**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













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

PARTE 48 self registration of user PAG 149