo privado(al inicio) y propiedad

private bool \_isRunning;

public bool IsRunning

{

get => \_isRunning;

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

}

1. Para el caso de IsEnabled es igual
2. Los bool por defecto arrancan en false en el constructor para el isenabled se debe de colocar en verdadero para que se habilite para el caso del isrunning no se jace nada

IsEnabled = true;

1. Para el caso del binfing con el LoginCommand como es un botón

* Se crea propiedad privada de tipo DelegateCommand

private DelegateCommand \_loginCommand;

* Se crea la propiedad publica que en la forma nueva queda así solo ejecute la acción cuando lo primero sea nulo

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(Login));

l

* Crear el método Login

private async void Login()

{

//validar que se haya digitado email y pass

if(string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "Usted debe ingresar un email", "Accept");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert("Error", "Usted debe ingresar una contraseña", "Accept");

return;

}

await App.Current.MainPage.DisplayAlert("Oh", "Listo", "Accept");

}

1. Para consumir los servicios se crean unas clases agregar en la carpeta common la clase response que será genérica y se utilizara para dar el estado de cuando se consuma cualquier cosa en el API

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

1. Se crea otra clase para que devuelva el token para login con los atributos correspondientes user y pass

public class TokenRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

1. Lo anterior devuelve el token y la fecha de expiración para lo cual toca crear la clase tokenResponse

public class TokenResponse

{

public string Token { get; set; }

public DateTime Expiration { get; set; }

public DateTime ExpirationLocal => Expiration.ToLocalTime();

}

1. Se crea clase genérica que se va poder usar trasnversalmente en toda la aplicación en la arpeta ccomon se agrega carpeta service dentro de la cual se agrega la interface IApiService tendra dos métodos que se habilitan en el token

public interface IApiService

{

Task<Response> GetOwnerByEmail(

string urlBase,

string servicePrefix,

string controller,

string tokenType, // para el caso es bearer

string accessToken, // num largo que genera

string email);

Task<Response> GetTokenAsync( //se obtiene token con este metodo

string urlBase, // url de publicacion

string servicePrefix, //ruteo del controlador

string controller, // nombre controlador y la accion

TokenRequest request); // este ultimo combinacion d usuario y pasa

}

1. Descerializar coger un string y volverlo objeto y serializar es coger un objeto y volverlo string
2. La implementación de esos métodos en la clase ApiService

public class ApiService : IApiService

{

public async Task<Response> GetTokenAsync(

string urlBase,

string servicePrefix,

string controller,

TokenRequest request) //request combinacion email - pass

{

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. Se necesita matricular la dirección url para no tenr que firmar esa url en cada método y poderlo reusar esto se haría en el diccionario de recursos donde se pueden meter muchas cosas archivo de confirguracion general que tienen las aplicaciones de movilidad
2. En el app.xaml se coloca un parámetro string que se llama urlAPI DONDE SE COLOCA la dirección de azure

<Application.Resources>

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">hthttps://myvirtualweb.azurewebsites.net</x:String>

</ResourceDictionary>

</Application.Resources>

</prism:PrismApplication>

1. En la clase del app.xaml se inyecta el apiservice

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<LoginPage, LoginPageViewModel>();

}

1. Algunas adiciones al loginPageViewModel

public class LoginPageViewModel : ViewModelBase

{

private readonly IApiService \_apiService;

private string \_password;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

public LoginPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

Title = "Ingresar";

IsEnabled = true;

\_apiService = apiService;

}

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);

}

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(Login));

private async void Login()

{

//validar que se haya digitado email y pass

if (string.IsNullOrEmpty(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "Usted debe ingresar un email", "Accept");

return;

}

if (string.IsNullOrEmpty(Password))

{

await App.Current.MainPage.DisplayAlert("Error", "Usted debe ingresar una contraseña", "Accept");

return;

}

//mientras opere

IsRunning = true;

IsEnabled = false;

var request = new TokenRequest

{

Password = Password,

Username = Email,

};

var url = App.Current.Resources["UrlAPI"].ToString(); //urlapi es como se llama en el app.xaml

var response = await \_apiService.GetTokenAsync(url, "/Account", "/CreateToken", request);

if (!response.IsSuccess)

{

//cuando termine

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert("Error", "User or password incorrect.", "Accept");

Password = string.Empty;

return;

}

//aquí va código para navegar a otra pagina

}

}

1. Crear otra pagina donde se muestra la lista de mascotas que tiene el propietario que se logueo esta nueva pagina creada va a ser de tipo Prism content page llamada PetsPage a la cual inicialmente se
2. En la petsPageViewModel se hace el cambio de quien va a heredar no es el que sale por defecto que es **BindableBase sino de la ViewModelBase** y se necesita al igual que en la anterior inyectar **INavigation Service** por lo que en el constructor se coloca como parámetro mandado a la clase padre y aquí se colocaría el valor d ela propiedad **Title**

public class PetsPageViewModel : ViewModelBase

{

public PetsPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = “Pets”;

}

}

1. **En la loginPageViewModel se inyecta el navigationService pero no se inicializo como capo lo que se debe hacer para poderlo usar por lo que ese es el paso siguiente**

public class LoginPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private string \_password;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

public LoginPageViewModel(

INavigationService navigationService,

IApiService apiService ) : base(navigationService)

{

Title = "Ingresar";

IsEnabled = true;

\_navigationService = navigationService;

\_apiService = apiService;

}

1. El pedazo que quedo pte en el 167en el loginPageViewModel diciendo que navegue a la ste pagina quedaría así

Password = string.Empty;

await \_navigationService.NavigateAsync("PetsPage");

1. Consumir wl owner agregando las líneas de código en el loginViewModel para que salga la lista de mascotas de ese propietario

if (!response.IsSuccess)

{

//cuando termine

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert("Error", "Usuario o contraseña incorrecta", "Accept");

Password = string.Empty;

return;

}

//consumir owner

var token = (TokenResponse)response.Result;

var response2 = await \_apiService.GetOwnerByEmailAsync(url, "api", "/Owners/GetOwnerByEmail", "bearer",token.Token,Email);

if (!response2.IsSuccess)

{

IsRunning = false;

IsEnabled = true;

await App.Current.MainPage.DisplayAlert("Error", "Este usuario tiene un problema. Llama a soporte técnico", "Accept");

Password = string.Empty;

return;

}

//descerialzar objeto usuario

var owner = (OwnerResponse)response2.Result;

var parameters = new NavigationParameters

{

{ "owner", owner }

};

IsRunning = false;

IsEnabled = true;

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

Password = string.Empty;

1. En la petsPage se sobreescribe el método onNavigationgTo y se crea la propiedad privada \_owner

private OwnerResponse \_owner;

public PetsPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Pets";

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if(parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner"); Title = $"Mascotas de: { \_owner.FullName}";

}

}

1. Probar
2. **El petPage quedaría así**

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

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

BackgroundColor="Silver"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ListView

BackgroundColor="Transparent"

HasUnevenRows="True"

ItemsSource="{Binding Pets}"

SeparatorVisibility="None">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Frame

CornerRadius="10"

HasShadow="True"

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

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding ImageUrl}"

WidthRequest="100"/>

<Grid

Grid.Column="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="Nombre"/>

<Label

Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

Text="{Binding Name}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="Born"/>

<Label

Grid.Column="1"

Grid.Row="1"

FontAttributes="Bold"

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

<Label

Grid.Column="0"

Grid.Row="2"

Text="Raza"/>

<Label

Grid.Column="1"

Grid.Row="2"

FontAttributes="Bold"

Text="{Binding Race}"/>

</Grid>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. En el petPageViewModel toca crear el atributo privado de tipo observablecollection

private ObservableCollection<PetResponse> \_pets;

1. Su respectiva propiedad publica

public ObservableCollection<PetResponse> Pets

{

get => \_pets;

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

}

1. En el método que se sobrescribió ya se puede cargar la propiedad

if(parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner");

Title = $"Mascotas de: { \_owner.FullName}";

Pets = new ObservableCollection<PetResponse>(\_owner.Pets);

}

1. Probar
2. Buscar iconos en el android asset studio Iconos genéricos en carpeta de recursos de proyecto android crear carpeta drawable dentro de la cual guardar icono > se puede hacer con todos los tamaños pero inicialmente con colocar el tamaño estándar es suficiente para el proyecto ios se pone en la raíz de la carpeta resources y en el proyecto uwp se coloca iconos en la raíz del proyecto
3. Se adicina en la grid principal otra columna para esa imagen quedando así

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding ImageUrl}"

WidthRequest="100"/>

<Grid

. . .

</Grid>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"

</Image>

</Grid>

1. Xamarin form control pagna controles nativos
2. Syncfusion xamarin controles avanzados
3. En el proyecto compartido en los viewModel agregar un viewModel origen de una lista el cual se llamara PetItemViewModel
4. En el petPageViewModel cambiar el modelo en el que se base la observableCollection que es petResponse al nuevo modelo creado en el paso anterior se hace cambio en el atributo privado como en la propiedad publica y en el método la instancia también se cambia y ya no se manda la colección de pet \_owner.Pets toca hacer una transformación con la función select

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if(parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner");

Title = $"Mascotas de: { \_owner.FullName}";

Pets = new ObservableCollection<PetItemViewModel>(\_owner.Pets.Select(p => new PetItemViewModel

{

Born = p.Born,

Histories = p.Histories,

Id = p.Id,

ImageUrl = p.ImageUrl,

Name = p.Name,

PetType = p.PetType,

Race = p.Race,

Remarks = p.Remarks

}).ToList());

}

}

1. En el pagePage hacerle cambios a Image quedando así

<Image

Grid.Column="2"

Source="ic\_chevron\_right">

<Image.GestureRecognizers>

<TapGestureRecognizer

Command="{Binding SelectPetCommand}"/>

</Image.GestureRecognizers>

</Image>

1. Ese selectPetCommand se llama en el petItemViewModel
2. Con este se debe navagar a otra pagina en views construir nueva contentPage de Prism llamada PetPagea la cual se le binding el titulo

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

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

Title="{Binding Title}">

</ContentPage>

1. En el petItemViewModel agrega constructor

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();

parameters.Add("pet", this);

await \_navigationService.NavigateAsync("PetPage");

}

}

1. El petPageViewModel

public class PetPageViewModel : ViewModelBase

{

private PetResponse \_pet;

public PetPageViewModel(INavigationService navigationService) : base(navigationService)

{

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if(parameters.ContainsKey("pet"))

{

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

Title = \_pet.Name;

}

}

}

181 y en el petsPageViewModel toca inyectar navigatorservice

public class PetsPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private OwnerResponse \_owner;

private ObservableCollection<PetItemViewModel> \_pets;

public PetsPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Pets";

\_navigationService = navigationService;

}

public ObservableCollection<PetItemViewModel> Pets

{

get => \_pets;

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

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if(parameters.ContainsKey("owner"))

{

\_owner = parameters.GetValue<OwnerResponse>("owner");

Title = $"Mascotas de: { \_owner.FullName}";

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());

}

1. Si se quiere que pase a la otra pagina tocando desde cualquier parte de la pagina el binding para selectPetCommand en la petPage cambia de lugar y el código quedaría así

<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>

<Image

Grid.Column="0"

Source="{Binding ImageUrl}"

WidthRequest="100"/>

<Grid>

.. .

</Grid>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"/>

</Grid>

</Frame>

1. En el pagePetViewModel se crea la propiedad publica

public PetResponse Pet

{

get => \_pet;

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

}

1. En el petPageViewModel establecer el pet SE HACE UNOS CAMBIOS

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if(parameters.ContainsKey("pet"))

{

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

Title = Pet.Name;

}

}

1. En el 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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

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

Title="{Binding Title}">

<StackLayout

Padding="10">

<Image

HeightRequest="250"

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

<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"/>

</Grid.RowDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="Nombre"/>

<Label

Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

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

<Label

Grid.Column="0"

Grid.Row="1"

Text="Born"/>

<Label

Grid.Column="1"

Grid.Row="1"

FontAttributes="Bold"

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

<Label

Grid.Column="0"

Grid.Row="2"

Text="Raza"/>

<Label

Grid.Column="1"

Grid.Row="2"

FontAttributes="Bold"

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

<Label

Grid.Column="0"

Grid.Row="3"

Text="Tipo mascota"/>

<Label

Grid.Column="1"

Grid.Row="3"

FontAttributes="Bold"

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

<Label

Grid.Column="0"

Grid.Row="4"

Text="Comentarios"/>

<Label

Grid.Column="1"

Grid.Row="4"

FontAttributes="Bold"

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

</Grid>

</StackLayout>

</ContentPage>

1. Para areglar problemas para mostrar imágenes en android se debe actualizar la librería de xamarin forn a la del video de la 3.6.0344457 a 4.2.0.709249 la xamarin.android a la versión 28.0.0.1
2. En la solución agrefar el nuget xamarin.FFImageLoading.Forms en Prism, android y ios
3. En android en el main.activity se debe inicializar el render de la librería que se agrego

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

LoadApplication(new App(new AndroidInitializer()));

}

1. En el appdelegate en el proyecto ios se hace lo mismo pero en este no se coloca el true

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

LoadApplication(new App(new iOSInitializer()));

return base.FinishedLaunching(app, options);

}

1. En la petsPage que es donde se va usar donde se necesita mostrar las imágenes se agrega referencia de esa clase que se va a utilizar

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

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

xmlns:prism="http://prismlibrary.com"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

1. donde establa el atributo image en esa pet se reemplaza por

<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"

WdthRequest="100"/>

1. en la pagina pet también se referencia y se usa la propiedad binding iria Pet.ImageUrl y el withrequest de 300 tambien se agrega un heigtRequest de 300 y un aspect de aspetfit
2. En todas las page login, Pets y pet meter los stackloyout dentro de un scrooview
3. Arreglo con respecto a la conexión de internet agregar nuget al proyecto common llamado **Xam.Plugin.Connectivity**
4. **En la interface Iapiservice agregar nurvo método**

Task<bool> CheckConnection(string url);

1. **e implementarlo en la clase**

public async Task<bool> CheckConnection(string url)

{

if (!CrossConnectivity.Current.IsConnected)

{

return false;

}

return await CrossConnectivity.Current.IsRemoteReachable(url);

}

1. Se modifica la loginPageViewModel antes d econsumir el servicio se valida que todo esta bien

//mientras opere

IsRunning = true;

IsEnabled = false;

var url = App.Current.Resources["UrlAPI"].ToString(); //urlapi es como se llama en el app.xaml

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

{

Password = Password,

Username = Email,

};

var response = await \_apiService.GetTokenAsync(url, "/Account", "/CreateToken", request);

1. Probar7
2. Del proyecto common se hace modificacione a la clase response para que sea mas genérica

public class Response<T> where T : class

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public T Result { get; set; }

}

1. Hacer los cambios en Iapiservice Y en la clase donde diga solo response agregar ese pedazo

public interface IApiService

{

Task<Response<OwnerResponse>> GetOwnerByEmailAsync(

string urlBase,

string servicePrefix,

string controller,

string tokenType, // para el caso es bearer

string accessToken, // num largo que genera

string email);

Task<Response<TokenResponse>> GetTokenAsync( //se obtiene token con este metodo

string urlBase, // url de publicacion

string servicePrefix, //ruteo del controlador

string controller, // nombre controlador y la accion

TokenRequest request); // este ultimo combinacion d usuario y pasa

1. Crear una nueva contentpage llamada 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="http://prismlibrary.com"

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="Fecha"/>

<Label

Grid.Column="1"

FontAttributes="Bold"

Text="Tipo de servicio"/>

<Label

Grid.Column="2"

FontAttributes="Bold"

Text="Descripción"/>

</Grid>

<ListView

HasUnevenRows="True"

ItemsSource="{Binding Histories}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<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}'}"

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"

VerticalOptions="Center"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

1. Hacer los cambios en la historiesPageViewModel

public class HistoriesPageViewModel : ViewModelBase

{

private PetResponse \_pet;

private ObservableCollection<HistoryResponse> \_histories;

public HistoriesPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Historias clinicas";

}

public ObservableCollection<HistoryResponse> Histories

{

get => \_histories;

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

}

public PetResponse Pet

{

get => \_pet;

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

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

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

Title = $"Historias clinicas de { Pet.Name}";

Histories = new ObservableCollection<HistoryResponse>(Pet.Histories);

}

}

}

1. Para probar en petitemviewmodel de forma temporal cambiar a donde maneja de petPage a historiespage
2. Se necesita una clase que sirva de origen para la lista de los detalles de la historia clínicas de las mascotas por lo que se crea clase llamada historyItemViewModel

public class HistoryItemViewModel : HistoryResponse

{

private readonly INavigationService \_navigationService;

public HistoryItemViewModel(INavigationService navigationService)

{

\_navigationService = navigationService;

}

}

1. En la historiesPageViewModel se debe cambiar el parámetro que recibe de historyResponse a historyItemViewModel y en los lugares que lo requieran la clase quedarías así

public class HistoriesPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PetResponse \_pet;

private ObservableCollection<HistoryItemViewModel> \_histories;

public HistoriesPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Historias clinicas";

\_navigationService = navigationService;

}

public ObservableCollection<HistoryItemViewModel> Histories

{

get => \_histories;

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

}

public PetResponse Pet

{

get => \_pet;

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

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

if (parameters.ContainsKey("pet"))

{

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

Title = $"Historias clinicas de { Pet.Name}";

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. En la historiesPage

<Grid>

<Grid.GestureRecognizers>

<TapGestureRecognizer

Command="{Binding SelectHistoryCommand}"/>

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

1. Adicionar líneas de código HistoryItemViewModel

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. En vistas agregar una nueva pagina de contenido llamada 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="http://prismlibrary.com"

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="Auto"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

FontAttributes="Bold"

Text="Fecha"/>

<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="Tipo de servicio"/>

<Label

Grid.Row="1"

Grid.Column="1"

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

<Label

Grid.Row="2"

Grid.Column="0"

FontAttributes="Bold"

Text="Descripción"/>

<Label

Grid.Row="2"

Grid.Column="1"

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

<Label

Grid.Row="3"

Grid.Column="0"

FontAttributes="Bold"

Text="Comentarios"/>

<Label

Grid.Row="3"

Grid.Column="1"

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

</Grid>

</StackLayout>

</ScrollView>

</ContentPage>

1. Adicionar en las view una tabbedPage llamada 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="http://prismlibrary.com"

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. En la petTabbedViewModel se hacen los cambios que siempre se han hecho quedando así

using Prism.Navigation;

namespace MyVet.Prism.ViewModels

{

public class PetTabbedPageViewModel : ViewModelBase

{

public PetTabbedPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Mascota";

}

}

}

1. Cuando se ingresa a pet se pasa mascota y cuando se ingresa a historia también se debe pasar la mascota si es una tabbed page no se le puede pasar el objeto a las dos paginas por lo que el objeto pet se guarda donde se pueda recuperar después fácilmente para eso se agrega nuget llamado **xam.plugin.setting** al proyecto common agrega clase que da la posibilidad de guardar datos en persistencia en la aplicación para poder recuperar ese objeto cuando se requiera
2. Agregar carpeta helpers dentro del proyecto common dentro de la cual se agrega la clase 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. Se debe hacer cambio rn la petItemViewModel cambiando unas cosas por otras

private async void SelectPet()

{

Settings.Pet = JsonConvert.SerializeObject(this);

await \_navigationService.NavigateAsync("PetTabbedPage");

}

1. Otros cambios en el petTabbedPageViewModel

public PetTabbedPageViewModel(INavigationService navigationService) : base(navigationService)

{

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

Title = $"Mascota: {pet.Name}";

}

1. Algunos cambios en la pagePageViewModel

public class PetPageViewModel : ViewModelBase

{

private PetResponse \_pet;

public PetPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Detalles";

}

public PetResponse Pet

{

get => \_pet;

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

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

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

}

}

1. Algunos cambios en el historiesPageViewModel

public class HistoriesPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PetResponse \_pet;

private ObservableCollection<HistoryItemViewModel> \_histories;

public HistoriesPageViewModel(

INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Historias clinicas";

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

LoadHistories();

}

public ObservableCollection<HistoryItemViewModel> Histories

{

get => \_histories;

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

}

public PetResponse Pet

{

get => \_pet;

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

}

//se borra overrride

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. Probar
2. Colocar iconos de list y pet se le agrega a las petPage e historiesPage los inconos para que los reconozca con esta propiedad

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

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

xmlns:prism="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

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

IconImageSource="ic\_list"

Title="{Binding Title}">

1. Git worshop

* Antes de empezar a trabajar en master sync - fetch para verificar que xi hay algo pte

crear ramas para trabajar cambios

- Hacer push a las ramas creadas

- Subir los cambios colocarle nombre hacer un commint ans sync de los cambios hechos para subirlos al git sobre l rama creada

- pasarse a la rama master

- se debe garantizar que master local esta al dia con la master remota dándole en sync y luego fetch si no sale nada es xq esta al dia y se da clic cualquiera que fuera el resultado en sync

- hacer un merch con el fin de pasar lo que se hizo en una subrama a la principal lo primero que se hace es pasarse a la rama donde se va hacer el merch (ya debe estar ahí)

- ahí se sabe si ha conflictos se debe escoger la opción de comparar archivos en caso de que haya se debe hacer commit que se refieran a la resolución de conflictos y commit all ans sync

- Si los cambios aun no están subidos y no funcionan se pueden deshacer haciendo clic derecho undo

- Si los cambios ya se subieron se verifica en la historia

- se va a donde esta guardada la solución y en la barra de dirección se coloca el cmd

Git –version muestra la versión del git

* Según a donde se desee devolver se crea otra rama a partir de la parte donde se desea devolver
* Otro comando git checkout –b nombreNuevaRamaMaster IdDondeDevolverse con el fin de crear otra rama a partir de esa rama y ya después borrar la rama master que se daño

1. **XAMARIN FORM – SYNCFUSION XAMARIN CONTROLES**
2. Colocarle imagen al login agregar al loginPage las líneas de código correspondientes

<ScrollView>

<StackLayout

Padding="10">

<Image

HeightRequest="200"

Source="LogoVet"/>

1. Cambiar el activity indicator a uno mejor de los que están en syncfusion
2. Licencia de syncfusion para xamarin

**MTgyMDY0QDMxMzcyZTMzMmUzMEZ5aTN2Tkk2ZSttcFhvRHM5eXlRb25Ca1RYNDRhbWJHV0xFTW96WVZtcTQ9**

1. Agregar nuget Syncfusion.Xamarin.SfBusyIndicator versión 17.2.0.49 se le agrega al Prism al ios y al android y hay nuget exclusivo para android y ios que se deben de agregar a cada proyexto como tal
2. Se debe de agregar la licencia obtenida en la clase app

protected override async void OnInitialized()

{

Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("MTgyMDY1QDMxMzcyZTMzMmUzMEZ5aTN2Tkk2ZSttcFhvRHM5eXlRb25Ca1RYNDRhbWJHV0xFTW96WVZtcTQ9");

InitializeComponent();

await NavigationService.NavigateAsync("NavigationPage/LoginPage");

}

1. Como son controles adicionales de xamarin se deben de renderizar para el caso del proyecto android es la clase por donde arranca la cual es mainActivity.cs

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfBusyIndicatorRenderer();

LoadApplication(new App(new AndroidInitializer()));

}

1. Para el caso del proyecto ios es en el archivo appDelegate

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

new SfBusyIndicatorRenderer();

LoadApplication(new App(new iOSInitializer()));

return base.FinishedLaunching(app, options);

}

1. Para poderlo utilizar se va a la pagina donde se quiere colocar para el caso el login y se agrega la referencia

<?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="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

1. Reemplazar el activity indicator estándar por el nuevo en la pagina

<?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="http://prismlibrary.com"

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">

<Image

HeightRequest="230"

Source="LogoVet"/>

<Label

Text="Email"/>

<Entry

Keyboard="Email"

Placeholder="Ingrese su correo..."

Text="{Binding Email}"/>

<Label

Text="Contraseña"/>

<Entry

IsPassword="True"

Placeholder="Ingrese su contraseña..."

Text="{Binding Password}"/>

<Button

BackgroundColor="Navy"

BorderRadius="23"

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

HeightRequest="46"

Text="Ingresar"

TextColor="White"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="Silver"

HorizontalOptions="Center"

TextColor="White"

IsVisible="{Binding IsRunning}"

Title="Cargando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

Atributos privados

* Constructor
* Propiedades publicas
* Métodos públicos
* Métodos privados

1. Se vuelve al backend para hacer las paginas de error
2. En el accountcontroller crear método llamado NotAuthorized

public IActionResult NotAuthorized()

{

return View();

}

1. Crear la correspondiente vista

@{

ViewData["Title"] = "NotAuthorized";

}

<br />

<br />

<img src="~/images/notAuthorized.png" />

<h2>Usted no tiene permiso para realizar esta acción!</h2>

1. Modificación en el Starup ara decir que se van a usar esas paginas

public void ConfigureServices(IServiceCollection services)

{

services.ConfigureApplicationCookie(options =>

{

options.LoginPath = "/Account/NotAuthorized";

options.AccessDeniedPath = "/Account/NotAuthorized";

});

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;

});

1. En ese mismo archivo también se adiciona una línea donde llama al error quedando así

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();

1. En el home controller crear una acción llamada error

[Route("error/404")]

public IActionResult Error404()

{

return View();

}

1. También crear la correspondiente vista

@{

ViewData["Title"] = "Error404";

}

<br />

<br />

<img src="~/images/notAuthorized.png" />

<h2>Lo sentimos. Página no encontrada</h2>

1. Implementar la función para el autoregistro en el account controller colocar la acción get de Register

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

private readonly DataContext \_context;

public AccountController(

IUserHelper userHelper,

IConfiguration configuration,

DataContext context)

{

\_userHelper = userHelper;

\_configuration = configuration;

\_context = context;

}

public IActionResult Register()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(AddUserViewModel model)

{

if (ModelState.IsValid)

{

var user = await AddUserAsync(model);

if (user == null)

{

ModelState.AddModelError(string.Empty, "Correo ya existe.");

return View(model);

}

var owner = new Owner

{

Pets = new List<Pet>(),

User = user,

};

\_context.Owners.Add(owner);

await \_context.SaveChangesAsync();

var loginViewModel = new LoginViewModel

{

Password = model.Password,

RememberMe = false,

Username = model.Username

};

var result2 = await \_userHelper.LoginAsync(loginViewModel);

if (result2.Succeeded)

{

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

}

}

return View(model);

}

private async Task<User> AddUserAsync(AddUserViewModel model)

{

var user = new User

{

Address = model.Address,

Document = model.Document,

Email = model.Username,

FirstName = model.FirstName,

LastName = model.LastName,

PhoneNumber = model.PhoneNumber,

UserName = model.Username

};

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

if (result != IdentityResult.Success)

{

return null;

}

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

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

return newUser;

}

1. La vista de Register

@model MyVet.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Registrarse";

}

<h2>Registrarse</h2>

<h4>Propietario</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>

<**partial** **name**="\_User"/>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Probar
2. Usuario cambie pass es importante que lo haga en pantalla diferente a la de actualizar datos
3. View model para camnio de pass

public class ChangePasswordViewModel

{

[Display(Name = "Contraseña actual")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe contener entre {2} y {1} caracteres.")]

public string OldPassword { get; set; }

[Display(Name = "Nueva contraseña")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe contener entre {2} y {1} caracteres.")]

public string NewPassword { get; set; }

[Display(Name = "Confirmar contraseña")]

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[DataType(DataType.Password)]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe contener entre {2} y {1} caracteres.")]

[Compare("NewPassword")]

public string Confirm { get; set; }

}

1. Adicionar en la interface de userhelper el método para cambiar el password

Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword);

1. Implementar el método que se creo en la clase userhelper

public async Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword)

{

return await \_userManager.ChangePasswordAsync(user, oldPassword, newPassword);

}

1. Agregar la acción get en el controlador account para cabiar usuario

public async Task<IActionResult> ChangeUser()

{

var owner = await \_context.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);

}

1. Se adiciona la viista de changeUser

@model MyVet.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Propietario</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="Guardar" class="btn btn-primary" />

<a **asp-action**="ChangePassword" class="btn btn-warning">Cambiar contraseña</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. La acción post para cambiar los datos del usuario

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangeUser(EditUserViewModel model)

{

if (ModelState.IsValid)

{

var owner = await \_context.Owners

.Include(o => o.User)

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

owner.User.Document = model.Document;

owner.User.FirstName = model.FirstName;

owner.User.LastName = model.LastName;

owner.User.Address = model.Address;

owner.User.PhoneNumber = model.PhoneNumber;

await \_userHelper.UpdateUserAsync(owner.User);

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

}

return View(model);

}

1. Probar
2. Agregar en el accountcontroller acción get y post para cambio de pass

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, "Usuario no encontrado.");

}

}

return View(model);

}

1. Y añadir la vista

@model MyVet.Web.Models.ChangePasswordViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Cambiar contraseña</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">Contraseña actual</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">Nueva contraseña</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">Confirmar contraseña</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="Cambiar contraseña" class="btn btn-primary" />

<a **asp-action**="ChangeUser" class="btn btn-success">Regresar a usuario</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Probar
2. Se debe de confirmar el correo de registro por lo que los correos registrados deben ser validos ya que al email llegara correo para activar cuenta la cual se podrá usar después de realizar la confirmación
3. Se debe hacer cambios en el archivo de configuración del proyecto en el starup se agregan las líneas de código

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. Activar en google si la cuenta es gmail el acceso a app poco seguras

<https://myaccount.google.com/lesssecureapps>

1. En el archivo json appsetting se deben ingresar las credenciales validas de un correo que va a mandar los correos

"Mail": {

"From": "diana.russiposadamail@gmail.com",

"Smtp": "smtp.gmail.com",

"Port": 587,

"Password": "Knenita1020"

}

1. Se debe instalar un nuget al proyecto web llamado mailkit 2.3.0
2. En la carpeta helper adicionar interface y clase para crear emaill IMailHelper

public interface IMailHelper

{

void SendMail(string to, string subject, string body);

}

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. Configurar la inyección del nuevo servicio la nueva interface en el starup

services.AddScoped<IImageHelper, ImageHelper>();

services.AddScoped<IConverterHelper, ConverterHelper>();

services.AddScoped<IMailHelper, MailHelper>();

1. Se debe agregar 3 metodos mas a la interface del userHelper

Task<string> GenerateEmailConfirmationTokenAsync(User user);

Task<IdentityResult> ConfirmEmailAsync(User user, string token);

Task<User> GetUserByIdAsync(string userId);

1. Su respectiva implementación en la clase

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. Inyectar el emailhelper en el account controller

public class AccountController : Controller

{

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

private readonly DataContext \_context;

private readonly ImageHelper \_mailHelper;

public AccountController(

IUserHelper userHelper,

IConfiguration configuration,

DataContext context,

ImageHelper mailHelper)

{

\_userHelper = userHelper;

\_configuration = configuration;

\_context = context;

\_\_mailHelper = mailHelper;

}

1. Modifica la acción register post en el método account controller

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(AddUserViewModel model)

{

if (ModelState.IsValid)

{

var user = await AddUserAsync(model);

if (user == null)

{

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

return View(model);

}

var owner = new Owner

{

Pets = new List<Pet>(),

User = user,

};

\_context.Owners.Add(owner);

await \_context.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(model.Username, "Confirmación de email", $"<h1>Email Confirmation</h1>" +

$"Para activar el usuario, " +

$"por favor da clic en este link:</br></br><a href = \"{tokenLink}\">Confirmación de email</a>");

ViewBag.Message = "Las instrucciones para activar el usuario han sido enviadas a tu correo.";

return View(model);

}

return View(model);

}

1. Adicionar en la vista register al final las líneas de código para que muestre los mensajes

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

1. Crear acción get para confirmar email en el 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. Crear la vista para confirmar email

@{

ViewData["Title"] = "Confirm email";

}

<h2>@ViewData["Title"]</h2>

<div>

<p>

Gracias por confirmar tu correo. Ya puedes ingresar en el sistema.

</p>

</div>

1. En el owner controller tb se debe de inyectar el mailhelper

public class OwnersController : Controller

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly ICombosHelper \_combosHelper;

private readonly IImageHelper \_imageHelper;

private readonly IConverterHelper \_converterHelper;

private readonly IMailHelper \_mailHelper;

public OwnersController(

DataContext context,

IUserHelper userHelper,

ICombosHelper combosHelper,

IImageHelper imageHelper,

IConverterHelper converterHelper,

IMailHelper mailHelper)

{

\_context = context;

\_userHelper = userHelper;

\_combosHelper = combosHelper;

\_imageHelper = imageHelper;

\_converterHelper = converterHelper;

\_mailHelper = mailHelper;

}

Como se envía email se modifica la acción post de créate en el controlador de owner

// POST: Owners/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel model)

{

if (ModelState.IsValid)

{

//crear usuario

var user = new User

{

Address = model.Address,

Document = model.Document,

Email = model.Username,

FirstName = model.FirstName,

LastName = model.LastName,

PhoneNumber = model.PhoneNumber,

UserName = model.Username

};

var response = await \_userHelper.AddUserAsync(user, model.Password);

if (response.Succeeded)

{

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

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

//crear owner

var owner = new Owner

{

Agendas = new List<Agenda>(),

Pets = new List<Pet>(),

User = userInDB

};

//mandar bd

\_context.Owners.Add(owner);

try

{

await \_context.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(model.Username, "Confirmación de email", $"<h1>Email Confirmation</h1>" +

$"Para activar el usuario, " +

$"por favor da clic en este link:</br></br><a href = \"{tokenLink}\">Confirmación de email</a>");

return RedirectToAction(nameof(Index));

}

catch (Exception ex)

{

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

return View(model);

}

}

ModelState.AddModelError(string.Empty, response.Errors.FirstOrDefault().Description);

}

return View(model);

}

1. Se debe de borrar la bd xq tiene usuarios los cuales no se activaron por correo antes de poder ingresar a la cuenta los usuarios que se creen a partir de ahora se debe hacer con correo valido

drop-database

1. Se debe modificar método checkuser en el seedb que toca modificarle y adicionarle las líneas para que para el caso no se mande al correo si no que haga la acción de activarlo por debajo teniendo en cuenta que es usuario admin

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. Se adiciona codigo en el mismo seeddb el metodo addpet

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. Prueba
2. En el xamarin se pueden ver los cambios en caliente con una funcioalidad que sacaron nueva actualizar en línea la intrface de usuario se llama hot reload y para habilitarlo se debe

* Tener la ultima versión del vs preview (versión del profe)
* Actualizar nuget de xamarin form a la ultima versión
* En tolos - option – xamarin – hot reload – enabled

1. Para recuperar la contraseña en la vista login agregar un nuevo link para recuperar contraseña

<div class="form-group">

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

<a **asp-action**="Register" class="btn btn-primary">Registrar nuevo usuario</a>

<a **asp-action**="RecoverPassword" class="btn btn-link">Olvido su contraseña?</a>

</div>

1. Adicionar un nuevo modelo para recuperar contraseña

public class RecoverPasswordViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

1. Adicionar otro modelo para resetear la contraseña

public class ResetPasswordViewModel

{

[Required]

[Display(Name = "Email")]

public string UserName { get; set; }

[Required]

[Display(Name = "Nueva contraseña")]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe contener entre {2} y {1} caracteres.")]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[Display(Name = "Confirmar contraseña")]

[StringLength(20, MinimumLength = 6, ErrorMessage = "El campo {0} debe contener entre {2} y {1} caracteres.")]

[DataType(DataType.Password)]

[Compare("Password",ErrorMessage ="La nueva contraseña y la confirmación deben ser iguales")]

public string ConfirmPassword { get; set; }

[Required]

public string Token { get; set; }

}

1. Agregarle a IUserHelper 2 metodos para genere token y para que resetee el pass

Task<string> GeneratePasswordResetTokenAsync(User user);

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

1. Implementarlos en la clase

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. En el account controller se adicionan las acciones get para recuperar ontraseña

public IActionResult RecoverPassword()

{

return View();

}

1. Y la acción post

[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 no corresponde al usuario registrado.");

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, "Cambiar contraseña", $"<h1>Cambiar contraseña</h1>" +

$"Para cambiar tu contraseña por favor da clic en este link:</br></br>" +

$"<a href = \"{link}\">Reset Password</a>");

ViewBag.Message = "Las instrucciones para recuperar tu contraseña han sido enviadas a tu correo.";

return View();

}

return View(model);

}

1. y resetear la contraseña acción get

public IActionResult ResetPassword(string token)

{

return View();

}

1. Y la post

[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 = "Contraseña cambiada satisfactoriamente.";

return View();

}

ViewBag.Message = "Error al cambiar la contraseña.";

return View(model);

}

ViewBag.Message = "Usuario no encontrado.";

return View(model);

}

1. Se agrga la vista para recuperar la contraseña

@model MyVet.Web.Models.RecoverPasswordViewModel

@{

ViewData["Title"] = "Recover Password";

}

<h2>Recuperar contraseña</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="Recuperar contraseña" class="btn btn-primary" />

<a **asp-action**="Login" class="btn btn-success">Regresar a ingresar</a>

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

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

}

1. Se agrega vista de cambiar contraseña

@model MyVet.Web.Models.ResetPasswordViewModel

@{

ViewData["Title"] = "Reset Password";

}

<h1>Cambie su contraseña</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">Nueva contraseña</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">Confirmación contraseña</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="Cambiar contraseña" class="btn btn-primary" />

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

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

}

1. Probar
2. Borrar bd en azure y volver a publicar
3. Se le hacen unos arreglos o cambios a la vista index de Owners para que se vea mejor

@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> Nuevo</a>

</p>

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

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

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

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

…..

<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>

<**partial** **name**="\_DeleteDialog" />

@section Scripts {

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

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

<script type="text/javascript">

$(document).ready(function () {

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

// Delete item

…

});

</script>

}

1. Todos los mismos cambiosen todos las vistas de los controladores
2. Arreglos y cambios al controlador de manager

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;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.Web.Controllers

{

[Authorize(Roles = "Admin")]

public class ManagersController : Controller

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly IMailHelper \_mailHelper;

public ManagersController(

DataContext context,

IUserHelper userHelper,

IMailHelper mailHelper

)

{

\_context = context;

\_userHelper = userHelper;

\_mailHelper = mailHelper;

}

// GET: Managers

public IActionResult Index()

{

return View(\_context.Managers.Include(m => m.User));

}

// GET: Managers/Details/5

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

{

if (id == null)

{

return NotFound();

}

var manager = await \_context.Managers

.Include(o => o.User)

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

if (manager == null)

{

return NotFound();

}

return View(manager);

}

// GET: Managers/Create

public IActionResult Create()

{

return View();

}

// POST: Managers/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel model)

{

if (ModelState.IsValid)

{

var user = await AddUser(model);

if (user == null)

{

ModelState.AddModelError(string.Empty, "Este correo ya existe.");

return View(model);

}

var manager = new Manager { User = user };

\_context.Add(manager);

await \_context.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(model.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(model);

}

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;

}

// GET: Managers/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var manager = await \_context.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);

}

// POST: Managers/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditUserViewModel view)

{

if (ModelState.IsValid)

{

var owner = await \_context.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);

}

// GET: Managers/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var manager = await \_context.Managers

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

if (manager == null)

{

return NotFound();

}

\_context.Managers.Remove(manager);

await \_context.SaveChangesAsync();

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

return RedirectToAction(nameof(Index));

}

private bool ManagerExists(int id)

{

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

}

}

}

1. Vista créate de manager

@model MyVet.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Nuevo</h2>

<h4>Administrador</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>

<**partial** **name**="\_User"/>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Vista edit de manager

@model MyVet.Web.Models.EditUserViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Administrador</h4>

<hr />

<div class="row">

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

<form **asp-action**="Edit">

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

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

<**partial** **name**="\_User" />

<div class="form-group">

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

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Vista details de manager

@model MyVet.Web.Data.Entities.Manager

@{

ViewData["Title"] = "Details";

}

<h2>Administrador</h2>

<div>

<h4>Detalles</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">Editar</a>

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</a>

</div>

1. Vista index de manager

@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> Crear nuevo</a>

</p>

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Aministradores</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-warning" **asp-route-id**="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a **asp-action**="Details" class="btn btn-info" **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>Está seguro de borrar este registro?</p>

</div>

<div class="modal-footer">

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

<button type="button" class="btn btn-danger" id="btnYesDelete">Borrar</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. Se crea controlador para mascotas

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 \_context;

public PetsController(

ICombosHelper combosHelper,

DataContext dataContext)

{

\_combosHelper = combosHelper;

\_context = dataContext;

}

public IActionResult Index()

{

return View(\_context.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 \_context.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 \_context.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 \_context.Owners.FindAsync(view.OwnerId),

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

Race = view.Race,

Remarks = view.Remarks

};

\_context.Pets.Update(pet);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(view);

}

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.Pets

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

if (pet == null)

{

return NotFound();

}

\_context.Pets.Remove(pet);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

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

{

if (id == null)

{

return NotFound();

}

var history = await \_context.Histories

.Include(h => h.Pet)

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

if (history == null)

{

return NotFound();

}

\_context.Histories.Remove(history);

await \_context.SaveChangesAsync();

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

}

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

{

if (id == null)

{

return NotFound();

}

var history = await \_context.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 \_context.Pets.FindAsync(view.PetId),

Remarks = view.Remarks,

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

};

\_context.Histories.Update(history);

await \_context.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 \_context.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 \_context.Pets.FindAsync(view.PetId),

Remarks = view.Remarks,

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

};

\_context.Histories.Add(history);

await \_context.SaveChangesAsync();

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

}

return View(view);

}

}

}

1. Vista index para Pets

@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">Mascotas</h3>

</div>

<div class="panel-body">

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

<thead>

<tr>

<th>

Propietario

</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>

Historias

</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-warning" **asp-route-id**="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a **asp-action**="Details" class="btn btn-info" **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-->

<**partial** **name**="\_DeleteDialog"/>

@section Scripts {

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

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

<script type="text/javascript">

$(document).ready(function () {

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

// Delete item

var item\_to\_delete;

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

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

});

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

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

});

});

</script>

}

1. Vista edit de pet

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Mascota</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" />

<**partial** **name**="\_Pet" />

<div class="form-group">

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

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</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. Vista detalle de mascotas

@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>Mascota</h2>

<div>

<h4>Detalles</h4>

<hr />

<div class="row">

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

<dl class="dl-horizontal">

<dt>

Propietario

</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">Editar</a>

<a **asp-action**="AddHistory" **asp-route-id**="@Model.Id" class="btn btn-primary">Agregar historia</a>

<a **asp-action**="Index" **asp-route-id**="@Model.Id" class="btn btn-success">Regresar a la lista</a>

</div>

<hr />

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

{

<h4>No hay historias clinicas adicionadas aun.</h4>

}

else

{

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Historia clinica</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-success" **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-->

<**partial** **name**="\_DeleteDialog"/>

@section Scripts {

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

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

<script type="text/javascript">

$(document).ready(function () {

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

// Delete item

var item\_to\_delete;

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

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

});

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

window.location.href = '/Pets/DeleteHistory/' + item\_to\_delete;

});

});

</script>

}

1. Agregar la vista de addHistory cuya acción esta en el controlador de pet

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Nueva</h2>

<h4>Historia clinica</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" />

<**partial** **name**="\_Histories"/>

<div class="form-group">

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

<a **asp-action**="Details" **asp-route-id**="@Model.PetId" class="btn btn-success">Regresar a mascota</a>

</div>

</form>

</div>

</div>

1. Vista edit de mascota

@model MyVet.Web.Models.HistoryViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Historia clinica</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" />

<**partial** **name**="\_Histories" />

<div class="form-group">

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

<a **asp-action**="Details" **asp-route-id**="@Model.PetId" class="btn btn-success">Regresar a mascota</a>

</div>

</form>

</div>

</div>

1. Probar
2. En el \_layout cambiar nombre de la veterinaria y pie de pagina
3. Buscar imágenes de 1200 x 400 para los slide de la pagina ppal
4. La vista index del home quedaría asi

@{

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>

<li data-target="#myCarousel" data-slide-to="8"></li>

</ol>

<div class="carousel-inner" role="listbox">

<div class="item active">

<img src="~/images/1.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/2.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/3.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/4.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/5.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/6.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/7.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/8.jpg" class="img-responsive" />

</div>

<div class="item">

<img src="~/images/9.png" 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">Anterior</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">Siguiente</span>

</a>

</div>

<hr />

<div class="row">

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

<a **asp-action**="MyPets" class="btn btn-primary">Mis mascotas</a>

</div>

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

<a **asp-action**="MyAgenda" class="btn btn-warning">Mi agenda</a>

</div>

</div>

1. Adicionar una interface helper para agenda IAgendaHelper

public interface IAgendaHelper

{

Task AddDays(int days);

}

1. Adicionar la respectiva clase

using MyVet.Web.Data;

using MyVet.Web.Data.Entities;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace MyVet.Web.Helpers

{

public class AgendaHelper : IAgendaHelper

{

private readonly DataContext \_context;

public AgendaHelper(DataContext dataContext)

{

\_context = dataContext;

}

public async Task AddDays(int days)

{

DateTime initialDate;

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

{

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

}

else

{

var agenda = \_context.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)

{

\_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. Hacer la inyección del helper en el Starup

services.AddScoped<IConverterHelper, ConverterHelper>();

services.AddScoped<IMailHelper, MailHelper>();

services.AddScoped<IAgendaHelper, AgendaHelper>();

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

1. Reemplazar controlador de agenda por este

public class AgendaController : Controller

{

private readonly DataContext \_context;

private readonly IAgendaHelper \_agendaHelper;

public AgendaController(

DataContext context,

IAgendaHelper agendaHelper)

{

\_context = context;

\_agendaHelper = agendaHelper;

}

// GET: Agenda

public IActionResult Index()

{

return View(\_context.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.AddDaysAsync(7);

return RedirectToAction(nameof(Index));

}

}

1. Modificar la vista index del controlador de agendas

@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> Adicionar dias</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>

Propietario

</th>

<th>

Mascota

</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">Cancelar cita</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>Está seguro de cancelar esta cita?</p>

</div>

<div class="modal-footer">

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

<button type="button" class="btn btn-danger" id="btnYesDelete">Cancelar cita</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. Como el seeder crea unos registros por defecto se omentarea el método y se borran los registros en un query de la tabla agenda
2. Probar
3. Se deben de adicionar dos métodos en la interface ICombosHelper

public interface ICombosHelper

{

IEnumerable<SelectListItem> GetComboPetTypes();

IEnumerable<SelectListItem> GetComboServiceTypes();

IEnumerable<SelectListItem> GetComboOwners();

IEnumerable<SelectListItem> GetComboPets(int ownerId);

}

1. Se implementan en la clase comboHelper

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 = "(Selecciona un propietario...)",

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 = "(Selecciona una mascota...)",

Value = "0"

});

return list;

}

1. Crear el modelo agendaViewModel el cual queda asi

public class AgendaViewModel : Agenda

{

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[Display(Name = "Propietario")]

[Range(1, int.MaxValue, ErrorMessage = "Usted debe seleccionar un propietario.")]

public int OwnerId { get; set; }

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

[Required(ErrorMessage = "El campo {0} es obligatorio.")]

[Display(Name = "Mascota")]

[Range(1, int.MaxValue, ErrorMessage = "Usted debe seleccionar una mascota.")]

public int PetId { get; set; }

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

}

1. Adicionar en el controlador de agenda la acción get para asignar citas se debe inyectar también enn ese controlador el combohelper

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_context.Agendas

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

if (agenda == null)

{

return NotFound();

}

var model = new AgendaViewModel

{

Id = agenda.Id,

Owners = \_combosHelper.GetComboOwners(),

Pets = \_combosHelper.GetComboPets(0)

};

return View(model);

}

1. Acción post para asignar citas

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Assing(AgendaViewModel model)

{

if (ModelState.IsValid)

{

var agenda = await \_context.Agendas.FindAsync(model.Id);

if (agenda != null)

{

agenda.IsAvailable = false;

agenda.Owner = await \_context.Owners.FindAsync(model.OwnerId);

agenda.Pet = await \_context.Pets.FindAsync(model.PetId);

agenda.Remarks = model.Remarks;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

}

model.Owners = \_combosHelper.GetComboOwners();

model.Pets = \_combosHelper.GetComboPets(model.OwnerId);

return View(model);

}

public async Task<JsonResult> GetPetsAsync(int ownerId)

{

var pets = await \_context.Pets

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

.OrderBy(p => p.Name)

.ToListAsync();

return Json(pets);

}

1. Vista de asignar citas

@model MyVet.Web.Models.AgendaViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Asignar</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="Asignar" class="btn btn-primary" />

<a **asp-action**="Index" class="btn btn-success">Regresar a la lista</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">(Seleccione una mascota...)</option>');

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

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

+ pet.id + '">'

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

});

},

error: function (ex) {

alert('Fallo al recuperar las mascotas.' + ex.statusText);

}

});

return false;

})

});

</script>

}

1. Hacer get para desasignar agenda

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_context.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;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

1. Probar
2. Inyectar el datacontexr en el homecontroller ya que este controlador ya tiene que acceder a la bd

private readonly DataContext \_context;

public HomeController(DataContext context)

{

\_context = context;

}

1. Adicionar acción mypets en el home controller

[Authorize(Roles = "Customer")]

public IActionResult MyPets()

{

return View(\_context.Pets

.Include(p => p.PetType)

.Include(p => p.Histories)

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

}

1. Vista mypets

@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">Mascotas</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>

Historias clinicas

</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-warning" **asp-route-id**="@item.Id"><i class="glyphicon glyphicon-pencil"></i> </a>

<a **asp-action**="Details" class="btn btn-info" **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-->

<**partial** **name**="\_DeleteDialog"/>

@section Scripts {

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

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

<script type="text/javascript">

$(document).ready(function () {

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

// Delete item

var item\_to\_delete;

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

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

});

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

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

});

});

</script>

}

1. Inyectarse el comboshelper en el homecontroller
2. Adicionar acción edit el get y el post en el controlador de home

[Authorize(Roles = "Customer")]

public async Task<IActionResult> Edit(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.Value);

if (pet == null)

{

return NotFound();

}

var model = 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(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(PetViewModel model)

{

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 = model.Born,

Id = model.Id,

ImageUrl = path,

Name = model.Name,

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

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

Race = model.Race,

Remarks = model.Remarks

};

\_context.Pets.Update(pet);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(MyPets));

}

model.PetTypes = \_combosHelper.GetComboPetTypes();

return View(model);

}

1. Adicionar vista edit

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Editar</h2>

<h4>Mascotas</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" />

<**partial** **name**="\_Pet"/>

<div class="form-group">

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

<a **asp-action**="MyPets" class="btn btn-success">Regresar a mis mascotas</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. Probar
2. Crear la acción de detalles

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.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. Crear la vista para detalles

@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>Mascota</h2>

<div>

<h4>Detalles</h4>

<hr />

<div class="row">

<div class="col-md-8">

<dl class="dl-horizontal">

<dt>

Propietario

</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">Editar</a>

<a **asp-action**="MyPets" **asp-route-id**="@Model.Id" class="btn btn-success">Regresar a mis mascotas</a>

</div>

<hr />

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

{

<h4>No hay historias clinicas adicionadas.</h4>

}

else

{

<div class="row">

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

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Historia clinica</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. Crar acción de delete

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var pet = await \_context.Pets

.Include(p => p.Histories)

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

if (pet == null)

{

return NotFound();

}

if (pet.Histories.Count > 0)

{

return RedirectToAction(nameof(MyPets));

}

\_context.Pets.Remove(pet);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(MyPets));

}

1. Acción get y post para créate

[Authorize(Roles = "Customer")]

public async Task<IActionResult> Create()

{

var owner = await \_context.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 model)

{

if (ModelState.IsValid)

{

var path = string.Empty;

if (model.ImageFile != null && model.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 model.ImageFile.CopyToAsync(stream);

}

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

}

var pet = new Pet

{

Born = model.Born,

ImageUrl = path,

Name = model.Name,

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

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

Race = model.Race,

Remarks = model.Remarks

};

\_context.Pets.Add(pet);

await \_context.SaveChangesAsync();

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

}

model.PetTypes = \_combosHelper.GetComboPetTypes();

return View(model);

}

1. Vista para créate

@model MyVet.Web.Models.PetViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Nueva</h2>

<h4>Mascota</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" />

<**partial** **name**="\_Pet"/>

<div class="form-group">

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

<a **asp-action**="MyPets" **asp-route-id**="@Model.OwnerId" class="btn btn-success">Regresar a mis mascotas</a>

</div>

</form>

</div>

</div>

1. Agregar 2 propiedades a la agendaviewmodel

public class AgendaViewModel : Agenda

{

…

public int PetId { get; set; }

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

public bool IsMine { get; set; }

public string Reserved => "Reserved";

}

1. En el home controller adicionar método myagenda

[Authorize(Roles = "Customer")]

public async Task<IActionResult> MyAgenda()

{

var agendas = await \_context.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. Adicionar la vista para agenda

@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>

Propietario

</th>

<th>

Mascota

</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-primary" **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">Cancelar cita</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>Está seguro de cancelar la cita?</p>

</div>

<div class="modal-footer">

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

<button type="button" class="btn btn-danger" id="btnYesDelete">Cancelar cita</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. Acción get y post en el home controller para asignar agenda

[Authorize(Roles = "Customer")]

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_context.Agendas

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

if (agenda == null)

{

return NotFound();

}

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

if (owner == null)

{

return NotFound();

}

var model = new AgendaViewModel

{

Id = agenda.Id,

OwnerId = owner.Id,

Pets = \_combosHelper.GetComboPets(owner.Id)

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Assing(AgendaViewModel model)

{

if (ModelState.IsValid)

{

var agenda = await \_context.Agendas.FindAsync(model.Id);

if (agenda != null)

{

agenda.IsAvailable = false;

agenda.Owner = await \_context.Owners.FindAsync(model.OwnerId);

agenda.Pet = await \_context.Pets.FindAsync(model.PetId);

agenda.Remarks = model.Remarks;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(MyAgenda));

}

}

model.Pets = \_combosHelper.GetComboPets(model.Id);

return View(model);

}

}

1. Vista para asignar la agenda

@model MyVet.Web.Models.AgendaViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Asignar</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" />

<**partial** **name**="\_Assing" />

<div class="form-group">

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

<a **asp-action**="MyAgenda" class="btn btn-success">Regresar a mi agenda</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Método para desasignar una agenda en el home controller

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

{

if (id == null)

{

return NotFound();

}

var agenda = await \_context.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;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(MyAgenda));

}

1. Se sigue con la parte móvil
2. Adicionar nuevo modelo llamado userRequest en el proyecto common

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. Adicionar controlador api para account

using Microsoft.AspNetCore.Http;

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;

using System.Linq;

using System.Threading.Tasks;

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<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

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

if (user != null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Este correo ya existe."

});

}

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 de confirmación", $"<h1>Email de confirmación</h1>" +

$"Para activar su usuario, " +

$"por favor clic en este link:</br></br><a href = \"{tokenLink}\">Email de confirmación</a>");

return Ok(new Response<object>

{

IsSuccess = true,

Message = "Se le ha enviado un email a su cuenta de correo. Porfavor confirme para su correo ingresar a la aplicaciòn."

});

}

}

}

1. Probar en postman recovery pass local
2. Se crea la acción de recuperar contraseña en el account controler del api

[HttpPost]

[Route("RecoverPassword")]

public async Task<IActionResult> RecoverPassword([FromBody] EmailRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

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

if (user == null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Este correo no existe en el sistema."

});

}

var myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

var link = Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Cambio de contraseña", $"<h1>Recuperación de contraseña</h1>" +

$"TPara cambiar tu contraseña haz clic en el siguiente link:</br></br>" +

$"<a href = \"{link}\">RCambiar contraseña</a>");

return Ok(new Response<object>

{

IsSuccess = true,

Message = "Se ha enviad a tu correo instrucciones para cambiar tu contraseña."

});

}

}

1. Probar en postman se debe correr proyecto
2. Crear acción para modificar usuario en el controlador account del api para esta opción necesario token el usuario debe etar logueado para poder hacerlo

[HttpPut]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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. Probar en postman put modificar usuario
2. Se debe adicionar un modelo llamado changepasswordrequest

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. Acción post en el controlador account del api para cambiar contraseña método se rutea xq ya existen varios post y x el nombre que no empieza por post

[HttpPost]

[Route("ChangePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "Bad request"

});

}

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

if (user == null)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = "El correo ingresado no existe."

});

}

var result = await \_userHelper.ChangePasswordAsync(user, request.OldPassword, request.NewPassword);

if (!result.Succeeded)

{

return BadRequest(new Response<object>

{

IsSuccess = false,

Message = result.Errors.FirstOrDefault().Description

});

}

return Ok(new Response<object>

{

IsSuccess = true,

Message = "La contraseña fue cambiada exitosamente!"

});

}

1. Probar en postman método post para cambiar pass
2. Se debe crear dentro del proyecto common un fileHelper para poder convertir un array en un archivo (para el caso de mascota que necesita foto. El memory stream es la lectura del archivo en memoria

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. Crear el modelo petRequest

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. Adicionar estos métodos a la interface de converter helper

public interface IConverterHelper

{

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

PetViewModel ToPetViewModel(Pet pet);

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

HistoryViewModel ToHistoryViewModel(History history);

PetResponse ToPetResponse(Pet pet);

OwnerResponse ToOwnerResposne(Owner owner);

}

1. Implementar los métodos en la clase

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. Adicionar controlador api para mascotas

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class PetsController : ControllerBase

{

private readonly DataContext \_dataContext;

private readonly IConverterHelper \_converterHelper;

public PetsController(

DataContext dataContext,

IConverterHelper converterHelper)

{

\_dataContext = dataContext;

\_converterHelper = converterHelper;

}

[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.ToUniversalTime(),

ImageUrl = imageUrl,

Name = request.Name,

Owner = owner,

PetType = petType,

Race = request.Race,

Remarks = request.Remarks

};

\_dataContext.Pets.Add(pet);

await \_dataContext.SaveChangesAsync();

return Ok(\_converterHelper.ToPetResponse(pet));

}

1. Probar crear mascotas en postman
2. Método para actualizar mascotas que se copia en el pet controller

[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("Mascota no existe.");

}

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

if (petType == null)

{

return BadRequest("Tipo de mascota no valido.");

}

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.ToUniversalTime();

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(\_converterHelper.ToPetResponse(oldPet));

}

}

1. Probar en postman la modificación de mascotas
2. Método para obtener los tipos de macotas que se hace en el controlador de pettypes del api

[HttpGet]

public IEnumerable<PetType> GetPetTypes()

{

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

}

1. Probar en postman
2. Crear el agendaResponse en el proyecto común

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. Crear el controlador api para agenda y crear la acción para obtener agenda por propietario

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

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public class AgendaController : Controller

{

private readonly DataContext \_context;

private readonly IConverterHelper \_converterHelper;

public AgendaController(

DataContext dataContext,

IConverterHelper converterHelper)

{

\_context = dataContext;

\_converterHelper = converterHelper;

}

[HttpPost]

[Route("GetAgendaForOwner")]

public async Task<IActionResult> GetAgendaForOwner(EmailRequest emailRequest)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var agendas = await \_context.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. Probar en postman por el backend antes crear agenda a las mascotas
2. Crear clase para asignar agenda

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. Agregar en el controlador de agenda del api la acción para asignar agenda

[HttpPost]

[Route("AssignAgenda")]

public async Task<IActionResult> AssignAgenda(AssignRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

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

if (agenda == null)

{

return BadRequest("Cita no existe.");

}

if (!agenda.IsAvailable)

{

return BadRequest("Cita no disponible.");

}

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

if (owner == null)

{

return BadRequest("Propietario no existe.");

}

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

if (pet == null)

{

return BadRequest("Mascota no existe.");

}

agenda.IsAvailable = false;

agenda.Remarks = request.Remarks;

agenda.Owner = owner;

agenda.Pet = pet;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return Ok(true);

}

1. Probar en postman
2. Crear el request para desasignar agenda

public class UnAssignRequest

{

[Required]

public int AgendaId { get; set; }

}

1. Crear la acción en el controlador de agenda para desasignarla

[HttpPost]

[Route("UnAssignAgenda")]

public async Task<IActionResult> UnAssignAgenda(UnAssignRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var agenda = await \_context.Agendas

.Include(a => a.Owner)

.Include(a => a.Pet)

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

if (agenda == null)

{

return BadRequest("Cita no existe.");

}

if (agenda.IsAvailable)

{

return BadRequest("Cita no disponible.");

}

agenda.IsAvailable = true;

agenda.Remarks = null;

agenda.Owner = null;

agenda.Pet = null;

\_context.Agendas.Update(agenda);

await \_context.SaveChangesAsync();

return Ok(agenda);

}

1. Se debe borrar la base de datos en azure entrando al portal – mis recursos y volver a publicar
2. En el proyecto common en la carpeta helper en la clase setting se agregan dos constantes que permitirán que en el dispositivo se pueda recordar usuario y pass

public static class Settings

{

private const string \_pet = "Pet";

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 Pet

{

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

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

}

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. . en el proyecto Prism en la loginpageviewModel se adicionan las líneas para guardar el propietario y el token

//descerialzar objeto usuario

var owner =(OwnerResponse)response2.Result;

Settings.Owner = JsonConvert.SerializeObject(owner);

Settings.Token = JsonConvert.SerializeObject(token);

var parameters = new NavigationParameters

{

{ "owner", owner }

};

IsRunning = false;

IsEnabled = true;

await \_navigationService.NavigateAsync("PetsPage");

1. En el petpageviewModel se hacen cambios y la pagina queda asi

public class PetsPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private OwnerResponse \_owner;

private ObservableCollection<PetItemViewModel> \_pets;

public PetsPageViewModel(INavigationService navigationService) : base(navigationService)

{

\_navigationService = navigationService;

Title = "Pets";

LoadOwner();

}

public ObservableCollection<PetItemViewModel> Pets

{

get => \_pets;

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

}

private void LoadOwner()

{

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

Title = $"Mascotas de: { \_owner.FullName}";

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());

}

}

1. Probar la app
2. En el proyecto common en los modelos agregar nueva clase que se va a llamar menú

public class Menu

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

}

1. En los view model se crea menumodelviewmodel

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("LoginPage"))

{

await \_navigationService.NavigateAsync("/NavigationPage/LoginPage");

return;

}

await \_navigationService.NavigateAsync($"/VeterinaryMasterDetailPage/NavigationPage/{PageName}");

}

}

1. Crear una master detail de prism para veterinary

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

x:Class="MyVet.Prism.Views.VeterinaryMasterDetailPage">

<MasterDetailPage.Master>

<ContentPage

Title="Menu">

<StackLayout Padding="20">

<Image

HeightRequest="150"

Source="LogoVet"/>

<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"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage>

</MasterDetailPage.Master>

</MasterDetailPage>

1. El modelo de master detail pageviewmodel quedaría asi

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());

}

}

xmlns:prism="clr-namespace:prism.Mvvm;assembly=Prism.Forms"

1. Agregar las diferentes content page para poder probar ybindarlas con la propiedad title y de donde heredan
2. Mappage

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="CenterAndExpand">

<Label Text="{Binding Title}"/>

</StackLayout>

1. Agendapage

<StackLayout

HorizontalOptions="CenterAndExpand"

VerticalOptions="CenterAndExpand">

<Label Text="{Binding Title}"/>

</StackLayout>

1. En la loginviewmodel se debe cambiar

await \_navigationService.NavigateAsync("/VeterinaryMasterDetailPage/NavigationPage/PetsPage");

1. Agregar para el splash de android imagen de 480 x 800
2. En la hoja de stilos agregar estas líneas de código

<style name="Theme.Splash" parent="android:Theme">

<item name="android:windowBackground">@drawable/splash</item>

<item name="android:windowNoTitle">true</item>

</style>

1. En el proyecto android crear en a raíz un archivo de tipo activity llamado 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. Se debe cambiar en el main activity el mainlauncher a false
2. En properties se cambia el nombre de la aplicación a my vet
3. Cambiar el incono el cual tiene por defecto que se llama launcher
4. Probar la app
5. Cambiar icono en ios
6. Pagina para bucar iconos make app icon
7. Los iconos en ios no pueden ser transparetes
8. En una pagina llamada makeappicon se crea los iconos en la diferentes resoluciones para ios
9. Archivo info.plist se cambia el nombre de la app y el mske interface es launch screen para que cree la tarjeta donde se coloca el splash
10. En la 2da pestaña que es visual assert la source utiza el appicon
11. En el archivo asset darle unset a las imágenes por defecto
12. Se debe tener mac a partir de aca ir a archivo launchHistoryBoard
13. Colocar fondo blanco
14. Toolbox- imageView – propiedad image se coloca el logo en content mode seleccionar aspect fit
15. Ios tiene algo llamado constrain que son las reglas de como se comportan los elementos en las diferentes pantallas seleccionando la imagen se debe activar ese editor de constrain y anclarla a la parte sup, inf, der e izq
16. En el diccionario de recursos agregar colores archivo app.xaml

<ResourceDictionary>

<!-- Parameters -->

<x:String x:Key="UrlAPI">https://myvirtualweb.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>

1. En tpdas las page agregar por ejemplo para el comor de fondo la ste línea de código

BackgroundColor="{StaticResource colorBackground}"

1. Por ejemplo para el boto en la loginpage quedaría asi

BackgroundColor="{StaticResource colorPrimary}"

TextColor="{StaticResource colorFontInverse}"

1. El de activity indicator en el login quedaría asi

BackgroundColor="{StaticResource colorFontDanger}"

TextColor="{StaticResource colorFontInverse}"

1. Darle un estilo único a todos los botones y a los label colocand en el diccionario de recursos las propiedades que se repiten para todos

<!-- Styles -->

<Style TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource colorPrimary}" />

<Setter Property="HorizontalOptions" Value="FillAndExpand" />

<Setter Property="TextColor" Value="{StaticResource colorFontInverse}" />

</Style>

</Style>

<Style TargetType="Label">

<Setter Property="TextColor" Value="{StaticResource colorFont}" />

</Style>

1. En la loginpage se agrega botón para registraese y el otro botón se coloca dentro del stacklayout

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Ingresar"/>

<Button

Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource ColorSecondary}"

Text="Regitrarse" />

</StackLayout>

1. Agrega estilo adicional para botón en el caso de login de registro

<Style x:Key="SecondaryButton" TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource ColorSecondary}" />

</Style>

1. Crear pagina tipo content para registrar usuarios y cambiar en la view model la respectiva propiedad al contructor y agregarle el binding de titulo y el static resource del bacgroundcolor
2. Implmementar el comando registerCommand en la login page view model su atributo privado y publico y su método privado para posteriormente probar

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

private DelegateCommand \_registerCommand;

…

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(Login));

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(Register));

private async void Register()

{

await \_navigationService.NavigateAsync("RegisterPage");//sin slash la pag de register se sobrepone a la de login

}

1. El código de la pagina de registro

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

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

BackgroundColor="{StaticResource colorBackground}"

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="Documento"

VerticalOptions="Center"/>

<Entry

Grid.Row="0"

Grid.Column="1"

Placeholder="Ingrese su documento..."

Text="{Binding Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="Nombres"

VerticalOptions="Center"/>

<Entry

Grid.Row="1"

Grid.Column="1"

Placeholder="Ingrese su nombre..."

Text="{Binding FirstName}"/>

<Label

Grid.Row="2"

Grid.Column="0"

Text="Apellidos"

VerticalOptions="Center"/>

<Entry

Grid.Row="2"

Grid.Column="1"

Placeholder="Ingrese sus apellidos..."

Text="{Binding LastName}"/>

<Label

Grid.Row="3"

Grid.Column="0"

Text="Dirección"

VerticalOptions="Center"/>

<Entry

Grid.Row="3"

Grid.Column="1"

Placeholder="Ingrese su dirección..."

Text="{Binding Address}"/>

<Label

Grid.Row="4"

Grid.Column="0"

Text="Correo"

VerticalOptions="Center"/>

<Entry

Grid.Row="4"

Grid.Column="1"

Keyboard="Email"

Placeholder="Ingrese su correo..."

Text="{Binding Email}"/>

<Label

Grid.Row="5"

Grid.Column="0"

Text="Teléfono"

VerticalOptions="Center"/>

<Entry

Grid.Row="5"

Grid.Column="1"

Keyboard="Telephone"

Placeholder="Ingrese su teléfono..."

Text="{Binding Phone}"/>

<Label

Grid.Row="6"

Grid.Column="0"

Text="Contraseña"

VerticalOptions="Center"/>

<Entry

Grid.Row="6"

Grid.Column="1"

IsPassword="True"

Placeholder="Ingrese su contraseña..."

Text="{Binding Password}"/>

<Label

Grid.Row="7"

Grid.Column="0"

Text="Confirmar contraseña"

VerticalOptions="Center"/>

<Entry

Grid.Row="7"

Grid.Column="1"

IsPassword="True"

Placeholder="Ingrese su confirmacipon de contraseña"

Text="{Binding PasswordConfirm}"/>

</Grid>

<Button

Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Register"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal">

<Label

Text="Recordar en este dispositivo"

VerticalOptions="Center"/>

<CheckBox

IsChecked="{Binding IsRemember}"/>

</StackLayout>

<Label

HorizontalOptions="Center"

Text="Olvido su contraseña?"

TextColor="{StaticResource colorAccent}">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding LoginCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Ingresar"/>

<Button

Command="{Binding RegisterCommand}"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="Regitrarse" />

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator

AnimationType="Gear"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

BackgroundColor="{StaticResource colorDanger}"

HorizontalOptions="Center"

TextColor="{StaticResource colorFontInverse}"

IsVisible="{Binding IsRunning}"

Title="Loading..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Crear helper para expresiones regulares ya que se desea validar email llamdo regexHelper

public static bool IsValidEmail(string emailaddress)

{

try

{

var mail = new MailAddress(emailaddress);

return true;

}

catch (FormatException)

{

return false;

}

}

1. El view model para registro quedaría asi

public class RegisterPageViewModel : ViewModelBase

{

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

public RegisterPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Registrar nuevo usuario";

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", "Debe ingresar un documento.", "Accept");

return false;

}

if (string.IsNullOrEmpty(FirstName))

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar su nombre.", "Accept");

return false;

}

if (string.IsNullOrEmpty(LastName))

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar su apellido.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Address))

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar su dirección.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Email) || !RegexHelper.IsValidEmail(Email))

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar un correo valido.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Phone))

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar un teléfono.", "Accept");

return false;

}

if (string.IsNullOrEmpty(Password) || Password.Length < 6)

{

await App.Current.MainPage.DisplayAlert("Error", "Debe ingresar contraseña de al menos 6 catacteres.", "Accept");

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert("Error", "Ingresa una confirmación de contraseña.", "Accept");

return false;

}

if (!Password.Equals(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert("Error", "La contraseña y la confirmación deben ser iguales.", "Accept");

return false;

}

return true;

}

}

1. En la interface api Service en el proyecto commons en la carpeta modelos y subcarpeta servicios adicionar el método para registrar usuarios

Task<Response<object>> RegisterUserAsync(

string urlBase,

string servicePrefix,

string controller,

UserRequest userRequest);

1. Y hacer la respectiva implementación en la clase

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. En la registerPageViewModel se debe hacer inyección del apiservice y el navigator colocarlo como campo

public class RegisterPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_registerCommand;

public RegisterPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

Title = "Registrar nuevo usuario";

IsEnabled = true;

\_navigationService = navigationService;

\_apiService = apiService;

}

1. Implementar método para registrar usuario completo

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. Habilitar la cuenta para que deje enviar el correo desde <https://accounts.google.com/DisplayUnlockCaptcha>
2. Probar en el teléfono
3. En la rememberpage colocar el binding para el titulo y que usa el backgroun del diccionario de recursos

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

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

BackgroundColor="{StaticResource colorBackground}"

Title="{Binding Title}">

1. En la registerPageViewModel se colocaría esto

public class RememberPasswordPageViewModel : ViewModelBase

{

public RememberPasswordPageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Recordar contraseña";

}

}

1. Adiciona esto en loginpageviewmodel

public class LoginPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private string \_password;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_loginCommand;

private DelegateCommand \_registerCommand;

private DelegateCommand \_forgotPasswordCommand;

public DelegateCommand LoginCommand => \_loginCommand ?? (\_loginCommand = new DelegateCommand(Login));

public DelegateCommand RegisterCommand => \_registerCommand ?? (\_registerCommand = new DelegateCommand(Register));

public DelegateCommand ForgotPasswordCommand => \_forgotPasswordCommand ?? (\_forgotPasswordCommand = new DelegateCommand(ForgotPassword));

private async void ForgotPassword()

{

await \_navigationService.NavigateAsync("RememberPasswordPage");

}

1. Probar en dispositivo
2. La interface para la pagina de recordar pass queda asi

<?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="http://prismlibrary.com"

prism:ViewModelLocator.AutowireViewModel="True"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

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

BackgroundColor="{StaticResource colorBackground}"

Title="{Binding Title}">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Label

Text="Email"/>

<Entry

Keyboard="Email"

Placeholder="Ingrese su correo..."

Text="{Binding Email}"/>

<Button

Command="{Binding RecoverCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Recordar contraseña"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator

BackgroundColor="{StaticResource colorDanger}"

TextColor="{StaticResource colorFontInverse}"

AnimationType="DoubleCircle"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

HorizontalOptions="Center"

IsVisible="{Binding IsRunning}"

Title="Recuperando Contraseña..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Probar de nuevo mostrando interface
2. Se debe ir al apiservice xq esta no se tiene preparada para consumir el nuevo método de recuperar contraseña colocarlo en la interface del apiservice e implementarlo en la clase

Task<Response<object>> RecoverPasswordAsync(

string urlBase,

string servicePrefix,

string controller,

EmailRequest emailRequest);

}

Implementación

public async Task<Response<object>> 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<object>>(answer);

return obj;

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. En la remeber view model se inyecta el api Service y también se necesita colocar a nivel de toda la clase el navigator Service para que se pueda devolver pantalla hacia atrás cuando recupere la contraseña el cdigo de esa pagina queda asi

using MyVet.Common.Helpers;

using MyVet.Common.Models;

using MyVet.Common.Models.Services;

using Prism.Commands;

using Prism.Navigation;

using System.Threading.Tasks;

namespace MyVet.Prism.ViewModels

{

public class RememberPasswordPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

private bool \_isRunning;

private bool \_isEnabled;

private DelegateCommand \_recoverCommand;

public RememberPasswordPageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

Title = "Recordar contraseña";

IsEnabled = true;

\_navigationService = navigationService;

\_apiService = apiService;

}

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(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

private async Task<bool> ValidateData()

{

if (string.IsNullOrEmpty(Email) || !RegexHelper.IsValidEmail(Email))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debes ingresar un correo valido.",

"Aceptar");

return false;

}

return true;

}

}

}

1. Probar
2. En la clase setting del proyecto commons se debe de adicionar la constante y una propiedad de solo lectura que por defecto inicia en false

public static class Settings

{

private const string \_pet = "Pet";

private const string \_token = "Token";

private const string \_owner = "Owner";

private const string \_isRemembered = "IsRemembered";

private static readonly bool \_boolDefault = false;

private static readonly string \_stringDefault = string.Empty;

1. También se define en ese mismo archivo la propiedad publica para asi cuando haya logueo se pregunta el estado con que el usuario se logueo y este se guarda en persistencia para la próxima vez que haya logueo si es el mismo usuario mandarlo de una dentro de la aplicación o a la masterpage según este check o no el chulo de remeber

public static bool IsRemembered

{

get => AppSettings.GetValueOrDefault(\_isRemembered, \_boolDefault);

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

}

1. En la login page view model se llama desde el setting para que funcione

Settings.Token = JsonConvert.SerializeObject(token);

Settings.IsRemembered = IsRemember;

1. En el archivo app.xaml es donde se valida la expiración del token

protected override async void OnInitialized()

{

Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("MTgyMDY1QDMxMzcyZTMzMmUzMEZ5aTN2Tkk2ZSttcFhvRHM5eXlRb25Ca1RYNDRhbWJHV0xFTW96WVZtcTQ9");

InitializeComponent();

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

if (Settings.IsRemembered && token?.Expiration > DateTime.Now)

{

await NavigationService.NavigateAsync("/VeterinaryMasterDetailPage/NavigationPage/PetsPage");

}

else

{

await NavigationService.NavigateAsync("/NavigationPage/LoginPage");

}

}

1. En la menú ítem view model se agrega lo siguiente la propiedad de remember en false

if (PageName.Equals("LoginPage"))

{

Settings.IsRemembered = false;

await \_navigationService.NavigateAsync("/NavigationPage/LoginPage");

return;

}

1. Probar
2. Hacer la interface para la profilePage sirve para modificar los datos del usuario

<?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="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

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

BackgroundColor="{StaticResource colorBackground}"

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="Documento"

VerticalOptions="Center"/>

<Entry

Grid.Row="0"

Grid.Column="1"

Placeholder="Ingrese su documento"

Text="{Binding Owner.Document}"/>

<Label

Grid.Row="1"

Grid.Column="0"

Text="Nombres"

VerticalOptions="Center"/>

<Entry

Grid.Row="1"

Grid.Column="1"

Placeholder="Ingrese su nombre"

Text="{Binding Owner.FirstName}"/>

<Label

Grid.Row="2"

Grid.Column="0"

Text="Apellidos"

VerticalOptions="Center"/>

<Entry

Grid.Row="2"

Grid.Column="1"

Placeholder="Ingrese su apellido"

Text="{Binding Owner.LastName}"/>

<Label

Grid.Row="3"

Grid.Column="0"

Text="Dirección"

VerticalOptions="Center"/>

<Entry

Grid.Row="3"

Grid.Column="1"

Placeholder="Ingrese su dirección"

Text="{Binding Owner.Address}"/>

<Label

Grid.Row="4"

Grid.Column="0"

Text="Teléfono"

VerticalOptions="Center"/>

<Entry

Grid.Row="4"

Grid.Column="1"

Keyboard="Telephone"

Placeholder="Ingrese su teléfono"

Text="{Binding Owner.PhoneNumber}"/>

</Grid>

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding SaveCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="Guardar"/>

<Button

Command="{Binding ChangePasswordCommand}"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Style="{StaticResource SecondaryButton}"

Text="Cambiar contraseña"/>

</StackLayout>

</StackLayout>

<busyindicator:SfBusyIndicator

BackgroundColor="{StaticResource colorDanger}"

TextColor="{StaticResource colorFontInverse}"

AnimationType="DoubleCircle"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

HorizontalOptions="Center"

IsVisible="{Binding IsRunning}"

Title="Guardando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

</ContentPage>

1. Probar que si muestre la interface para pasar a la parte donde se muetran los datos a modificar
2. Modificación de la profilePageViewModel

public class ProfilePageViewModel : ViewModelBase

{

//atributos privados

private bool \_isRunning;

private bool \_isEnabled;

private OwnerResponse \_owner;

private DelegateCommand \_saveCommand;

public ProfilePageViewModel(

INavigationService navigationService) : base(navigationService)

{

Title = "Modificar perfil";

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("Error","Debes ingresar un documento.", "Aceptar");

return false;

}

if (string.IsNullOrEmpty(Owner.FirstName))

{

await App.Current.MainPage.DisplayAlert("Error", "Debes ingresar un nombre.", "Aceptar");

return false;

}

if (string.IsNullOrEmpty(Owner.LastName))

{

await App.Current.MainPage.DisplayAlert("Error", "Debes ingresar un apellido.", "Aceptar");

}

if (string.IsNullOrEmpty(Owner.Address))

{

await App.Current.MainPage.DisplayAlert("Error", "Debes ingresar una dirección.", "Aceptar");

return false;

}

return true;

}

}

1. Probar
2. En la interface del apiservice

Task<Response<object>> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken);

1. En el apiService la clase colocar la implementación del método anterior

public async Task<Response<Object>> PutAsync<T>(

string urlBase,

string servicePrefix,

string controller,

T model,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(model);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response<Object>

{

IsSuccess = false,

Message = answer,

};

}

var obj = JsonConvert.DeserializeObject<T>(answer);

return new Response<Object>

{

IsSuccess = true,

Result = obj,

};

}

catch (Exception ex)

{

return new Response<Object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. En la view model de profile colocamos como campo el apiservice para que se puedan usar en todas las clases se debe de inyectar
2. y el código quedaría asi para la view model de profile el método save completo

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(

"Error",

response.Message,

"Aceptar");

return;

}

Settings.Owner = JsonConvert.SerializeObject(Owner);

await App.Current.MainPage.DisplayAlert(

"Ok",

"Usuario actualizado correctamente",

"Aceptar");

}

1. Probar que si modifique los datos de usuario desde app
2. En el apiservice en la interface agregar método para cambiar contraseña

Task<Response<object>> ChangePasswordAsync(

string urlBase,

string servicePrefix,

string controller,

ChangePasswordRequest changePasswordRequest,

string tokenType,

string accessToken);

}

1. E implementar el método anterior en la respectiva clase

public async Task<Response<object>> ChangePasswordAsync(

string urlBase,

string servicePrefix,

string controller,

ChangePasswordRequest changePasswordRequest,

string tokenType,

string accessToken)

{

try

{

var request = JsonConvert.SerializeObject(changePasswordRequest);

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

var client = new HttpClient

{

BaseAddress = new Uri(urlBase)

};

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

var url = $"{servicePrefix}{controller}";

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

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

var obj = JsonConvert.DeserializeObject<Response<object>>(answer);

return obj;

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Se crea pagina de cambiar contraseña a la cual se le binda el tiutlo y se le agrega el bacgroun definido en el diccionario de recursos y llamar el activity indicator y el resto del código queda asi

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All"

Padding="10">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="Contraseña actual"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

IsPassword="True"

Placeholder="Ingrese su contraseña actual"

Text="{Binding CurrentPassword}"/>

<Label

Grid.Column="0"

Grid.Row="1"

Text="Nueva contraseña"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

IsPassword="True"

Placeholder="Ingrese la nueva contraseña."

Text="{Binding NewPassword}"/>

<Label

Grid.Column="0"

Grid.Row="2"

Text="Confirmar contraseña"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="2"

IsPassword="True"

Placeholder="Ingrese confirmación de contraseña"

Text="{Binding PasswordConfirm}"/>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand"/>

<Button

Command="{Binding ChangePasswordCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Cambiar contraseña"

VerticalOptions="EndAndExpand"/>

</StackLayout>

<busyindicator:SfBusyIndicator

BackgroundColor="{StaticResource colorDanger}"

TextColor="{StaticResource colorFontInverse}"

AnimationType="DoubleCircle"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

HorizontalOptions="Center"

IsVisible="{Binding IsRunning}"

Title="Cambiando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

1. La view model de cambiar contraseña quedaría asi

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 = "Cambiar contraseña";

}

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(

"Error",

"Usted debe ingresar su contraseña actual",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(NewPassword) || NewPassword?.Length < 6)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debe ingresar una nueva contraseña de al menos seis caracteres.",

"Aceptar");

return false;

}

if (string.IsNullOrEmpty(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Debes ingresar la confirmación de la contraseña",

"Aceptar");

return false;

}

if (!NewPassword.Equals(PasswordConfirm))

{

await App.Current.MainPage.DisplayAlert(

"Error",

"La nueva contraseña y la confirmación no son iguales",

"Aceptar");

return false;

}

return true;

}

}

1. En la profile view model se debe agregar el nuevo comando para cambiar pass crear su atributo privado, propiedad publica y el método privado y el inavigator Service crearlo como campo para que se pueda usar en toda la clase

public class ProfilePageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IApiService \_apiService;

//atributos privados

private bool \_isRunning;

private bool \_isEnabled;

private OwnerResponse \_owner;

private DelegateCommand \_saveCommand;

private DelegateCommand \_changePasswordCommand;

public ProfilePageViewModel(

INavigationService navigationService,

IApiService apiService) : base(navigationService)

{

Title = "Modificar perfil";

IsEnabled = true;

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

\_navigationService = navigationService;

\_apiService = apiService;

}

public DelegateCommand ChangePasswordCommand => \_changePasswordCommand ?? (\_changePasswordCommand = new DelegateCommand(ChangePassword));

public DelegateCommand SaveCommand => \_saveCommand ?? (\_saveCommand = new DelegateCommand(Save));

….

private async void ChangePassword()

{

await \_navigationService.NavigateAsync("ChangePasswordPage");

}

…

1. Probar la navegación a la nueva pagina
2. Modificar el changePassViewModel el método para cambiar la pass se completa quedando asi

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(

"Error",

response.Message,

"Aceptar");

return;

}

await App.Current.MainPage.DisplayAlert(

"Ok",

response.Message,

"Aceptar");

await \_navigationService.GoBackAsync();

}

1. Se va adicionar un nuevo icono a la barra de hmientas se descarga de asset studio se guarda en la carpeta resources
2. En la petspage se agrega asi

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

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

xmlns:prism="http://prismlibrary.com"

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

prism:ViewModelLocator.AutowireViewModel="True"

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

IconImageSource="ic\_pet"

BackgroundColor="{StaticResource colorBackground}"

Title="{Binding Title}">

<ContentPage.ToolbarItems>

<ToolbarItem IconImageSource="ic\_action\_add\_circle" Command="{Binding AddPetCommand}"/>

</ContentPage.ToolbarItems>

1. En la Pets page view model

private DelegateCommand \_addPetCommand;

…

public DelegateCommand AddPetCommand => \_addPetCommand ?? (\_addPetCommand = new DelegateCommand(AddPet));

…

private async void AddPet()

{

await \_navigationService.NavigateAsync("EditPage");

}

1. Se agrega nueva pagina de tipo content para adicionar y editar mascotas se lecoloca el binding del titulo lo del activity indicator y lo del diccionario de recursos para que tenga fondo del mismo color azul y en la view model cambiar de donde hereda
2. Prueba navegabilidad
3. En el diccionario de recursos app.xaml crear nevo estilo para botón de borrar

<Color x:Key="ColorDanger">#D9042B</Color>

<Style x:Key="DangerButton" TargetType="Button">

<Setter Property="BackgroundColor" Value="{StaticResource ColorDanger}" />

<Setter Property="TextColor" Value="{StaticResource ColorFontInverse}" />

</Style>

1. La pagina para editar la amscota queda asi

<?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="http://prismlibrary.com"

xmlns:busyindicator="clr-namespace:Syncfusion.SfBusyIndicator.XForms;assembly=Syncfusion.SfBusyIndicator.XForms"

prism:ViewModelLocator.AutowireViewModel="True"

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

BackgroundColor="{StaticResource colorBackground}"

Title="{Binding Title}">

<StackLayout

Padding="10">

<ScrollView>

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All">

<Image

HeightRequest="150"

Source="{Binding ImageSource}">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Label

FontSize="Micro"

HorizontalOptions="Center"

Text="Cambiar imagen"/>

<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="Nombre"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="0"

Placeholder="Ingrese nombre de la mascota"

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

<Label

Grid.Column="0"

Grid.Row="1"

Text="Raza"

VerticalOptions="Center"/>

<Entry

Grid.Column="1"

Grid.Row="1"

Placeholder="Ingrese raza de la mascota"

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

<Label

Grid.Column="0"

Grid.Row="2"

Text="Tipo de mascota"

VerticalOptions="Center"/>

<Picker

Grid.Column="1"

Grid.Row="2"

ItemDisplayBinding="{Binding Name}"

ItemsSource="{Binding PetTypes}"

SelectedItem="{Binding PetType}"

Title="Seleccione tipo de mascota"/>

<Label

Grid.Column="0"

Grid.Row="3"

Text="Fecha de nacimiento"

VerticalOptions="Center"/>

<DatePicker

Grid.Column="1"

Grid.Row="3"

Date="{Binding Pet.Born}"/>

<Label

Grid.Column="0"

Grid.Row="4"

Text="Comentarios"

VerticalOptions="Center"/>

<Editor

Grid.Column="1"

Grid.Row="4"

HeightRequest="80"

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

</Grid>

</StackLayout>

<busyindicator:SfBusyIndicator

BackgroundColor="{StaticResource colorDanger}"

TextColor="{StaticResource colorFontInverse}"

AnimationType="DoubleCircle"

AbsoluteLayout.LayoutBounds=".5,.5,.5,.5"

AbsoluteLayout.LayoutFlags="All"

HorizontalOptions="Center"

IsVisible="{Binding IsRunning}"

Title="Guardando..."

VerticalOptions="Center"

ViewBoxWidth="80"

ViewBoxHeight="80" />

</AbsoluteLayout>

</ScrollView>

<StackLayout

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Button

Command="{Binding SaveCommand}"

IsEnabled="{Binding IsEnabled}"

Text="Guardar"/>

<Button

Command="{Binding DeleteCommand}"

IsEnabled="{Binding IsEnabled}"

IsVisible="{Binding IsEdit}"

Style="{StaticResource DangerButton}"

Text="Borrar"/>

</StackLayout>

</StackLayout>

</ContentPage>

1. Probar
2. En la edit page quedria asi

public class EditPageViewModel : ViewModelBase

{

private PetResponse \_pet;

private ImageSource \_imageSource;

private bool \_isRunning;

private bool \_isEnabled;

private bool \_isEdit;

public EditPageViewModel(

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 = "Editar Mascota";

}

else

{

Pet = new PetResponse { Born = DateTime.Today };

ImageSource = "NOIMAGE";

IsEdit = false;

Title = "Nueva Mascota";

}

1. Probar
2. Crear una clase en el proyecto common en los modelos que se va a llamar el petresponse

public class PetTypeResponse

{

public int Id { get; set; }

public string Name { get; set; }

}

1. En la interface apiservice crear método genérico para obtener listas

Task<Response<object>> GetListAsync<T>(

string urlBase,

string servicePrefix,

string controller,

string tokenType,

string accessToken);

1. Y la implementación en la respectiva clase

public async Task<Response<object>> 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<object>

{

IsSuccess = false,

Message = result,

};

}

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

return new Response<object>

{

IsSuccess = true,

Result = list

};

}

catch (Exception ex)

{

return new Response<object>

{

IsSuccess = false,

Message = ex.Message

};

}

}

1. En la editpage inyectar el apiservice y el navigator Service colocarlo como cmpo para toda la clase
2. Otros cambios en la editpage agregando propiedad privada y publica de tipo obdrvsble collection para pet types y de tipo response para pet type

private ObservableCollection<PetTypeResponse> \_petTypes;

private PetTypeResponse \_petType;

.....

public ObservableCollection<PetTypeResponse> PetTypes

{

get => \_petTypes;

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

}

public PetTypeResponse PetType

{

get => \_petType;

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

}

….

LoadPetTypesAsync();

}

private async void LoadPetTypesAsync()

{

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", "Revisa tu conexión a internet.", "Aceptar");

await \_navigationService.GoBackAsync();

return;

}

var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

var response = await \_apiService.GetListAsync<PetTypeResponse>(

url,

"/api",

"/PetTypes",

"bearer",

token.Token);

IsEnabled = true;

if (!response.IsSuccess)

{

await App.Current.MainPage.DisplayAlert(

"Error",

"Obteniendo los tipos de mascota. Por favor trate despues.",

"Aceptar");

await \_navigationService.GoBackAsync();

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. Probar
2. En la page de editarpet hacer cambios

-scrollview meterlo dentro de otro stacklayout

- El padding que tenia el stack layout que había colocárselo al nuevo stack layout que es el queda como ppal y después del cierre del scollview colocar botón para editar

<Button

Command="{Binding EditPetCommand}"

Text="Editar mascota"

VerticalOptions="EndAndExpand"/>

1. Probar como se ve el botón que haga scroll todo menos el botón
2. La petPageViewModel queda asi

public class PetPageViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private PetResponse \_pet;

private DelegateCommand \_editPetCommand;

public PetPageViewModel(INavigationService navigationService) : base(navigationService)

{

Title = "Detalles";

\_navigationService = navigationService;

}

public DelegateCommand EditPetCommand => \_editPetCommand ?? (\_editPetCommand = new DelegateCommand(EditPetAsync));

public PetResponse Pet

{

get => \_pet;

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

}

public override void OnNavigatedTo(INavigationParameters parameters)

{

base.OnNavigatedTo(parameters);

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

}

private async void EditPetAsync()

{

var parameters = new NavigationParameters

{

{ "pet", Pet }

};

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

}

}

1. En el editPage se debe colocar esta definición para cambiar como se muestra la imagen paraque funcione para crear y editar

xmlns:ffimageloading="clr-namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"

…..

<AbsoluteLayout>

<StackLayout

AbsoluteLayout.LayoutBounds="0,0,1,1"

AbsoluteLayout.LayoutFlags="All">

<ffimageloading:CachedImage

Aspect="AspectFit"

Source="{Binding ImageSource}"

LoadingPlaceholder="LoaderImage"

ErrorPlaceholder="ErrorImage"

CacheDuration="50"

RetryCount="3"

RetryDelay="600"

DownsampleToViewSize="True">

<ffimageloading:CachedImage.GestureRecognizers>

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

</ffimageloading:CachedImage.GestureRecognizers>

</ffimageloading:CachedImage>

1. Probar y commit
2. Adicionar plugin a todos los proyectos de movilidad xam.plugin.Meda
3. En el archivo mainActivity adicionar líneas para que se de los permisos de acceder a la cámara para tomar fotos

using Android.App;

using Android.Content.PM;

using Android.OS;

using Prism;

using Prism.Ioc;

using Syncfusion.SfBusyIndicator.XForms.Droid;

using Android.Runtime;

using Plugin.CurrentActivity;

using Plugin.Permissions;

namespace MyVet.Prism.Droid

{

[Activity(

Label = "MyVet",

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);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

new SfBusyIndicatorRenderer();

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. Dentro de properties en el proyecto android en el archivo 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.companyname.appname" 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. Dentro de la carpeta resources crear carpeta **xml y dentro de ella crear un archivo de tipo xml llamado file\_paths.xml dentro del cual se copia lo siguiente**

<?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. Para ios se modifica el info.plist se abre con editor de texto y al final se agregan estas líneas

<key>UILaunchStoryboardName</key>

<string>LaunchScreen</string>

<key>UIMainStoryboardFile</key>

<string>LaunchScreen</string>

<key>XSAppIconAssets</key>

<string>Assets.xcassets/AppIcon.appiconset</string>

<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>

</dict>

</plist>

1. Ahora la lógica en el editpage se agrega los siguiente propiedades privadas

private MediaFile \_file;

private DelegateCommand \_changeImageCommand;

1. Se agregan las propiedades publicas

public DelegateCommand ChangeImageCommand => \_changeImageCommand ?? (\_changeImageCommand = new DelegateCommand(ChangeImageAsync));

1. Se agrega el método privado al final para cambiar imagen

private async void ChangeImageAsync()

{

await CrossMedia.Current.Initialize();

var source = await Application.Current.MainPage.DisplayActionSheet(

"De donde quieres obtener la foto?",

"Cancelar",

null,

"Desde galeria",

"Desde camara");

if (source == "Cancelar")

{

\_file = null;

return;

}

if (source == "Desde camara")

{

\_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. Probar
2. Actualizar desde los xamarin from packages los proyectos de movilidad se hace proyecto por proyecto en el Prism, android e iosal mas actual
3. Actualizar desde solución el amarin de maps
4. Para obtener latitud y longitud agregar otro nuget al proyecto common y a los de movilidad se adiciona desde solución llamado xam.plugin.geolocator
5. Ingresar a <https://developers.google.com/maps/?hl=es-419> donde se matricula una llave para maoas google te da la llave que se debe colocar en el proyecto para el caso se usa la del video
6. En el android manifest se adiciona lo siguiente para ubicar la posición del usuario debe dar los permisos correspondientes

<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" />

1. Para el caso de ios en el archivo info.plist adicionar al final esto

<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. En el main activity agrega la siguiente línea para inicializar el render sw xMrin form maps

global::Xamarin.Forms.Forms.Init(this, bundle);

Xamarin.FormsMaps.Init(this, bundle);

FFImageLoading.Forms.Platform.CachedImageRenderer.Init(true);

1. Para ios se haría en el appdelegate

global::Xamarin.Forms.Forms.Init();

Xamarin.FormsMaps.Init();

FFImageLoading.Forms.Platform.CachedImageRenderer.Init();

1. En el proyecto de Prism hay una mappage donde se hace lo siguiente agregar referencia en el namespace y agregar lo que ve el usuario

<?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="http://prismlibrary.com"

xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

prism:ViewModelLocator.AutowireViewModel="True"

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

BackgroundColor="{StaticResource colorBackground}"

Title="{Binding Title}">

<StackLayout>

<maps:Map

x:Name="MyMap"

IsShowingUser="true"

MapType="Street"/>

</StackLayout>

</ContentPage>

1. Probar
2. Adiconar en la carpeta services una interface llamada **IGeolocatorService con su respectiva clase**
3. **La interface tiene lo siguiente**

public interface IGeolocatorService

{

double Latitude { get; set; }

double Longitude { get; set; }

Task GetLocationAsync();

}

1. Yla implementación se hace en la clase que quedaría asi

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. Se debe inyecta en el inyector de dependencias en el archivo app.xml.cs agregar

protected override void RegisterTypes(IContainerRegistry containerRegistry)

{

containerRegistry.Register<IApiService, ApiService>();

containerRegistry.Register<IGeolocatorService, GeolocatorService>();

containerRegistry.RegisterForNavigation<NavigationPage>();

containerRegistry.RegisterForNavigation<LoginPage, LoginPageViewModel>();

…..

1. Modifica el map.xml.cs

using MyVet.Common.Models.Services;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

namespace MyVet.Prism.Views

{

public partial class MapPage : ContentPage

{

private readonly IGeolocatorService \_geolocatorService;

public MapPage(IGeolocatorService geolocatorService)

{

InitializeComponent();

\_geolocatorService = geolocatorService;

MoveMapToCurrentPositionAsync();

}

private async void MoveMapToCurrentPositionAsync()

{

await \_geolocatorService.GetLocationAsync();

var position = new Position(

\_geolocatorService.Latitude,

\_geolocatorService.Longitude);

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(

position,

Distance.FromKilometers(.5)));

}

}

}

1. Adicionar en la mappage esta línea de código

<StackLayout>

<maps:Map

x:Name="MyMap"

IsShowingUser="True"

MapType="Street"/>

</StackLayout>

1. probar
2. Puntos en el mapa modificar la entidad usuario en el backend agregando dos propiedades latitud y longitud
3. Como se hace cambio en bd se debe agregar migración si esta no se puede hacer se borra y actualiza la bd y se debe hacer lo mismo con la de azure
4. Actualizar la vista index de owner agregando esos nuevos campos después del campo teléfono

<th>@Html.DisplayNameFor(model => model.User.Latitude) </th>

<th>@Html.DisplayNameFor(model => model.User.Longitude) </th>

<td>@Html.DisplayFor(modelItem => item.User.Latitude) </td>

<td>@Html.DisplayFor(modelItem => item.User.Longitude) </td>

1. Modificar el editUserViewModel

*[Display(Name = "Latitud")]*

*public double Latitude { get; set; }*

*[Display(Name = "Longitud")]*

*public double Longitude { get; set; }*

1. *En la vista parcial de usuarios se debe adicionar los dos atributos*

<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. Modificar controlador de owner la acción de créate post

*public async Task<IActionResult> Create(AddUserViewModel model)*

*{*

*if (ModelState.IsValid)*

*{*

*//crear usuario*

*var user = new User*

*{*

*Address = model.Address,*

*Document = model.Document,*

*Email = model.Username,*

*FirstName = model.FirstName,*

*LastName = model.LastName,*

*PhoneNumber = model.PhoneNumber,*

*UserName = model.Username*

*Latitude = model.Latitude,*

*Longitude = model.Longitude*

*};*

1. *Adicionar esos campos en el controlador de owner las acciones edit get y post*

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

{

if (id == null)

{

return NotFound();

}

var owner = await \_context.Owners

.Include(o => o.User)

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

if (owner == null)

{

return NotFound();

}

var model = 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

};

return View(model);

}

// POST: Owners/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

*public async Task<IActionResult> Edit(EditUserViewModel model)*

*{*

*if (ModelState.IsValid)*

*{*

*var owner = await \_context.Owners*

*.Include(o => o.User)*

*.FirstOrDefaultAsync(o => o.Id == model.Id);*

*owner.User.Document = model.Document;*

*owner.User.FirstName = model.FirstName;*

*owner.User.LastName = model.LastName;*

*owner.User.Address = model.Address;*

*owner.User.PhoneNumber = model.PhoneNumber;*

*owner.User.Latitude = model.Latitude;*

*owner.User.Longitude = model.Longitude;*

*await \_userHelper.UpdateUserAsync(owner.User);*

*return RedirectToAction(nameof(Index));*

*}*

*return View(model);*

*}*

1. Probar, posiciones cerca de latitud 6 and longitud -75.

PARTE 89 add latitud y longitud in backend PAG 434

Plugin xam.plugin.media 4.0.1.5

ANDROID ASSET

MAKEAPPICON

COLOR.ADOBE

ICON FINDER

XAMARIN FORM NOMBRE PROPIEDAD

Syncfusion.xamaryn versión 17.2.0.49 es la video y la mia es 17.3.0.34 y la actual al 23 de dic 17.4.0.39

Prism.dryloc.forms versión 7.1.0.4.431 es la video y la mia es 7.2.0.1367 y la actual al 23 de dic 7.2.0.1422

Xam.geolocator es el 4.5.0.6 el del video y el actual al 23 de dic tb

Xamarin form map del video es 4.2.0.848062

Xamarin.form 4.2.0.709249 la del video y la mia es la misma y la actual es 4.4.0.991265 se actualizara a la que actualicen en el video que es la 4.2.0848062 se actualzo al mas actual