Fantasy

With .NET & Blazor

Juan Carlos Zuluaga

2024, Semestre 2

# Generales

## Links de interes

* Videos del ejemplo del proyecto anterior: <https://www.youtube.com/playlist?list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2>
* Los videos de este proyecto en: <https://www.youtube.com/playlist?list=PLuEZQoW9bRnQZftBlazC2AEHlzBVtiqhU>
* La URL del repositorio como lo llevo en clase es: <https://github.com/Zulu55/Fantasy>
* La URL del repositorio terminando (o como lo llevo preparadas las clases) lo puede encontrar en: <https://github.com/Zulu55/FantasyPrep>
* URL de la aplicación terminada: <https://fantasyzulu.azurewebsites.net>.

## Fantasy, ejemplo del 2024-II

Sistema donde diferentes grupos de amigos pueden hacer predicciones sobre torneos de fútbol. En Colombia, se le llama “Polla”; en Argentina, “Prode”; y en Estados Unidos, “Fantasy”. La idea es que cualquier número de torneos de fútbol, como la Copa América, el Mundial, la Eurocopa, la Champions League, o el Torneo Colombiano, entre otros, pueda ser registrado. Los grupos de amigos podrán formar sus propias “Pollas” y realizar predicciones sobre los partidos. Una vez completados los partidos y aplicadas las reglas de negocio, el participante que acumule más puntos ganará la “Fantasy”, la “Polla” o como se le denomine en su país.

Cada usuario podrá crear múltiples grupos o unirse a grupos existentes para participar en cualquier torneo de fútbol habilitado por el administrador. El creador del grupo será considerado el administrador de dicho grupo y podrá definir las condiciones de reparto del premio, por ejemplo:

* 70% para el primer puesto.
* 20% para el segundo puesto.
* 10% para el tercer puesto.

El administrador también tendrá la facultad de activar o desactivar a los miembros de su grupo. Por ejemplo, si un miembro no ha pagado el valor correspondiente a la polla, el administrador podrá desactivarlo, y un usuario inactivo no podrá ingresar predicciones.

La forma de obtener puntos es la siguiente:

* 5 puntos por acertar el ganador o predecir un empate.
* 2 puntos adicionales por predecir los goles del equipo local.
* 2 puntos adicionales por predecir los goles del equipo visitante.
* 1 punto por acertar la diferencia de goles.

El máximo de puntos por partido será 10, en caso de acertar el resultado perfecto. Ten en cuenta las siguientes consideraciones:

* Solo se podrán ingresar o modificar predicciones hasta 10 minutos antes de iniciar un partido.
* El resultado se basará en los 90 minutos de tiempo reglamentario más las adiciones. No se tendrán en cuenta los goles en tiempos extra o penales.
* Los partidos de segunda ronda en adelante otorgarán el doble de puntos.

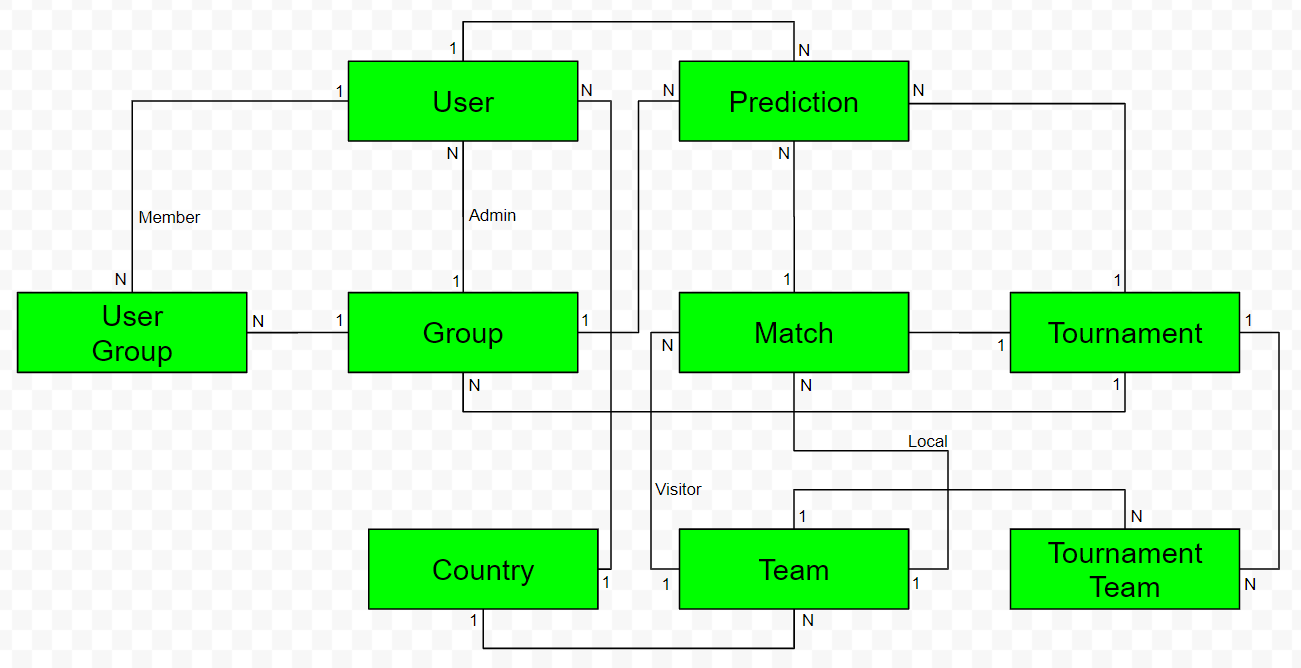
## Matriz de funcionalidad

En en siguiente vídeo encontrará la explicación de esta parte, así como indicaciones de como instalar el ambiente de desarrollo:

| **Funcionalidad** | **Administrador** | **Usuario** | **Anónimo** |
| --- | --- | --- | --- |
| Ingresar al sistema con email y contraseña | X | X |  |
| Editar datos de usuario (incluyendo foto de perfi) | X | X |  |
| Cambiar contraseña | X | X |  |
| Recuperar contraseña, si el usuario olvida la contraseña se le enviará un correo con un token para poder recuperar contraseña. | X | X |  |
| Administrar usuarios, el decir podrá ver todos los usuarios del sistema y crear nuevos administradores | X |  |  |
| Administrar: paises, equipos, torneos y partidos | X |  |  |
| Cerrar los partidos luego de terminados para que el sistema haga los calculos de puntos obtenidos en cada grupo | X |  |  |
| Podra crear grupos de amigos para crear una nueva “polla” inscrita un torneo |  | X |  |
| El usuario que cree el grupo será conocido como el “administrador” del grupo y podrá marcar si los miembros ya pagaron o no pagaron el valor apostado en la polla |  | X |  |
| Ingresar/modificar las predicciones hasta 10 minutos antes de empezar el partido |  | X |  |
| Ver las predicciones que hicieron todos los miembros en un grupo cuando falten 10 minutos para empezar el partido o despues |  | X | X |
| Ver tabla de posiciones en la polla |  | X | X |

## Diagrama Entidad Relación

Vamos a crear un sencillo sistema de ventas que va a utilizar el siguiente modelo de datos:



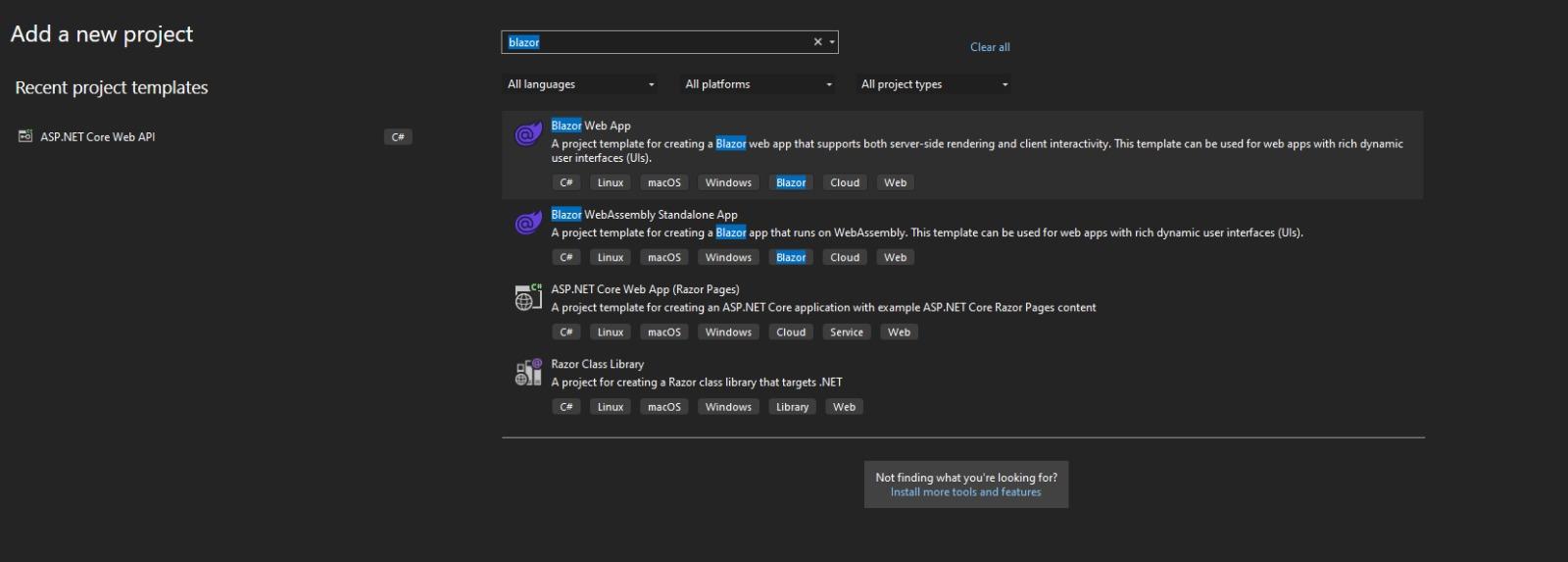
## Estructura básica de proyecto



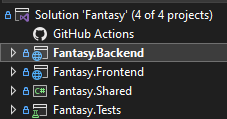
Vamos a crear esta estructura en Visual Studio (asegurese de poner todos los proyectos rn :

* Una solución en blanco llamada **Fantasy**.
* A la solución le agregamos un proyecto tipo: **ASP.NET Core Frontend Backend**, llamado **Fantasy.Backend**. (Backend)
* A la solución le agregamos un proyecto tipo: **Blazor FrontendAssembly App**, llamado **Fantasy. Frontend**. (Frontend)
* A la solución le agregamos un proyecto tipo: **Class Library**, llamado **Fantasy.Shared**.
* A la solución le agregamos un proyecto tipo: **MS Test**, llamado **Fantasy.Tests**.

**Nota**: en algunas instalaciones de Visual Studio no lo puedes ver como **Blazor FrontendAssembly App** sino como **Blazor WebAssembly Standalone App**, usa esta.



Debe quedar algo como esto:



Hacemos el primer commit en nuestro repositorio.

## Creando la base de datos con Entity Framework



Recomiendo buscar y leer documentación sobre Code First y Database First. En este curso trabajaremos con EF Code First, si están interesados en conocer más sobre EF Database First acá les dejo un enlace:<https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

1. Agregamos la extesnión al Visual Studio **Code Maid**, para mantener nuestro código, limpiamente formateado.
2. Empecemos creando la carpeta **Entites** y dentro de esta la entidad **Country** en el proyecto **Shared**:

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.Entities;

public class Country

{

public int Id { get; set; }

[MaxLength(100)]

[Required]

public string Name { get; set; } = null!;

}

1. Actualizar Nuggets del proyecto **Backend**.
2. En el proyecto **Backend** creamos la carpeta **Data** y dentro de esta la clase **DataContext**:

using Fantasy.Shared.Entities;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Data;

public class DataContext : DbContext

{

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

{

}

public DbSet<Country> Countries { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Country>().HasIndex(x => x.Name).IsUnique();

}

}

1. Configurar el string de conexión en el **appsettings.json** del proyecto **Backend**:

{

"ConnectionStrings": {

"DockerConnection": "Data Source=.;Initial Catalog=Fantasy;User ID={Your user};Password={Your password};Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",

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

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

**Nota:** dejo los 2 string de conexión para que use el que más le convenga en el vídeo de clase explico mejor cual utilizar en cada caso.

1. Agregar/verificar los paquetes al proyecto **Backend**:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

1. Configurar la inyección del data context en el **Program** del proyecto **Backend**:

builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=LocalConnection"));

var app = builder.Build();

1. Correr los comandos:

add-migration InitialDb

update-database

1. Hacemos nuestro segundo **Commit**.

## Creando el primer controlador

1. En el proyecto **Backend** en la carpeta **Controllers** creamos la clase **CountriesController**:

using Fantasy.Backend.Data;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class CountriesController : ControllerBase

{

private readonly DataContext \_context;

public CountriesController(DataContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAsync()

{

return Ok(await \_context.Countries.ToListAsync());

}

[HttpGet("{id}")]

public async Task<IActionResult> GetAsync(int id)

{

var country = await \_context.Countries.FirstOrDefaultAsync(c => c.Id == id);

if (country == null)

{

return NotFound();

}

return Ok(country);

}

[HttpPost]

public async Task<IActionResult> PostAsync(Country country)

{

\_context.Add(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteAsync(int id)

{

var country = await \_context.Countries.FirstOrDefaultAsync(c => c.Id == id);

if (country == null)

{

return NotFound();

}

\_context.Remove(country);

await \_context.SaveChangesAsync();

return NoContent();

}

[HttpPut]

public async Task<IActionResult> PutAsync(Country country)

{

\_context.Update(country);

await \_context.SaveChangesAsync();

return Ok(country);

}

}

1. Agregamos estas líneas al **Program** del proyecto **Backend** para habilitar su consumo:

app.MapControllers();

app.UseCors(x => x

.AllowAnyMethod()

.AllowAnyHeader()

.SetIsOriginAllowed(origin => true)

.AllowCredentials());

app.Run();

1. Borramos las clases de **WeatherForecast**.
2. Probamos la creación y listado de paises por el **swagger** y por **Postman**.
3. Hacemos el **commit** de lo que llevamos.

# CRUDs Parte I

## Creando nuestros primeros componentes en Blazor

1. Le agregamos este nuget al **Fronted**: **System.Net.Http**.
2. Ahora vamos listar y crear países por la interfaz Frontend. Primero configuramos en el proyecto  **Frontend** la dirección por la cual sale nuestra **Backend**:

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7232") });

1. En el proyecto  **Frontend** creamos a carpeta **Repositories** y dentro de esta creamos la clase **HttpResponseWrapper** con el siguiente código:

using System.Net;

namespace Fantasy.Frontend.Repositories;

public class HttpResponseWrapper<T>

{

public HttpResponseWrapper(T? response, bool error, HttpResponseMessage httpResponseMessage)

{

Response = response;

Error = error;

HttpResponseMessage = httpResponseMessage;

}

public T? Response { get; }

public bool Error { get; }

public HttpResponseMessage HttpResponseMessage { get; }

public async Task<string?> GetErrorMessageAsync()

{

if (!Error)

{

return null;

}

var statusCode = HttpResponseMessage.StatusCode;

if (statusCode == HttpStatusCode.NotFound)

{

return "Recurso no encontrado.";

}

if (statusCode == HttpStatusCode.BadRequest)

{

return await HttpResponseMessage.Content.ReadAsStringAsync();

}

if (statusCode == HttpStatusCode.Unauthorized)

{

return "Tienes que estar logueado para ejecutar esta operación.";

}

if (statusCode == HttpStatusCode.Forbidden)

{

return "No tienes permisos para hacer esta operación.";

}

return "Ha ocurrido un error inesperado.";

}

}

1. En la misma carpeta creamos la interfaz **IRepository**:

namespace Fantasy.Frontend.Repositories;

public interface IRepository

{

Task<HttpResponseWrapper<T>> GetAsync<T>(string url);

Task<HttpResponseWrapper<object>> PostAsync<T>(string url, T model);

Task<HttpResponseWrapper<TActionResponse>> PostAsync<T, TActionResponse>(string url, T model);

}

1. En la misma carpeta creamos la case **Repository**:

using System.Text;

using System.Text.Json;

namespace Fantasy.Frontend.Repositories;

public class Repository : IRepository

{

private readonly HttpClient \_httpClient;

private JsonSerializerOptions \_jsonDefaultOptions => new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true,

};

public Repository(HttpClient httpClient)

{

\_httpClient = httpClient;

}

public async Task<HttpResponseWrapper<T>> GetAsync<T>(string url)

{

var responseHttp = await \_httpClient.GetAsync(url);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<T>(responseHttp);

return new HttpResponseWrapper<T>(response, false, responseHttp);

}

return new HttpResponseWrapper<T>(default, true, responseHttp);

}

public async Task<HttpResponseWrapper<object>> PostAsync<T>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);

}

public async Task<HttpResponseWrapper<TActionResponse>> PostAsync<T, TActionResponse>(string url, T model)

{

var messageJSON = JsonSerializer.Serialize(model);

var messageContet = new StringContent(messageJSON, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PostAsync(url, messageContet);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TActionResponse>(responseHttp);

return new HttpResponseWrapper<TActionResponse>(response, false, responseHttp);

}

return new HttpResponseWrapper<TActionResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);

}

private async Task<T> UnserializeAnswer<T>(HttpResponseMessage responseHttp)

{

var response = await responseHttp.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<T>(response, \_jsonDefaultOptions)!;

}

}

8

1. En el Program del proyecto Frontend configuramos la inyección del **Repository**:

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7230/") });

builder.Services.AddScoped<IRepository, Repository>();

await builder.Build().RunAsync();

1. En el proyecto del **Frontend** creamos las carpeta **Shared** y dentro de esta, creamos el componente genérico **GenericList.razor.cs**:

using Microsoft.AspNetCore.Components;

namespace Fantasy.Frontend.Shared;

public partial class GenericList<Titem>

{

[Parameter] public RenderFragment? Loading { get; set; }

[Parameter] public RenderFragment? NoRecords { get; set; }

[EditorRequired, Parameter] public RenderFragment Body { get; set; } = null!;

[EditorRequired, Parameter] public List<Titem> MyList { get; set; } = null!;

}

1. Y modifcamos el **GenericList.razor**:

@typeparam Titem

@if (MyList is null)

{

@if (Loading is null)

{

<div class="d-flex justify-content-center align-items-center">

<img src="https://www.wpfaster.org/wp-content/uploads/2013/06/loading-gif.gif" width="200" height="200"/>

</div>

}

else

{

@Loading

}

}

else if (MyList.Count == 0)

{

@if (NoRecords is null)

{

<p>No hay registros para mostrar...</p>

}

else

{

@NoRecords

}

}

else

{

@Body

}

1. En el proyecto **Frontend** Dentro de **Pages** creamos la carpeta **Countries** y dentro de esta carpeta creamos la página **CountriesIndex.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountriesIndex

{

[Inject] private IRepository Repository { get; set; } = null!;

private List<Country>? Countries { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHppt = await Repository.GetAsync<List<Country>>("api/countries");

Countries = responseHppt.Response!;

}

}

1. Y modificamos el **CountriesIndex.razor**:

<h3>Paises</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/countries/create">Nuevo País</a>

</div>

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>País</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

<a class="btn btn-warning">Editar</a>

<button class="btn btn-danger">Borrar</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

1. Cambiamos el menú en el **NavMenu.razor.cs**:

namespace Fantasy.Frontend.Layout;

public partial class NavMenu

{

private bool collapseNavMenu = true;

private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

private void ToggleNavMenu()

{

collapseNavMenu = !collapseNavMenu;

}

}

1. Cambiamos el menú en el **NavMenu.razor**:

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="">Fantasy</a>

<button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

</div>

</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">

<nav class="flex-column">

<div class="nav-item px-3">

<NavLink class="nav-link" href="" Match="NavLinkMatch.All">

<span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Inicio

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="countries">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span> Paises

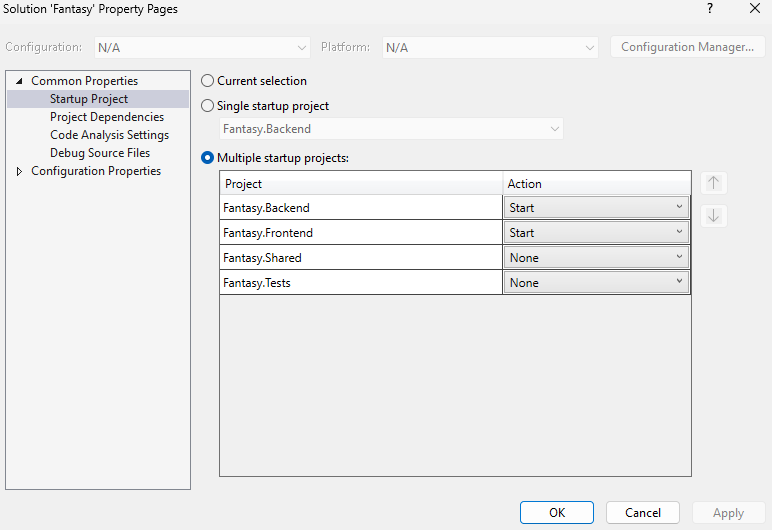
</NavLink>

</div>

</nav>

</div>

1. Configuramos nuestro proyecto para que inicie al mismo tiempo el proyecto **Backend** y el proyecto  **Frontend**:



1. Probamos y hacemos nuestro commit.

## Soportando múltiples idiomas

1. Al **Frontend** agregamos el Nuget: **Microsoft.Extensions.Localization**
2. En el program del Frontend configuramos el servicio de localización:

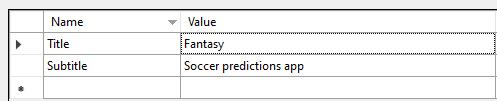
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7232") });

builder.Services.AddScoped<IRepository, Repository>();

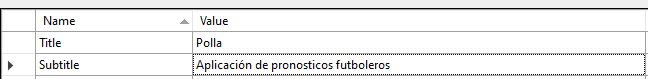
builder.Services.AddLocalization();

await builder.Build().RunAsync();

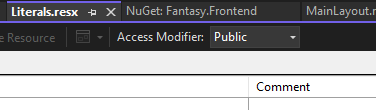
1. En el proyecto **Shared** creamos la carpeta **Resources** y dentro de esta el archivo de recursos por defecto **Literals.resx** y tantos archivos de idioma como deseemos, por ejemplo **Literals.es.resx**
2. Agregamos este par de literales para el archivo de recursos en Ingles:



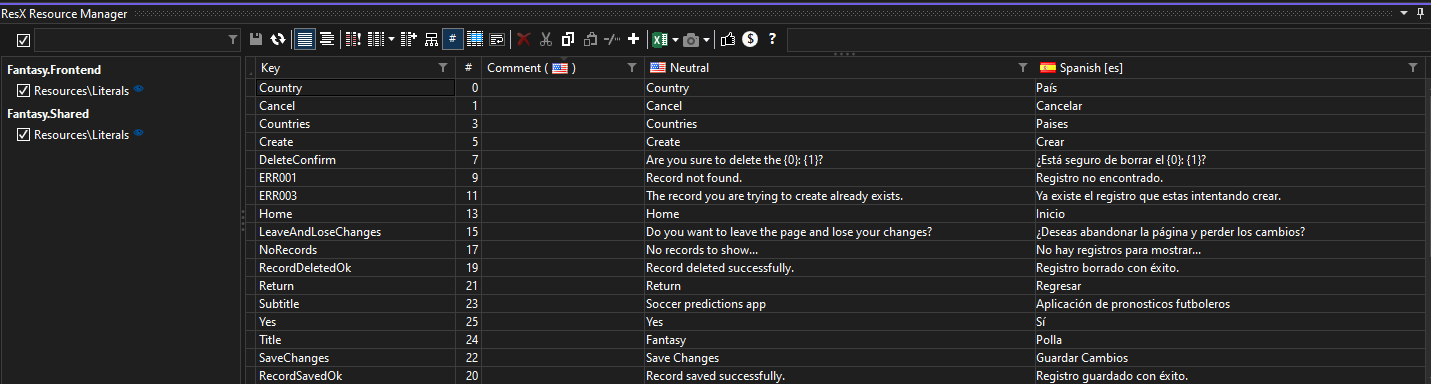
1. Agregamos este par de literales para el archivo de recursos en Español:



1. Y nos aseguramos que el archivo de recursos por defecto, tenga el modificador de acceso público:



1. **Nota**: para hacer una mejor administración de los archivos de recursos, siguiero instalarle al Visual Studio la extensión **ResXManager**.



1. Agregamos el archivo **Home.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages;

public partial class Home

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

}

1. Y modificamos el **Home.razor**:

@page "/"

<PageTitle>Home</PageTitle>

<h1>@Localizer["Title"]</h1>

<h2>@Localizer["Subtitle"]</h2>

1. Probamos y hacemos el commit.
2. Agregamos estos literales:

| About | About | Acerca de |
| --- | --- | --- |
| Countries | Countries | Paises |
| Delete | Delete | Borrar |
| Edit | Edit | Editar |
| Home | Home | Inicio |
| Image | Image | Imagén |
| NoRecords | No records to show... | No hay registros para mostrar... |

1. Creamos el **MainLayout.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Layout;

public partial class MainLayout

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

}

1. Modificamos el **MainLayout.razor**:

<a href="https://learn.microsoft.com/aspnet/core/" target="\_blank">@Localizer["About"]</a>

1. Modificamos el **NavMenu.razor.cs**:

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

1. Modificamos el **NavMenu.razor**:

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="">@Localizer["Title"]</a>

<button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">

<span class="navbar-toggler-icon"></span>

</button>

</div>

</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">

<nav class="flex-column">

<div class="nav-item px-3">

<NavLink class="nav-link" href="" Match="NavLinkMatch.All">

<span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span>@Localizer["Home"]

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="countries">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span>@Localizer["Countries"]

</NavLink>

</div>

</nav>

</div>

1. Modificamos el **CountriesIndex.razor.cs**:

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

1. Modificamos el **CountriesIndex.razor**:

@page "/countries"

<h3>@Localizer["Countries"]</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/countries/create">@Localizer["New"] @Localizer["Country"]</a>

</div>

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>@Localizer["Country"]</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

<a class="btn btn-warning">@Localizer["Edit"]</a>

<button class="btn btn-danger">@Localizer["Delete"]</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

1. Modificamos el **GenericList.razor.cs**:

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

1. Modificamos el **GenericList.razor**:

@if (NoRecords is null)

{

<p>@Localizer["NoRecords"]</p>

}

else

{

@NoRecords

}

1. Probamos y hacemos el commit.

## Completando las acciones de crear, editar y borrar países

1. Agregamos estos métodos a la interfaz **IRepository**.

Task<HttpResponseWrapper<object>> DeleteAsync(string url);

Task<HttpResponseWrapper<object>> PutAsync<T>(string url, T model);

Task<HttpResponseWrapper<TActionResponse>> PutAsync<T, TActionResponse>(string url, T model);

1. Y los implementamos la clase **Repository** (antes renombramos el **UnserializeAnswer** a **UnserializeAnswerAsync** que nos habia quedado mal).

public async Task<HttpResponseWrapper<object>> DeleteAsync(string url)

{

var responseHttp = await \_httpClient.DeleteAsync(url);

return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);

}

public async Task<HttpResponseWrapper<object>> PutAsync<T>(string url, T model)

{

var messageJson = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJson, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);

}

public async Task<HttpResponseWrapper<TActionResponse>> PutAsync<T, TActionResponse>(string url, T model)

{

var messageJson = JsonSerializer.Serialize(model);

var messageContent = new StringContent(messageJson, Encoding.UTF8, "application/json");

var responseHttp = await \_httpClient.PutAsync(url, messageContent);

if (responseHttp.IsSuccessStatusCode)

{

var response = await UnserializeAnswer<TActionResponse>(responseHttp);

return new HttpResponseWrapper<TActionResponse>(response, false, responseHttp);

}

return new HttpResponseWrapper<TActionResponse>(default, true, responseHttp);

}

1. Vamos agregarle al proyecto  **Frontend** el paquete **CurrieTechnologies.Razor.SweetAlert2**, que nos va a servir para mostrar modeles de alertas muy bonitos.
2. Vamos a la página de Sweet Alert 2 ([Basaingeal/Razor.SweetAlert2: A Razor class library for interacting with SweetAlert2 (github.com)](https://github.com/Basaingeal/Razor.SweetAlert2) y copiamos el script que debemos de agregar al **index.html** que está en el **wwwroot** de nuestro proyecto  **Frontend**.

<script src="\_framework/blazor. Frontendassembly.js"></script>

<script src="\_content/CurrieTechnologies.Razor.SweetAlert2/sweetAlert2.min.js"></script>

</body>

1. En el proyecto  **Frontend** configuramos la inyección del servicio de alertas:

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7232") });

builder.Services.AddScoped<IRepository, Repository>();

builder.Services.AddLocalization();

builder.Services.AddSweetAlert2();

1. Creamos el componente gérico **Loading.razor**:

<div class="d-flex justify-content-center align-items-center">

<img src="https://www.wpfaster.org/wp-content/uploads/2013/06/loading-gif.gif" width="200" height="200" />

</div>

1. Modificamos el **GenericList.razor**:

@if (Loading is null)

{

<Loading/>

}

1. Agregamos estos líterales:

| Confirmation | Confirmation | Confirmación |
| --- | --- | --- |
| LeaveAndLoseChanges | Do you want to leave the page and lose your changes? | ¿Deseas abandonar la página y perder los cambios? |
| SaveChanges | Save Changes | Guardar Cambios |
| Return | Return | Regresar |
| Create | Create | Crear |
| Cancel | Cancel | Cancelar |
| RecordSavedOk | Record saved successfully. | Registro guardado con éxito. |
| Error | Error | Error |
| DeleteConfirm | Are you sure to delete the {0}: {1}? | ¿Está seguro de borrar el {0}: {1}? |
| RecordDeletedOk | Record deleted successfully. | Registro borrado con éxito. |
| Yes | Yes | Sí |

1. En la carpeta **Countries** agregar el componente **CountryForm.razor** y **CountryForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountryForm

{

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(Country);

}

[EditorRequired, Parameter] public Country Country { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Modificamos el **CountryForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<div class="mb-3">

<label>@Localizer["Country"]:</label>

<div>

<InputText class="form-control" @bind-Value="@Country.Name" />

<ValidationMessage For="@(() => Country.Name)" />

</div>

</div>

<button class="btn btn-primary" type="submit">@Localizer["SaveChanges"]</button>

<button class="btn btn-success" @onclick="ReturnAction">@Localizer["Return"]</button>

</EditForm>

1. En la carpeta **Countries** agregar el componente **CountryCreate.razor** y **CountryCreate.razor.cs:**

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountryCreate

{

private CountryForm? countryForm;

private Country country = new();

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/countries", country);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], message);

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordCreatedOk"]);

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/countries");

}

}

1. Modificamos el **CountryCreate.razor**:

@page "/countries/create"

<h3>Crear País</h3>

<CountryForm @ref="countryForm" Country="country" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Probamos la creación de países por interfaz. **Asegurate que luego de correr el proyecto, presionar Ctrl + F5, para que te tome los cambios**.
2. Hacemos el commit.
3. Ahora creamos el componente **CountryEdit.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountryEdit

{

private Country? country;

private CountryForm? countryForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Country>($"api/countries/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("countries");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], messageError, SweetAlertIcon.Error);

}

}

else

{

country = responseHttp.Response;

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/countries", country);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], mensajeError, SweetAlertIcon.Error);

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordSavedOk"]);

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("countries");

}

}

1. Modificamos el **CountryEdit.razor**:

@page "/countries/edit/{Id:int}"

<h3>@Localizer["Edit"] @Localizer["Country"]</h3>

@if (country is null)

{

<Loading />

}

else

{

<CountryForm @ref="countryForm" Country="country" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

1. Luego modificamos el componente **CountriesIndex.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountriesIndex

{

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private List<Country>? Countries { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHppt = await Repository.GetAsync<List<Country>>("api/countries");

if (responseHppt.Error)

{

var message = await responseHppt.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], message, SweetAlertIcon.Error);

return;

}

Countries = responseHppt.Response!;

}

private async Task DeleteAsync(Country country)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = string.Format(Localizer["DeleteConfirm"], Localizer["Country"], country.Name),

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"]

});

var confirm = string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"api/countries/{country.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], mensajeError, SweetAlertIcon.Error);

}

return;

}

await LoadAsync();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000,

ConfirmButtonText = Localizer["Yes"]

});

toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordDeletedOk"]);

}

}

1. Luego modificamos el componente **CountriesIndex.razor**:

<a class="btn btn-warning" href="/countries/edit/@country.Id">@Localizer["Edit"]</a>

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(country))>@Localizer["Delete"]</button>

1. Y probamos la edición y eliminación de países por interfaz. No olvides hacer el **commit**.

## Creando controladores genéricos y solucionando el problema de registros duplicados

Material complementario: <https://www.netmentor.es/entrada/repository-pattern>



1. Creamos la entidad **Team**:

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.Entities;

public class Team

{

public int Id { get; set; }

[MaxLength(100)]

[Required]

public string Name { get; set; } = null!;

public string? Image { get; set; }

public Country Country { get; set; } = null!;

public int CountryId { get; set; }

}

1. Modificamos la entidad **Country**:

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.Entities;

public class Country

{

public int Id { get; set; }

[MaxLength(100)]

[Required]

public string Name { get; set; } = null!;

public ICollection<Team>? Teams { get; set; }

public int TeamsCount => Teams == null ? 0 : Teams.Count;

}

1. Modificamos el **DataContext**:

using Fantasy.Shared.Entities;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Data;

public class DataContext : DbContext

{

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

{

}

public DbSet<Country> Countries { get; set; }

public DbSet<Team> Teams { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Country>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<Team>().HasIndex(x => new { x.CountryId, x.Name }).IsUnique();

DisableCascadingDelete(modelBuilder);

}

private void DisableCascadingDelete(ModelBuilder modelBuilder)

{

var relationships = modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys());

foreach (var relationship in relationships)

{

relationship.DeleteBehavior = DeleteBehavior.Restrict;

}

}

}

1. Agregamos la migración y actualizamos la BD.
2. En **Shared** creamos la carpeta **Responses** y dentro de esta la clase **ActionResponse**:

namespace Fantasy.Shared.Responses;

public class ActionResponse<T>

{

public bool WasSuccess { get; set; }

public string? Message { get; set; }

public T? Result { get; set; }

}

1. En **Backend** creamos la carpeta **Repositories/Interfaces** y dentro de esta la interfaz **IGenericRepository**:

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface IGenericRepository<T> where T : class

{

Task<ActionResponse<T>> GetAsync(int id);

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<T>> AddAsync(T entity);

Task<ActionResponse<T>> DeleteAsync(int id);

Task<ActionResponse<T>> UpdateAsync(T entity);

}

1. Creanis la carpeta **UnitsOfWork/Interfaces** y dentro de esta creamos la interfaz **IGenericUnitOfWork**:

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IGenericUnitOfWork<T> where T : class

{

Task<ActionResponse<IEnumerable<T>>> GetAsync();

Task<ActionResponse<T>> AddAsync(T model);

Task<ActionResponse<T>> UpdateAsync(T model);

Task<ActionResponse<T>> DeleteAsync(int id);

Task<ActionResponse<T>> GetAsync(int id);

}

1. Agregar estos literales:

| ERR001 | Record not found. | Registro no encontrado. |
| --- | --- | --- |
| ERR002 | Cannot be deleted because it has related records. | No se puede borrar, porque tiene registros relacionados. |
| ERR003 | The record you are trying to create already exists. | Ya existe el registro que estas intentando crear. |

1. En **Backend** creamos la carpeta **Repositories/Implementations** y dentro de esta la clase **GenericRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class GenericRepository<T> : IGenericRepository<T> where T : class

{

private readonly DataContext \_context;

private readonly DbSet<T> \_entity;

public GenericRepository(DataContext context)

{

\_context = context;

\_entity = context.Set<T>();

}

public virtual async Task<ActionResponse<T>> AddAsync(T entity)

{

\_context.Add(entity);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

Result = entity

};

}

catch (DbUpdateException)

{

return DbUpdateExceptionActionResponse();

}

catch (Exception exception)

{

return ExceptionActionResponse(exception);

}

}

public virtual async Task<ActionResponse<T>> DeleteAsync(int id)

{

var row = await \_entity.FindAsync(id);

if (row == null)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "ERR001"

};

}

try

{

\_entity.Remove(row);

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

};

}

catch

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "ERR002"

};

}

}

public virtual async Task<ActionResponse<T>> GetAsync(int id)

{

var row = await \_entity.FindAsync(id);

if (row != null)

{

return new ActionResponse<T>

{

WasSuccess = true,

Result = row

};

}

return new ActionResponse<T>

{

WasSuccess = false,

Message = "ERR001"

};

}

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync()

{

return new ActionResponse<IEnumerable<T>>

{

WasSuccess = true,

Result = await \_entity.ToListAsync()

};

}

public virtual async Task<ActionResponse<T>> UpdateAsync(T entity)

{

try

{

\_context.Update(entity);

await \_context.SaveChangesAsync();

return new ActionResponse<T>

{

WasSuccess = true,

Result = entity

};

}

catch (DbUpdateException)

{

return DbUpdateExceptionActionResponse();

}

catch (Exception exception)

{

return ExceptionActionResponse(exception);

}

}

private ActionResponse<T> ExceptionActionResponse(Exception exception)

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = exception.Message

};

}

private ActionResponse<T> DbUpdateExceptionActionResponse()

{

return new ActionResponse<T>

{

WasSuccess = false,

Message = "ERR003"

};

}

}

1. En **Backend** creamos la carpeta **UnitsOfWork/Implementations** y dentro de esta la clase **GenericUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class GenericUnitOfWork<T> : IGenericUnitOfWork<T> where T : class

{

private readonly IGenericRepository<T> \_repository;

public GenericUnitOfWork(IGenericRepository<T> repository)

{

\_repository = repository;

}

public virtual async Task<ActionResponse<T>> AddAsync(T model) => await \_repository.AddAsync(model);

public virtual async Task<ActionResponse<T>> DeleteAsync(int id) => await \_repository.DeleteAsync(id);

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync() => await \_repository.GetAsync();

public virtual async Task<ActionResponse<T>> GetAsync(int id) => await \_repository.GetAsync(id);

public virtual async Task<ActionResponse<T>> UpdateAsync(T model) => await \_repository.UpdateAsync(model);

}

1. En **Backend** en la carpeta **Controllers** y dentro de esta la clase **GenericController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

public class GenericController<T> : Controller where T : class

{

private readonly IGenericUnitOfWork<T> \_unitOfWork;

public GenericController(IGenericUnitOfWork<T> unitOfWork)

{

\_unitOfWork = unitOfWork;

}

[HttpGet]

public virtual async Task<IActionResult> GetAsync()

{

var action = await \_unitOfWork.GetAsync();

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public virtual async Task<IActionResult> GetAsync(int id)

{

var action = await \_unitOfWork.GetAsync(id);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return NotFound();

}

[HttpPost]

public virtual async Task<IActionResult> PostAsync(T model)

{

var action = await \_unitOfWork.AddAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut]

public virtual async Task<IActionResult> PutAsync(T model)

{

var action = await \_unitOfWork.UpdateAsync(model);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpDelete("{id}")]

public virtual async Task<IActionResult> DeleteAsync(int id)

{

var action = await \_unitOfWork.DeleteAsync(id);

if (action.WasSuccess)

{

return NoContent();

}

return BadRequest(action.Message);

}

}

1. Configuramos las inyecciones en el **Program** del **Backend**:

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=DockerConnection"));

builder.Services.AddScoped(typeof(IGenericUnitOfWork<>), typeof(GenericUnitOfWork<>));

builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

1. Reemplazamos el **CountriesController** por esto:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class CountriesController : GenericController<Country>

{

public CountriesController(IGenericUnitOfWork<Country> unit) : base(unit)

{

}

}

1. Probamos que la aplicación siga funcionando como si no hubieramos echo nada.
2. En teoría podríamos crear el **TeamsController** con el siguiente código:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class TeamsController : GenericController<Team>

{

public TeamsController(IGenericUnitOfWork<Team> unitOfWork) : base(unitOfWork)

{

}

}

1. Probamos y ya tenemos la plantilla, pero no podemos agregar el equipo porque la Entidad ya no nos sirve como modelo. Lo corregimos un par de títulos más adelante.

## Solucionando problema de las validaciones de campos

Podemos observar que los mensajes de validación del formulario de países no se está mostrando correctamente según el idioma. Para solucionar esto, sigamos estos pasos.

1. Cremos estos literales:

| Country | Country | País |
| --- | --- | --- |
| Team | Team | Equipo |
| MaxLength | Field {0} cannot be longer than {1} characters. | El campo {0} no puede tener más de {1} caracteres. |
| RequiredField | Field {0} is required. | El campo {0} es obligatorio. |

1. Modificamos la entidad **Country**:

[Display(Name = "Country", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

1. Probamos.
2. También tenemos un problemita con el mensaje de los registros duplicados, vamos a corregirlo. Modificamos el **CountryCreate.razor.cs**:

await SweetAlertService.FireAsync(Localizer["Error"], Localizer[message!], SweetAlertIcon.Error);

1. Y modificamos el **CountryEdit.razor.cs**:

await SweetAlertService.FireAsync(Localizer["Error"], Localizer[mensajeError!], SweetAlertIcon.Error);

1. Aunque no tenemos el cRUD de **Team** coloquemos las data annotations en **Team** antes que se nos olvide:

[Display(Name = "Team", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

1. Probamos y hacemos el commit.

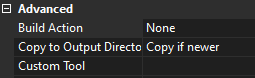
## Configurando un repositorio para trabajo en equipo, resolver conflictos y obtener estadísticas de código

Este tema está explicado en los vídeos:

* <https://www.youtube.com/watch?v=GtN6N11qSgA&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=16>
* <https://www.youtube.com/watch?v=5ycMPV5qGMg&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=17>
* <https://www.youtube.com/watch?v=-_rCQGG7lEs&list=PLuEZQoW9bRnRBThyGs208ZMrCYBRTvIg2&index=18>

## Adicionando un Seeder a la base de datos

1. Agregue l archivo **Countries.sql** en la carpeta **Data** del **Backend** y asegúrese de que tenga la propiedad “**Copy if newer**”:



1. Creamos en el proyecto **Backend** dentro de la carpeta **Data** la clase **SeedDb**:

using Fantasy.Shared.Entities;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Data;

public class SeedDb

{

private readonly DataContext \_context;

public SeedDb(DataContext context)

{

\_context = context;

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckTeamsAsync();

}

private async Task CheckCountriesAsync()

{

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

{

var countriesSQLScript = File.ReadAllText("Data\\Countries.sql");

await \_context.Database.ExecuteSqlRawAsync(countriesSQLScript);

}

}

private async Task CheckTeamsAsync()

{

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

{

foreach (var country in \_context.Countries)

{

\_context.Teams.Add(new Team { Name = country.Name, Country = country! });

if (country.Name == "Colombia")

{

\_context.Teams.Add(new Team { Name = "Medellín", Country = country! });

\_context.Teams.Add(new Team { Name = "Nacional", Country = country! });

\_context.Teams.Add(new Team { Name = "Millonarios", Country = country! });

\_context.Teams.Add(new Team { Name = "Junior", Country = country! });

}

}

await \_context.SaveChangesAsync();

}

}

}

1. Luego modificamos el **Program** del proyecto **Backend** para llamar el alimentador de la BD:

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=LocalConnection"));

builder.Services.AddTransient<SeedDb>();

var app = builder.Build();

SeedData(app);

void SeedData(WebApplication app)

{

var scopedFactory = app.Services.GetService<IServiceScopeFactory>();

using var scope = scopedFactory!.CreateScope();

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

service!.SeedAsync().Wait();

}

1. Borramos la base de datos con el comando **drop-database**.
2. Probamos y hacemos el **commit**.

## Creando el CountriesRepository

Cuando ya necesitamos algo particular de un repositorio, como son los datos de país y equipos relacionados, ya no nos sirve ‘al 100%’ el repositorio genérico y debemos de hacer implementaciones específicas que se ajusten a nuestras necesidades. Entonces procedamos con los siguientes pasos:

1. Para evitar la redundancia ciclica en la respuesta de los JSON vamos a agregar la siguiente línea en el **Program** del **Backend**:

builder.Services.AddControllers().AddJsonOptions(x => x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);

1. Creamos el **ICountriesRepository**:

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface ICountriesRepository

{

Task<ActionResponse<Country>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

Task<IEnumerable<Country>> GetComboAsync();

}

1. Creamos el **CountriesRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class CountriesRepository : GenericRepository<Country>, ICountriesRepository

{

private readonly DataContext \_context;

public CountriesRepository(DataContext context) : base(context)

{

\_context = context;

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync()

{

var countries = await \_context.Countries

.Include(c => c.Teams)

.ToListAsync();

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = countries

};

}

public override async Task<ActionResponse<Country>> GetAsync(int id)

{

var country = await \_context.Countries

.Include(c => c.Teams)

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

if (country == null)

{

return new ActionResponse<Country>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Country>

{

WasSuccess = true,

Result = country

};

}

public async Task<IEnumerable<Country>> GetComboAsync()

{

return await \_context.Countries

.OrderBy(c => c.Name)

.ToListAsync();

}

}

1. Creamos el **ICountriesUnitOfWork**:

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface ICountriesUnitOfWork

{

Task<ActionResponse<Country>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Country>>> GetAsync();

Task<IEnumerable<Country>> GetComboAsync();

}

1. Creamos el **CountriesUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class CountriesUnitOfWork : GenericUnitOfWork<Country>, ICountriesUnitOfWork

{

private readonly ICountriesRepository \_countriesRepository;

public CountriesUnitOfWork(IGenericRepository<Country> repository, ICountriesRepository countriesRepository) : base(repository)

{

\_countriesRepository = countriesRepository;

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync() => await \_countriesRepository.GetAsync();

public override async Task<ActionResponse<Country>> GetAsync(int id) => await \_countriesRepository.GetAsync(id);

public async Task<IEnumerable<Country>> GetComboAsync() => await \_countriesRepository.GetComboAsync();

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped(typeof(IGenericUnitOfWork<>), typeof(GenericUnitOfWork<>));

builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

var app = builder.Build();

1. Modificamos el **CountriesController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class CountriesController : GenericController<Country>

{

private readonly ICountriesUnitOfWork \_countriesUnitOfWork;

public CountriesController(IGenericUnitOfWork<Country> unit, ICountriesUnitOfWork countriesUnitOfWork) : base(unit)

{

\_countriesUnitOfWork = countriesUnitOfWork;

}

[HttpGet("combo")]

public async Task<IActionResult> GetComboAsync()

{

return Ok(await \_countriesUnitOfWork.GetComboAsync());

}

[HttpGet]

public override async Task<IActionResult> GetAsync()

{

var response = await \_countriesUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_countriesUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

}

1. Probamos los cambios por el **swagger**.
2. Modificamos el **CountriesIndex.razor**:

<GenericList MyList="Countries">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>@Localizer["Country"]</th>

<th># @Localizer["Teams"]</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var country in Countries!)

{

<tr>

<td>

@country.Name

</td>

<td>

@country.TeamsCount

</td>

<td>

<a class="btn btn-warning" href="/countries/edit/@country.Id">@Localizer["Edit"]</a>

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(country))>@Localizer["Delete"]</button>

</td>

</tr>

}

</tbody>

</table>

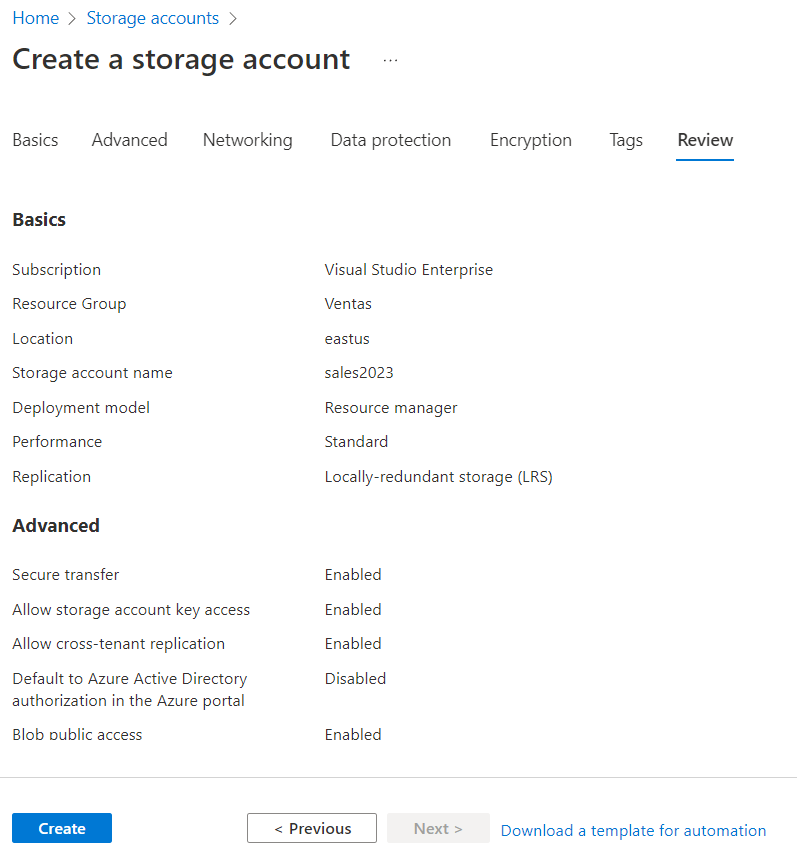
</Body>

</GenericList>

1. Probamos y hacemos el **commit**.

## Creando el TeamsRepository

1. Los equipos necesitan almacenar la imagén/bandera/escudo del equipo, y estas imágenes las vamos a almacenar en Azure, por eso vamos a crear el **blob** en **Azure**:



1. Luego que termine copiamos el connection string que necesitamos para acceder a nuestro blob storage. Agregamos ese connection string en el **appsettings** de nuestro proyecto **Backend**:

"ConnectionStrings": {

"DockerConnection": "Data Source=.;Initial Catalog=Orders;User ID={Your user};Password={Your password};Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False",

"LocalConnection": "Server=(localdb)\\MSSQLLocalDB;Database=Orders2023;Trusted\_Connection=True;MultipleActiveResultSets=true",

"AzureStorage": "{Your azure connection string}"

},

1. En el proyecto **Backend** en la carpeta **Helpers** creamos la interfaz **IFileStorage**:

namespace Orders.Backend.Helpers

{

public interface IFileStorage

{

Task<string> SaveFileAsync(byte[] content, string extention, string containerName);

Task RemoveFileAsync(string path, string containerName);

}

}

1. En la misma carpeta creamos la implementation **FileStorage**:

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

namespace Orders.Backend.Helpers

{

public class FileStorage : IFileStorage

{

private readonly string \_connectionString;

public FileStorage(IConfiguration configuration)

{

\_connectionString = configuration.GetConnectionString("AzureStorage")!;

}

public async Task RemoveFileAsync(string path, string containerName)

{

var client = new BlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

var fileName = Path.GetFileName(path);

var blob = client.GetBlobClient(fileName);

await blob.DeleteIfExistsAsync();

}

public async Task<string> SaveFileAsync(byte[] content, string extention, string containerName)

{

var client = new BlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

client.SetAccessPolicy(PublicAccessType.Blob);

var fileName = $"{Guid.NewGuid()}{extention}";

var blob = client.GetBlobClient(fileName);

using (var ms = new MemoryStream(content))

{

await blob.UploadAsync(ms);

}

return blob.Uri.ToString();

}

}

}

1. Configuramos la nueva inyección en el **Program** del **Backend**:

builder.Services.AddScoped<IFileStorage, FileStorage>();

1. En el **Shared** agregamos el líteral para **Image**.
2. En el **Shared** creamos la carpeta **DTOs** y dentro de esta el **TeamDTO**:

using Fantasy.Shared.Resources;

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.DTOs;

public class TeamDTO

{

public int Id { get; set; }

[Display(Name = "Team", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

[Display(Name = "Image", ResourceType = typeof(Literals))]

public string? Image { get; set; }

[Display(Name = "Country", ResourceType = typeof(Literals))]

public int CountryId { get; set; }

}

1. Adicionamos los siguientes literales:

| ERR004 | The country Id is not valid. | El código del país no es válido. |
| --- | --- | --- |
| ERR005 | The team Id is not valid. | El código del equipo no es válido. |

1. Creamos el **ITeamsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface ITeamsRepository

{

Task<IEnumerable<Team>> GetComboAsync(int countryId);

Task<ActionResponse<Team>> AddAsync(TeamDTO teamDTO);

Task<ActionResponse<Team>> UpdateAsync(TeamDTO teamDTO);

Task<ActionResponse<Team>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Team>>> GetAsync();

}

1. Creamos el **TeamsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entites;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations

{

public class TeamsRepository : GenericRepository<Team>, ITeamsRepository

{

private readonly DataContext \_context;

private readonly IFileStorage \_fileStorage;

public TeamsRepository(DataContext context, IFileStorage fileStorage) : base(context)

{

\_context = context;

\_fileStorage = fileStorage;

}

public override async Task<ActionResponse<IEnumerable<Team>>> GetAsync()

{

var teams = await \_context.Teams

.Include(x => x.Country)

.OrderBy(x => x.Name)

.ToListAsync();

return new ActionResponse<IEnumerable<Team>>

{

WasSuccess = true,

Result = teams

};

}

public override async Task<ActionResponse<Team>> GetAsync(int id)

{

var team = await \_context.Teams

.Include(x => x.Country)

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

if (team == null)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Team>

{

WasSuccess = true,

Result = team

};

}

public async Task<ActionResponse<Team>> AddAsync(TeamDTO teamDTO)

{

var country = await \_context.Countries.FindAsync(teamDTO.CountryId);

if (country == null)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR004"

};

}

var team = new Team

{

Country = country,

Name = teamDTO.Name,

};

if (!string.IsNullOrEmpty(teamDTO.Image))

{

var imageBase64 = Convert.FromBase64String(teamDTO.Image!);

team.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "teams");

}

\_context.Add(team);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Team>

{

WasSuccess = true,

Result = team

};

}

catch (DbUpdateException)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<IEnumerable<Team>> GetComboAsync(int countryId)

{

return await \_context.Teams

.Where(x => x.CountryId == countryId)

.OrderBy(x => x.Name)

.ToListAsync();

}

public async Task<ActionResponse<Team>> UpdateAsync(TeamDTO teamDTO)

{

var currentTeam = await \_context.Teams.FindAsync(teamDTO.Id);

if (currentTeam == null)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR005"

};

}

var country = await \_context.Countries.FindAsync(teamDTO.CountryId);

if (country == null)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR004"

};

}

if (!string.IsNullOrEmpty(teamDTO.Image))

{

var imageBase64 = Convert.FromBase64String(teamDTO.Image!);

currentTeam.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "teams");

}

currentTeam.Country = country;

currentTeam.Name = teamDTO.Name;

\_context.Update(currentTeam);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Team>

{

WasSuccess = true,

Result = currentTeam

};

}

catch (DbUpdateException)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Team>

{

WasSuccess = false,

Message = exception.Message

};

}

}

}

}

1. Creamos el **ITeamsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface ITeamsUnitOfWork

{

Task<IEnumerable<Team>> GetComboAsync(int countryId);

Task<ActionResponse<Team>> AddAsync(TeamDTO teamDTO);

Task<ActionResponse<Team>> UpdateAsync(TeamDTO teamDTO);

Task<ActionResponse<Team>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Team>>> GetAsync();

}

1. Creamos el **TeamsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class TeamsUnitOfWork : GenericUnitOfWork<Team>, ITeamsUnitOfWork

{

private readonly ITeamsRepository \_teamsRepository;

public TeamsUnitOfWork(IGenericRepository<Team> repository, ITeamsRepository teamsRepository) : base(repository)

{

\_teamsRepository = teamsRepository;

}

public async Task<ActionResponse<Team>> AddAsync(TeamDTO teamDTO) => await \_teamsRepository.AddAsync(teamDTO);

public async Task<IEnumerable<Team>> GetComboAsync(int countryId) => await \_teamsRepository.GetComboAsync(countryId);

public async Task<ActionResponse<Team>> UpdateAsync(TeamDTO teamDTO) => await \_teamsRepository.UpdateAsync(teamDTO);

public override async Task<ActionResponse<Team>> GetAsync(int id) => await \_teamsRepository.GetAsync(id);

public override async Task<ActionResponse<IEnumerable<Team>>> GetAsync() => await \_teamsRepository.GetAsync();

}

1. Registramos las nuevas inyecciones:

builder.Services.AddScoped<ITeamsRepository, TeamsRepository>();

builder.Services.AddScoped<ITeamsUnitOfWork, TeamsUnitOfWork>();

1. Creamos el **TeamsController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class TeamsController : GenericController<Team>

{

private readonly ITeamsUnitOfWork \_teamsUnitOfWork;

public TeamsController(IGenericUnitOfWork<Team> unitOfWork, ITeamsUnitOfWork teamsUnitOfWork) : base(unitOfWork)

{

\_teamsUnitOfWork = teamsUnitOfWork;

}

[HttpGet]

public override async Task<IActionResult> GetAsync()

{

var response = await \_teamsUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_teamsUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpGet("combo/{countryId:int}")]

public async Task<IActionResult> GetComboAsync(int countryId)

{

return Ok(await \_teamsUnitOfWork.GetComboAsync(countryId));

}

[HttpPost]

public async Task<IActionResult> PostAsync(TeamDTO teamDTO)

{

var action = await \_teamsUnitOfWork.AddAsync(teamDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut]

public async Task<IActionResult> PutAsync(TeamDTO teamDTO)

{

var action = await \_teamsUnitOfWork.UpdateAsync(teamDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Probamos.
2. Corregimos el problema de los anottation del HTTP:

[HttpPost("full")]

…

[HttpPut("full")]

1. Probamos y hacemos el **commit**.

## Index de Teams

1. Creamos un literal para **Teams**.
2. Dentro de **Pages** creamos la carpeta **Teams** y dentro de esta creamos el **TeamsIndex.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamsIndex

{

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private List<Team>? Teams { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

var responseHppt = await Repository.GetAsync<List<Team>>("api/teams");

if (responseHppt.Error)

{

var message = await responseHppt.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], message, SweetAlertIcon.Error);

return;

}

Teams = responseHppt.Response!;

}

private async Task DeleteAsync(Team team)

{

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = string.Format(Localizer["DeleteConfirm"], Localizer["Team"], team.Name),

Icon = SweetAlertIcon.Question,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"]

});

var confirm = string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"api/teams/{team.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/");

}

else

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], mensajeError, SweetAlertIcon.Error);

}

return;

}

await LoadAsync();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000,

ConfirmButtonText = Localizer["Yes"]

});

toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordDeletedOk"]);

}

}

1. Modificamos el **TeamsIndex.razor**:

@page "/teams"

<h3>@Localizer["Teams"]</h3>

<div class="mb-3">

<a class="btn btn-primary" href="/teams/create">@Localizer["New"] @Localizer["Team"]</a>

</div>

<GenericList MyList="Teams">

<Body>

<table class="table table-striped">

<thead>

<tr>

<th>@Localizer["Team"]</th>

<th>@Localizer["Image"]</th>

<th>@Localizer["Country"]</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var team in Teams!)

{

<tr>

<td>

@team.Name

</td>

<td>

<img src="@team.Image" style="width:80px;" />

</td>

<td>

@team.Country.Name

</td>

<td>

<a class="btn btn-warning" href="/teams/edit/@team.Id">@Localizer["Edit"]</a>

<button class="btn btn-danger" @onclick=@(() => DeleteAsync(team))>@Localizer["Delete"]</button>

</td>

</tr>

}

</tbody>

</table>

</Body>

</GenericList>

1. Modificamos el **NavMenu.razor**:

<div class="nav-item px-3">

<NavLink class="nav-link" href="countries">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span>@Localizer["Countries"]

</NavLink>

</div>

<div class="nav-item px-3">

<NavLink class="nav-link" href="teams">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span>@Localizer["Teams"]

</NavLink>

</div>

1. Probamos.
2. Vamos a crear la carpeta **images** en **wwwroot** y vamos a colocar una imagén para cuando no hay imagén. No olvidar poner la propiedad de **Copy to Output Directory** en **Copy if newer**.
3. Agregamos esta propiedad a **Teams**:

public string ImageFull => string.IsNullOrEmpty(Image) ? "/images/NoImage.png" : Image;

1. Modificamos el **TeamIndex.razor**:

<td><img src="@team.ImageFull" style="width:90px;height:60px;" /></td>

1. Probamos y hacemos el **commit**.

## Creando y Editando equipos

1. Agregamos los siguientes literales:

| SelectFile | Select a File… | Seleccione un archivo... |
| --- | --- | --- |
| Search | Search… | Buscar… |

1. Para poder capturar imágenes creamos el componente genérico **InputImg.razor** y **InputImg.razor.cs**:

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Shared;

public partial class InputImg

{

private string? imageBase64;

private string? fileName;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public string? Label { get; set; }

[Parameter] public string? ImageURL { get; set; }

[Parameter] public EventCallback<string> ImageSelected { get; set; }

protected override void OnParametersSet()

{

base.OnParametersSet();

if (string.IsNullOrWhiteSpace(Label))

{

Label = Localizer["Image"];

}

}

private async Task OnChange(InputFileChangeEventArgs e)

{

var file = e.File;

if (file != null)

{

fileName = file.Name;

var arrBytes = new byte[file.Size];

await file.OpenReadStream().ReadAsync(arrBytes);

imageBase64 = Convert.ToBase64String(arrBytes);

ImageURL = null;

await ImageSelected.InvokeAsync(imageBase64);

StateHasChanged();

}

}

}

1. Modificamos el **InputImg.razor**:

<div class="mb-3">

<label class="form-label">@Label</label>

<div class="input-group">

<input type="text"

class="form-control"

placeholder="@Localizer["SelectFile"]"

readonly="readonly"

value="@fileName" />

<label class="input-group-btn">

<span class="btn btn-primary">

@Localizer["Search"]<InputFile OnChange="OnChange" class="d-none" accept=".jpg,.jpeg,.png" />

</span>

</label>

</div>

</div>

<div>

@if (imageBase64 is not null)

{

<div>

<div style="margin: 10px">

<img src="data:image/jpeg;base64,@imageBase64" style="height:200px;" />

</div>

</div>

}

@if (ImageURL is not null && ImageURL.StartsWith("http"))

{

<div>

<div style="margin: 10px">

<img src="@ImageURL" style="height:200px;" />

</div>

</div>

}

</div>

1. Creamos este literal:

| SelectACountry | -- Select a Country --- | -- Selecciona un País --- |
| --- | --- | --- |

1. Creamos el **TeamForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamForm

{

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(TeamDTO);

}

[EditorRequired, Parameter] public TeamDTO TeamDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private List<Country>? countries;

private string? imageUrl;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

protected override void OnParametersSet()

{

base.OnParametersSet();

if (!string.IsNullOrEmpty(TeamDTO.Image))

{

imageUrl = TeamDTO.Image;

TeamDTO.Image = null;

}

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Country>>("/api/countries/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);

return;

}

countries = responseHttp.Response;

}

private void ImageSelected(string imagenBase64)

{

TeamDTO.Image = imagenBase64;

imageUrl = null;

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Creamos el **TeamForm.cs**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<div class="mb-3">

<label>@Localizer["Team"]:</label>

<div>

<InputText class="form-control" @bind-Value="@TeamDTO.Name" />

<ValidationMessage For="@(() => TeamDTO.Name)" />

</div>

</div>

<div class="mb-3">

<label>@Localizer["Country"]:</label>

<div>

<select class="form-select" @bind="TeamDTO.CountryId">

<option value="0">@Localizer["SelectACountry"]</option>

@if (countries is not null)

{

@foreach (var country in countries)

{

<option value="@country.Id">@country.Name</option>

}

}

</select>

<ValidationMessage For="@(() => TeamDTO.CountryId)" />

</div>

</div>

<div class="mb-3">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

<button class="btn btn-primary" type="submit">@Localizer["SaveChanges"]</button>

<button class="btn btn-success" @onclick="ReturnAction">@Localizer["Return"]</button>

</EditForm>

1. Creamos el **TeamCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamCreate

{

private TeamForm? teamForm;

private TeamDTO teamDTO = new();

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/teams/full", teamDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], Localizer[message!], SweetAlertIcon.Error);

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordCreatedOk"]);

}

private void Return()

{

teamForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/teams");

}

}

1. Creamos el **TeamCreate.razor**:

@page "/teams/create"

<h3>@Localizer["Create"] @Localizer["Team"]</h3>

<TeamForm @ref="teamForm" TeamDTO="teamDTO" OnValidSubmit="CreateAsync" ReturnAction="Return" />

1. Probamos.
2. Creamos el **TeamEdit.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamEdit

{

private TeamDTO? teamDTO;

private TeamForm? teamForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Team>($"api/teams/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("teams");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], messageError, SweetAlertIcon.Error);

}

}

else

{

var team = responseHttp.Response;

teamDTO = new TeamDTO()

{

Id = team!.Id,

Name = team!.Name,

Image = team.Image,

CountryId = team.CountryId

};

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/teams/full", teamDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync(Localizer["Error"], Localizer[mensajeError!], SweetAlertIcon.Error);

return;

}

Return();

var toast = SweetAlertService.Mixin(new SweetAlertOptions

{

Toast = true,

Position = SweetAlertPosition.BottomEnd,

ShowConfirmButton = true,

Timer = 3000

});

await toast.FireAsync(icon: SweetAlertIcon.Success, message: Localizer["RecordSavedOk"]);

}

private void Return()

{

teamForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("teams");

}

}

1. Creamos el **TeamEdit.razor**:

@page "/teams/edit/{Id:int}"

<h3>@Localizer["Edit"] @Localizer["Team"]</h3>

@if (teamDTO is null)

{

<Loading />

}

else

{

<TeamForm @ref="teamForm" TeamDTO="teamDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

}

1. Probamos y hacemos el **commit**.

## Colocando las banderas de los países por el Seeder

1. Dentro del backend creamos la carpeta Images/Flags y dentro de esta ponemos los archivos de banderas (los puedes descargar de mi repositorio). **Nota**: No olvidar poner la propiedad de **Copy to Output Directory** en **Copy if newer**.
2. Modificamos el **SeedDb,** primero inyectamos el **IFileStorage**:

private async Task CheckTeamsAsync()

{

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

{

foreach (var country in \_context.Countries)

{

var imagePath = string.Empty;

var filePath = $"{Environment.CurrentDirectory}\\Images\\Flags\\{country.Name}.png";

if (File.Exists(filePath))

{

var fileBytes = File.ReadAllBytes(filePath);

imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "flags");

}

\_context.Teams.Add(new Team { Name = country.Name, Country = country!, Image = imagePath });

}

await \_context.SaveChangesAsync();

}

}

1. Probamos y hacemos el **commit**.

# Aca vamos…

## Agregando paginación y filtros desde el backend

1. En la carpeta **DTOs** creamos la clase **PaginationDTO**:

namespace Fantasy.Shared.DTOs;

public class PaginationDTO

{

public int Id { get; set; }

public int Page { get; set; } = 1;

public int RecordsNumber { get; set; } = 10;

public string? Filter { get; set; }

}

1. En el proyecto **Backend** en el folder **Helpers** creamos la clase **QueryableExtensions**:

using Fantasy.Shared.DTOs;

namespace Fantasy.Backend.Helpers;

public static class QueryableExtensions

{

public static IQueryable<T> Paginate<T>(this IQueryable<T> queryable, PaginationDTO pagination)

{

return queryable

.Skip((pagination.Page - 1) \* pagination.RecordsNumber)

.Take(pagination.RecordsNumber);

}

}

1. Modificamos el **IGenericRepository**, agregandole otra sobre carga el GET.

Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync();

1. Modificamos el **GenericRepository**:

public virtual async Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_entity.AsQueryable();

return new ActionResponse<IEnumerable<T>>

{

WasSuccess = true,

Result = await queryable

.Paginate(pagination)

.ToListAsync()

};

}

public virtual async Task<ActionResponse<int>> GetTotalRecordsAsync()

{

var queryable = \_entity.AsQueryable();

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **IGenericUnitOfWork**:

Task<ActionResponse<IEnumerable<T>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync();

1. Modificamos el **IGenericUnitOfWork**:

public virtual async Task<ActionResponse<int>> GetTotalRecordsAsync() => await \_repository.GetTotalRecordsAsync();

public virtual async Task<ActionResponse<T>> UpdateAsync(T model) => await \_repository.UpdateAsync(model);

1. Modificamos el **GenericController**:

[HttpGet("paginated")]

public virtual async Task<IActionResult> GetAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_unitOfWork.GetAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("totalRecords")]

public virtual async Task<IActionResult> GetTotalRecordsAsync()

{

var action = await \_unitOfWork.GetTotalRecordsAsync();

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Modificamos el **ICountriesRepository**:

Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Modificamos el **CountriesRepository**:

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync()

{

var countries = await \_context.Countries

.Include(c => c.Teams)

.OrderBy(c => c.Name)

.ToListAsync();

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = countries

};

}

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Countries

.Include(x => x.Teams)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Countries.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **ICountriesUnitOfWork**:

Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Modificamos el **CountriesUnitOfWork**:

public override async Task<ActionResponse<IEnumerable<Country>>> GetAsync(PaginationDTO pagination) => await \_countriesRepository.GetAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_countriesRepository.GetTotalRecordsAsync(pagination);

1. Modificamos el **CountriesController**:

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_countriesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_countriesUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Probamos en swagger.
2. Modificamos el **ITeamsRepository**:

Task<ActionResponse<IEnumerable<Team>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Modificamos el **TeamsRepository**:

public override async Task<ActionResponse<IEnumerable<Team>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Teams

.Include(x => x.Country)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Team>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Teams.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **ITeamsUnitOfWork**:

Task<ActionResponse<IEnumerable<Team>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Modificamos el **TeamsUnitOfWork**:

public override async Task<ActionResponse<IEnumerable<Team>>> GetAsync(PaginationDTO pagination) => await \_teamsRepository.GetAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_teamsRepository.GetTotalRecordsAsync(pagination);

1. Modificamos el **TeamsController**:

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_teamsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_teamsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Probamos en swagger y hacemos el commit.

## Cambiando el look & feel con MudBlazor

Vamos a utilizar las librerías de **MudBlazor**, la documentación está en <https://mudblazor.com/getting-started/installation#prerequisites> primero procedemos con la instalación

1. En el **FrontEnd** usamos el NuGet Package Manager para instalar **MudBlazor**.
2. Luego de instalar el paquete añadimos en el archivo **\_imports.razor** la siguiente línea:

@using MudBlazor

1. Modificamos el archivo **index.html** para agregar los estilos y scripts de MudBlazor:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Fantasy.Frontend</title>

<base href="/" />

<link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />

<link rel="stylesheet" href="css/app.css" />

<link rel="icon" type="image/png" href="favicon.png" />

<link href="Fantasy.Frontend.styles.css" rel="stylesheet" />

<link href="\_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />

</head>

<body>

<div id="app">

<svg class="loading-progress">

<circle r="40%" cx="50%" cy="50%" />

<circle r="40%" cx="50%" cy="50%" />

</svg>

<div class="loading-progress-text"></div>

</div>

<div id="blazor-error-ui">

An unhandled error has occurred.

<a href="" class="reload">Reload</a>

<a class="dismiss">🗙</a>

</div>

<script src="\_framework/blazor.webassembly.js"></script>

<script src="\_content/CurrieTechnologies.Razor.SweetAlert2/sweetAlert2.min.js"></script>

<script src="\_content/MudBlazor/MudBlazor.min.js"></script>

</body>

</html>

1. En el archivo **Program.cs** añadimos el servicio:

builder.Services.AddMudServices();

1. Modificamos todo el contenido de **MainLayout.razor,** en este paso adaptamos todo el contenido del layout principal con componentes de MudBlazor. En este fragmento usamos varios componentes de MudBlazor, como el **MudLayout** que define el layout de la aplicación, El **MudAppBa**r que genera una barra superior, **MudIconButtons** que son botones definidos por iconos, el **MudMen**u que ofrece un menú desplegable y el **MudDrawer** que es donde se encuentra el NavMenu. Primero modificamos el **MainLayout.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Layout;

public partial class MainLayout

{

private bool \_drawerOpen = true;

private string \_icon = Icons.Material.Filled.DarkMode;

private bool \_darkMode { get; set; } = true;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private void DrawerToggle()

{

\_drawerOpen = !\_drawerOpen;

}

private void DarkModeToggle()

{

\_darkMode = !\_darkMode;

\_icon = \_darkMode ? Icons.Material.Filled.LightMode : Icons.Material.Filled.DarkMode;

}

}

1. Luego modificamos el **MainLayout.razor**:

@inherits LayoutComponentBase

<MudThemeProvider IsDarkMode="\_darkMode" />

<MudDialogProvider />

<MudSnackbarProvider />

<MudPopoverProvider />

<MudLayout>

<MudAppBar Elevation="1">

<MudIconButton Icon="@Icons.Material.Filled.Menu"

Color="Color.Inherit"

Edge="Edge.Start"

OnClick="@((e) => DrawerToggle())" />

<MudLink Href="/"

Typo="Typo.h5"

Class="ml-3"

Color="Color.Inherit">

@Localizer["Title"]

</MudLink>

<MudSpacer />

<MudMenu Icon="@Icons.Material.Filled.Settings"

Color="Color.Inherit"

ActivationEvent="@MouseEvent.MouseOver"

AnchorOrigin="Origin.BottomRight"

TransformOrigin="Origin.TopRight">

@\* <AuthLinks /> \*@

</MudMenu>

<MudIconButton Icon="@\_icon"

Color="Color.Inherit"

Edge="Edge.Start"

OnClick="@((e) => DarkModeToggle())" />

</MudAppBar>

<MudDrawer @bind-Open="\_drawerOpen"

ClipMode="DrawerClipMode.Always"

Elevation="2">

<NavMenu />

</MudDrawer>

<MudMainContent>

<MudContainer MaxWidth="MaxWidth.Large" Style="margin-top: 3rem">

@Body

</MudContainer>

</MudMainContent>

</MudLayout>

1. Probamos, recuerda correr con Ctrl + F5 para que tome los nuevos Scripts y Estilos.
2. Modificamos todo el contenido de **NavMenu.razor,** En este código utilizamos los componente **MudNavMenu** que entrega la navegación de la aplicación, los **MudNavLinks** que define las rutas de la navegación, y el **MudDivider** que entrega una separación entre componentes, además se usan los iconos incluidos en la librería de **MudBlazor**. En la siguiente URL puede buscar los íconos que necesite <https://fonts.google.com/icons>:

<MudNavMenu>

<MudNavLink Href="/" Match="NavLinkMatch.All" Icon="@Icons.Material.Rounded.Home">@Localizer["Home"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/countries" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Public">@Localizer["Countries"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/teams" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Groups">@Localizer["Teams"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/about" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Info">@Localizer["About"]</MudNavLink>

</MudNavMenu>

1. Modificamos el **Home.Razor**:
2. Probamos y hacemos el commit.

## Agregando paginación/filtrado de paises desde el frontend

1. Aquí es donde realmente se comienzan a ver los cambios en los componentes, modificamos el archivo **CountriesIndex.razor,** En este código usamos el componente **MudPagination**, que es un paginador nativo de **Mudblazor** que recibe como parámetro el total de páginas y ejecuta una función con el método **SelectedChange** al que le pasa el número de página seleccionado.
2. Primero agregamos los siguientes literales:

| RecordsNumber | RecordsNumber | Número de Registros: |
| --- | --- | --- |
| All | All | Todos |
| Search | Search… | Buscar… |
| Filter | Filter | Filtrar |
| Clean | Clean | Limpiar |
| Actions | Actions | Acciones |

1. Agregamos el archivo **question.png** en **wwwroot/images**.
2. En **Frontend/Shared** cremos el **FilterComponent.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Shared;

public partial class FilterComponent

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public string FilterValue { get; set; } = string.Empty;

[Parameter] public EventCallback<string> ApplyFilter { get; set; }

private async Task CleanFilter()

{

FilterValue = string.Empty;

await ApplyFilter.InvokeAsync(FilterValue);

}

private async Task OnFilterApply()

{

await ApplyFilter.InvokeAsync(FilterValue);

}

}

1. En **Frontend/Shared** modificamos el **FilterComponent.razor**:

<div class="mb-2" style="display: flex; flex-wrap:wrap; align-items: center;">

<div>

<MudTextField @bind-Value="FilterValue"

Placeholder=@Localizer["Search"]

Adornment="Adornment.Start"

AdornmentIcon="@Icons.Material.Filled.Search"

IconSize="Size.Medium" Class="mt-0"/>

</div>

<div class="mx-1">

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.FilterList"

Color="Color.Primary"

@onclick="OnFilterApply">

@Localizer["Filter"]

</MudButton>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Delete"

Color="Color.Error"

@onclick="CleanFilter">

@Localizer["Clean"]

</MudButton>

</div>

</div>

1. En **Frontend/Shared** cremos el **ConfirmDialog.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Shared;

public partial class ConfirmDialog

{

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public string Message { get; set; } = null!;

private void Accept()

{

MudDialog.Close(DialogResult.Ok(true));

}

private void Cancel()

{

MudDialog.Close(DialogResult.Cancel());

}

}

1. En **Frontend/Shared** modificamos el **ConfirmDialog.razor**:

<MudDialog>

<DialogContent>

<div style="display: flex; justify-content: center; align-items: center;">

<MudImage Src="images/question.png" Width="130" Class="mb-3" />

</div>

<MudText>@Message</MudText>

</DialogContent>

<DialogActions>

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Check" Color="Color.Primary" OnClick="Accept" FullWidth="true">

@Localizer["Yes"]

</MudButton>

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Cancel" Color="Color.Secondary" OnClick="Cancel" FullWidth="true">

@Localizer["No"]

</MudButton>

</DialogActions>

</MudDialog>

1. Modificamos el **CountriesIndex.razor.cs**:

using System.Net;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountriesIndex

{

private List<Country>? Countries { get; set; }

private MudTable<Country> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/countries";

private string infoFormat = "{first\_item}-{last\_item} => {all\_items}";

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadTotalRecordsAsync();

}

private async Task LoadTotalRecordsAsync()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

totalRecords = responseHttp.Response;

loading = false;

}

private async Task<TableData<Country>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated/?page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Country>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Country> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Country> { Items = [], TotalItems = 0 };

}

return new TableData<Country>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

IDialogReference? dialog;

if (isEdit)

{

var parameters = new DialogParameters

{

{ "Id", id }

};

dialog = DialogService.Show<CountryEdit>($"{Localizer["Edit"]} {Localizer["Country"]}", parameters, options);

}

else

{

dialog = DialogService.Show<CountryCreate>($"{Localizer["New"]} {Localizer["Country"]}", options);

}

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

private async Task DeleteAsync(Country country)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Country"], country.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrl}/{country.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/countries");

}

else

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

}

return;

}

await LoadTotalRecordsAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Modificamos el **CountriesIndex.razor**:

@page "/countries"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@Countries"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<div class="d-flex justify-content-between">

<MudText Typo="Typo.h6" Class="me-4"> @Localizer["Countries"]</MudText>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info" OnClick="@(() => ShowModalAsync())">

@Localizer["New"]

</MudButton>

</div>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Country"]</MudTh>

<MudTh># @Localizer["Teams"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>@context.TeamsCount</MudTd>

<MudTd>

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Delete"]">

<MudButton Variant="Variant.Filled"

Color="Color.Error"

OnClick="@(() => DeleteAsync(@context))">

<MudIcon Icon="@Icons.Material.Filled.Delete" />

</MudButton>

</MudTooltip>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos.
2. Modificamos el **CountryForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Country"]"

@bind-Value="@Country.Name"

For="@(() => Country.Name)"

Class="mb-4" />

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **CountryCreate.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountryCreate

{

private CountryForm? countryForm;

private Country country = new();

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/countries", country);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordCreatedOk"], Severity.Success);

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/countries");

}

}

1. Modificamos el **CountryCreate.razor**:

<MudDialog>

<DialogContent>

<CountryForm @ref="countryForm" Country="country" OnValidSubmit="CreateAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

1. Modificamos el **CountryEdit.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Countries;

public partial class CountryEdit

{

private Country? country;

private CountryForm? countryForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Country>($"api/countries/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("countries");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

else

{

country = responseHttp.Response;

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/countries", country);

if (responseHttp.Error)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

countryForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("countries");

}

}

1. Modificamos el **CountryEdit.razor**:

@if(country is null)

{

<Loading/>

}

else

{

<MudDialog>

<DialogContent>

<CountryForm @ref="countryForm" Country="country" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Cambios el componete **Loading.razor.cs**:

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Shared;

public partial class Loading

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public string? Label { get; set; }

protected override void OnParametersSet()

{

base.OnParametersSet();

if (string.IsNullOrEmpty(Label))

{

Label = Localizer["PleaseWait"];

}

}

}

1. Cambios el componete **Loading.razor**:

<MudCard>

<div class="overlay d-flex flex-column justify-content-center align-items-center p-3">

<MudProgressCircular Indeterminate="true" Color="Color.Primary" Class="mb-3" />

<MudText Typo="Typo.h5">@Label</MudText>

</div>

</MudCard>

1. Probamos y hacemos el commit.

## Agregando paginación/filtrado de equipos desde el frontend

1. Modificamos el **TeamsIndex.razor.cs**:

using System.Net;

using Fantasy.Frontend.Pages.Countries;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamsIndex

{

private List<Team>? Teams { get; set; }

private MudTable<Team> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/teams";

private string infoFormat = "{first\_item}-{last\_item} => {all\_items}";

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadTotalRecordsAsync();

}

private async Task LoadTotalRecordsAsync()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

totalRecords = responseHttp.Response;

loading = false;

}

private async Task<TableData<Team>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated/?page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Team>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Team> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Team> { Items = [], TotalItems = 0 };

}

return new TableData<Team>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadTotalRecordsAsync();Close

await table.ReloadServerData();

}

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

IDialogReference? dialog;

if (isEdit)

{

var parameters = new DialogParameters

{

{ "Id", id }

};

dialog = DialogService.Show<CountryEdit>($"{Localizer["Edit"]} {Localizer["Team"]}", parameters, options);

}

else

{

dialog = DialogService.Show<CountryCreate>($"{Localizer["New"]} {Localizer["Team"]}", options);

}

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

private async Task DeleteAsync(Team team)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Team"], team.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrl}/{team.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/teams");

}

else

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

}

return;

}

await LoadTotalRecordsAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Modificamos el **TeamsIndex.razor**:

@page "/teams"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@Teams"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<div class="d-flex justify-content-between">

<MudText Typo="Typo.h6" Class="me-4"> @Localizer["Teams"]</MudText>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info" OnClick="@(() => ShowModalAsync())">

@Localizer["New"]

</MudButton>

</div>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Team"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Country"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>

<MudImage Src="@context.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>@context.Country.Name</MudTd>

<MudTd>

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Delete"]">

<MudButton Variant="Variant.Filled"

Color="Color.Error"

OnClick="@(() => DeleteAsync(@context))">

<MudIcon Icon="@Icons.Material.Filled.Delete" />

</MudButton>

</MudTooltip>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos.
2. Modificamos el **TeamForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamForm

{

private EditContext editContext = null!;

private Country selectedCountry = new();

protected override void OnInitialized()

{

editContext = new(TeamDTO);

}

[EditorRequired, Parameter] public TeamDTO TeamDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private List<Country>? countries;

private string? imageUrl;

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

protected override void OnParametersSet()

{

base.OnParametersSet();

if (!string.IsNullOrEmpty(TeamDTO.Image))

{

imageUrl = TeamDTO.Image;

TeamDTO.Image = null;

}

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Country>>("/api/countries/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);

return;

}

countries = responseHttp.Response;

}

private void ImageSelected(string imagenBase64)

{

TeamDTO.Image = imagenBase64;

imageUrl = null;

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

private async Task<IEnumerable<Country>> SearchCountry(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return countries!;

}

return countries!

.Where(x => x.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private void CountryChanged(Country country)

{

selectedCountry = country;

TeamDTO.CountryId = country.Id;

}

}

1. Modificamos el **TeamForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Team"]"

@bind-Value="@TeamDTO.Name"

For="@(() => TeamDTO.Name)"

Class="mb-4" />

<MudAutocomplete T="Country"

Label=@Localizer["Country"]

Placeholder=@Localizer["SelectACountry"]

SearchFunc="SearchCountry"

Value="selectedCountry"

ValueChanged="CountryChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<div class="my-2">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **TeamCreate.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamCreate

{

private TeamForm? teamForm;

private TeamDTO teamDTO = new();

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/teams/full", teamDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordCreatedOk"], Severity.Success);

}

private void Return()

{

teamForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/teams");

}

}

1. Modificamos el **TeamCreate.razor**:

<MudDialog>

<DialogContent>

<TeamForm @ref="teamForm" TeamDTO="teamDTO" OnValidSubmit="CreateAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

1. Modificamos el **TeamsEdit.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Teams;

public partial class TeamEdit

{

private TeamDTO? teamDTO;

private TeamForm? teamForm;

private Country selectedCountry = new();

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Team>($"api/teams/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("teams");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

else

{

var team = responseHttp.Response;

teamDTO = new TeamDTO()

{

Id = team!.Id,

Name = team!.Name,

Image = team.Image,

CountryId = team.CountryId

};

selectedCountry = team.Country;

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/teams/full", teamDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

teamForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("teams");

}

}

1. Modificamos el **TeamEdit.razor**:

@if(teamDTO is null)

{

<Loading/>

}

else

{

<MudDialog>

<DialogContent>

<TeamForm @ref="teamForm" TeamDTO="teamDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el commit.

# Sistema de Seguridad

## Creando las tablas de usuarios

1. Agregamos los siguientes literales:

| Admin | Admin | Administrador |
| --- | --- | --- |
| User | User | Usuario |
| FirstName | First Name | Nombres |
| LastName | Last Name | Apellidos |
| UserType | User Type | Tipo de Usuario |

1. Como vamos a tener dos tipos de usuarios; administradores y usuarios. Vamos a crear una enumeración para diferenciarlos. Creamos la carpeta **Enums** en el proyecto **Shared** y dentro de esta carpeta la enumeración **UserType**:

namespace Fantasy.Shared.Enums;

public enum UserType

{

Admin,

User

}

1. En el proyecto **Shared** el nuget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.
2. En el proyecto **Shared** en la carpeta **Entities**, crear la entidad **User**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Enums;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Identity;

namespace Fantasy.Shared.Entities;

public class User : IdentityUser

{

[Display(Name = "FirstName", ResourceType = typeof(Literals))]

[MaxLength(50, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string FirstName { get; set; } = null!;

[Display(Name = "LastName", ResourceType = typeof(Literals))]

[MaxLength(50, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string LastName { get; set; } = null!;

[Display(Name = "Image", ResourceType = typeof(Literals))]

public string? Photo { get; set; }

[Display(Name = "UserType", ResourceType = typeof(Literals))]

public UserType UserType { get; set; }

public Country Country { get; set; } = null!;

[Display(Name = "Country", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int CountryId { get; set; }

[Display(Name = "User", ResourceType = typeof(Literals))]

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

}

1. Modificamos la entidad **Country** para definir la relación a ambos lados de esta:

public ICollection<User>? Users { get; set; }

public int UsersCount => Users == null ? 0 : Users.Count;

1. En el proyecto **Backend** instalar el nugget **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.
2. Modificar el **DataContext**:

public class DataContext : IdentityDbContext<User>

1. Creamos el **IUsersRepository**:

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Identity;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface IUsersRepository

{

Task<User> GetUserAsync(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. Creamos el **UsersRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class UsersRepository : IUsersRepository

{

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public UsersRepository(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager)

{

\_context = context;

\_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> GetUserAsync(string email)

{

var user = await \_context.Users

.Include(u => u.Country)

.FirstOrDefaultAsync(x => x.Email == email);

return user!;

}

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

{

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

}

}

1. Creamos el **IUsersUnitOfWork**:

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Identity;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IUsersUnitOfWork

{

Task<User> GetUserAsync(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. Creamos el **UsersUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Identity;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class UsersUnitOfWork : IUsersUnitOfWork

{

private readonly IUsersRepository \_usersRepository;

public UsersUnitOfWork(IUsersRepository usersRepository)

{

\_usersRepository = usersRepository;

}

public async Task<IdentityResult> AddUserAsync(User user, string password) => await \_usersRepository.AddUserAsync(user, password);

public async Task AddUserToRoleAsync(User user, string roleName) => await \_usersRepository.AddUserToRoleAsync(user, roleName);

public async Task CheckRoleAsync(string roleName) => await \_usersRepository.CheckRoleAsync(roleName);

public async Task<User> GetUserAsync(string email) => await \_usersRepository.GetUserAsync(email);

public async Task<bool> IsUserInRoleAsync(User user, string roleName) => await \_usersRepository.IsUserInRoleAsync(user, roleName);

}

1. Matriculamos la nueva inyección en el **Program** del proyecto **Backend**, y otras modificaciones para configurar el manejo de usuarios:

builder.Services.AddScoped<ITeamsRepository, TeamsRepository>();

builder.Services.AddScoped<ITeamsUnitOfWork, TeamsUnitOfWork>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

builder.Services.AddIdentity<User, IdentityRole>(x =>

{

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

var app = builder.Build();

1. Modificamos el **SeedDb** (Primero inyectamos el **IUsersUnitOfWork**):

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckTeamsAsync();

await CheckRolesAsync();

await CheckUserAsync("Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", UserType.Admin);

}

private async Task CheckRolesAsync()

{

await \_usersUnitOfWork.CheckRoleAsync(UserType.Admin.ToString());

await \_usersUnitOfWork.CheckRoleAsync(UserType.User.ToString());

}

private async Task<User> CheckUserAsync(string firstName, string lastName, string email, string phone, UserType userType)

{

var user = await \_usersUnitOfWork.GetUserAsync(email);

if (user == null)

{

var country = await \_context.Countries.FirstOrDefaultAsync(x => x.Name == "Colombia");

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Country = country!,

UserType = userType,

};

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

await \_usersUnitOfWork.AddUserToRoleAsync(user, userType.ToString());

}

return user;

}

1. Corremos los siguientes comandos:

PM> drop-database

PM> add-migration AddUsersEntities

PM> update-database

1. Probamos y hacemos el **commit**.

## Creando sistema de seguridad

1. Agregamos los siguientes literales:

| NotFound | Not Found | No Encontrado |
| --- | --- | --- |
| NothingInRoute | Sorry, there is nothing on this route. | Lo sentimos no hay nada en esta ruta. |
| Authorizing | Authorizing… | Autorizando... |
| NotAuthorized | You are not authorized to view this content... | No estas autorizado para ver este contenido... |

1. Al proyecto **Frontend** agregamos el paquete:

**Microsoft.AspNetCore.Components.WebAssembly.Authentication**

1. Agregamos este using en el **\_Imports**:

@using Microsoft.AspNetCore.Components.Authorization

1. En el proyecto **Frontend** creamos la carpeta **AuthenticationProviders** y dentro de esta la clase **AuthenticationProviderTest**:

using Microsoft.AspNetCore.Components.Authorization;

using System.Security.Claims;

namespace Fantasy.Frontend.AuthenticationProviders;

public class AuthenticationProviderTest : AuthenticationStateProvider

{

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonimous)));

}

}

1. Modificamos el **Program** del proyecto **Frontend**:

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7232") });

builder.Services.AddScoped<IRepository, Repository>();

builder.Services.AddLocalization();

builder.Services.AddSweetAlert2();

builder.Services.AddMudServices();

builder.Services.AddAuthorizationCore();

builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderTest>();

1. Creamos el **App.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend;

public partial class App

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

}

1. Modificamos el **App.razor**:

<Router AppAssembly="@typeof(App).Assembly">

<Found Context="routeData">

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

<FocusOnNavigate RouteData="@routeData" Selector="h1" />

</Found>

<NotFound>

<CascadingAuthenticationState>

<PageTitle>@Localizer["NotFound"]</PageTitle>

<LayoutView Layout="@typeof(MainLayout)">

<p role="alert">@Localizer["NothingInRoute"]</p>

</LayoutView>

</CascadingAuthenticationState>

</NotFound>

</Router>

1. Probamos y vemos que aparentemente no pasa nada, ahora a nuestro **AuthenticationProviderTest** le vamos a colocar un tiempo de espera:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

await Task.Delay(3000);

var anonimous = new ClaimsIdentity();

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonimous)));

}

1. Probamos de nuevo y vemos que tarda los 3 segundos haciendo la autorización.
2. Si queremos cambiar el mensaje, modificamos el **App.razor**:

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">

<Authorizing>

<p>@Localizer["Authorizing"]</p>

</Authorizing>

</AuthorizeRouteView>

1. Probamos de nuevo.
2. Modificacmos el **Home.razor**.

@page "/"

<AuthorizeView>

<p>Estas autenticado</p>

</AuthorizeView>

1. Modificamos el **AuthenticationProviderTest**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

var user = new ClaimsIdentity(authenticationType: "test");

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(user)));

}

1. Cambiamos el **Home.razor**.

<AuthorizeView>

<Authorized>

<p>Estas autenticado</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Y jugamos con el **AuthenticationProviderTest** para ver que pasa con el usuario **anonimous** y con el usuario **user**.
2. Modificamos nuestro **AuthenticationProviderTest**, para agregar algunos **Claims**:

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var anonimous = new ClaimsIdentity();

var user = new ClaimsIdentity(authenticationType: "test");

var admin = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Juan"),

new Claim("LastName", "Zulu"),

new Claim(ClaimTypes.Name, "zulu@yopmail.com")

},

authenticationType: "test");

return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(admin)));

}

1. Modificamos el **Home.razor** y probamos:

<AuthorizeView>

<Authorized>

<p>Estas autenticado, @context.User.Identity?.Name</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Probamos.
2. Modificamos de nuevo el **Index.razor** para crear un **Role** y probamos:

<AuthorizeView Roles="Admin">

<Authorized>

<p>Estas autenticado y autorizado, @context.User.Identity?.Name</p>

</Authorized>

<NotAuthorized>

<p>No estas autorizado</p>

</NotAuthorized>

</AuthorizeView>

1. Modificamos nuestro **AuthenticationProviderTest**, para agregar el **Claim** de **Role** y probamos:

var admin = new ClaimsIdentity(new List<Claim>

{

new Claim("FirstName", "Juan"),

new Claim("LastName", "Zulu"),

new Claim(ClaimTypes.Name, "zulu@yopmail.com"),

new Claim(ClaimTypes.Role, "Admin")

},

authenticationType: "test");

1. Ahora cambiamos nuestro **NavMenu** para mostrar la opción de países y equipos solo a los administradores, y jugamos con nuestro **AuthenticationProviderTest** para cambiarle el rol al usuario:

<MudNavMenu>

<MudNavLink Href="/" Match="NavLinkMatch.All" Icon="@Icons.Material.Rounded.Home">@Localizer["Home"]</MudNavLink>

<MudDivider />

<AuthorizeView Roles="Admin">

<Authorized>

<MudNavLink Href="/countries" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Public">@Localizer["Countries"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/teams" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Groups">@Localizer["Teams"]</MudNavLink>

<MudDivider />

</Authorized>

</AuthorizeView>

<MudNavLink Href="/about" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Info">@Localizer["About"]</MudNavLink>

</MudNavMenu>

1. Pero nótese que solo estamos ocultando la opción, si el usuario por la URL introduce la dirección de países, pues podrá acceder a nuestras páginas, lo cual es algo que no queremos. Para evitar esto le colocamos este atributo a todos los componentes a los que navegamos y queremos proteger, agregamos este decorador a las clases **CountriesIndex.razor.cs** y **TeamsIndex.razor.cs**:

[Authorize(Roles = "Admin")]

1. Ahora si queremos personalizar el mensaje podemos modificar nuestro **App.razor**:

<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">

<Authorizing>

<p>@Localizer["Authorizing"]</p>

</Authorizing>

<NotAuthorized>

<p>@Localizer["NotAuthorized"]</p>

</NotAuthorized>

</AuthorizeRouteView>

1. Probamos y hacemos el **commit**.

## Seguridad desde el backend

1. Agregamos los siguientes literales en ambos archivos de literales:

| Password | Password | Contraseña |
| --- | --- | --- |
| LengthField | Field {0} must be between {2} and {1} characters. | El campo {0} debe tener entre {2} y {1} carácteres. |
| PasswordAndConfirmationDifferent | The password and confirmation are not the same. | La contraseña y la confirmación no son iguales. |
| PasswordConfirm | Password Confirm | Confirmación de contraseña |
| Email | Email | Correo electrónico |
| ValidEmail | You must enter a valid email. | Debes ingresar un correo válido. |
| MinLength | The {0} field must have at least {1} characters. | El campo {0} debe tener al menos {1} carácteres. |
| ERR006 | Incorrect email or password. | Email o contraseña incorrectos. |

1. Agregamos al proyecto **Backend** el paquete **Microsoft.AspNetCore.Authentication.JwtBearer**.
2. Creamos el parámetro **jwtKey** en el **appsettings** del proyecto **Backend** (cualquier cosa, entre mas larga mejor):

"jwtKey": "[Put your own long key]",

"Logging": {

1. Modificamos el **Program** del proyecto **Backend**:

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(x => x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["jwtKey"]!)),

ClockSkew = TimeSpan.Zero

});

var app = builder.Build();

1. Y corrijamos para que el **UseCors** este antes de **UseHttpsRedirection**:

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseCors(x => x

.AllowAnyMethod()

.AllowAnyHeader()

.SetIsOriginAllowed(origin => true)

.AllowCredentials());

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **UserDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class UserDTO : User

{

[DataType(DataType.Password)]

[Display(Name = "Password", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string Password { get; set; } = null!;

[Compare("Password", ErrorMessageResourceName = "PasswordAndConfirmationDifferent", ErrorMessageResourceType = typeof(Literals))]

[Display(Name = "PasswordConfirm", ResourceType = typeof(Literals))]

[DataType(DataType.Password)]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string PasswordConfirm { get; set; } = null!;

}

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **TokenDTO**:

namespace Fantasy.Shared.DTOs;

public class TokenDTO

{

public string Token { get; set; } = null!;

public DateTime Expiration { get; set; }

}

1. En el proyecto **Shared** en la carpeta **DTOs** creamos el **LoginDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class LoginDTO

{

[Display(Name = "Email", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[EmailAddress(ErrorMessageResourceName = "ValidEmail", ErrorMessageResourceType = typeof(Literals))]

public string Email { get; set; } = null!;

[Display(Name = "Password", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[MinLength(6, ErrorMessageResourceName = "MinLength", ErrorMessageResourceType = typeof(Literals))]

public string Password { get; set; } = null!;

}

1. Agregamos estos métodos al **IUsersRepository**:

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

1. Los implementamos en el **UsersRepository**:

…

private readonly DataContext \_context;

private readonly UserManager<User> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly SignInManager<User> \_signInManager;

public UsersRepository(DataContext context, UserManager<User> userManager, RoleManager<IdentityRole> roleManager, SignInManager<User> signInManager)

{

\_context = context;

\_userManager = userManager;

\_roleManager = roleManager;

\_signInManager = signInManager;

}

public async Task<SignInResult> LoginAsync(LoginDTO model)

{

return await \_signInManager.PasswordSignInAsync(model.Email, model.Password, false, false);

}

public async Task LogoutAsync()

{

await \_signInManager.SignOutAsync();

}

…

1. Agregamos estos métodos al **IUsersUnitOfWork**:

Task<SignInResult> LoginAsync(LoginDTO model);

Task LogoutAsync();

1. Los implementamos en el **UsersUnitOfWork**:

public async Task<SignInResult> LoginAsync(LoginDTO model) => await \_usersRepository.LoginAsync(model);

public async Task LogoutAsync() => await \_usersRepository.LogoutAsync();

1. Creamos el **AccountsController**:

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Route("/api/accounts")]

public class AccountsController : ControllerBase

{

private readonly IUsersUnitOfWork \_usersUnitOfWork;

private readonly IConfiguration \_configuration;

public AccountsController(IUsersUnitOfWork usersUnitOfWork, IConfiguration configuration)

{

\_usersUnitOfWork = usersUnitOfWork;

\_configuration = configuration;

}

[HttpPost("CreateUser")]

public async Task<IActionResult> CreateUser([FromBody] UserDTO model)

{

User user = model;

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

if (result.Succeeded)

{

await \_usersUnitOfWork.AddUserToRoleAsync(user, user.UserType.ToString());

return Ok(BuildToken(user));

}

return BadRequest(result.Errors.FirstOrDefault());

}

[HttpPost("Login")]

public async Task<IActionResult> LoginAsync([FromBody] LoginDTO model)

{

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

if (result.Succeeded)

{

var user = await \_usersUnitOfWork.GetUserAsync(model.Email);

return Ok(BuildToken(user));

}

return BadRequest("ERR006");

}

private TokenDTO BuildToken(User user)

{

var claims = new List<Claim>

{

new(ClaimTypes.Name, user.Email!),

new(ClaimTypes.Role, user.UserType.ToString()),

new("FirstName", user.FirstName),

new("LastName", user.LastName),

new("Photo", user.Photo ?? string.Empty),

new("CountryId", user.Country.Id.ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["jwtKey"]!));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var expiration = DateTime.UtcNow.AddDays(30);

var token = new JwtSecurityToken(

issuer: null,

audience: null,

claims: claims,

expires: expiration,

signingCredentials: credentials);

return new TokenDTO

{

Token = new JwtSecurityTokenHandler().WriteToken(token),

Expiration = expiration

};

}

}

1. Luego le colocamos autorización a los controladores **CountriesController** y **TeamsController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

1. Podemos probar por **POSTMAN** como está funcionando nuestro token, y con <https://jwt.io/> probamos como está quedando nuestro token.
2. Probamos en la interfaz Frontend, y nos debe salir un error porque aun no le mandamos ningún token a nuestra Backend. Hacemos el **commit**.

## Habilitando tokens en swagger

1. Modificamos el **Program** del **Backend**:

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "Orders Backend", Version = "v1" });

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = @"JWT Authorization header using the Bearer scheme. <br /> <br />

Enter 'Bearer' [space] and then your token in the text input below.<br /> <br />

Example: 'Bearer 12345abcdef'<br /> <br />",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement()

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

},

Scheme = "oauth2",

Name = "Bearer",

In = ParameterLocation.Header,

},

new List<string>()

}

});

});

builder.Services.AddDbContext<DataContext>(x => x.UseSqlServer("name=DockerConnection"));

1. Probamos y hacemos el **commit**.

## Implementando login & logout

1. Creamos los siguientes literales:

| Hello | Hello | Hola |
| --- | --- | --- |
| Logout | Logout | Cerrar Sesión |
| Register | Register as new user | Regitrarse como nuevo usuario |
| Login | Login | Iniciar Sesión |
| NotUserYet | Not a user yet? Register here | ¿No eres usuario aún? Resgitrate aquí |
| LogoutConfirm | Are you sure you want to log out? | ¿Estás seguro que deseas cerrar sesión? |
| LogoutMessage | If you log out, you will need to log back in to access your account. | Si cierras sesión, tendrás que volver a iniciar sesión para acceder a tu cuenta. |
| EditUserProfile | Edit User Profile | Editar Perfil de Usuario |

1. En el proyecto  **Frontend** Instalamos el paquete: **System.IdentityModel.Tokens.Jwt**.
2. En el proyecto  **Frontend** en la carpeta **Helpers** creamos el **IJSRuntimeExtensionMethods**:

using Microsoft.JSInterop;

namespace Fantasy.Frontend.Helpers;

public static class IJSRuntimeExtensionMethods

{

public static ValueTask<object> SetLocalStorage(this IJSRuntime js, string key, string content)

{

return js.InvokeAsync<object>("localStorage.setItem", key, content);

}

public static ValueTask<object> GetLocalStorage(this IJSRuntime js, string key)

{

return js.InvokeAsync<object>("localStorage.getItem", key);

}

public static ValueTask<object> RemoveLocalStorage(this IJSRuntime js, string key)

{

return js.InvokeAsync<object>("localStorage.removeItem", key);

}

}

1. En el proyecto  **Frontend** en la carpeta **Services** creamos el **ILoginService**:

namespace Fantasy.Frontend.Services;

public interface ILoginService

{

Task LoginAsync(string token);

Task LogoutAsync();

}

1. En el proyecto  **Frontend** en la carpeta **AuthenticationProviders** creamos el **AuthenticationProviderJWT**:

using Fantasy.Frontend.Helpers;

using Fantasy.Frontend.Services;

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.JSInterop;

using System.IdentityModel.Tokens.Jwt;

using System.Net.Http.Headers;

using System.Security.Claims;

namespace Fantasy.Frontend.AuthenticationProviders;

public class AuthenticationProviderJWT : AuthenticationStateProvider, ILoginService

{

private readonly IJSRuntime \_jSRuntime;

private readonly HttpClient \_httpClient;

private readonly string \_tokenKey;

private readonly AuthenticationState \_anonimous;

public AuthenticationProviderJWT(IJSRuntime jSRuntime, HttpClient httpClient)

{

\_jSRuntime = jSRuntime;

\_httpClient = httpClient;

\_tokenKey = "TOKEN\_KEY";

\_anonimous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));

}

public override async Task<AuthenticationState> GetAuthenticationStateAsync()

{

var token = await \_jSRuntime.GetLocalStorage(\_tokenKey);

if (token is null)

{

return \_anonimous;

}

return BuildAuthenticationState(token.ToString()!);

}

private AuthenticationState BuildAuthenticationState(string token)

{

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer", token);

var claims = ParseClaimsFromJWT(token);

return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));

}

private IEnumerable<Claim> ParseClaimsFromJWT(string token)

{

var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();

var unserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

return unserializedToken.Claims;

}

public async Task LoginAsync(string token)

{

await \_jSRuntime.SetLocalStorage(\_tokenKey, token);

var authState = BuildAuthenticationState(token);

NotifyAuthenticationStateChanged(Task.FromResult(authState));

}

public async Task LogoutAsync()

{

await \_jSRuntime.RemoveLocalStorage(\_tokenKey);

\_httpClient.DefaultRequestHeaders.Authorization = null;

NotifyAuthenticationStateChanged(Task.FromResult(\_anonimous));

}

}

1. Modificamos el **Program** del  **Frontend** para usar nuestro nuevo proveedor de autenticación:

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7201/") });

builder.Services.AddScoped<IRepository, Repository>();

builder.Services.AddSweetAlert2();

builder.Services.AddAuthorizationCore();

builder.Services.AddScoped<AuthenticationProviderJWT>();

builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

builder.Services.AddScoped<ILoginService, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

1. Creamos dentro de **Pages** la carpeta **Auth** y dentro de esta creamos el **Login.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Resources;

using Fantasy.Frontend.Services;

using Fantasy.Shared.DTOs;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class Login

{

private LoginDTO loginDTO = new();

private bool wasClose;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

private void CloseModal()

{

wasClose = true;

MudDialog.Cancel();

}

private async Task LoginAsync()

{

if (wasClose)

{

NavigationManager.NavigateTo("/");

return;

}

var responseHttp = await Repository.PostAsync<LoginDTO, TokenDTO>("/api/accounts/Login", loginDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message!], Severity.Error);

return;

}

await LoginService.LoginAsync(responseHttp.Response!.Token);

NavigationManager.NavigateTo("/");

}

}

1. Modificamos el **Login.razor**:

<MudDialog>

<DialogContent>

<EditForm Model="loginDTO" OnValidSubmit="LoginAsync">

<DataAnnotationsValidator />

<MudGrid Class="mb-4">

<MudItem xs="12" sm="12">

<MudTextField Label="Email" @bind-Value="@loginDTO.Email" InputType="InputType.Email" />

<ValidationMessage For="@(() => loginDTO.Email)" />

</MudItem>

<MudItem xs="12" sm="12">

<MudTextField Label="Contraseña" @bind-Value="@loginDTO.Password" InputType="InputType.Password" />

<ValidationMessage For="@(() => loginDTO.Password)" />

</MudItem>

</MudGrid>

<MudGrid Class="mb-4">

<MudItem xs="12" sm="6" Class="d-flex justify-content-center">

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Login" Color="Color.Primary" ButtonType="ButtonType.Submit" FullWidth="true">

@Localizer["Login"]

</MudButton>

</MudItem>

<MudItem xs="12" sm="6" Class="d-flex justify-content-center">

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Cancel" Color="Color.Error" OnClick="CloseModal" FullWidth="true">

@Localizer["Cancel"]

</MudButton>

</MudItem>

</MudGrid>

</EditForm>

</DialogContent>

<DialogActions>

<MudItem xs="12" sm="12">

<MudLink Href="/Register" Underline="Underline.Always">@Localizer["NotUserYet"]</MudLink>

</MudItem>

</DialogActions>

</MudDialog>

1. Creamos el **Logout.razor.cs**:

using Fantasy.Frontend.Resources;

using Fantasy.Frontend.Services;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class Logout

{

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

private async Task LogoutActionAsync()

{

await LoginService.LogoutAsync();

NavigationManager.NavigateTo("/");

CancelAction();

}

private void CancelAction()

{

MudDialog.Cancel();

}

}

1. Modificamos el **Logout.razor**:

<MudDialog>

<DialogContent>

<MudText Typo="Typo.h5">@Localizer["LogoutConfirm"]</MudText>

<MudText Typo="Typo.body2">@Localizer["LogoutMessage"]</MudText>

</DialogContent>

<DialogActions>

<MudButton Variant="Variant.Outlined" Color="Color.Tertiary" OnClick="CancelAction">@Localizer["Cancel"]</MudButton>

<MudSpacer />

<MudButton Variant="Variant.Outlined" Color="Color.Error" OnClick="LogoutActionAsync">@Localizer["Logout"]</MudButton>

</DialogActions>

</MudDialog>

1. Creamos el componente compartido **AuthLinks.razor.cs**:

using Fantasy.Frontend.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Shared;

public partial class AuthLinks

{

private string? photoUser;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[CascadingParameter] private Task<AuthenticationState> AuthenticationStateTask { get; set; } = null!;

protected override async Task OnParametersSetAsync()

{

var authenticationState = await AuthenticationStateTask;

var claims = authenticationState.User.Claims.ToList();

var photoClaim = claims.FirstOrDefault(x => x.Type == "Photo");

var nameClaim = claims.FirstOrDefault(x => x.Type == "UserName");

if (photoClaim is not null)

{

photoUser = photoClaim.Value;

}

}

private void EditAction()

{

NavigationManager.NavigateTo("/EditUser");

}

private void ShowModalLogIn()

{

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

DialogService.Show<Login>(Localizer["Login"], closeOnEscapeKey);

}

private void ShowModalLogOut()

{

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

DialogService.Show<Logout>(Localizer["Logout"], closeOnEscapeKey);

}

}

1. Modificamos el **AuthLinks.razor**:

<AuthorizeView>

<Authorized>

<MudContainer Style="width: 18rem; margin-top: 2rem; margin-bottom: 2rem;">

@if (!string.IsNullOrEmpty(photoUser))

{

<MudContainer Class="d-flex justify-content-center mb-3">

<MudBadge Color="Color.Success" Overlap="true" Bordered="true" Class="position-relative">

<MudAvatar Size="Size.Large" Class="mb-3 mx-auto d-block">

<MudImage Src="@photoUser"></MudImage>

</MudAvatar>

</MudBadge>

</MudContainer>

}

<MudText Typo="Typo.body1" Align="Align.Center">@Localizer["Hello"], @context.User.Identity!.Name</MudText>

<MudPaper Elevation="1" Class="d-flex justify-content-center align-content-center">

<MudStack Spacing="2">

<MudButton Variant="Variant.Text" OnClick="EditAction">@Localizer["EditUserProfile"] <MudIcon Icon="@Icons.Material.Filled.Person" /> </MudButton>

<MudButton Variant="Variant.Text" OnClick="ShowModalLogOut">@Localizer["Logout"] <MudIcon Icon="@Icons.Material.Filled.Login" /> </MudButton>

</MudStack>

</MudPaper>

</MudContainer>

</Authorized>

<NotAuthorized>

<MudContainer Style="width: 18rem; margin-top: 2rem; margin-bottom: 2rem;">

<MudStack Spacing="2">

<a href="/register" class="nav-link btn btn-link">@Localizer["Register"] <MudIcon Icon="@Icons.Material.Filled.HowToReg" /></a>

<MudMenuItem OnClick="ShowModalLogIn">@Localizer["Login"] <MudIcon Icon="@Icons.Material.Filled.Login" /> </MudMenuItem>

</MudStack>

</MudContainer>

</NotAuthorized>

</AuthorizeView>

1. Llamamos el nuevo componente desde el **MainLayout**:.

<MudMenu Icon="@Icons.Material.Filled.Settings"

Color="Color.Inherit"

ActivationEvent="@MouseEvent.MouseOver"

AnchorOrigin="Origin.BottomRight"

TransformOrigin="Origin.TopRight">

<AuthLinks />

</MudMenu>

1. Probamos lo que llevamos y hacemos el commit.

## Confirmar el registro de usuarios

1. Agregar los siguientes literales:

| ERR006 | Incorrect email or password. | Email o contraseña incorrectos. |
| --- | --- | --- |
| ERR007 | You have exceeded the maximum number of attempts, your account is blocked, please try again in 5 minutes. | Ha superado el máximo número de intentos, su cuenta está bloqueada, intente de nuevo en 5 minutos. |
| ERR008 | The user has not been enabled, you must follow the instructions in the email sent to enable the user. | El usuario no ha sido habilitado, debes de seguir las instrucciones del correo enviado para poder habilitar el usuario. |
| ConfirmedEmailMessage | Thank you for confirming your email, you can now log in to the system. | Gracias por confirmar su email, ahora puedes ingresar al sistema. |
| ConfirmEmail | Email confirmation | Confirmación de email |
| ConfirmEmailMessage | Press the button to confirm your account. | Presione el botón para confirmar su cuenta. |

1. Agregamos estos métodos al **IUsersRepository**:

Task<User> GetUserAsync(Guid userId);

Task<string> GenerateEmailConfirmationTokenAsync(User user);

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

1. Agregamos estos métodos al **UsersRepository** (primero inyectamos el **IFileStorage**):

public async Task<User> GetUserAsync(Guid userId)

{

var user = await \_context.Users

.Include(u => u.Country)

.FirstOrDefaultAsync(x => x.Id == userId.ToString());

return user!;

}

public async Task<string> GenerateEmailConfirmationTokenAsync(User user)

{

return await \_userManager.GenerateEmailConfirmationTokenAsync(user);

}

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

{

return await \_userManager.ConfirmEmailAsync(user, token);

}

public async Task<SignInResult> LoginAsync(LoginDTO model)

{

return await \_signInManager.PasswordSignInAsync(model.Email, model.Password, false, true);

}

1. Agregamos estos métodos al **IUsersUnitOfWork**:

Task<User> GetUserAsync(Guid userId);

Task<string> GenerateEmailConfirmationTokenAsync(User user);

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

1. Agregamos estos métodos al **UsersUnitOfWork**:

public async Task<User> GetUserAsync(Guid userId) => await \_usersRepository.GetUserAsync(userId);

public async Task<string> GenerateEmailConfirmationTokenAsync(User user) => await \_usersRepository.GenerateEmailConfirmationTokenAsync(user);

public async Task<IdentityResult> ConfirmEmailAsync(User user, string token) => await \_usersRepository.ConfirmEmailAsync(user, token);

1. Cambiamos la configuración de usuarios en el **Program** del **Backend**:

builder.Services.AddIdentity<User, IdentityRole>(x =>

{

x.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;

x.SignIn.RequireConfirmedEmail = true;

x.User.RequireUniqueEmail = true;

x.Password.RequireDigit = false;

x.Password.RequiredUniqueChars = 0;

x.Password.RequireLowercase = false;

x.Password.RequireNonAlphanumeric = false;

x.Password.RequireUppercase = false;

x.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);

x.Lockout.MaxFailedAccessAttempts = 3;

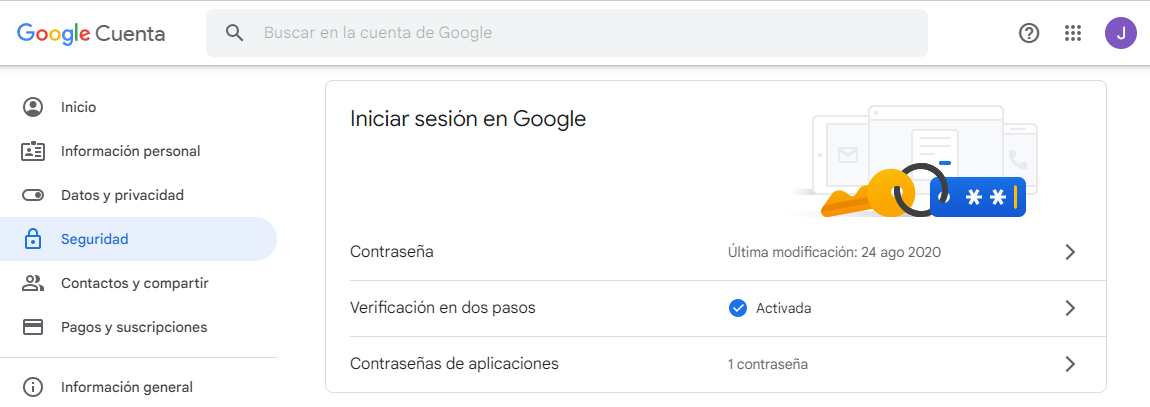
x.Lockout.AllowedForNewUsers = true;

})

.AddEntityFrameworkStores<DataContext>()

.AddDefaultTokenProviders();

1. Verificamos que la cuenta de Gmail con la que vamos a mandar los correos tenga lo siguiente:



1. Adicionamos estos parámetros a la configuración del **Backend**:

"Mail": {

"From": "{your email}",

"NameEs": "Soporte Polla - Aplicación Predicciones Fubtboleras",

"NameEn": "Fantasy Support - Football Predictions Application",

"SubjectConfirmationEs": "Pollas - Confirmación de cuenta",

"SubjectConfirmationEn": "Fantasy - Account confirmation",

"BodyConfirmationEs": "<h1>Pollas - Confirmación de cuenta</h1><p>Para habilitar el usuario, por favor hacer clic en el boton <b><a href ={0}>Confirmar Email</a></b></p>",

"BodyConfirmationEn": "<h1>Fantasy - Account confirmation</h1><p>To enable the user, please click clic on button <b><a href ={0}>Confirm Email</a></b></p>",

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

"Port": 587,

"Password": "{your password}"

},

"Url Frontend": "localhost:{your port}"

**Nota**: reemplazar los your por tus datos.

1. Adicionamos el nuget “**Mailkit**” al proyecto **Backend**:
2. En los **Helpers** del **Backend** adicionamos la interzar **IMailHelper**:

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Helpers;

public interface IMailHelper

{

ActionResponse<string> SendMail(string toName, string toEmail, string subject, string body, string language);

}

1. Luego agregamos la implementation **MailHelper**:

using Fantasy.Shared.Responses;

using MailKit.Net.Smtp;

using MimeKit;

namespace Fantasy.Backend.Helpers;

public class MailHelper : IMailHelper

{

private readonly IConfiguration \_configuration;

public MailHelper(IConfiguration configuration)

{

\_configuration = configuration;

}

public ActionResponse<string> SendMail(string toName, string toEmail, string subject, string body, string language)

{

try

{

var from = \_configuration["Mail:From"];

var name = \_configuration["Mail:NameEn"];

if (language == "es")

{

name = \_configuration["Mail:NameEs"];

}

var smtp = \_configuration["Mail:Smtp"];

var port = \_configuration["Mail:Port"];

var password = \_configuration["Mail:Password"];

var message = new MimeMessage();

message.From.Add(new MailboxAddress(name, from));

message.To.Add(new MailboxAddress(toName, toEmail));

message.Subject = subject;

BodyBuilder bodyBuilder = new BodyBuilder

{

HtmlBody = body

};

message.Body = bodyBuilder.ToMessageBody();

using (var client = new SmtpClient())

{

client.Connect(smtp, int.Parse(port!), false);

client.Authenticate(from, password);

client.Send(message);

client.Disconnect(true);

}

return new ActionResponse<string> { WasSuccess = true };

}

catch (Exception ex)

{

return new ActionResponse<string>

{

WasSuccess = false,

Message = ex.Message,

};

}

}

}

1. Configuramos la inyección del servicio:

builder.Services.AddScoped<IMailHelper, MailHelper>();

1. Agregamos esta propiedad al **UserDTO**:

public string Language { get; set; } = null!;

1. Modificamos la entidad **User**:

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public Country Country { get; set; } = null!;

1. Modificamos el CountriesController para hacer accesible el método combo sin token:

[AllowAnonymous]

[HttpGet("combo")]

1. Modificamos el **AccountsController** (primero inyectamos el **IMailHelper**):

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

namespace Fantasy.Backend.Controllers;

[ApiController]

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

public class AccountsController : ControllerBase

{

private readonly IUsersUnitOfWork \_usersUnitOfWork;

private readonly IConfiguration \_configuration;

private readonly IMailHelper \_mailHelper;

private readonly DataContext \_context;

public AccountsController(IUsersUnitOfWork usersUnitOfWork, IConfiguration configuration, IMailHelper mailHelper, DataContext context)

{

\_usersUnitOfWork = usersUnitOfWork;

\_configuration = configuration;

\_mailHelper = mailHelper;

\_context = context;

}

[HttpPost("CreateUser")]

public async Task<IActionResult> CreateUser([FromBody] UserDTO model)

{

var country = await \_context.Countries.FindAsync(model.CountryId);

if (country == null)

{

return BadRequest("ERR004");

}

User user = model;

user.Country = country;

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

if (result.Succeeded)

{

await \_usersUnitOfWork.AddUserToRoleAsync(user, user.UserType.ToString());

var response = await SendConfirmationEmailAsync(user, model.Language);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

return BadRequest(result.Errors.FirstOrDefault());

}

[HttpGet("ConfirmEmail")]

public async Task<IActionResult> ConfirmEmailAsync(string userId, string token)

{

token = token.Replace(" ", "+");

var user = await \_usersUnitOfWork.GetUserAsync(new Guid(userId));

if (user == null)

{

return NotFound();

}

var result = await \_usersUnitOfWork.ConfirmEmailAsync(user, token);

if (!result.Succeeded)

{

return BadRequest(result.Errors.FirstOrDefault());

}

return NoContent();

}

[HttpPost("Login")]

public async Task<IActionResult> LoginAsync([FromBody] LoginDTO model)

{

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

if (result.Succeeded)

{

var user = await \_usersUnitOfWork.GetUserAsync(model.Email);

return Ok(BuildToken(user));

}

if (result.IsLockedOut)

{

return BadRequest("ERR007");

}

if (result.IsNotAllowed)

{

return BadRequest("ERR008");

}

return BadRequest("ERR006");

}

private TokenDTO BuildToken(User user)

{

var claims = new List<Claim>

{

new(ClaimTypes.Name, user.Email!),

new(ClaimTypes.Role, user.UserType.ToString()),

new("FirstName", user.FirstName),

new("LastName", user.LastName),

new("Photo", user.Photo ?? string.Empty),

new("CountryId", user.Country.Id.ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["jwtKey"]!));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var expiration = DateTime.UtcNow.AddDays(30);

var token = new JwtSecurityToken(

issuer: null,

audience: null,

claims: claims,

expires: expiration,

signingCredentials: credentials);

return new TokenDTO

{

Token = new JwtSecurityTokenHandler().WriteToken(token),

Expiration = expiration

};

}

public async Task<ActionResponse<string>> SendConfirmationEmailAsync(User user, string language)

{

var myToken = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = Url.Action("ConfirmEmail", "accounts", new

{

userid = user.Id,

token = myToken

}, HttpContext.Request.Scheme, \_configuration["Url Frontend"]);

if (language == "es")

{

return \_mailHelper.SendMail(user.FullName, user.Email!, \_configuration["Mail:SubjectConfirmationEs"]!, string.Format(\_configuration["Mail:BodyConfirmationEs"]!, tokenLink), language);

}

return \_mailHelper.SendMail(user.FullName, user.Email!, \_configuration["Mail:SubjectConfirmationEn"]!, string.Format(\_configuration["Mail:BodyConfirmationEn"]!, tokenLink), language);

}

}

1. Dentro de **Pages/Auth** creamos la página **ConfirmEmail.razor** y **ConfirmEmail.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class ConfirmEmail

{

private string? message;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string UserId { get; set; } = string.Empty;

[Parameter, SupplyParameterFromQuery] public string Token { get; set; } = string.Empty;

protected async Task ConfirmAccountAsync()

{

var responseHttp = await Repository.GetAsync($"/api/accounts/ConfirmEmail/?userId={UserId}&token={Token}");

if (responseHttp.Error)

{

message = await responseHttp.GetErrorMessageAsync();

NavigationManager.NavigateTo("/");

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Snackbar.Add(Localizer["ConfirmedEmailMessage"], Severity.Success);

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

DialogService.Show<Login>(Localizer["Login"], closeOnEscapeKey);

}

}

1. Luego modicamos **ConfirmEmail.razor**:

@page "/api/accounts/ConfirmEmail"

<MudPaper Class="confirmation-container p-4 shadow-sm">

<MudGrid>

<MudItem xs="12" Class="text-center mb-4">

<MudText Typo="Typo.h3">@Localizer["ConfirmEmail"]</MudText>

</MudItem>

<MudItem xs="12" Class="text-center mb-4">

<MudText Typo="Typo.body1">@Localizer["ConfirmEmailMessage"]</MudText>

</MudItem>

<MudItem xs="12" Class="text-center">

<MudButton Variant="Variant.Filled" Color="Color.Primary" OnClick="ConfirmAccountAsync">@Localizer["ConfirmEmail"]</MudButton>

</MudItem>

</MudGrid>

</MudPaper>

1. Borramos los usuarios de la base de datos. Dentro de **Data** creamos el script **DeleteUsers.sql** con las siguientes intrucciones:

DELETE FROM AspNetUserRoles

DELETE FROM AspNetUsers

1. Modificamos el alimentador de la base de datos:

private async Task<User> CheckUserAsync(string firstName, string lastName, string email, string phone, UserType userType)

{

var user = await \_usersUnitOfWork.GetUserAsync(email);

if (user == null)

{

var country = await \_context.Countries.FirstOrDefaultAsync(x => x.Name == "Colombia");

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Country = country!,

UserType = userType,

};

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

await \_usersUnitOfWork.AddUserToRoleAsync(user, userType.ToString());

var token = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

await \_usersUnitOfWork.ConfirmEmailAsync(user, token);

}

return user;

}

1. Hacemos el **commit** no podemos probar hasta que no registremos los usuarios.

## Implementando el registro de usuarios

1. Agregar los siguientes literales:

| AdminRegister | Administrator Registration | Registro de Administrador |
| --- | --- | --- |
| UserRegister | User Registration | Registro de Usuario |
| SendEmailConfirmationMessage | Your account has been created successfully. An email has been sent to you with instructions on how to activate your account. | Su cuenta ha sido creada con exito. Se te ha enviado un correo electrónico con las instrucciones para activar tu usuario. |
| PhoneNumber | Phone Number | Número de teléfono |
| PleaseWait | Please wait... | Por favor espera... |
| EmailAlreadyExists | The email you entered already exists. | El correo que ingresaste ya existe. |

1. Modificamos el **Loading.razor.cs**:

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Shared;

public partial class Loading

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public string? Label { get; set; }

protected override void OnParametersSet()

{

base.OnParametersSet();

if (string.IsNullOrEmpty(Label))

{

Label = Localizer["PleaseWait"];

}

}

}

1. Modificamos el **Loading.razor**:

<MudCard>

<div class="overlay d-flex flex-column justify-content-center align-items-center p-3">

<MudProgressCircular Indeterminate="true" Color="Color.Primary" Class="mb-3" />

<MudText Typo="Typo.h5">@Label</MudText>

</div>

</MudCard>

1. Dentro de **Pages** en la carpeta **Auth** creamos el componente **Register.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Services;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Enums;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth

{

public partial class Register

{

private UserDTO userDTO = new();

private List<Country>? countries;

private bool loading;

private string? imageUrl;

private string? titleLabel;

private Country selectedCountry = new();

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ILoginService LogInService { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public bool IsAdmin { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

protected override void OnParametersSet()

{

base.OnParametersSet();

titleLabel = IsAdmin ? Localizer["AdminRegister"] : Localizer["UserRegister"];

}

private void ImageSelected(string imageBase64)

{

userDTO.Photo = imageBase64;

imageUrl = null;

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Country>>("/api/countries/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

countries = responseHttp.Response;

}

private void CountryChanged(Country country)

{

selectedCountry = country;

}

private async Task<IEnumerable<Country>> SearchCountries(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return countries!;

}

return countries!

.Where(c => c.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private void ReturnAction()

{

NavigationManager.NavigateTo("/");

}

private async Task CreateUserAsync()

{

if (!ValidateForm())

{

return;

}

userDTO.UserType = UserType.User;

userDTO.UserName = userDTO.Email;

userDTO.Country = selectedCountry;

userDTO.CountryId = selectedCountry.Id;

userDTO.Language = System.Globalization.CultureInfo.CurrentCulture.Name.Substring(0, 2);

if (IsAdmin)

{

userDTO.UserType = UserType.Admin;

}

loading = true;

var responseHttp = await Repository.PostAsync<UserDTO>("/api/accounts/CreateUser", userDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

if (message!.Contains("DuplicateUserName"))

{

Snackbar.Add(Localizer["EmailAlreadyExists"], Severity.Error);

return;

}

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

NavigationManager.NavigateTo("/");

await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["SendEmailConfirmationMessage"],

Icon = SweetAlertIcon.Info,

});

}

private bool ValidateForm()

{

var hasErrors = false;

if (string.IsNullOrEmpty(userDTO.FirstName))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["FirstName"])), Severity.Error);

hasErrors = true;

}

if (string.IsNullOrEmpty(userDTO.LastName))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["LastName"])), Severity.Error);

hasErrors = true;

}

if (string.IsNullOrEmpty(userDTO.PhoneNumber))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["PhoneNumber"])), Severity.Error);

hasErrors = true;

}

if (string.IsNullOrEmpty(userDTO.Email))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["Email"])), Severity.Error);

hasErrors = true;

}

if (string.IsNullOrEmpty(userDTO.Password))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["Password"])), Severity.Error);

hasErrors = true;

}

if (string.IsNullOrEmpty(userDTO.PasswordConfirm))

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["PasswordConfirm"])), Severity.Error);

hasErrors = true;

}

if (selectedCountry.Id == 0)

{

Snackbar.Add(string.Format(Localizer["RequiredField"], string.Format(Localizer["Country"])), Severity.Error);

hasErrors = true;

}

return !hasErrors;

}

}

}

1. Luego modificamos el **Register.razor**:

@page "/Register"

@if (loading)

{

<Loading />

}

else

{

<MudCard Class="p-2">

<MudItem>

<MudText Typo="Typo.h5">@titleLabel</MudText>

</MudItem>

<EditForm Model="userDTO">

<DataAnnotationsValidator />

<MudGrid>

<MudItem xs="12" sm="6">

<MudCardContent>

<MudTextField Label="@Localizer["FirstName"]"

@bind-Value="userDTO.FirstName"

For="@(() => userDTO.FirstName)" />

<MudTextField Label="@Localizer["LastName"]"

@bind-Value="userDTO.LastName"

For="@(() => userDTO.LastName)" />

<MudTextField Label="@Localizer["PhoneNumber"]"

@bind-Value="userDTO.PhoneNumber"

For="@(() => userDTO.PhoneNumber)"

InputType="InputType.Telephone" />

<MudTextField Label="@Localizer["Email"]"

@bind-Value="userDTO.Email"

For="@(() => userDTO.Email)"

InputType="InputType.Email" />

<MudTextField Label="@Localizer["Password"]"

InputType="InputType.Password"

@bind-Value="userDTO.Password"

For="@(() => userDTO.Password)" />

<MudTextField Label="@Localizer["PasswordConfirm"]"

InputType="InputType.Password"

@bind-Value="userDTO.PasswordConfirm"

For="@(() => userDTO.PasswordConfirm)" />

</MudCardContent>

</MudItem>

<MudItem xs="12" sm="6">

<MudCardContent>

<MudAutocomplete T="Country"

Label=@Localizer["Country"]

Placeholder=@Localizer["SelectACountry"]

SearchFunc="SearchCountries"

Value="selectedCountry"

ValueChanged="CountryChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

</MudCardContent>

<MudItem xs="12" sm="6">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</MudItem>

</MudItem>

<MudStack Class="m-2" Row="true">

<MudButton Variant="Variant.Outlined" StartIcon="@Icons.Material.Filled.ArrowBack" Color="Color.Info" OnClick="ReturnAction" Class="ms-8">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined" StartIcon="@Icons.Material.Filled.Check" Color="Color.Primary" OnClick="CreateUserAsync">

@Localizer["SaveChanges"]

</MudButton>

</MudStack>

</MudGrid>

</EditForm>

</MudCard>

}

1. Probamos y hacemos el **commit**.

## Reenviar correo de confirmación

1. Adicionamos los siguientes literales:

| Send | Send | Enviar |
| --- | --- | --- |
| MailForwarding | Mail forwarding | Reenvío de correo |
| ResendAccountActivationEmail | Resend account activation email | Reenviar correro de activación de cuenta |

1. En **Shared.DTOs** creamos la clase **EmailDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class EmailDTO

{

[Display(Name = "Email", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[EmailAddress(ErrorMessageResourceName = "ValidEmail", ErrorMessageResourceType = typeof(Literals))]

public string Email { get; set; } = null!;

public string Language { get; set; } = null!;

}

1. En el **Backend** creamos este método en el **AccountsController**:

[HttpPost("ResedToken")]

public async Task<IActionResult> ResedTokenAsync([FromBody] EmailDTO model)

{

var user = await \_usersUnitOfWork.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var response = await SendConfirmationEmailAsync(user, model.Language);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

1. Creamos el **ResendConfirmationEmailToken.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class ResendConfirmationEmailToken

{

private EmailDTO emailDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

private async Task ResendConfirmationEmailTokenAsync()

{

emailDTO.Language = System.Globalization.CultureInfo.CurrentCulture.Name.Substring(0, 2);

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/ResedToken", emailDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

MudDialog.Cancel();

NavigationManager.NavigateTo("/");

Snackbar.Add(Localizer["SendEmailConfirmationMessage"], Severity.Success);

}

}

1. Modficamos el **ResendConfirmationEmailToken.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<EditForm Model="emailDTO" OnValidSubmit="ResendConfirmationEmailTokenAsync">

<DataAnnotationsValidator />

<MudTextField Label=@Localizer["Email"] @bind-Value="@emailDTO.Email" InputType="InputType.Email" Class="mb-3" />

<ValidationMessage For="@(() => emailDTO.Email)" />

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Send" Color="Color.Primary" ButtonType="ButtonType.Submit" FullWidth="true">

@Localizer["Send"]

</MudButton>

</EditForm>

</DialogContent>

</MudDialog>

}

1. Modificamos el **Login.razor.cs**:

private void ShowModalResendConfirmationEmail()

{

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true, MaxWidth = MaxWidth.ExtraLarge };

DialogService.Show<ResendConfirmationEmailToken>(Localizer["MailForwarding"], closeOnEscapeKey);

}

1. Modificamos nuestro **Login.razor**:

<DialogActions>

<MudStack Spacing="2" Style="padding: 2rem;">

<MudItem xs="12" sm="12">

<MudLink Href="/Register" Underline="Underline.Always" Style="display: block; text-align: center;">

@Localizer["NotUserYet"]

</MudLink>

</MudItem>

<MudItem xs="12" sm="12">

<MudLink OnClick="ShowModalResendConfirmationEmail" Underline="Underline.Always" Style="display: block; text-align: center;">

@Localizer["ResendAccountActivationEmail"]

</MudLink>

</MudItem>

</MudStack>

</DialogActions>

1. Probamos y hacemos el **commit**.

## Editando el usuario

1. Agregamos los siguientes litarales:

| CurrentPassword | Current Password | Contraseña Actual |
| --- | --- | --- |
| NewPassword | New Password | Nueva Contraseña |
| PasswordChangedSuccessfully | Password Changed Successfully. | Contraseña Modificada con éxito. |
| ChangePassword | Change Password | Cambiar Contraseña |

1. Dentro de **Shared.DTOs** creamos el **ChangePasswordDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class ChangePasswordDTO

{

[DataType(DataType.Password)]

[Display(Name = "CurrentPassword", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string CurrentPassword { get; set; } = null!;

[DataType(DataType.Password)]

[Display(Name = "NewPassword", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string NewPassword { get; set; } = null!;

[Compare("NewPassword", ErrorMessageResourceName = "PasswordAndConfirmationDifferent", ErrorMessageResourceType = typeof(Literals))]

[Display(Name = "PasswordConfirm", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string Confirm { get; set; } = null!;

}

1. Modificamos el **IUsersRepository**:

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

Task<IdentityResult> UpdateUserAsync(User user);

1. Modificamos el **UsersRepository**:

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

{

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

}

public async Task<IdentityResult> UpdateUserAsync(User user)

{

return await \_userManager.UpdateAsync(user);

}

1. Modificamos el **IUsersUnitOfWork**:

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

Task<IdentityResult> UpdateUserAsync(User user);

1. Modificamos el **UsersUnitOfWork**:

public async Task<IdentityResult> ChangePasswordAsync(User user, string currentPassword, string newPassword) => await \_usersRepository.ChangePasswordAsync(user, currentPassword, newPassword);

public async Task<IdentityResult> UpdateUserAsync(User user) => await \_usersRepository.UpdateUserAsync(user);

1. Creamos estos métodos en el **AccountsController** (primero se inyecta el **IFileStorage**):

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPut]

public async Task<IActionResult> PutAsync(User user)

{

try

{

var currentUser = await \_usersUnitOfWork.GetUserAsync(User.Identity!.Name!);

if (currentUser == null)

{

return NotFound();

}

if (!string.IsNullOrEmpty(user.Photo))

{

var photoUser = Convert.FromBase64String(user.Photo);

user.Photo = await \_fileStorage.SaveFileAsync(photoUser, ".jpg", \_container);

}

currentUser.Document = user.Document;

currentUser.FirstName = user.FirstName;

currentUser.LastName = user.LastName;

currentUser.Address = user.Address;

currentUser.PhoneNumber = user.PhoneNumber;

currentUser.Photo = !string.IsNullOrEmpty(user.Photo) && user.Photo != currentUser.Photo ? user.Photo : currentUser.Photo;

currentUser.CityId = user.CityId;

var result = await \_usersUnitOfWork.UpdateUserAsync(currentUser);

if (result.Succeeded)

{

return Ok(BuildToken(currentUser));

}

return BadRequest(result.Errors.FirstOrDefault());

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpGet]

public async Task<IActionResult> GetAsync()

{

return Ok(await \_usersUnitOfWork.GetUserAsync(User.Identity!.Name!));

}

[HttpPost("changePassword")]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

public async Task<IActionResult> ChangePasswordAsync(ChangePasswordDTO model)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

var user = await \_usersUnitOfWork.GetUserAsync(User.Identity!.Name!);

if (user == null)

{

return NotFound();

}

var result = await \_usersUnitOfWork.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (!result.Succeeded)

{

return BadRequest(result.Errors.FirstOrDefault()!.Description);

}

return NoContent();

}

1. Dentro de **Orders. Frontend.Pages** creamos el **ChangePassword.razor** y **ChangePassword.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class ChangePassword

{

private ChangePasswordDTO changePasswordDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

private async Task ChangePasswordAsync()

{

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/changePassword", changePasswordDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

MudDialog.Cancel();

NavigationManager.NavigateTo("/EditUser");

Snackbar.Add(Localizer["PasswordChangedSuccessfully"], Severity.Success);

}

private void ReturnAction()

{

MudDialog.Cancel();

NavigationManager.NavigateTo("/EditUser");

}

}

1. Luego modificamos **ChangePassword.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<EditForm Model="changePasswordDTO">

<DataAnnotationsValidator />

<MudGrid Spacing="3">

<MudItem xs="12">

<MudTextField Label="@Localizer["CurrentPassword"]"

InputType="InputType.Password"

@bind-Value="@changePasswordDTO.CurrentPassword" />

<ValidationMessage For="@(() => changePasswordDTO.CurrentPassword)" />

</MudItem>

<MudItem xs="12">

<MudTextField Label="@Localizer["NewPassword"]"

InputType="InputType.Password"

@bind-Value="@changePasswordDTO.NewPassword" />

<ValidationMessage For="@(() => changePasswordDTO.NewPassword)" />

</MudItem>

<MudItem xs="12">

<MudTextField Label="@Localizer["PasswordConfirm"]"

InputType="InputType.Password"

@bind-Value="@changePasswordDTO.Confirm" />

<ValidationMessage For="@(() => changePasswordDTO.Confirm)" />

</MudItem>

</MudGrid>

</EditForm>

</DialogContent>

<DialogActions>

<MudItem>

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Check" Color="Color.Primary" OnClick="ChangePasswordAsync">

@Localizer["SaveChanges"]

</MudButton>

</MudItem>

<MudItem>

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.ArrowBack" Color="Color.Secondary" OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

</MudItem>

</DialogActions>

</MudDialog>

}

1. Creamos el **EditUser.razor** y **EditUser.razor.cs**:

using System.Net;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Services;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

[Authorize]

public partial class EditUser

{

private User? user;

private List<Country>? countries;

private bool loading = true;

private string? imageUrl;

private Country selectedCountry = new();

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ILoginService LoginService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LoadUserAsyc();

await LoadCountriesAsync();

selectedCountry = user!.Country!;

if (!string.IsNullOrEmpty(user!.Photo))

{

imageUrl = user.Photo;

user.Photo = null;

}

}

private void ShowModal()

{

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

DialogService.Show<ChangePassword>(Localizer["ChangePassword"], closeOnEscapeKey);

}

private async Task LoadUserAsyc()

{

var responseHttp = await Repository.GetAsync<User>($"/api/accounts");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/");

return;

}

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

return;

}

user = responseHttp.Response;

loading = false;

}

private void ImageSelected(string imagenBase64)

{

user!.Photo = imagenBase64;

imageUrl = null;

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Country>>("/api/countries/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

countries = responseHttp.Response;

}

private void CountryChanged(Country country)

{

selectedCountry = country;

}

private async Task<IEnumerable<Country>> SearchCountries(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return countries!;

}

return countries!

.Where(c => c.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private async Task SaveUserAsync()

{

var responseHttp = await Repository.PutAsync<User, TokenDTO>("/api/accounts", user!);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

await LoginService.LoginAsync(responseHttp.Response!.Token);

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

NavigationManager.NavigateTo("/");

}

private void ReturnAction()

{

NavigationManager.NavigateTo("/");

}

}

1. Modificamos **EditUser.razor**:

@page "/EditUser"

@if (loading)

{

<Loading />

}

else

{

<MudCard Class="p-4">

<MudItem>

<MudText Typo="Typo.h5">@Localizer["EditUserProfile"]</MudText>

</MudItem>

<EditForm Model="user" OnValidSubmit="SaveUserAsync">

<DataAnnotationsValidator />

<MudGrid>

<MudItem xs="12" sm="6">

<MudCardContent>

<MudTextField Label="@Localizer["FirstName"]"

@bind-Value="user!.FirstName"

For="@(() => user!.FirstName)" />

<MudTextField Label="@Localizer["LastName"]"

@bind-Value="user.LastName"

For="@(() => user.LastName)" />

<MudTextField Label="@Localizer["PhoneNumber"]"

@bind-Value="user.PhoneNumber"

For="@(() => user.PhoneNumber)"

InputType="InputType.Telephone" />

</MudCardContent>

</MudItem>

<MudItem xs="12" sm="6">

<MudCardContent>

<MudAutocomplete T="Country"

Label=@Localizer["Country"]

Placeholder=@Localizer["SelectACountry"]

SearchFunc="SearchCountries"

Value="selectedCountry"

ValueChanged="CountryChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

</MudCardContent>

<MudItem xs="12" sm="6">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</MudItem>

</MudItem>

<MudGrid Justify="Justify.Center" Class="mt-4">

<MudItem>

<MudButton Variant="Variant.Outlined" StartIcon="@Icons.Material.Filled.Check" Color="Color.Primary" OnClick="SaveUserAsync">

@Localizer["SaveChanges"]

</MudButton>

</MudItem>

<MudItem>

<MudButton Variant="Variant.Outlined" StartIcon="@Icons.Material.Filled.LockReset" Color="Color.Secondary" OnClick="ShowModal">

@Localizer["ChangePassword"]

</MudButton>

</MudItem>

<MudItem>

<MudButton Variant="Variant.Outlined" StartIcon="@Icons.Material.Filled.ArrowBack" Color="Color.Info" OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

</MudItem>

</MudGrid>

</MudGrid>

</EditForm>

</MudCard>

}

1. Probamos y hacemos el **commit**.

## Recuperación de contraseña

1. Adicionamos los siguientes literales:

| RecoverPasswordMessage | An email has been sent to you with instructions on how to recover your password. | Se te ha enviado un correo electrónico con las instrucciones para recuperar su contraseña. |
| --- | --- | --- |
| PasswordRecoveredMessage | Password changed successfully, you can now log in with your new password. | Contraseña cambiada con éxito, ahora puede ingresar con su nueva contraseña. |
| PasswordRecovery | Password Recovery | Recuperación de Contraseña |
| ForgottenYourPassword | Have you forgotten your password? | ¿Has olvidado tu contraseña? |

1. Adicionamos en **Shared.DTOs** la clase **ResetPasswordDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class ResetPasswordDTO

{

[Display(Name = "Email", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[EmailAddress(ErrorMessageResourceName = "ValidEmail", ErrorMessageResourceType = typeof(Literals))]

public string Email { get; set; } = null!;

[DataType(DataType.Password)]

[Display(Name = "NewPassword", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string NewPassword { get; set; } = null!;

[Compare("NewPassword", ErrorMessageResourceName = "PasswordAndConfirmationDifferent", ErrorMessageResourceType = typeof(Literals))]

[Display(Name = "PasswordConfirm", ResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

[StringLength(20, MinimumLength = 6, ErrorMessageResourceName = "LengthField", ErrorMessageResourceType = typeof(Literals))]

public string ConfirmPassword { get; set; } = null!;

public string Token { get; set; } = null!;

}

1. Adicionamos estos métodos al **IUsersRepository**:

Task<string> GeneratePasswordResetTokenAsync(User user);

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

Y la implementación:

public async Task<string> GeneratePasswordResetTokenAsync(User user)

{

return await \_userManager.GeneratePasswordResetTokenAsync(user);

}

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

{

return await \_userManager.ResetPasswordAsync(user, token, password);

}

1. Adicionamos estos métodos al **IUsersUnitOfWork**:

Task<string> GeneratePasswordResetTokenAsync(User user);

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

Y la implementación:

public async Task<string> GeneratePasswordResetTokenAsync(User user) => await \_usersRepository.GeneratePasswordResetTokenAsync(user);

public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password) => await \_usersRepository.ResetPasswordAsync(user, token, password);

1. Adicionamos estos métodos al **AccountController**:

[HttpPost("RecoverPassword")]

public async Task<IActionResult> RecoverPasswordAsync([FromBody] EmailDTO model)

{

var user = await \_usersUnitOfWork.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var response = await SendRecoverEmailAsync(user, model.Language);

if (response.WasSuccess)

{

return NoContent();

}

return BadRequest(response.Message);

}

[HttpPost("ResetPassword")]

public async Task<IActionResult> ResetPasswordAsync([FromBody] ResetPasswordDTO model)

{

var user = await \_usersUnitOfWork.GetUserAsync(model.Email);

if (user == null)

{

return NotFound();

}

var result = await \_usersUnitOfWork.ResetPasswordAsync(user, model.Token, model.NewPassword);

if (result.Succeeded)

{

return NoContent();

}

return BadRequest(result.Errors.FirstOrDefault()!.Description);

}

…

public async Task<ActionResponse<string>> SendRecoverEmailAsync(User user, string language)

{

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

var tokenLink = Url.Action("ResetPassword", "accounts", new

{

userid = user.Id,

token = myToken

}, HttpContext.Request.Scheme, \_configuration["Url Frontend"]);

if (language == "es")

{

return \_mailHelper.SendMail(user.FullName, user.Email!, \_configuration["Mail:SubjectRecoveryEs"]!, string.Format(\_configuration["Mail:BodyRecoveryEs"]!, tokenLink), language);

}

return \_mailHelper.SendMail(user.FullName, user.Email!, \_configuration["Mail:SubjectRecoveryEn"]!, string.Format(\_configuration["Mail:BodyRecoveryEn"]!, tokenLink), language);

}

1. Dentro de **Pages/Auth** creamos el **RecoverPassword.razor** y **RecoverPassword.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class RecoverPassword

{

private EmailDTO emailDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[CascadingParameter] private MudDialogInstance MudDialog { get; set; } = null!;

private async Task SendRecoverPasswordEmailTokenAsync()

{

emailDTO.Language = System.Globalization.CultureInfo.CurrentCulture.Name.Substring(0, 2);

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/RecoverPassword", emailDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

MudDialog.Cancel();

NavigationManager.NavigateTo("/");

Snackbar.Add(Localizer["RecoverPasswordMessage"], Severity.Success);

}

}

1. Creamos el **RecoverPassword.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<EditForm Model="emailDTO" OnValidSubmit="SendRecoverPasswordEmailTokenAsync">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Email"]" @bind-Value="@emailDTO.Email" InputType="InputType.Email" Class="mb-3" />

<ValidationMessage For="@(() => emailDTO.Email)" />

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Send" Color="Color.Primary" ButtonType="ButtonType.Submit" FullWidth="true">

@Localizer["Send"]

</MudButton>

</EditForm>

</DialogContent>

</MudDialog>

}

1. Dentro de **Pages/Auth** creamos el **ResetPassword.razor** y **ResetPassword.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

public partial class ResetPassword

{

private ResetPasswordDTO resetPasswordDTO = new();

private bool loading;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Token { get; set; } = string.Empty;

private async Task ChangePasswordAsync()

{

resetPasswordDTO.Token = Token;

loading = true;

var responseHttp = await Repository.PostAsync("/api/accounts/ResetPassword", resetPasswordDTO);

loading = false;

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Snackbar.Add(Localizer["PasswordRecoveredMessage"], Severity.Success);

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

DialogService.Show<Login>(Localizer["Login"], closeOnEscapeKey);

}

}

1. Creamos el **ResetPassword.razor**:

@page "/api/accounts/ResetPassword"

@if (loading)

{

<Loading />

}

<EditForm Model="resetPasswordDTO" OnValidSubmit="ChangePasswordAsync">

<DataAnnotationsValidator />

<MudGrid>

<MudItem xs="12" sm="7">

<MudCard>

<MudCardHeader>

<CardHeaderContent>

<MudText Typo="Typo.h6">@Localizer["ChangePassword"]</MudText>

</CardHeaderContent>

<CardHeaderActions>

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Check" Color="Color.Primary" ButtonType="ButtonType.Submit">

@Localizer["ChangePassword"]

</MudButton>

</CardHeaderActions>

</MudCardHeader>

<MudCardContent>

<MudItem xs="12" sm="12">

<MudTextField Label="@Localizer["Email"]" @bind-Value="@resetPasswordDTO.Email" InputType="InputType.Email" />

<ValidationMessage For="@(() => resetPasswordDTO.Email)" />

</MudItem>

<MudItem xs="12" sm="12">

<MudTextField Label="@Localizer["NewPassword"]" InputType="InputType.Password" @bind-Value="@resetPasswordDTO.NewPassword" />

<ValidationMessage For="@(() => resetPasswordDTO.NewPassword)" />

</MudItem>

<MudItem xs="12" sm="12">

<MudTextField Label="@Localizer["PasswordConfirm"]" InputType="InputType.Password" @bind-Value="@resetPasswordDTO.ConfirmPassword" />

<ValidationMessage For="@(() => resetPasswordDTO.ConfirmPassword)" />

</MudItem>

</MudCardContent>

</MudCard>

</MudItem>

</MudGrid>

</EditForm>

1. Modificamos el **Login.razor.cs**:

private void ShowModalRecoverPassword()

{

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true, MaxWidth = MaxWidth.ExtraLarge };

DialogService.Show<RecoverPassword>(Localizer["PasswordRecovery"], closeOnEscapeKey);

}

1. Modificamos el **Login.razor**:

<MudDialog>

<DialogContent>

<EditForm Model="loginDTO" OnValidSubmit="LoginAsync">

<DataAnnotationsValidator />

<MudGrid Class="mb-4">

<MudItem xs="12" sm="12">

<MudTextField Label="Email" @bind-Value="@loginDTO.Email" InputType="InputType.Email" />

<ValidationMessage For="@(() => loginDTO.Email)" />

</MudItem>

<MudItem xs="12" sm="12">

<MudTextField Label="Contraseña" @bind-Value="@loginDTO.Password" InputType="InputType.Password" />

<ValidationMessage For="@(() => loginDTO.Password)" />

</MudItem>

</MudGrid>

<MudGrid Class="mb-4">

<MudItem xs="12" sm="6" Class="d-flex justify-content-center">

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Login" Color="Color.Primary" ButtonType="ButtonType.Submit" FullWidth="true">

@Localizer["Login"]

</MudButton>

</MudItem>

<MudItem xs="12" sm="6" Class="d-flex justify-content-center">

<MudButton Variant="Variant.Filled" StartIcon="@Icons.Material.Filled.Cancel" Color="Color.Error" OnClick="CloseModal" FullWidth="true">

@Localizer["Cancel"]

</MudButton>

</MudItem>

</MudGrid>

</EditForm>

<MudStack Spacing="2" AlignItems="AlignItems.Center" Justify="Justify.Center">

<MudLink Href="/Register" Underline="Underline.Always" Color="Color.Info" Class="mt-4">

@Localizer["NotUserYet"]

</MudLink>

<MudLink OnClick="ShowModalRecoverPassword" Underline="Underline.Always" Color="Color.Secondary">

@Localizer["ForgottenYourPassword"]

</MudLink>

<MudLink OnClick="ShowModalResendConfirmationEmail" Underline="Underline.Always" Color="Color.Warning">

@Localizer["ResendAccountActivationEmail"]

</MudLink>

</MudStack>

</DialogContent>

</MudDialog>

1. Probamos y hacemos el **commit**.

# CRUDs Parte II

## Creando el controlador de torneos

1. Adicionamos los siguientes literales:

| Tournament | Tournament | Torneo |
| --- | --- | --- |
| Tournaments | Tournaments | Torneos |
| IsActive | Is Active | Esta Activo |
| Remarks | Remarks | Comentarios |

1. Creamos la entidad **Tournament**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.Entities;

public class Tournament

{

public int Id { get; set; }

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

public string? Image { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

[Display(Name = "Remarks", ResourceType = typeof(Literals))]

public string? Remarks { get; set; }

public string ImageFull => string.IsNullOrEmpty(Image) ? "/images/NoImage.png" : Image;

}

1. Creamos la entidad **TournamentTeam**:

namespace Fantasy.Shared.Entities;

public class TournamentTeam

{

public int Id { get; set; }

public Tournament Tournament { get; set; } = null!;

public int TournamentId { get; set; }

public Team Team { get; set; } = null!;

public int TeamId { get; set; }

}

1. Modificamos la entidad **Tournament**:

public ICollection<TournamentTeam>? TournamentTeams { get; set; }

public int TeamsCount => TournamentTeams == null ? 0 : TournamentTeams.Count;

1. Modificamos la entidad **Team**:

public ICollection<TournamentTeam>? TournamentTeams { get; set; }

public int TournamentsCount => TournamentTeams == null ? 0 : TournamentTeams.Count;

1. Modificamos el **DataContext**:

public DbSet<Country> Countries { get; set; }

public DbSet<Team> Teams { get; set; }

public DbSet<Tournament> Tournaments { get; set; }

public DbSet<TournamentTeam> TournamentTeams { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Country>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<Team>().HasIndex(x => new { x.CountryId, x.Name }).IsUnique();

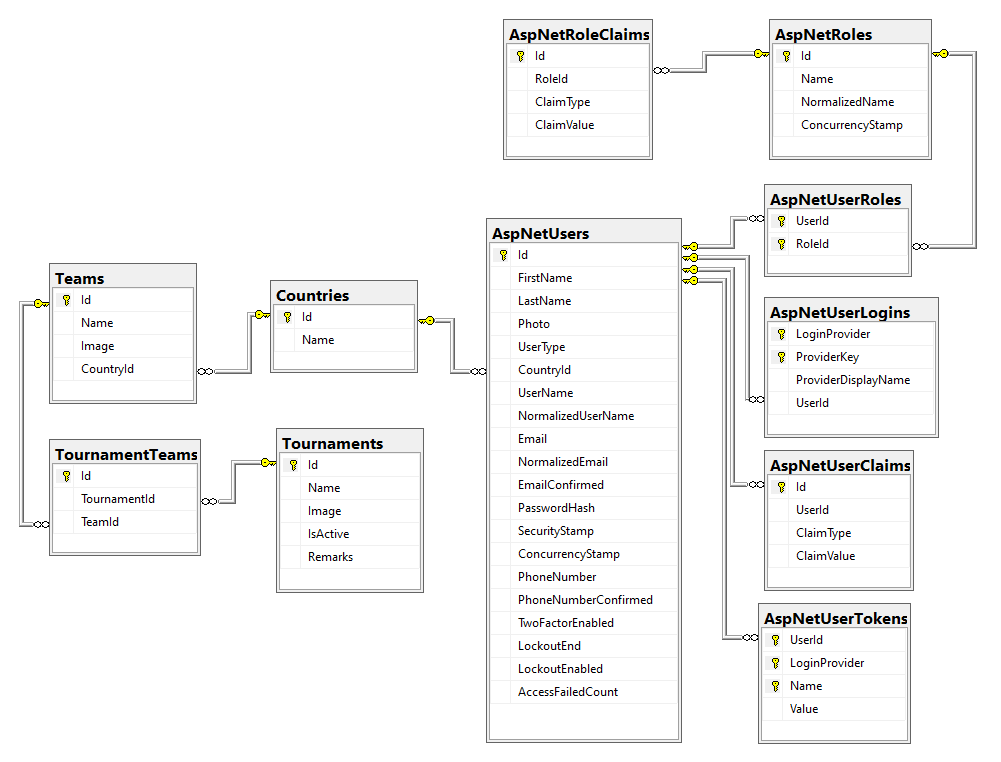
modelBuilder.Entity<Tournament>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<TournamentTeam>().HasIndex(x => new { x.TournamentId, x.TeamId }).IsUnique();

DisableCascadingDelete(modelBuilder);

}

1. Adicionamos la migración y actualizamos la base de datos.
2. Asi llevamos nuestra base de datos:



1. Modificamos el **SeedDb**:

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckTeamsAsync();

await CheckRolesAsync();

await CheckUserAsync("Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", UserType.Admin);

await CheckTournamentsAsync();

}

private async Task CheckTournamentsAsync()

{

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

{

var colombia = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Colombia")!;

var peru = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Peru");

var ecuador = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Ecuador");

var venezuela = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Venezuela");

var brazil = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Brazil");

var argentina = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Argentina");

var uruguay = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Uruguay");

var chile = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Chile");

var bolivia = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Bolivia");

var paraguay = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Paraguay");

var unitedStates = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "United States");

var canada = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Canada");

var mexico = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Mexico");

var panama = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Panama");

var costaRica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Costa Rica ");

var honduras = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Honduras");

var jamaica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Jamaica");

var guatemala = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Guatemala");

var barbados = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Barbados");

var dominica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Dominica");

var copaAmerica = new Tournament

{

IsActive = true,

Name = "Copa America - 2025",

TournamentTeams =

[

new TournamentTeam { Team = colombia! },

new TournamentTeam { Team = peru! },

new TournamentTeam { Team = ecuador! },

new TournamentTeam { Team = venezuela! },

new TournamentTeam { Team = brazil! },

new TournamentTeam { Team = argentina! },

new TournamentTeam { Team = uruguay! },

new TournamentTeam { Team = chile! },

new TournamentTeam { Team = bolivia! },

new TournamentTeam { Team = paraguay! },

new TournamentTeam { Team = unitedStates! },

new TournamentTeam { Team = canada! },

]

};

var copaOro = new Tournament

{

IsActive = true,

Name = "Copa Oro - 2025",

TournamentTeams =

[

new TournamentTeam { Team = unitedStates! },

new TournamentTeam { Team = canada! },

new TournamentTeam { Team = mexico! },

new TournamentTeam { Team = panama! },

new TournamentTeam { Team = costaRica! },

new TournamentTeam { Team = honduras! },

new TournamentTeam { Team = jamaica! },

new TournamentTeam { Team = guatemala! },

new TournamentTeam { Team = barbados! },

new TournamentTeam { Team = dominica! },

new TournamentTeam { Team = colombia! },

new TournamentTeam { Team = uruguay! },

]

};

\_context.Tournaments.Add(copaAmerica);

\_context.Tournaments.Add(copaOro);

await \_context.SaveChangesAsync();

}

}

1. Creamos el **TournamentDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class TournamentDTO

{

public int Id { get; set; }

[Display(Name = "Team", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

[Display(Name = "Image", ResourceType = typeof(Literals))]

public string? Image { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

[Display(Name = "Remarks", ResourceType = typeof(Literals))]

public string? Remarks { get; set; }

}

1. Creamos el **TournamentTeamDTO**:

namespace Fantasy.Shared.DTOs;

public class TournamentTeamDTO

{

public int Id { get; set; }

public int TournamentId { get; set; }

public int TeamId { get; set; }

}

1. Creamos el **ITournamentsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces

{

public interface ITournamentsRepository

{

Task<IEnumerable<Tournament>> GetComboAsync();

Task<ActionResponse<Tournament>> AddAsync(TournamentDTO tournamentDTO);

Task<ActionResponse<Tournament>> UpdateAsync(TournamentDTO tournamentDTO);

Task<ActionResponse<Tournament>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync();

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

}

1. Creamos el **TournamentsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class TournamentsRepository : GenericRepository<Tournament>, ITournamentsRepository

{

private readonly DataContext \_context;

private readonly IFileStorage \_fileStorage;

public TournamentsRepository(DataContext context, IFileStorage fileStorage) : base(context)

{

\_context = context;

\_fileStorage = fileStorage;

}

public async Task<ActionResponse<Tournament>> AddAsync(TournamentDTO tournamentDTO)

{

var tournament = new Tournament

{

IsActive = false,

Name = tournamentDTO.Name,

Remarks = tournamentDTO.Remarks,

TournamentTeams = new List<TournamentTeam>()

};

if (!string.IsNullOrEmpty(tournamentDTO.Image))

{

var imageBase64 = Convert.FromBase64String(tournamentDTO.Image!);

tournament.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "tournaments");

}

\_context.Add(tournament);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Tournament>

{

WasSuccess = true,

Result = tournament

};

}

catch (DbUpdateException)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<IEnumerable<Tournament>> GetComboAsync()

{

return await \_context.Tournaments

.Where(x => x.IsActive)

.OrderBy(x => x.Name)

.ToListAsync();

}

public override async Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Tournaments

.Include(x => x.TournamentTeams!)

.ThenInclude(x => x.Team)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Tournament>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<Tournament>> GetAsync(int id)

{

var team = await \_context.Tournaments

.Include(x => x.TournamentTeams!)

.ThenInclude(x => x.Team)

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

if (team == null)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Tournament>

{

WasSuccess = true,

Result = team

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Tournaments.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<Tournament>> UpdateAsync(TournamentDTO tournamentDTO)

{

var currentTeam = await \_context.Tournaments.FindAsync(tournamentDTO.Id);

if (currentTeam == null)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = "ERR005"

};

}

if (!string.IsNullOrEmpty(tournamentDTO.Image))

{

var imageBase64 = Convert.FromBase64String(tournamentDTO.Image!);

currentTeam.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "tournaments");

}

currentTeam.Name = tournamentDTO.Name;

currentTeam.IsActive = tournamentDTO.IsActive;

currentTeam.Remarks = tournamentDTO.Remarks;

\_context.Update(currentTeam);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Tournament>

{

WasSuccess = true,

Result = currentTeam

};

}

catch (DbUpdateException)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Tournament>

{

WasSuccess = false,

Message = exception.Message

};

}

}

}

1. Creamos el **ITournamentsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface ITournamentsUnitOfWork

{

Task<IEnumerable<Tournament>> GetComboAsync();

Task<ActionResponse<Tournament>> AddAsync(TournamentDTO tournamentDTO);

Task<ActionResponse<Tournament>> UpdateAsync(TournamentDTO tournamentDTO);

Task<ActionResponse<Tournament>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync();

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **TournamentsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations

{

public class TournamentsUnitOfWork : GenericUnitOfWork<Tournament>, ITournamentsUnitOfWork

{

private readonly ITournamentsRepository \_tournamentsRepository;

public TournamentsUnitOfWork(IGenericRepository<Tournament> repository, ITournamentsRepository tournamentsRepository) : base(repository)

{

\_tournamentsRepository = tournamentsRepository;

}

public async Task<ActionResponse<Tournament>> AddAsync(TournamentDTO tournamentDTO) => await \_tournamentsRepository.AddAsync(tournamentDTO);

public async Task<IEnumerable<Tournament>> GetComboAsync() => await \_tournamentsRepository.GetComboAsync();

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_tournamentsRepository.GetTotalRecordsAsync(pagination);

public async Task<ActionResponse<Tournament>> UpdateAsync(TournamentDTO tournamentDTO) => await \_tournamentsRepository.UpdateAsync(tournamentDTO);

public override async Task<ActionResponse<Tournament>> GetAsync(int id) => await \_tournamentsRepository.GetAsync(id);

public override async Task<ActionResponse<IEnumerable<Tournament>>> GetAsync() => await \_tournamentsRepository.GetAsync();

public override async Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination) => await \_tournamentsRepository.GetAsync(pagination);

}

}

1. Creamos el **TournamentsController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class TournamentsController : GenericController<Tournament>

{

private readonly ITournamentsUnitOfWork \_tournamentsUnitOfWork;

public TournamentsController(IGenericUnitOfWork<Tournament> unitOfWork, ITournamentsUnitOfWork tournamentsUnitOfWork) : base(unitOfWork)

{

\_tournamentsUnitOfWork = tournamentsUnitOfWork;

}

[HttpGet]

public override async Task<IActionResult> GetAsync()

{

var response = await \_tournamentsUnitOfWork.GetAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_tournamentsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_tournamentsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_tournamentsUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpGet("combo")]

public async Task<IActionResult> GetComboAsync()

{

return Ok(await \_tournamentsUnitOfWork.GetComboAsync());

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(TournamentDTO tournamentDTO)

{

var action = await \_tournamentsUnitOfWork.AddAsync(tournamentDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutAsync(TournamentDTO tournamentDTO)

{

var action = await \_tournamentsUnitOfWork.UpdateAsync(tournamentDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Matriculamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<ITeamsRepository, TeamsRepository>();

builder.Services.AddScoped<ITeamsUnitOfWork, TeamsUnitOfWork>();

builder.Services.AddScoped<ITournamentsRepository, TournamentsRepository>();

builder.Services.AddScoped<ITournamentsUnitOfWork, TournamentsUnitOfWork>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

1. Probamos por swagger y hacemos el commit.

## Creando el controlador de torneos/equipos

1. Agregamos el siguiente literal:

| ERR009 | The tournament Id is not valid. | El código de torneo no es válido. |
| --- | --- | --- |

1. Creamos el **ITournamentTeamsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface ITournamentTeamsRepository

{

Task<IEnumerable<Tournament>> GetComboAsync(int tournamentId);

Task<ActionResponse<Tournament>> AddAsync(TournamentTeamDTO tournamentTeamDTO);

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **TournamentTeamsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class TournamentTeamsRepository : GenericRepository<TournamentTeam>, ITournamentTeamsRepository

{

private readonly DataContext \_context;

public TournamentTeamsRepository(DataContext context) : base(context)

{

\_context = context;

}

public async Task<ActionResponse<TournamentTeam>> AddAsync(TournamentTeamDTO tournamentTeamDTO)

{

var tournament = await \_context.Tournaments.FindAsync(tournamentTeamDTO.TournamentId);

if (tournament == null)

{

return new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = "ERR009"

};

}

var team = await \_context.Teams.FindAsync(tournamentTeamDTO.TeamId);

if (team == null)

{

return new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = "ERR005"

};

}

var tournamentTeam = new TournamentTeam

{

Tournament = tournament,

Team = team,

};

\_context.Add(tournamentTeam);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<TournamentTeam>

{

WasSuccess = true,

Result = tournamentTeam

};

}

catch (DbUpdateException)

{

return new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<IEnumerable<TournamentTeam>> GetComboAsync(int tournamentId)

{

return await \_context.TournamentTeams

.Include(x => x.Team)

.Where(x => x.TournamentId == tournamentId)

.OrderBy(x => x.Team.Name)

.ToListAsync();

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.TournamentTeams.AsQueryable();

queryable = queryable.Where(x => x.TournamentId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Team.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public override async Task<ActionResponse<IEnumerable<TournamentTeam>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.TournamentTeams

.Include(x => x.Team)

.AsQueryable();

queryable = queryable.Where(x => x.TournamentId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Team.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<TournamentTeam>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Team.Name)

.Paginate(pagination)

.ToListAsync()

};

}

}

1. Creamos el **ITournamentTeamsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface ITournamentTeamsUnitOfWork

{

Task<IEnumerable<Tournament>> GetComboAsync(int tournamentId);

Task<ActionResponse<Tournament>> AddAsync(TournamentTeamDTO tournamentTeamDTO);

Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **TournamentTeamsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class TournamentTeamsUnitOfWork : GenericUnitOfWork<TournamentTeam>, ITournamentTeamsUnitOfWork

{

private readonly ITournamentTeamsRepository \_tournamentTeamsRepository;

public TournamentTeamsUnitOfWork(IGenericRepository<TournamentTeam> repository, ITournamentTeamsRepository tournamentTeamsRepository) : base(repository)

{

\_tournamentTeamsRepository = tournamentTeamsRepository;

}

public async Task<ActionResponse<TournamentTeam>> AddAsync(TournamentTeamDTO tournamentTeamDTO) => await \_tournamentTeamsRepository.AddAsync(tournamentTeamDTO);

public async Task<IEnumerable<TournamentTeam>> GetComboAsync(int tournamentId) => await \_tournamentTeamsRepository.GetComboAsync(tournamentId);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_tournamentTeamsRepository.GetTotalRecordsAsync(pagination);

public override async Task<ActionResponse<IEnumerable<TournamentTeam>>> GetAsync(PaginationDTO pagination) => await \_tournamentTeamsRepository.GetAsync(pagination);

}

1. Creamos el **TournamentTeamsController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class TournamentTeamsController : GenericController<TournamentTeam>

{

private readonly ITournamentTeamsUnitOfWork \_tournamentTeamsUnitOfWork;

public TournamentTeamsController(IGenericUnitOfWork<TournamentTeam> unitOfWork, ITournamentTeamsUnitOfWork tournamentTeamsUnitOfWork) : base(unitOfWork)

{

\_tournamentTeamsUnitOfWork = tournamentTeamsUnitOfWork;

}

[HttpGet("combo/{tournamentId}")]

public async Task<IActionResult> GetComboAsync(int tournamentId)

{

return Ok(await \_tournamentTeamsUnitOfWork.GetComboAsync(tournamentId));

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(TournamentTeamDTO tournamentTeamDTO)

{

var action = await \_tournamentTeamsUnitOfWork.AddAsync(tournamentTeamDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_tournamentTeamsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_tournamentTeamsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

}

1. Matriculamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<ITournamentsRepository, TournamentsRepository>();

builder.Services.AddScoped<ITournamentsUnitOfWork, TournamentsUnitOfWork>();

builder.Services.AddScoped<ITournamentTeamsRepository, TournamentTeamsRepository>();

builder.Services.AddScoped<ITournamentTeamsUnitOfWork, TournamentTeamsUnitOfWork>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

1. Probamos por swagger y hacemos el commit.

## Index de torneos

1. Adicionamos los siguientes literales:

| TournamentActive | The tournament is active | El torneo está activo |
| --- | --- | --- |
| TournamentInactive | The tournament is inactive | El torneo está inactivo |
| Activate | Activate | Activar |
| Deactivate | Deactivate | Desactivar |

1. Dentro de **Pages** creamos la carpeta **Tournaments** y dentro de esta creamos el **TournamentCreate.razor.cs** temporal, luego lo completamos:

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class TournamentCreate

{

}

1. Luego modificamos el **TournamentCreate.razor**:

<h3>TournamentCreate</h3>

1. En la carpeta **Tournaments** y dentro de esta creamos el **TournamentEdit.razor.cs** temporal, luego lo completamos:

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class TournamentEdit

{

}

1. Luego modificamos el **TournamentEdit.razor**:

<h3>TournamentEdit</h3>

1. Dentro de **Pages** creamos la carpeta **Tournaments** y dentro de esta creamos el **TournamentsIndex.razor.cs**:

using System.Net;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

[Authorize(Roles = "Admin")]

public partial class TournamentsIndex

{

private List<Tournament>? Tournaments { get; set; }

private MudTable<Tournament> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/tournaments";

private string infoFormat = "{first\_item}-{last\_item} => {all\_items}";

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadTotalRecordsAsync();

}

private void TeamsAction(Tournament tournament)

{

NavigationManager.NavigateTo($"/tournament/teams/{tournament.Id}");

}

private async Task LoadTotalRecordsAsync()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

totalRecords = responseHttp.Response;

loading = false;

}

private async Task<TableData<Tournament>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated/?page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Tournament>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Tournament> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Tournament> { Items = [], TotalItems = 0 };

}

return new TableData<Tournament>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

IDialogReference? dialog;

if (isEdit)

{

var parameters = new DialogParameters

{

{ "Id", id }

};

dialog = DialogService.Show<TournamentEdit>($"{Localizer["Edit"]} {Localizer["Tournament"]}", parameters, options);

}

else

{

dialog = DialogService.Show<TournamentCreate>($"{Localizer["New"]} {Localizer["Tournament"]}", options);

}

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

private async Task DeleteAsync(Tournament team)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Tournament"], team.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrl}/{team.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/tournaments");

}

else

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

}

return;

}

await LoadTotalRecordsAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Modificamos el **TournamentsIndex.razor**:

@page "/tournaments"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@Tournaments"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<div class="d-flex justify-content-between">

<MudText Typo="Typo.h6" Class="me-4"> @Localizer["Tournaments"]</MudText>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info" OnClick="@(() => ShowModalAsync())">

@Localizer["New"]

</MudButton>

</div>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Tournament"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["IsActive"]</MudTh>

<MudTh>@Localizer["Remarks"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>

<MudImage Src="@context.ImageFull" Width="80" />

</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>@context.Remarks</MudTd>

<MudTd>

<MudTooltip Text="@Localizer["Teams"]">

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Primary"

OnClick="@(() => TeamsAction(@context))" style="width: 100px;">

@context.TeamsCount

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Matches"]">

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Sports"

Color="Color.Success"

OnClick="@(() => MatchesAction(@context))" style="width: 100px;">

@context.MatchesCount

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Delete"]">

<MudButton Variant="Variant.Filled"

Color="Color.Error"

OnClick="@(() => DeleteAsync(@context))">

<MudIcon Icon="@Icons.Material.Filled.Delete" />

</MudButton>

</MudTooltip>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **NavMenu.razor**:

<MudNavLink Href="/teams" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Groups">@Localizer["Teams"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/tournaments" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Star">@Localizer["Tournaments"]</MudNavLink>

<MudDivider />

1. Probamos y hacemos el **commit**.

## Creando y Editando Torneos

1. Creamos el **TournamentForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class TournamentForm

{

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(TournamentDTO);

}

[EditorRequired, Parameter] public TournamentDTO TournamentDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

private string? imageUrl;

private string? isActiveMessage;

protected override void OnParametersSet()

{

base.OnParametersSet();

if (!string.IsNullOrEmpty(TournamentDTO.Image))

{

imageUrl = TournamentDTO.Image;

TournamentDTO.Image = null;

isActiveMessage = TournamentDTO.IsActive ? Localizer["TournamentActive"] : Localizer["TournamentInactive"];

}

}

private void ImageSelected(string imagenBase64)

{

TournamentDTO.Image = imagenBase64;

imageUrl = null;

}

private void SetTournamentOff()

{

TournamentDTO.IsActive = false;

isActiveMessage = Localizer["TournamentInactive"];

}

private void SetTournamentOn()

{

TournamentDTO.IsActive = true;

isActiveMessage = Localizer["TournamentActive"];

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Creamos el **TournamentForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Tournament"]"

@bind-Value="@TournamentDTO.Name"

For="@(() => TournamentDTO.Name)"

Class="mb-4" />

<MudTextField Label="@Localizer["Remarks"]"

@bind-Value="@TournamentDTO.Remarks"

For="@(() => TournamentDTO.Remarks)"

Class="mb-4"

Lines="5" />

<MudGrid Justify="Justify.SpaceBetween">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@isActiveMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (TournamentDTO.IsActive)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="SetTournamentOff">

@Localizer["Deactivate"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="SetTournamentOn">

@Localizer["Activate"]

</MudButton>

}

</MudItem>

</MudGrid>

<div class="my-2">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **TournamentCreate.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class TournamentCreate

{

private TournamentForm? tournamentForm;

private TournamentDTO tournamentDTO = new() { IsActive = true };

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/tournaments/full", tournamentDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordCreatedOk"], Severity.Success);

}

private void Return()

{

tournamentForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/tournaments");

}

}

1. Creamos el **TournamentCreate.razor**:

<MudDialog>

<DialogContent>

<TournamentForm @ref="tournamentForm" TournamentDTO="tournamentDTO" OnValidSubmit="CreateAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

1. Probamos.
2. Modificamos el **TournamentEdit.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class TournamentEdit

{

private TournamentDTO? tournamentDTO;

private TournamentForm? tournamentForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Tournament>($"api/tournaments/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("tournaments");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

else

{

var tournament = responseHttp.Response;

tournamentDTO = new TournamentDTO()

{

Id = tournament!.Id,

Name = tournament!.Name,

Image = tournament.Image,

IsActive = tournament!.IsActive,

Remarks = tournament!.Remarks,

};

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/tournaments/full", tournamentDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

tournamentForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("tournaments");

}

}

1. Creamos el **TournamentEdit.razor**:

@if(tournamentDTO is null)

{

<Loading/>

}

else

{

<MudDialog>

<DialogContent>

<TournamentForm @ref="tournamentForm" TournamentDTO="tournamentDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el **commit**.

## Listando equipos del torneo

1. Agregamos lo siguientes literales:

| AddTeamToTournament | Add Team To Tournament | Adicionar Equipo a Torneo |
| --- | --- | --- |
| AddTeam | Add Team | Adicionar Equipo |

1. Creamos el **AddTeam.razor.cs** temporal:

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class AddTeam

{

}

1. Modificamos el **AddTeam.razor** temporal:

<h3>AddTeam</h3>

1. Adicionamos el **TournamentTeams.razor.cs**:

using System.Net;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

[Authorize(Roles = "Admin")]

public partial class TournamentTeams

{

private Tournament? tournament;

private List<TournamentTeam>? tournamentTeams;

private MudTable<TournamentTeam> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrlTournament = "api/tournaments";

private const string baseUrlTournamentTeam = "api/tournamentTeams";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

[Parameter] public int TournamentId { get; set; }

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTournamentAsync()

{

var responseHttp = await Repository.GetAsync<Tournament>($"{baseUrlTournament}/{TournamentId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/tournaments");

return false;

}

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

tournament = responseHttp.Response;

return true;

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

if (tournament is null)

{

var ok = await LoadTournamentAsync();

if (!ok)

{

NoCountry();

return false;

}

}

var url = $"{baseUrlTournamentTeam}/totalRecordsPaginated/?id={TournamentId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<TournamentTeam>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrlTournamentTeam}/paginated?id={TournamentId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<TournamentTeam>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<TournamentTeam> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<TournamentTeam> { Items = [], TotalItems = 0 };

}

return new TableData<TournamentTeam>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo("/tournaments");

}

private async Task ShowModalAsync()

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

var parameters = new DialogParameters

{

{ "Id", TournamentId }

};

var dialog = DialogService.Show<AddTeam>(Localizer["AddTeamToTournament"], parameters, options);

await dialog.Result;

await LoadAsync();

await table.ReloadServerData();

}

private void NoCountry()

{

NavigationManager.NavigateTo("/tournaments");

}

private async Task DeleteAsync(TournamentTeam tournamentTeam)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Team"], tournamentTeam.Team.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrlTournamentTeam}/{tournamentTeam.Id}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

await LoadAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Modificamos el **TournamentTeams.razor**:

@page "/tournament/teams/{TournamentId:int}"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@tournamentTeams"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<MudImage Src="@tournament!.ImageFull" Width="80" Height="80" />

<MudText Typo="Typo.h6" Class="mr-4">@tournament?.Name</MudText>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info"

OnClick="@(() => ShowModalAsync())">

@Localizer["Team"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Team"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Team.Name</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

<MudImage Src="@context.Team.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>

<MudTooltip Text="@Localizer["Delete"]">

<MudButton Variant="Variant.Filled"

Color="Color.Error"

OnClick="@(() => DeleteAsync(@context))">

<MudIcon Icon="@Icons.Material.Filled.Delete" />

</MudButton>

</MudTooltip>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos y hacemos el **commit**.

## Agregar equipos al torneo

1. Agregamos el siguiente literal:

| SelectATeam | -- Select a Team -- | -- Selecciona un Equipo -- |
| --- | --- | --- |

1. Creamos al **AddTeamForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using Microsoft.AspNetCore.Components.Routing;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class AddTeamForm

{

private EditContext editContext = null!;

private Country selectedCountry = new();

private Team selectedTeam = new();

private List<Country>? countries;

private List<Team>? teams;

private string? imageUrl;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[EditorRequired, Parameter] public TournamentTeamDTO TournamentTeamDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

protected override void OnInitialized()

{

editContext = new(TournamentTeamDTO);

}

protected override async Task OnInitializedAsync()

{

await LoadCountriesAsync();

}

private async Task LoadCountriesAsync()

{

var responseHttp = await Repository.GetAsync<List<Country>>("/api/countries/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

countries = responseHttp.Response;

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

private async Task<IEnumerable<Country>> SearchCountry(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return countries!;

}

return countries!

.Where(x => x.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private async Task<IEnumerable<Team>> SearchTeam(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return teams!;

}

return teams!

.Where(x => x.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private async Task CountryChangedAsync(Country country)

{

selectedCountry = country;

selectedTeam = new Team();

teams = null;

await LoadTeamsAsyn(country.Id);

}

private async Task LoadTeamsAsyn(int countryId)

{

var responseHttp = await Repository.GetAsync<List<Team>>($"/api/teams/combo/{countryId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

teams = responseHttp.Response;

}

private void TeamChanged(Team team)

{

selectedTeam = team;

imageUrl = team.ImageFull;

TournamentTeamDTO.TeamId = team.Id;

}

}

1. Modificamos el **AddTeamForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">

<DataAnnotationsValidator />

<MudAutocomplete T="Country"

Label=@Localizer["Country"]

Placeholder=@Localizer["SelectACountry"]

SearchFunc="SearchCountry"

Value="selectedCountry"

ValueChanged="CountryChangedAsync"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")"

Class="mb-2">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<MudAutocomplete T="Team"

Label=@Localizer["Team"]

Placeholder=@Localizer["SelectATeam"]

SearchFunc="SearchTeam"

Value="selectedTeam"

ValueChanged="TeamChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")"

Class="mb-2">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<div class="mb-2">

<MudImage Src="@imageUrl" Width="90" Height="60" />

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **AddTeam.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class AddTeam

{

private TournamentTeamDTO? tournamentTeamDTO;

private AddTeamForm? addTeamForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override void OnParametersSet()

{

base.OnParametersSet();

tournamentTeamDTO = new TournamentTeamDTO()

{

TournamentId = Id,

};

}

private async Task AddAsync()

{

var responseHttp = await Repository.PostAsync("api/TournamentTeams/full", tournamentTeamDTO);

if (responseHttp.Error)

{

var menssageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[menssageError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordCreatedOk"], Severity.Success);

}

private void Return()

{

addTeamForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/tournament/teams/{Id}");

}

}

1. Modificamos el **AddTeam.razor**:

@if (tournamentTeamDTO is null)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<AddTeamForm @ref="addTeamForm" TournamentTeamDTO="tournamentTeamDTO" OnValidSubmit="AddAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el **commit**.

## Creando entidad partidos

1. Adicionamos los siguientes literales:

| Date | Date | Fecha |
| --- | --- | --- |
| Local | Local | Local |
| Visitor | Visitor | Visitante |
| GoalsLocal | Goals Local | Goles del Local |
| GoalsVisitor | Goals Visitor | Goles del Visitante |

1. Creamos la entidad **Match**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.Entities;

public class Match

{

public int Id { get; set; }

public Tournament Tournament { get; set; } = null!;

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

[Display(Name = "Date", ResourceType = typeof(Literals))]

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

public DateTime Date { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

public Team Local { get; set; } = null!;

[Display(Name = "Local", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int LocalId { get; set; }

public Team Visitor { get; set; } = null!;

[Display(Name = "Visitor", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int VisitorId { get; set; }

[Display(Name = "GoalsLocal", ResourceType = typeof(Literals))]

public int? GoalsLocal { get; set; }

[Display(Name = "GoalsVisitor", ResourceType = typeof(Literals))]

public int? GoalsVisitor { get; set; }

[Display(Name = "Date", ResourceType = typeof(Literals))]

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

public DateTime DateLocal => Date.ToLocalTime();

public bool IsClosed { get; set; }

}

1. Modificamos la entidad **Tournament**:

public ICollection<Match>? Matches { get; set; }

public int MatchesCount => Matches == null ? 0 : Matches.Count;

1. Modificamos el **DataContext**:

public DbSet<Country> Countries { get; set; }

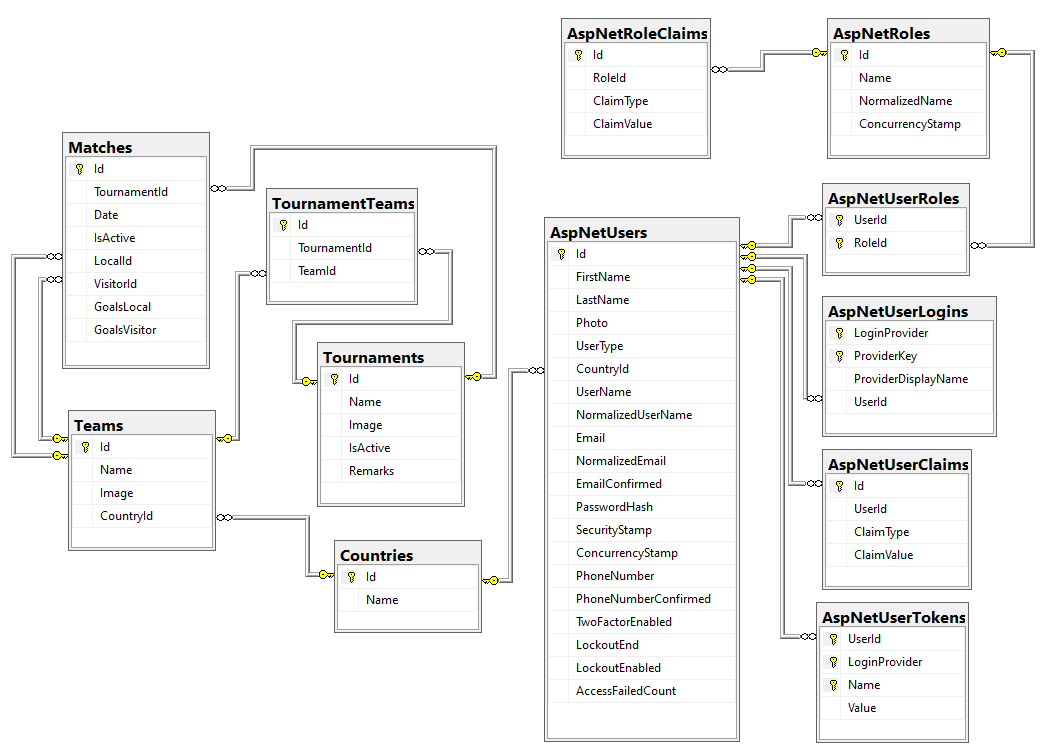
public DbSet<Match> Matches { get; set; }

public DbSet<Team> Teams { get; set; }

public DbSet<Tournament> Tournaments { get; set; }

public DbSet<TournamentTeam> TournamentTeams { get; set; }

1. Adicionamos la migración y la aplicamos.
2. Asi va nuestra base de datos:



1. En la carpeta **Images** creamos la carpeta **Tournaments** y ahí adicionamos las imágenes de los torneos.
2. Creamos dentro de data el Script **DeleteTournaments.sql** y lo ejecutamos:

DELETE FROM Matches

DELETE FROM TournamentTeams

DELETE FROM Tournaments

1. Modificamos el **SeeDb**:

private async Task CheckTournamentsAsync()

{

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

{

var colombia = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Colombia")!;

var peru = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Peru");

var ecuador = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Ecuador");

var venezuela = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Venezuela");

var brazil = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Brazil");

var argentina = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Argentina");

var uruguay = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Uruguay");

var chile = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Chile");

var bolivia = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Bolivia");

var paraguay = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Paraguay");

var unitedStates = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "United States");

var canada = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Canada");

var mexico = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Mexico");

var panama = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Panama");

var costaRica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Costa Rica ");

var honduras = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Honduras");

var jamaica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Jamaica");

var guatemala = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Guatemala");

var barbados = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Barbados");

var dominica = await \_context.Teams.FirstOrDefaultAsync(x => x.Name == "Dominica");

var name = "Copa América - 2025";

var imagePath = string.Empty;

var filePath = $"{Environment.CurrentDirectory}\\Images\\Tournaments\\{name}.png";

if (File.Exists(filePath))

{

var fileBytes = File.ReadAllBytes(filePath);

imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "tournaments");

}

var copaAmerica = new Tournament

{

IsActive = true,

Name = name,

Image = imagePath,

TournamentTeams =

[

new TournamentTeam { Team = colombia! },

new TournamentTeam { Team = peru! },

new TournamentTeam { Team = ecuador! },

new TournamentTeam { Team = venezuela! },

new TournamentTeam { Team = brazil! },

new TournamentTeam { Team = argentina! },

new TournamentTeam { Team = uruguay! },

new TournamentTeam { Team = chile! },

new TournamentTeam { Team = bolivia! },

new TournamentTeam { Team = paraguay! },

new TournamentTeam { Team = unitedStates! },

new TournamentTeam { Team = canada! },

],

Matches =

[

new Match { Date = DateTime.Today.AddDays(1).AddHours(18).ToUniversalTime(), IsActive = true, Local = colombia!, Visitor = peru! },

new Match { Date = DateTime.Today.AddDays(1).AddHours(21).ToUniversalTime(), IsActive = true, Local = ecuador!, Visitor = canada! },

new Match { Date = DateTime.Today.AddDays(2).AddHours(18).ToUniversalTime(), IsActive = true, Local = brazil!, Visitor = chile! },

new Match { Date = DateTime.Today.AddDays(2).AddHours(21).ToUniversalTime(), IsActive = true, Local = bolivia!, Visitor = uruguay! },

new Match { Date = DateTime.Today.AddDays(3).AddHours(18).ToUniversalTime(), IsActive = true, Local = argentina!, Visitor = unitedStates! },

new Match { Date = DateTime.Today.AddDays(3).AddHours(21).ToUniversalTime(), IsActive = true, Local = venezuela!, Visitor = paraguay! },

new Match { Date = DateTime.Today.AddDays(4).AddHours(18).ToUniversalTime(), IsActive = true, Local = canada!, Visitor = colombia! },

new Match { Date = DateTime.Today.AddDays(4).AddHours(21).ToUniversalTime(), IsActive = true, Local = peru!, Visitor = ecuador! },

new Match { Date = DateTime.Today.AddDays(5).AddHours(18).ToUniversalTime(), IsActive = true, Local = uruguay!, Visitor = chile! },

new Match { Date = DateTime.Today.AddDays(5).AddHours(21).ToUniversalTime(), IsActive = true, Local = chile!, Visitor = bolivia! },

new Match { Date = DateTime.Today.AddDays(6).AddHours(18).ToUniversalTime(), IsActive = true, Local = argentina!, Visitor = paraguay! },

new Match { Date = DateTime.Today.AddDays(6).AddHours(21).ToUniversalTime(), IsActive = true, Local = unitedStates!, Visitor = venezuela! },

new Match { Date = DateTime.Today.AddDays(7).AddHours(19).ToUniversalTime(), IsActive = true, Local = peru!, Visitor = canada! },

new Match { Date = DateTime.Today.AddDays(7).AddHours(19).ToUniversalTime(), IsActive = true, Local = colombia!, Visitor = ecuador! },

new Match { Date = DateTime.Today.AddDays(8).AddHours(19).ToUniversalTime(), IsActive = true, Local = chile!, Visitor = uruguay! },

new Match { Date = DateTime.Today.AddDays(8).AddHours(19).ToUniversalTime(), IsActive = true, Local = bolivia!, Visitor = brazil! },

new Match { Date = DateTime.Today.AddDays(9).AddHours(19).ToUniversalTime(), IsActive = true, Local = unitedStates!, Visitor = paraguay! },

new Match { Date = DateTime.Today.AddDays(9).AddHours(19).ToUniversalTime(), IsActive = true, Local = argentina!, Visitor = venezuela! },

]

};

name = "Copa Oro - 2025";

imagePath = string.Empty;

filePath = $"{Environment.CurrentDirectory}\\Images\\Tournaments\\{name}.png";

if (File.Exists(filePath))

{

var fileBytes = File.ReadAllBytes(filePath);

imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "tournaments");

}

var copaOro = new Tournament

{

IsActive = true,

Name = name,

Image = imagePath,

TournamentTeams =

[

new TournamentTeam { Team = unitedStates! },

new TournamentTeam { Team = canada! },

new TournamentTeam { Team = mexico! },

new TournamentTeam { Team = panama! },

new TournamentTeam { Team = costaRica! },

new TournamentTeam { Team = honduras! },

new TournamentTeam { Team = jamaica! },

new TournamentTeam { Team = guatemala! },

new TournamentTeam { Team = barbados! },

new TournamentTeam { Team = dominica! },

new TournamentTeam { Team = colombia! },

new TournamentTeam { Team = uruguay! },

]

};

\_context.Tournaments.Add(copaAmerica);

\_context.Tournaments.Add(copaOro);

await \_context.SaveChangesAsync();

}

}

1. Probamos y hacemos el **commit**.

## Creando el controlador de partidos

1. Adicionamos los siguientes literales:

| ERR010 | The local Id is not valid. | El código del equipo local no es válido. |
| --- | --- | --- |
| ERR011 | The visitor Id is not valid. | El código del equipo visitante no es válido. |
| ERR012 | The match Id is not valid. | El código del partido no es válido. |

1. Creamos el **MatchDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class MatchDTO

{

public int Id { get; set; }

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

[Display(Name = "Date", ResourceType = typeof(Literals))]

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

public DateTime Date { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

[Display(Name = "Local", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int LocalId { get; set; }

[Display(Name = "Visitor", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int VisitorId { get; set; }

[Display(Name = "GoalsLocal", ResourceType = typeof(Literals))]

public int? GoalsLocal { get; set; }

[Display(Name = "GoalsVisitor", ResourceType = typeof(Literals))]

public int? GoalsVisitor { get; set; }

}

1. Creamos el **IMatchesRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces

{

public interface IMatchesRepository

{

Task<ActionResponse<Match>> AddAsync(MatchDTO matchDTO);

Task<ActionResponse<Match>> UpdateAsync(MatchDTO matchDTO);

Task<ActionResponse<Match>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Match>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

}

1. Creamos el **MatchesRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Enums;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class MatchesRepository : GenericRepository<Match>, IMatchesRepository

{

private readonly DataContext \_context;

public MatchesRepository(DataContext context) : base(context)

{

\_context = context;

}

public async Task<ActionResponse<Match>> AddAsync(MatchDTO matchDTO)

{

var tournament = await \_context.Tournaments.FindAsync(matchDTO.TournamentId);

if (tournament == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR009"

};

}

var local = await \_context.Teams.FindAsync(matchDTO.LocalId);

if (local == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR010"

};

}

var visitor = await \_context.Teams.FindAsync(matchDTO.VisitorId);

if (visitor == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR011"

};

}

var match = new Match

{

IsActive = matchDTO.IsActive,

Date = matchDTO.Date,

Tournament = tournament,

Local = local,

Visitor = visitor,

DoublePoints = matchDTO.DoublePoints,

};

\_context.Add(match);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Match>

{

WasSuccess = true,

Result = match

};

}

catch (DbUpdateException)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Matches.AsQueryable();

queryable = queryable.Where(x => x.TournamentId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public override async Task<ActionResponse<IEnumerable<Match>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Matches

.Include(x => x.Tournament)

.Include(x => x.Local)

.Include(x => x.Visitor)

.AsQueryable();

queryable = queryable.Where(x => x.TournamentId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Match>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.IsClosed)

.ThenBy(x => x.Date)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<Match>> GetAsync(int id)

{

var team = await \_context.Matches

.Include(x => x.Tournament)

.Include(x => x.Local)

.Include(x => x.Visitor)

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

if (team == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Match>

{

WasSuccess = true,

Result = team

};

}

public async Task<ActionResponse<Match>> UpdateAsync(MatchDTO matchDTO)

{

var currentMatch = await \_context.Matches.FindAsync(matchDTO.Id);

if (currentMatch == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR012"

};

}

var tournament = await \_context.Tournaments.FindAsync(matchDTO.TournamentId);

if (tournament == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR009"

};

}

var local = await \_context.Teams.FindAsync(matchDTO.LocalId);

if (local == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR010"

};

}

var visitor = await \_context.Teams.FindAsync(matchDTO.VisitorId);

if (visitor == null)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR011"

};

}

currentMatch.Local = local;

currentMatch.Visitor = visitor;

currentMatch.GoalsVisitor = matchDTO.GoalsVisitor;

currentMatch.GoalsLocal = matchDTO.GoalsLocal;

currentMatch.Date = matchDTO.Date;

currentMatch.IsActive = matchDTO.IsActive;

currentMatch.DoublePoints = matchDTO.DoublePoints;

\_context.Update(currentMatch);

try

{

await \_context.SaveChangesAsync();

if (currentMatch.GoalsLocal != null && currentMatch.GoalsVisitor != null)

{

await CloseMatchAsync(currentMatch);

}

return new ActionResponse<Match>

{

WasSuccess = true,

Result = currentMatch

};

}

catch (DbUpdateException)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Match>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task CloseMatchAsync(Match match)

{

match.IsClosed = true;

\_context.Update(match);

var predictions = await \_context.Predictions

.Where(x => x.MatchId == match.Id)

.ToListAsync();

foreach (var prediction in predictions)

{

var points = CalculatePoints(match, prediction);

prediction.Points = points;

\_context.Update(prediction);

}

await \_context.SaveChangesAsync();

}

public int CalculatePoints(Match match, Prediction prediction)

{

int points = 0;

if (prediction.GoalsLocal == null || prediction.GoalsVisitor == null)

{

return points;

}

var matchStatus = GetMatchStatus(match.GoalsLocal!.Value, match.GoalsVisitor!.Value);

var predictionStatus = GetMatchStatus(prediction.GoalsLocal!.Value, prediction.GoalsVisitor!.Value);

if (matchStatus == predictionStatus) points += 5;

if (match.GoalsLocal == prediction.GoalsLocal) points += 2;

if (match.GoalsVisitor == prediction.GoalsVisitor) points += 2;

if (Math.Abs((decimal)match.GoalsLocal! - (decimal)match.GoalsVisitor!) == Math.Abs((decimal)prediction.GoalsLocal! - (decimal)prediction.GoalsVisitor!)) points++;

if (match.DoublePoints) points \*= 2;

return points;

}

public MatchStatus GetMatchStatus(int goalsLocal, int goalsVisitor)

{

if (goalsLocal > goalsVisitor) return MatchStatus.LocalWin;

if (goalsLocal < goalsVisitor) return MatchStatus.VisitorWin;

return MatchStatus.Tie;

}

}

1. Creamos el **IMatchesUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IMatchesUnitOfWork

{

Task<ActionResponse<Match>> AddAsync(MatchDTO matchDTO);

Task<ActionResponse<Match>> UpdateAsync(MatchDTO matchDTO);

Task<ActionResponse<Match>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Match>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **MatchesUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations

{

public class MatchesUnitOfWork : GenericUnitOfWork<Match>, IMatchesUnitOfWork

{

private readonly IMatchesRepository \_matchesRepository;

public MatchesUnitOfWork(IGenericRepository<Match> repository, IMatchesRepository matchesRepository) : base(repository)

{

\_matchesRepository = matchesRepository;

}

public override async Task<ActionResponse<Match>> GetAsync(int id) => await \_matchesRepository.GetAsync(id);

public override async Task<ActionResponse<IEnumerable<Match>>> GetAsync(PaginationDTO pagination) => await \_matchesRepository.GetAsync(pagination);

public async Task<ActionResponse<Match>> AddAsync(MatchDTO matchDTO) => await \_matchesRepository.AddAsync(matchDTO);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_matchesRepository.GetTotalRecordsAsync(pagination);

public async Task<ActionResponse<Match>> UpdateAsync(MatchDTO matchDTO) => await \_matchesRepository.UpdateAsync(matchDTO);

}

}

1. Creamos el **MatchesController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class MatchesController : GenericController<Match>

{

private readonly IMatchesUnitOfWork \_matchesUnitOfWork;

public MatchesController(IGenericUnitOfWork<Match> unitOfWork, IMatchesUnitOfWork matchesUnitOfWork) : base(unitOfWork)

{

\_matchesUnitOfWork = matchesUnitOfWork;

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_matchesUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_matchesUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_matchesUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(MatchDTO matchDTO)

{

var action = await \_matchesUnitOfWork.AddAsync(matchDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutAsync(MatchDTO matchDTO)

{

var action = await \_matchesUnitOfWork.UpdateAsync(matchDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Agregamos la nueva inyección en el **Program**:

builder.Services.AddScoped<ITeamsRepository, TeamsRepository>();

builder.Services.AddScoped<ITeamsUnitOfWork, TeamsUnitOfWork>();

builder.Services.AddScoped<IMatchesRepository, MatchesRepository>();

builder.Services.AddScoped<IMatchesUnitOfWork, MatchesUnitOfWork>();

builder.Services.AddScoped<ITournamentsRepository, TournamentsRepository>();

builder.Services.AddScoped<ITournamentsUnitOfWork, TournamentsUnitOfWork>();

1. Probamos en **Swagger** y hacemos el **commit**.

## Listando partidos del torneo

1. Agregamos lo siguientes literales:

| AddMatchToTournament | Add Match to Tournament | Adicionar partido a torneo |
| --- | --- | --- |
| Match | Match | Partido |
| Matches | Matches | Partidos |

1. Creamos el **AddMatch.razor.cs** temporal:

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class AddMatch

{

}

1. Creamos el **AddMatch.razor** temporal:

<h3>AddMatch</h3>

1. Creamos el **TournamentMatches.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using System.Net;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

[Authorize(Roles = "Admin")]

public partial class TournamentMatches

{

private Tournament? tournament;

private List<Match>? matches;

private MudTable<Match> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrlTournament = "api/tournaments";

private const string baseUrlMatch = "api/matches";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

[Parameter] public int TournamentId { get; set; }

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTournamentAsync()

{

var responseHttp = await Repository.GetAsync<Tournament>($"{baseUrlTournament}/{TournamentId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/tournaments");

return false;

}

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

tournament = responseHttp.Response;

return true;

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

if (tournament is null)

{

var ok = await LoadTournamentAsync();

if (!ok)

{

NoTournament();

return false;

}

}

var url = $"{baseUrlMatch}/totalRecordsPaginated/?id={TournamentId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<Match>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrlMatch}/paginated?id={TournamentId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Match>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Match> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Match> { Items = [], TotalItems = 0 };

}

return new TableData<Match>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo("/tournaments");

}

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

IDialogReference? dialog;

if (isEdit)

{

var parameters = new DialogParameters

{

{ "Id", id }

};

dialog = DialogService.Show<EditMatch>($"{Localizer["Edit"]} {Localizer["Match"]}", parameters, options);

}

else

{

var parameters = new DialogParameters

{

{ "Id", TournamentId }

};

dialog = DialogService.Show<AddMatch>(Localizer["AddMatchToTournament"], parameters, options);

}

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadAsync();

await table.ReloadServerData();

}

}

private void NoTournament()

{

NavigationManager.NavigateTo("/tournaments");

}

private async Task DeleteAsync(Match match)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Match"], $"{match.Local.Name} Vs. {match.Visitor.Name}") }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrlMatch}/{match.Id}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

await LoadAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Creamos el **TournamentMatches.razor**:

@page "/tournament/matches/{TournamentId:int}"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@matches"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<MudImage Src="@tournament!.ImageFull" Width="80" Height="80" />

<MudText Typo="Typo.h6" Class="mr-4">@tournament?.Name</MudText>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info"

OnClick="@(() => ShowModalAsync())">

@Localizer["Match"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Date"]</MudTh>

<MudTh>@Localizer["IsActive"]</MudTh>

<MudTh>@Localizer["Local"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["GoalsLocal"]</MudTh>

<MudTh>@Localizer["GoalsVisitor"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.DateLocal</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>@context.Local.Name</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

<MudImage Src="@context.Local.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsLocal</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsVisitor</MudText>

</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

<MudImage Src="@context.Visitor.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>@context.Visitor.Name</MudTd>

<MudTd>

<MudStack Row="true">

<MudTooltip Text="@Localizer["CloseMatch"]">

<MudButton Variant="Variant.Filled"

Color="Color.Info"

OnClick="@(() => CloseMatchAsync(context.Id))"

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

<MudIcon Icon="@Icons.Material.Filled.Close" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))"

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["Delete"]">

<MudButton Variant="Variant.Filled"

Color="Color.Error"

OnClick="@(() => DeleteAsync(@context))"

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

<MudIcon Icon="@Icons.Material.Filled.Delete" />

</MudButton>

</MudTooltip>

</MudStack>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **TournamentsIndex.razor.cs**:

private void MatchesAction(Tournament tournament)

{

NavigationManager.NavigateTo($"/tournament/matches/{tournament.Id}");

}

1. Modificamos el **TournamentsIndex.razor**:

<HeaderContent>

<MudTh>@Localizer["Tournament"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["IsActive"]</MudTh>

<MudTh>@Localizer["Remarks"]</MudTh>

<MudTh># @Localizer["Teams"]</MudTh>

<MudTh># @Localizer["Matches"]</MudTh>

<MudTh style="width: 300px;">@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudText Style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis; max-width: 200px;">

@context.Name

</MudText>

</MudTd>

<MudTd>

<img src="@context.ImageFull" style="width:80px;" />

</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>

<MudText Style="white-space: nowrap; overflow: hidden; text-overflow: ellipsis; max-width: 340px;">

@context.Remarks

</MudText>

</MudTd>

<MudTd>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Info"

OnClick="@(() => TeamsAction(@context))" style="width: 100px;">

@context.TeamsCount

</MudButton>

</MudTd>

<MudTd>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Sports"

Color="Color.Warning"

OnClick="@(() => MatchesAction(@context))" style="width: 100px;">

@context.MatchesCount

</MudButton>

</MudTd>

<MudTd>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Edit"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

@Localizer["Edit"]

</MudButton>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Delete"

Color="Color.Error"

OnClick=@(() => DeleteAsync(@context))>

@Localizer["Delete"]

</MudButton>

</MudTd>

</RowTemplate>

1. Modificamos el **TournamentsRepository**:

public override async Task<ActionResponse<IEnumerable<Tournament>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Tournaments

.Include(x => x.Matches!)

.Include(x => x.TournamentTeams!)

.ThenInclude(x => x.Team)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Tournament>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

1. Probamos y hacemos el **commit**.

## Agregar partidos al torneo

1. Agregamos los siguientes literales:

| MatchInactive | Match Inactive | El partido está inactivo |
| --- | --- | --- |
| MatchActive | Match Active | El partido está activo |
| SelectDate | Select a Date... | Seleccione una fecha... |
| SelectTime | Select a Time... | Seleccione una hora... |
| MustSelectLocalTeam | You must select a local team. | Debe seleccionar un equipo local. |
| MustSelectVisitorTeam | You must select a visitor team. | Debe seleccionar un equipo visitante. |

1. Creamos al **MatchForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class MatchForm

{

private EditContext editContext = null!;

private Team selectedLocal = new();

private Team selectedVisitor = new();

private List<Team>? teams;

private string? imageUrlLocal;

private string? imageUrlVisitor;

private string? isActiveMessage;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[EditorRequired, Parameter] public MatchDTO MatchDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

private DateTime? selectedDate { get; set; } = DateTime.Now.Date;

private TimeSpan? selectedTime { get; set; } = DateTime.Now.TimeOfDay;

public bool FormPostedSuccessfully { get; set; } = false;

protected override void OnInitialized()

{

base.OnInitialized();

editContext = new(MatchDTO);

}

protected override async Task OnParametersSetAsync()

{

base.OnParametersSet();

await LoadMatchesAsync();

isActiveMessage = MatchDTO.IsActive ? Localizer["MatchActive"] : Localizer["MatchInactive"];

if (MatchDTO.Id != 0)

{

LoadInitialValues();

}

else

{

MatchDTO.Date = DateTime.Now;

}

}

private void LoadInitialValues()

{

var local = teams!.FirstOrDefault(x => x.Id == MatchDTO.LocalId)!;

var visitor = teams!.FirstOrDefault(x => x.Id == MatchDTO.VisitorId)!;

if (local != null)

{

selectedLocal = local;

imageUrlLocal = local.ImageFull;

}

if (visitor != null)

{

selectedVisitor = visitor;

imageUrlVisitor = visitor.ImageFull;

}

selectedDate = MatchDTO.Date.ToLocalTime().Date;

selectedTime = MatchDTO.Date.ToLocalTime().TimeOfDay;

}

private async Task LoadMatchesAsync()

{

var responseHttp = await Repository.GetAsync<List<TournamentTeam>>($"/api/tournamentTeams/combo/{MatchDTO.TournamentId}");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

var tournamentTeams = responseHttp.Response;

teams = tournamentTeams!.Select(t => t.Team).ToList();

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

private async Task<IEnumerable<Team>> SearchTeam(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return teams!;

}

return teams!

.Where(x => x.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private void LocalChanged(Team team)

{

selectedLocal = team;

MatchDTO.LocalId = team.Id;

imageUrlLocal = team.ImageFull;

}

private void VisitorChanged(Team team)

{

selectedVisitor = team;

MatchDTO.VisitorId = team.Id;

imageUrlVisitor = team.ImageFull;

}

private void OnDateChanged(DateTime? newDate)

{

selectedDate = newDate;

UpdateMatchDate();

}

private void OnTimeChanged(TimeSpan? newTime)

{

selectedTime = newTime;

UpdateMatchDate();

}

private void UpdateMatchDate()

{

if (selectedDate.HasValue && selectedTime.HasValue)

{

MatchDTO.Date = selectedDate.Value.Date + selectedTime.Value;

}

}

private void SetTournamentOff()

{

MatchDTO.IsActive = false;

isActiveMessage = Localizer["MatchInactive"];

}

private void SetTournamentOn()

{

MatchDTO.IsActive = true;

isActiveMessage = Localizer["MatchActive"];

}

private async Task OnSubmitAsync()

{

if (ValidateForm())

{

await OnValidSubmit.InvokeAsync(null);

}

}

private bool ValidateForm()

{

var hasErros = false;

if (selectedLocal.Id == 0)

{

Snackbar.Add(Localizer["MustSelectLocalTeam"], Severity.Error);

hasErros = true;

}

if (selectedVisitor.Id == 0)

{

Snackbar.Add(Localizer["MustSelectVisitorTeam"], Severity.Error);

hasErros = true;

}

return !hasErros;

}

}

1. Modificamos el **MatchForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnSubmit="OnSubmitAsync">

<DataAnnotationsValidator />

<MudAutocomplete T="Team"

Label=@Localizer["Local"]

Placeholder=@Localizer["SelectATeam"]

SearchFunc="SearchTeam"

Value="selectedLocal"

ValueChanged="LocalChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")"

Class="mb-2">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<MudAutocomplete T="Team"

Label=@Localizer["Visitor"]

Placeholder=@Localizer["SelectATeam"]

SearchFunc="SearchTeam"

Value="selectedVisitor"

ValueChanged="VisitorChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")"

Class="mb-2">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<MudDatePicker Label=@Localizer["SelectDate"]

Date="selectedDate"

DateChanged="OnDateChanged"

DateFormat="yyyy/MM/dd"

Class="mb-2" />

<MudTimePicker Label=@Localizer["SelectTime"]

Time="selectedTime"

TimeChanged="OnTimeChanged"

TimeFormat="HH:mm"

AmPm="false"

Class="mb-2" />

<MudGrid Justify="Justify.SpaceBetween">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@isActiveMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (MatchDTO.IsActive)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="SetTournamentOff">

@Localizer["Deactivate"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="SetTournamentOn">

@Localizer["Activate"]

</MudButton>

}

</MudItem>

</MudGrid>

<div style="display: flex; align-items: center; justify-content: center; margin-top: 30px; margin-bottom: 30px;">

<div class="mb-2" style="margin-right: 10px;">

<MudImage Src="@imageUrlLocal" Width="90" Height="60" />

</div>

<MudText Typo="Typo.h3" Align="Align.Center" Class="mx-2">Vs</MudText>

<div class="mb-2" style="margin-left: 10px;">

<MudImage Src="@imageUrlVisitor" Width="90" Height="60" />

</div>

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **AddMatch.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class AddMatch

{

private MatchDTO? matchDTO;

private MatchForm? addMatchForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override void OnParametersSet()

{

base.OnParametersSet();

matchDTO = new MatchDTO

{

IsActive = true,

TournamentId = Id,

};

}

private async Task AddAsync()

{

matchDTO!.Date = matchDTO.Date.ToUniversalTime();

var responseHttp = await Repository.PostAsync("api/Matches/full", matchDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordCreatedOk"], Severity.Success);

}

private void Return()

{

addMatchForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/tournament/matches/{Id}");

}

}

1. Modificamos el **AddMatch.razor**:

@if (matchDTO is null)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<MatchForm @ref="addMatchForm" MatchDTO="matchDTO" OnValidSubmit="AddAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el **commit**.
2. Creamos el **EditMatch.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class EditMatch

{

private MatchDTO? matchDTO;

private MatchForm? addMatchForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Match>($"api/Matches/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("tournaments");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

else

{

var match = responseHttp.Response;

matchDTO = new MatchDTO()

{

Id = match!.Id,

IsActive = match!.IsActive,

Date = match!.Date,

GoalsLocal = match!.GoalsLocal,

GoalsVisitor = match!.GoalsVisitor,

LocalId = match!.LocalId,

TournamentId = match!.TournamentId,

VisitorId = match!.VisitorId,

};

}

}

private async Task EditAsync()

{

matchDTO!.Date = matchDTO.Date.ToUniversalTime();

var responseHttp = await Repository.PutAsync("api/Matches/full", matchDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

addMatchForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/tournament/matches/{matchDTO!.TournamentId}");

}

}

1. Modificamos el **EditMatch.razor**:

@if (matchDTO is null)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<MatchForm @ref="addMatchForm" MatchDTO="matchDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el **commit**.

## Agregando entidades de Group, UserGroup y Prediction

1. Agregamos los siguientes literales:

| Group | Group | Grupo |
| --- | --- | --- |
| Groups | Groups | Grupos |
| Code | Code | Código |

1. Creamos la entidad **Group**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.Entities;

public class Group

{

public int Id { get; set; }

[Display(Name = "Group", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

public User Admin { get; set; } = null!;

[Display(Name = "Admin", ResourceType = typeof(Literals))]

[MaxLength(450, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string AdminId { get; set; } = null!;

public Tournament Tournament { get; set; } = null!;

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

[Display(Name = "Code", ResourceType = typeof(Literals))]

[MaxLength(6, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Code { get; set; } = null!;

public string? Image { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

[Display(Name = "Remarks", ResourceType = typeof(Literals))]

public string? Remarks { get; set; }

public ICollection<UserGroup>? Members { get; set; }

public string ImageFull => string.IsNullOrEmpty(Image) ? "/images/NoImage.png" : Image;

}

1. Creamos la entidad **UserGroup**:

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.Entities;

public class UserGroup

{

public int Id { get; set; }

public User User { get; set; } = null!;

[MaxLength(450)]

public string UserId { get; set; } = null!;

public Group Group { get; set; } = null!;

public int GroupId { get; set; }

public bool IsActive { get; set; }

}

1. Modificamos la entidad **User**:

public ICollection<Group>? GroupsManaged { get; set; }

public ICollection<UserGroup>? GroupsBelong { get; set; }

1. Modificamos la entidad **Tournament**:

public ICollection<Group>? Groups { get; set; }

public int GroupsCount => Groups == null ? 0 : Groups.Count;

1. Creamos la entidad **Prediction**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.Entities;

public class Prediction

{

public int Id { get; set; }

public Tournament Tournament { get; set; } = null!;

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

public Group Group { get; set; } = null!;

[Display(Name = "Group", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int GroupId { get; set; }

public Match Match { get; set; } = null!;

[Display(Name = "Match", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int MatchId { get; set; }

public User User { get; set; } = null!;

[Display(Name = "User", ResourceType = typeof(Literals))]

[MaxLength(450, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string UserId { get; set; } = null!;

[Display(Name = "GoalsLocal", ResourceType = typeof(Literals))]

public int? GoalsLocal { get; set; }

[Display(Name = "GoalsVisitor", ResourceType = typeof(Literals))]

public int? GoalsVisitor { get; set; }

[Display(Name = "Points", ResourceType = typeof(Literals))]

public int? Points { get; set; }

}

1. Modificamos las entidades **Tournament**, **Group**, **Match** y **User**, agregando estas propiedades:

public ICollection<Prediction>? Predictions { get; set; }

public int PredictionsCount => Predictions == null ? 0 : Predictions.Count;

1. Modificamos el **DataContext**:

public DbSet<Country> Countries { get; set; }

public DbSet<Group> Groups { get; set; }

public DbSet<Match> Matches { get; set; }

public DbSet<Prediction> Predictions { get; set; }

public DbSet<Team> Teams { get; set; }

public DbSet<Tournament> Tournaments { get; set; }

public DbSet<TournamentTeam> TournamentTeams { get; set; }

public DbSet<UserGroup> UserGroups { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Country>().HasIndex(x => x.Name).IsUnique();

modelBuilder.Entity<Group>().HasIndex(x => x.Code).IsUnique();

modelBuilder.Entity<Prediction>().HasIndex(x => new { x.GroupId, x.MatchId, x.UserId }).IsUnique();

modelBuilder.Entity<Team>().HasIndex(x => new { x.CountryId, x.Name }).IsUnique();

modelBuilder.Entity<Tournament>().HasIndex(x => x.Name).IsUnique();

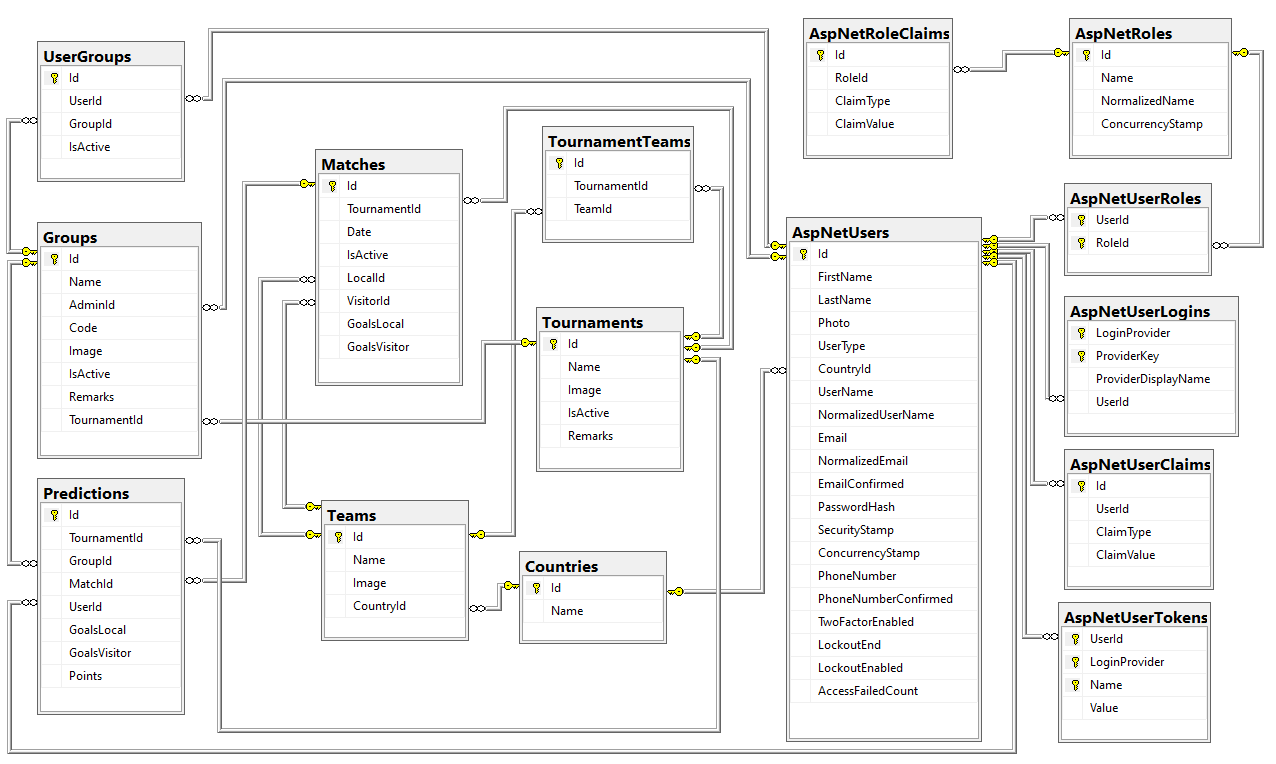
modelBuilder.Entity<TournamentTeam>().HasIndex(x => new { x.TournamentId, x.TeamId }).IsUnique();

modelBuilder.Entity<UserGroup>().HasIndex(x => new { x.UserId, x.GroupId }).IsUnique();

DisableCascadingDelete(modelBuilder);

}

1. Adicionamos la migración y la aplicamos.
2. Así queda nuestra base de datos:



1. Creamos la carpeta **Images/Users** y ahí copiamos las fotos de los usuarios.
2. Modificamos el **SeedDb**:

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckTeamsAsync();

await CheckRolesAsync();

await CheckUsersAsync();

await CheckTournamentsAsync();

await CheckGroupsAsync();

}

private async Task CheckUsersAsync()

{

await CheckUserAsync("Juan", "Zuluaga", "zulu@yopmail.com", "322 311 4620", "JuanZuluaga.jpg", UserType.Admin);

await CheckUserAsync("Ledys", "Bedoya", "ledys@yopmail.com", "322 311 4620", "LedysBedoya.jpg", UserType.User);

await CheckUserAsync("Brad", "Pitt", "brad@yopmail.com", "322 311 4620", "Brad.jpg", UserType.User);

await CheckUserAsync("Angelina", "Jolie", "angelina@yopmail.com", "322 311 4620", "Angelina.jpg", UserType.User);

await CheckUserAsync("Bob", "Marley", "bob@yopmail.com", "322 311 4620", "bob.jpg", UserType.User);

await CheckUserAsync("Celia", "Cruz", "celia@yopmail.com", "322 311 4620", "celia.jpg", UserType.Admin);

await CheckUserAsync("Fredy", "Mercury", "fredy@yopmail.com", "322 311 4620", "fredy.jpg", UserType.User);

await CheckUserAsync("Hector", "Lavoe", "hector@yopmail.com", "322 311 4620", "hector.jpg", UserType.User);

await CheckUserAsync("Liv", "Taylor", "liv@yopmail.com", "322 311 4620", "liv.jpg", UserType.User);

await CheckUserAsync("Otep", "Shamaya", "otep@yopmail.com", "322 311 4620", "otep.jpg", UserType.User);

await CheckUserAsync("Ozzy", "Osbourne", "ozzy@yopmail.com", "322 311 4620", "ozzy.jpg", UserType.User);

await CheckUserAsync("Selena", "Quintanilla", "selena@yopmail.com", "322 311 4620", "selena.jpg", UserType.User);

}

private async Task<User> CheckUserAsync(string firstName, string lastName, string email, string phone, string image, UserType userType)

{

var user = await \_usersUnitOfWork.GetUserAsync(email);

if (user == null)

{

var filePath = $"{Environment.CurrentDirectory}\\Images\\users\\{image}";

var fileBytes = File.ReadAllBytes(filePath);

var imagePath = await \_fileStorage.SaveFileAsync(fileBytes, "jpg", "users");

var country = await \_context.Countries.FirstOrDefaultAsync(x => x.Name == "Colombia");

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Country = country!,

UserType = userType,

Photo = imagePath

};

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

await \_usersUnitOfWork.AddUserToRoleAsync(user, userType.ToString());

var token = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

await \_usersUnitOfWork.ConfirmEmailAsync(user, token);

}

return user;

}

private async Task CheckGroupsAsync()

{

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

{

var zulu = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "zulu@yopmail.com");

var ledys = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "ledys@yopmail.com");

var brad = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "brad@yopmail.com");

var angelina = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "angelina@yopmail.com");

var bob = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "bob@yopmail.com");

var celia = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "celia@yopmail.com");

var fredy = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "fredy@yopmail.com");

var hector = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "hector@yopmail.com");

var liv = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "liv@yopmail.com");

var otep = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "otep@yopmail.com");

var ozzy = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "ozzy@yopmail.com");

var selena = await \_context.Users.FirstOrDefaultAsync(x => x.UserName == "selena@yopmail.com");

var copaAmerica = await \_context.Tournaments.FirstOrDefaultAsync(x => x.Name == "Copa América - 2025");

var zuluGoup = new Group

{

Admin = zulu!,

Code = Guid.NewGuid().ToString().Substring(0, 6).ToUpper(),

Name = "Gupo Zulu Copa America",

Remarks = "Valor COP$50,000. Primer puesto 70% del premio, segundo puesto 30% del premio",

Tournament = copaAmerica!,

Image = copaAmerica?.Image,

IsActive = true,

Members =

[

new UserGroup { IsActive = true, User = zulu! },

new UserGroup { IsActive = true, User = ledys! },

new UserGroup { IsActive = true, User = brad! },

new UserGroup { IsActive = true, User = angelina! },

new UserGroup { IsActive = true, User = bob! },

new UserGroup { IsActive = true, User = celia! },

new UserGroup { IsActive = true, User = fredy! },

new UserGroup { IsActive = true, User = selena! },

],

};

\_context.Add(zuluGoup);

var selenaGoup = new Group

{

Admin = selena!,

Code = Guid.NewGuid().ToString().Substring(0, 6).ToUpper(),

Name = "Gupo Selena Copa America",

Remarks = "Valor USD$30.00. Primer puesto 80% del premio, segundo puesto 20% del premio",

Tournament = copaAmerica!,

Image = copaAmerica?.Image,

IsActive = true,

Members =

[

new UserGroup { IsActive = true, User = zulu! },

new UserGroup { IsActive = true, User = celia! },

new UserGroup { IsActive = true, User = fredy! },

new UserGroup { IsActive = true, User = hector! },

new UserGroup { IsActive = true, User = liv! },

new UserGroup { IsActive = true, User = otep! },

new UserGroup { IsActive = true, User = ozzy! },

new UserGroup { IsActive = true, User = selena! },

],

};

\_context.Add(selenaGoup);

await \_context.SaveChangesAsync();

}

}

private async Task CheckPredictionsAsync()

{

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

{

var random = new Random();

var predictions = new List<Prediction>();

var groups = await \_context.Groups

.Include(x => x.Tournament)

.ThenInclude(x => x.Matches)

.Include(x => x.Members)

.ToListAsync();

foreach (var group in groups)

{

foreach (var match in group.Tournament.Matches!)

{

foreach (var member in group.Members!)

{

predictions.Add(new Prediction

{

GoalsLocal = random.Next(4),

GoalsVisitor = random.Next(4),

Group = group,

Match = match,

Tournament = group.Tournament,

User = member.User,

});

}

}

}

\_context.AddRange(predictions);

await \_context.SaveChangesAsync();

}

}

1. Modificamos el **UsersRepository**:

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

{

if (!string.IsNullOrEmpty(user.Photo) && !user.Photo.StartsWith("http"))

{

var imageBase64 = Convert.FromBase64String(user.Photo!);

user.Photo = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "users");

}

var result = await \_userManager.CreateAsync(user, password);

return result;

}

1. Actualizar el scrip de borrado de usuarios **DeleteUsers.sql**:

DELETE FROM UserGroups

DELETE FROM Groups

DELETE FROM AspNetUserRoles

DELETE FROM AspNetUsers

1. Probamos y hacemos el **commit**.

## Listar usuarios y crear nuevos administradores

1. Adicionamos los siguientes literales:

| Users | Users | Usuarios |
| --- | --- | --- |
| Add | Add | Adicionar |
| Confirmed | Confirmed | Confirmado |

1. Adicionamos estos métodos al **IUsersRepository**:

Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Adicionamos estos métodos al **UsersRepository**:

public async Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Users

.Include(x => x.Country)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<User>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.FirstName)

.ThenBy(x => x.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Users.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Adicionamos estos métodos al **IUsersUnitOfWork**:

Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

1. Adicionamos estos métodos al **UsersUnitOfWork**:

public async Task<ActionResponse<IEnumerable<User>>> GetAsync(PaginationDTO pagination) => await \_usersRepository.GetAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_usersRepository.GetTotalRecordsAsync(pagination);

1. Adicionamos estos métodos al **AccountController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpGet("paginated")]

public async Task<IActionResult> GetAsync([FromQuery] PaginationDTO pagination)

{

var response = await \_usersUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetPagesAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_usersUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Modificamos el **NavMenu.razor**:

<MudNavLink Href="/tournaments" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Star">@Localizer["Tournaments"]</MudNavLink>

<MudDivider />

<MudNavLink Href="/users" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.People">@Localizer["Users"]</MudNavLink>

<MudDivider />

1. Creamos el **UserIndex.razor.cs** dentro de **Pages/Auth**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Auth;

[Authorize(Roles = "Admin")]

public partial class UserIndex

{

public List<User>? Users { get; set; }

private MudTable<User> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private string baseUrl = "api/accounts";

private string infoFormat = "{first\_item}-{last\_item} => {all\_items}";

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<User>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated?page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<User>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<User> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<User> { Items = [], TotalItems = 0 };

}

return new TableData<User>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

}

1. Modificamos el **UserIndex.razor** dentro de **Pages/Auth**:

@page "/users"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@Users"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<div class="d-flex justify-content-between">

<MudText Typo="Typo.h6" Class="me-4">@Localizer["Users"]</MudText>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info"

Href="/register/?IsAdmin=true">

@Localizer["Users"] @Localizer["Admin"]

</MudButton>

</div>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["User"]</MudTh>

<MudTh>@Localizer["PhoneNumber"]</MudTh>

<MudTh>@Localizer["Email"]</MudTh>

<MudTh>@Localizer["Confirmed"]</MudTh>

<MudTh>@Localizer["UserType"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudImage Src="@context.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

</MudTd>

<MudTd>@context.FullName</MudTd>

<MudTd>@context.PhoneNumber</MudTd>

<MudTd>@context.Email</MudTd>

<MudTd>@context.EmailConfirmed</MudTd>

<MudTd>@context.UserType</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos y hacemos el **commit**.

# Funcionalidad de la aplicación

## Creando el controlador grupos

1. Adicionamos los siguientes literales:

| ERR013 | The user Id is not valid. | El código de usuario no es válido. |
| --- | --- | --- |
| ERR014 | The group Id is not valid. | El código del grupo no es válido. |

1. Modificamos el **PaginationDTO**:

public string? Email { get; set; }

1. Creamos el **GroupDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class GroupDTO

{

public int Id { get; set; }

[Display(Name = "Group", ResourceType = typeof(Literals))]

[MaxLength(100, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Name { get; set; } = null!;

[Display(Name = "Admin", ResourceType = typeof(Literals))]

[MaxLength(450, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

public string AdminId { get; set; } = null!;

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

[Display(Name = "Code", ResourceType = typeof(Literals))]

[MaxLength(6, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

public string Code { get; set; } = null!;

public string? Image { get; set; }

[Display(Name = "IsActive", ResourceType = typeof(Literals))]

public bool IsActive { get; set; }

[Display(Name = "Remarks", ResourceType = typeof(Literals))]

public string? Remarks { get; set; }

}

1. Creamos el **IGroupsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface IGroupsRepository

{

Task<ActionResponse<Group>> AddAsync(GroupDTO groupDTO);

Task<ActionResponse<Group>> UpdateAsync(GroupDTO groupDTO);

Task<ActionResponse<Group>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Group>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **IGroupsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IGroupsUnitOfWork

{

Task<ActionResponse<Group>> AddAsync(GroupDTO groupDTO);

Task<ActionResponse<Group>> UpdateAsync(GroupDTO groupDTO);

Task<ActionResponse<Group>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Group>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **GroupsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class GroupsRepository : GenericRepository<Group>, IGroupsRepository

{

private readonly DataContext \_context;

private readonly IFileStorage \_fileStorage;

private readonly IUsersRepository \_usersRepository;

public GroupsRepository(DataContext context, IFileStorage fileStorage, IUsersRepository usersRepository) : base(context)

{

\_context = context;

\_fileStorage = fileStorage;

\_usersRepository = usersRepository;

}

public override async Task<ActionResponse<IEnumerable<Group>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Groups

.Include(x => x.Members!)

.ThenInclude(x => x.User)

.Include(x => x.Tournament)

.AsQueryable();

queryable = queryable.Where(x => x.Members!.Any(x => x.User.Email == pagination.Email));

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Group>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Name)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<Group>> GetAsync(int id)

{

var group = await \_context.Groups

.Include(x => x.Members!)

.ThenInclude(x => x.User)

.Include(x => x.Tournament)

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

if (group == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Group>

{

WasSuccess = true,

Result = group

};

}

public async Task<ActionResponse<Group>> AddAsync(GroupDTO groupDTO)

{

var admin = await \_usersRepository.GetUserAsync(groupDTO.AdminId);

if (admin == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR013"

};

}

var tournament = await \_context.Tournaments.FindAsync(groupDTO.TournamentId);

if (tournament == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR009"

};

}

var code = string.Empty;

var exists = true;

do

{

code = Guid.NewGuid().ToString().Substring(0, 6).ToUpper();

var currentGroup = await \_context.Groups.FirstOrDefaultAsync(x => x.Code == code);

exists = currentGroup != null;

} while (exists);

var group = new Group

{

Admin = admin,

Tournament = tournament,

Code = code,

IsActive = true,

Name = groupDTO.Name,

Remarks = groupDTO.Remarks,

Members = [

new UserGroup { User = admin, IsActive = true },

]

};

if (!string.IsNullOrEmpty(groupDTO.Image))

{

var imageBase64 = Convert.FromBase64String(groupDTO.Image!);

group.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "groups");

}

\_context.Add(group);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Group>

{

WasSuccess = true,

Result = group

};

}

catch (DbUpdateException)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Groups.AsQueryable();

queryable = queryable.Where(x => x.Members!.Any(x => x.User.Email == pagination.Email));

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<Group>> UpdateAsync(GroupDTO groupDTO)

{

var currentGroup = await \_context.Groups.FindAsync(groupDTO.Id);

if (currentGroup == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR014"

};

}

if (!string.IsNullOrEmpty(groupDTO.Image))

{

var imageBase64 = Convert.FromBase64String(groupDTO.Image!);

currentGroup.Image = await \_fileStorage.SaveFileAsync(imageBase64, ".jpg", "groups");

}

currentGroup.Name = groupDTO.Name;

currentGroup.IsActive = groupDTO.IsActive;

currentGroup.Remarks = groupDTO.Remarks;

\_context.Update(currentGroup);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Group>

{

WasSuccess = true,

Result = currentGroup

};

}

catch (DbUpdateException)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<Group>> GetAsync(string code)

{

var group = await \_context.Groups.FirstOrDefaultAsync(x => x.Code == code);

if (group == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Group>

{

WasSuccess = true,

Result = group

};

}

public async Task CheckPredictionsForAllMatchesAsync(int id)

{

var group = await \_context.Groups

.Include(x => x.Members)

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

if (group == null)

{

return;

}

var tournament = await \_context.Tournaments

.Include(x => x.Matches)

.FirstOrDefaultAsync(x => x.Id == group.TournamentId);

if (tournament == null)

{

return;

}

var newPredictions = new List<Prediction>();

foreach (var userGroup in group.Members!)

{

foreach (var match in tournament!.Matches!)

{

var prediction = await \_context.Predictions.FirstOrDefaultAsync(x => x.GroupId == group.Id &&

x.Match.Id == match.Id &&

x.UserId == userGroup.UserId &&

x.TournamentId == tournament.Id);

if (prediction == null)

{

newPredictions.Add(new Prediction

{

Group = group,

Match = match,

Tournament = tournament,

User = userGroup.User,

UserId = userGroup.UserId,

});

}

}

}

if (newPredictions.Count > 0)

{

\_context.AddRange(newPredictions);

await \_context.SaveChangesAsync();

}

}

public async Task<ActionResponse<IEnumerable<Group>>> GetAllAsync()

{

var groups = await \_context.Groups

.Include(x => x.Admin)

.Include(x => x.Tournament)

.Where(x => x.IsActive)

.OrderBy(x => x.Name)

.ToListAsync();

return new ActionResponse<IEnumerable<Group>>

{

WasSuccess = true,

Result = groups

};

}

}

1. Creamos el **GroupsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class GroupsUnitOfWork : GenericUnitOfWork<Group>, IGroupsUnitOfWork

{

private readonly IGroupsRepository \_groupsRepository;

public GroupsUnitOfWork(IGenericRepository<Group> repository, IGroupsRepository groupsRepository) : base(repository)

{

\_groupsRepository = groupsRepository;

}

public override async Task<ActionResponse<IEnumerable<Group>>> GetAsync(PaginationDTO pagination) => await \_groupsRepository.GetAsync(pagination);

public override async Task<ActionResponse<Group>> GetAsync(int id) => await \_groupsRepository.GetAsync(id);

public async Task<ActionResponse<Group>> AddAsync(GroupDTO groupDTO) => await \_groupsRepository.AddAsync(groupDTO);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_groupsRepository.GetTotalRecordsAsync(pagination);

public async Task<ActionResponse<Group>> UpdateAsync(GroupDTO groupDTO) => await \_groupsRepository.UpdateAsync(groupDTO);

}

1. Creamos el **GroupsController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class GroupsController : GenericController<Group>

{

private readonly IGroupsUnitOfWork \_groupsUnitOfWork;

public GroupsController(IGenericUnitOfWork<Group> unitOfWork, IGroupsUnitOfWork groupsUnitOfWork) : base(unitOfWork)

{

\_groupsUnitOfWork = groupsUnitOfWork;

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

pagination.Email = User.Identity!.Name;

var response = await \_groupsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

pagination.Email = User.Identity!.Name;

var action = await \_groupsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_groupsUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(GroupDTO groupDTO)

{

groupDTO.AdminId = User.Identity!.Name!;

var action = await \_groupsUnitOfWork.AddAsync(groupDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutAsync(GroupDTO groupDTO)

{

var action = await \_groupsUnitOfWork.UpdateAsync(groupDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<ICountriesRepository, CountriesRepository>();

builder.Services.AddScoped<ICountriesUnitOfWork, CountriesUnitOfWork>();

builder.Services.AddScoped<IGroupsRepository, GroupsRepository>();

builder.Services.AddScoped<IGroupsUnitOfWork, GroupsUnitOfWork>();

builder.Services.AddScoped<ITeamsRepository, TeamsRepository>();

builder.Services.AddScoped<ITeamsUnitOfWork, TeamsUnitOfWork>();

1. Probamos en **Swagger** y hacemos el **commit**.

## Creando el controlador usuarios-grupos

1. Adicionamos el siguiente literal:

| ERR015 | The user group is not valid. | El códio de usario grupo no es válido. |
| --- | --- | --- |

1. Creamos el **UserGroupDTO**:

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.DTOs;

public class UserGroupDTO

{

public int Id { get; set; }

[MaxLength(450)]

public string UserId { get; set; } = null!;

public int GroupId { get; set; }

public bool IsActive { get; set; }

}

1. Creamos el **IUserGroupsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface IUserGroupsRepository

{

Task<ActionResponse<UserGroup>> AddAsync(UserGroupDTO userGroupDTO);

Task<ActionResponse<UserGroup>> UpdateAsync(UserGroupDTO userGroupDTO);

Task<ActionResponse<UserGroup>> GetAsync(int id);

Task<ActionResponse<IEnumerable<UserGroup>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **IUserGroupsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IUserGroupsUnitOfWork

{

Task<ActionResponse<UserGroup>> AddAsync(UserGroupDTO userGroupDTO);

Task<ActionResponse<UserGroup>> UpdateAsync(UserGroupDTO userGroupDTO);

Task<ActionResponse<UserGroup>> GetAsync(int id);

Task<ActionResponse<IEnumerable<UserGroup>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **UserGroupsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class UserGroupsRepository : GenericRepository<UserGroup>, IUserGroupsRepository

{

private readonly DataContext \_context;

private readonly IUsersRepository \_usersRepository;

public UserGroupsRepository(DataContext context, IUsersRepository usersRepository) : base(context)

{

\_context = context;

\_usersRepository = usersRepository;

}

public override async Task<ActionResponse<IEnumerable<UserGroup>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.UserGroups

.Include(x => x.User)

.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<UserGroup>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.User.FirstName)

.ThenBy(x => x.User.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<UserGroup>> GetAsync(int id)

{

var userGroup = await \_context.UserGroups

.Include(x => x.User)

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

if (userGroup == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<UserGroup>

{

WasSuccess = true,

Result = userGroup

};

}

public async Task<ActionResponse<UserGroup>> AddAsync(UserGroupDTO userGroupDTO)

{

var user = await \_usersRepository.GetUserAsync(Guid.Parse(userGroupDTO.UserId));

if (user == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR013"

};

}

var group = await \_context.Groups.FindAsync(userGroupDTO.GroupId);

if (group == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR014"

};

}

var userGroup = new UserGroup

{

Group = group,

User = user

};

\_context.Add(userGroup);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<UserGroup>

{

WasSuccess = true,

Result = userGroup

};

}

catch (DbUpdateException)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.UserGroups.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<UserGroup>> UpdateAsync(UserGroupDTO userGroupDTO)

{

var currentUserGroup = await \_context.UserGroups.FindAsync(userGroupDTO.Id);

if (currentUserGroup == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR015"

};

}

currentUserGroup.IsActive = userGroupDTO.IsActive;

\_context.Update(currentUserGroup);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<UserGroup>

{

WasSuccess = true,

Result = currentUserGroup

};

}

catch (DbUpdateException)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = exception.Message

};

}

}

}

1. Creamos el **UserGroupsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class UserGroupsUnitOfWork : GenericUnitOfWork<UserGroup>, IUserGroupsUnitOfWork

{

private readonly IUserGroupsRepository \_userGroupsRepository;

public UserGroupsUnitOfWork(IGenericRepository<UserGroup> repository, IUserGroupsRepository userGroupsRepository) : base(repository)

{

\_userGroupsRepository = userGroupsRepository;

}

public override async Task<ActionResponse<IEnumerable<UserGroup>>> GetAsync(PaginationDTO pagination) => await \_userGroupsRepository.GetAsync(pagination);

public override async Task<ActionResponse<UserGroup>> GetAsync(int id) => await \_userGroupsRepository.GetAsync(id);

public async Task<ActionResponse<UserGroup>> AddAsync(UserGroupDTO userGroupDTO) => await \_userGroupsRepository.AddAsync(userGroupDTO);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination) => await \_userGroupsRepository.GetTotalRecordsAsync(pagination);

public async Task<ActionResponse<UserGroup>> UpdateAsync(UserGroupDTO userGroupDTO) => await \_userGroupsRepository.UpdateAsync(userGroupDTO);

}

1. Creamos el **UserGroupsController**:

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class UserGroupsController : GenericController<UserGroup>

{

private readonly IUserGroupsUnitOfWork \_userGroupsUnitOfWork;

public UserGroupsController(IGenericUnitOfWork<UserGroup> unitOfWork, IUserGroupsUnitOfWork userGroupsUnitOfWork) : base(unitOfWork)

{

\_userGroupsUnitOfWork = userGroupsUnitOfWork;

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

var response = await \_userGroupsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_userGroupsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_userGroupsUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(UserGroupDTO userGroupDTO)

{

var action = await \_userGroupsUnitOfWork.AddAsync(userGroupDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutAsync(UserGroupDTO userGroupDTO)

{

var action = await \_userGroupsUnitOfWork.UpdateAsync(userGroupDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<ITournamentTeamsRepository, TournamentTeamsRepository>();

builder.Services.AddScoped<ITournamentTeamsUnitOfWork, TournamentTeamsUnitOfWork>();

builder.Services.AddScoped<IUserGroupsRepository, UserGroupsRepository>();

builder.Services.AddScoped<IUserGroupsUnitOfWork, UserGroupsUnitOfWork>();

builder.Services.AddScoped<IUsersRepository, UsersRepository>();

builder.Services.AddScoped<IUsersUnitOfWork, UsersUnitOfWork>();

1. Probamos en **Swagger** y hacemos el **commit**.

## Creando el controlador predicciones

1. Adicionamos los siguientes literales:

| Points | Points | Puntos |
| --- | --- | --- |
| ERR016 | The prediction Id is not valid. | El código de la predicción no es válido. |

1. Creamos el **PredictionDTO**:

using System.ComponentModel.DataAnnotations;

using Fantasy.Shared.Resources;

namespace Fantasy.Shared.DTOs;

public class PredictionDTO

{

public int Id { get; set; }

[Display(Name = "Tournament", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int TournamentId { get; set; }

[Display(Name = "Group", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int GroupId { get; set; }

[Display(Name = "Match", ResourceType = typeof(Literals))]

[Range(1, int.MaxValue, ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public int MatchId { get; set; }

[Display(Name = "User", ResourceType = typeof(Literals))]

[MaxLength(450, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string UserId { get; set; } = null!;

[Display(Name = "GoalsLocal", ResourceType = typeof(Literals))]

public int? GoalsLocal { get; set; }

[Display(Name = "GoalsVisitor", ResourceType = typeof(Literals))]

public int? GoalsVisitor { get; set; }

[Display(Name = "Points", ResourceType = typeof(Literals))]

public int? Points { get; set; }

}

1. Creamos el **IPredictionsRepository**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.Repositories.Interfaces;

public interface IPredictionsRepository

{

Task<ActionResponse<Prediction>> AddAsync(PredictionDTO predictionDTO);

Task<ActionResponse<Prediction>> UpdateAsync(PredictionDTO predictionDTO);

Task<ActionResponse<Prediction>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Prediction>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **IPredictionsUnitOfWork**:

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Interfaces;

public interface IPredictionsUnitOfWork

{

Task<ActionResponse<Prediction>> AddAsync(PredictionDTO predictionDTO);

Task<ActionResponse<Prediction>> UpdateAsync(PredictionDTO predictionDTO);

Task<ActionResponse<Prediction>> GetAsync(int id);

Task<ActionResponse<IEnumerable<Prediction>>> GetAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination);

}

1. Creamos el **PredictionsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Backend.Repositories.Implementations;

public class PredictionsRepository : GenericRepository<Prediction>, IPredictionsRepository

{

private readonly DataContext \_context;

private readonly IUsersRepository \_usersRepository;

public PredictionsRepository(DataContext context, IUsersRepository usersRepository) : base(context)

{

\_context = context;

\_usersRepository = usersRepository;

}

public override async Task<ActionResponse<IEnumerable<Prediction>>> GetAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Prediction>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.Match.IsClosed)

.ThenBy(x => x.Match.Date)

.Paginate(pagination)

.ToListAsync()

};

}

public override async Task<ActionResponse<Prediction>> GetAsync(int id)

{

var prediction = await \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

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

if (prediction == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Prediction>

{

WasSuccess = true,

Result = prediction

};

}

public async Task<ActionResponse<Prediction>> AddAsync(PredictionDTO predictionDTO)

{

var user = await \_usersRepository.GetUserAsync(Guid.Parse(predictionDTO.UserId));

if (user == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR013"

};

}

var group = await \_context.Groups.FindAsync(predictionDTO.GroupId);

if (group == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR014"

};

}

var tournament = await \_context.Tournaments.FindAsync(predictionDTO.TournamentId);

if (tournament == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR009"

};

}

var match = await \_context.Matches.FindAsync(predictionDTO.MatchId);

if (match == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR012"

};

}

var prediction = new Prediction

{

GoalsLocal = predictionDTO.GoalsLocal,

GoalsVisitor = predictionDTO.GoalsVisitor,

Group = group,

Tournament = tournament,

Match = match,

User = user,

};

\_context.Add(prediction);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Prediction>

{

WasSuccess = true,

Result = prediction

};

}

catch (DbUpdateException)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<Prediction>> UpdateAsync(PredictionDTO predictionDTO)

{

var currentPrediction = await \_context.Predictions

.Include(x => x.Match)

.FirstOrDefaultAsync(x => x.Id == predictionDTO.Id);

if (currentPrediction == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR016"

};

}

if (currentPrediction.Match.GoalsLocal != null || currentPrediction.Match.GoalsVisitor != null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR018"

};

}

if (CanWatch(currentPrediction))

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR018"

};

}

currentPrediction.GoalsLocal = predictionDTO.GoalsLocal;

currentPrediction.GoalsVisitor = predictionDTO.GoalsVisitor;

currentPrediction.Points = predictionDTO.Points;

\_context.Update(currentPrediction);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<Prediction>

{

WasSuccess = true,

Result = currentPrediction

};

}

catch (DbUpdateException)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = exception.Message

};

}

}

public async Task<ActionResponse<IEnumerable<PositionDTO>>> GetPositionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Where(x => x.GroupId == pagination.Id && x.Points.HasValue)

.GroupBy(x => x.User)

.Select(g => new PositionDTO

{

User = g.Key,

Points = g.Sum(x => x.Points ?? 0)

})

.OrderByDescending(x => x.Points)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<PositionDTO>>

{

WasSuccess = true,

Result = await queryable

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsForPositionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Where(x => x.GroupId == pagination.Id && x.Points.HasValue)

.GroupBy(x => x.User)

.Select(g => new PositionDTO

{

User = g.Key,

Points = g.Sum(x => x.Points ?? 0)

})

.OrderByDescending(x => x.Points)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<IEnumerable<Prediction>>> GetAllPredictionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

.Include(x => x.User)

.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.MatchId == pagination.Id2);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Prediction>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.User.FirstName)

.ThenBy(x => x.User.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAllPredictionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.MatchId == pagination.Id2);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public async Task<ActionResponse<IEnumerable<Prediction>>> GetBalanceAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

.Include(x => x.User)

.AsQueryable();

queryable = queryable.Where(x => x.Match.GoalsLocal != null && x.Match.GoalsVisitor != null);

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Prediction>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.User.FirstName)

.ThenBy(x => x.User.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsBalanceAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions.AsQueryable();

queryable = queryable.Where(x => x.Match.GoalsLocal != null && x.Match.GoalsVisitor != null);

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

public virtual bool CanWatch(Prediction prediction)

{

if (prediction.Match.GoalsLocal != null || prediction.Match.GoalsVisitor != null)

{

return true;

}

var dateMatch = prediction.Match.Date.ToLocalTime();

var currentDate = DateTime.Now;

var minutesMatch = dateMatch.Subtract(DateTime.MinValue).TotalMinutes;

var minutesNow = currentDate.Subtract(DateTime.MinValue).TotalMinutes;

var difference = minutesNow - minutesMatch;

var canWatch = difference >= -10;

return canWatch;

}

}

1. Creamos el **PredictionsUnitOfWork**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

namespace Fantasy.Backend.UnitsOfWork.Implementations;

public class PredictionsUnitOfWork : GenericUnitOfWork<Prediction>, IPredictionsUnitOfWork

{

private readonly IPredictionsRepository \_predictionsRepository;

public PredictionsUnitOfWork(IGenericRepository<Prediction> repository, IPredictionsRepository predictionsRepository) : base(repository)

{

\_predictionsRepository = predictionsRepository;

}

public override async Task<ActionResponse<IEnumerable<Prediction>>> GetAsync(PaginationDTO pagination) => await \_predictionsRepository.GetAsync(pagination);

public override async Task<ActionResponse<Prediction>> GetAsync(int id) => await \_predictionsRepository.GetAsync(id);

public async Task<ActionResponse<Prediction>> AddAsync(PredictionDTO AddAsync) => await \_predictionsRepository.AddAsync(AddAsync);

public async Task<ActionResponse<int>> GetTotalRecordsAsync(PaginationDTO paginationDTO) => await \_predictionsRepository.GetTotalRecordsAsync(paginationDTO);

public async Task<ActionResponse<Prediction>> UpdateAsync(PredictionDTO predictionDTO) => await \_predictionsRepository.UpdateAsync(predictionDTO);

}

1. Creamos el **PredictionsController**:

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace Fantasy.Backend.Controllers;

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

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

public class PredictionsController : GenericController<Prediction>

{

private readonly IPredictionsUnitOfWork \_predictionsUnitOfWork;

public PredictionsController(IGenericUnitOfWork<Prediction> unitOfWork, IPredictionsUnitOfWork predictionsUnitOfWork) : base(unitOfWork)

{

\_predictionsUnitOfWork = predictionsUnitOfWork;

}

[HttpGet("paginated")]

public override async Task<IActionResult> GetAsync(PaginationDTO pagination)

{

pagination.Email = User.Identity!.Name;

var response = await \_predictionsUnitOfWork.GetAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginated")]

public async Task<IActionResult> GetTotalRecordsAsync([FromQuery] PaginationDTO pagination)

{

pagination.Email = User.Identity!.Name;

var action = await \_predictionsUnitOfWork.GetTotalRecordsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

{

var response = await \_predictionsUnitOfWork.GetAsync(id);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

[HttpPost("full")]

public async Task<IActionResult> PostAsync(PredictionDTO predictionDTO)

{

var action = await \_predictionsUnitOfWork.AddAsync(predictionDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

[HttpPut("full")]

public async Task<IActionResult> PutAsync(PredictionDTO predictionDTO)

{

var action = await \_predictionsUnitOfWork.UpdateAsync(predictionDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

}

1. Agregamos las nuevas inyecciones en el **Program**:

builder.Services.AddScoped<IMatchesRepository, MatchesRepository>();

builder.Services.AddScoped<IMatchesUnitOfWork, MatchesUnitOfWork>();

builder.Services.AddScoped<IPredictionsRepository, PredictionsRepository>();

builder.Services.AddScoped<IPredictionsUnitOfWork, PredictionsUnitOfWork>();

builder.Services.AddScoped<ITournamentsRepository, TournamentsRepository>();

builder.Services.AddScoped<ITournamentsUnitOfWork, TournamentsUnitOfWork>();

1. Probamos en **Swagger** y hacemos el **commit**.

## Listando los grupos a los que pertenezco

1. Adicionamos los siguientes literales:

| Members | Members | Miembros |
| --- | --- | --- |
| GroupDetails | Group Details | Detalles de Grupo |
| JoinGroup | Join Group | Unirme a un grupo |
| MyGroups | My Groups | Mis Grupos |
| NoGroups | You are not part of any group. You can join a group using the URL or code shared by the group administrator, or you can view the available groups on the homepage and then ask the administrator to activate you so you can enter your predictions. You can also create your own group of friends. | No perteneces a ningún grupo. Puedes unirte a uno con la URL o el código compartido por el administrador del grupo, o bien, ver los grupos disponibles en la página de inicio y luego pedirle al administrador que te active para poder ingresar tus predicciones. También puedes crear tu propio grupo de amigos. |

1. Creamos el **GroupCreate.razor.cs** temporal:

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupCreate

{

}

1. Creamos el **GroupCreate.razor** temporal:

<h3>GroupCreate</h3>

1. Creamos el **GroupEdit.razor.cs** temporal:

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupEdit

{

}

1. Creamos el **GroupEdit.razor** temporal:

<h3>GroupEdit</h3>

1. Creamos el **GroupsIndex.razor.cs**:

using System.Net;

using Fantasy.Frontend.Helpers;

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

[Authorize(Roles = "Admin, User")]

public partial class GroupsIndex

{

private List<Group>? Groups { get; set; }

private MudTable<Group> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/groups";

private string infoFormat = "{first\_item}-{last\_item} => {all\_items}";

private string username = string.Empty;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private AuthenticationStateProvider AuthenticationStateProvider { get; set; } = null!;

[Inject] private IClipboardService ClipboardService { get; set; } = null!;

[Inject] private IStringLocalizer<Parameters> Parameters { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadUserNameAsync();

await LoadTotalRecordsAsync();

}

private async Task LoadUserNameAsync()

{

var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();

var user = authState.User;

if (user.Identity != null && user.Identity.IsAuthenticated)

{

username = user.Identity.Name!;

}

}

private async Task AdminUsersGroupAsync(Group group)

{

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Large,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", group.Id },

};

var dialog = DialogService.Show<UsersGroup>(@Localizer["AdminUsersGroup"], parameters, options);

await dialog.Result;

}

}

private async Task CopyInvitationAsync(Group group)

{

var joinURL = $"{Parameters["URLFront"]}/groups/join/?code={group!.Code}";

await ClipboardService.CopyToClipboardAsync(joinURL);

var text = string.Format(Localizer["InvitationURLCopied"], group!.Name);

Snackbar.Add(text, Severity.Success);

}

private void GroupDetails(Group group)

{

NavigationManager.NavigateTo($"/groups/details/{group.Id}/false");

}

private async Task LoadTotalRecordsAsync()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"?filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

totalRecords = responseHttp.Response;

loading = false;

}

private async Task<TableData<Group>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated/?page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Group>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Group> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Group> { Items = [], TotalItems = 0 };

}

return new TableData<Group>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

private async Task ShowModalJoinAsync()

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

var dialog = DialogService.Show<JoinGroup>($"{Localizer["JoinExistingGroup"]}", options);

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

private async Task ShowModalAsync(int id = 0, bool isEdit = false)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

IDialogReference? dialog;

if (isEdit)

{

var parameters = new DialogParameters

{

{ "Id", id }

};

dialog = DialogService.Show<GroupEdit>($"{Localizer["Edit"]} {Localizer["Group"]}", parameters, options);

}

else

{

dialog = DialogService.Show<GroupCreate>($"{Localizer["New"]} {Localizer["Group"]}", options);

}

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

private async Task DeleteAsync(Group team)

{

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["DeleteConfirm"], Localizer["Group"], team.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.DeleteAsync($"{baseUrl}/{team.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("/groups");

}

else

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

}

return;

}

await LoadTotalRecordsAsync();

await table.ReloadServerData();

Snackbar.Add(Localizer["RecordDeletedOk"], Severity.Success);

}

}

1. Creamos el **GroupsIndex.razor**:

@page "/groups"

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@Groups"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true">

<ToolBarContent>

<div class="d-flex justify-content-between">

<MudText Typo="Typo.h6" Class="me-4"> @Localizer["MyGroups"]</MudText>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Info" OnClick="@(() => ShowModalAsync())"

class="me-2">

@Localizer["New"]

</MudButton>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Add"

Color="Color.Warning" OnClick="@(() => ShowModalJoinAsync())">

@Localizer["JoinGroup"]

</MudButton>

</div>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Grupo"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Admin"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Tournament"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Code"]</MudTh>

<MudTh style="width: 80px;">@Localizer["IsActive"]</MudTh>

<MudTh># @Localizer["Members"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>

<MudImage Src="@context.ImageFull" Width="80" />

</MudTd>

<MudTd>@context.Admin.FullName</MudTd>

<MudTd>

<MudImage Src="@context.Admin.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

</MudTd>

<MudTd>@context.Tournament.Name</MudTd>

<MudTd>

<MudImage Src="@context.Tournament.ImageFull" Width="80" />

</MudTd>

<MudTd>@context.Code</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>@context.MembersCount</MudTd>

<MudTd>

<MudStack Spacing="2">

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Info"

OnClick="@(() => GroupDetails(@context))"

Disabled="@(!context.IsActive)">

@Localizer["GroupDetails"]

</MudButton>

@if (context.Admin.UserName == username)

{

<MudStack Row="true" Spacing="2">

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["CopyInvitationURLTitle"]">

<MudButton Variant="Variant.Filled"

Color="Color.Secondary"

OnClick="@(() => CopyInvitationAsync(@context))"

Disabled="@(!context.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.ContentCopy" />

</MudButton>

</MudTooltip>

</MudStack>

}

</MudStack>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoGroups"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **NavMenu.razor**:

</AuthorizeView>

<AuthorizeView>

<MudNavLink Href="/groups" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.SportsSoccer">@Localizer["MyGroups"]</MudNavLink>

<MudDivider />

</AuthorizeView>

<MudNavLink Href="/about" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Info">@Localizer["About"]</MudNavLink>

1. Probamos y hacemos el **commit**.

## Creando y editando grupos

1. Adicionamos los siguientes literales:

| Inactive | Inactive | Inactivo |
| --- | --- | --- |
| SelectATournament | -- Select a Tournament -- | -- Seleccione un Torneo -- |
| GroupCreated | Group: {0} has been created, with code: {1}, please send the group code to people who want to join this group. | Se ha creado el grupo: {0}, con el código: {1}, por favor envíe el código del grupo a las personas que deseen uniserse a este grupo. |
| Active | Active | Activo |

1. Creamos el **GroupForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using Fantasy.Shared.Entities;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupForm

{

private string? imageUrl;

private string? isActiveMessage;

private EditContext editContext = null!;

private Tournament selectedTournament = new();

private List<Tournament>? tournaments;

protected override async Task OnInitializedAsync()

{

editContext = new(GroupDTO);

await LoadTournamentAsync();

}

[EditorRequired, Parameter] public GroupDTO GroupDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

protected override void OnParametersSet()

{

base.OnParametersSet();

if (!string.IsNullOrEmpty(GroupDTO.Image))

{

imageUrl = GroupDTO.Image;

GroupDTO.Image = null;

}

isActiveMessage = GroupDTO.IsActive ? $"{Localizer["Group"]} {Localizer["Active"]}" : $"{Localizer["Group"]} {Localizer["Inactive"]}";

}

private void OnInvalidSubmit(EditContext editContext)

{

var messages = editContext.GetValidationMessages();

foreach (var message in messages)

{

Snackbar.Add(Localizer[message], Severity.Error);

}

}

private async Task LoadTournamentAsync()

{

var responseHttp = await Repository.GetAsync<List<Tournament>>("/api/tournaments/combo");

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

await SweetAlertService.FireAsync("Error", message, SweetAlertIcon.Error);

return;

}

tournaments = responseHttp.Response;

}

private async Task<IEnumerable<Tournament>> SearchTournament(string searchText, CancellationToken cancellationToken)

{

await Task.Delay(5);

if (string.IsNullOrWhiteSpace(searchText))

{

return tournaments!;

}

return tournaments!

.Where(x => x.Name.Contains(searchText, StringComparison.InvariantCultureIgnoreCase))

.ToList();

}

private void TournamentChanged(Tournament tournament)

{

selectedTournament = tournament;

GroupDTO.TournamentId = tournament.Id;

}

private void ImageSelected(string imagenBase64)

{

GroupDTO.Image = imagenBase64;

imageUrl = null;

}

private void SetTournamentOff()

{

GroupDTO.IsActive = false;

isActiveMessage = $"{Localizer["Group"]} {Localizer["Inactive"]}";

}

private void SetTournamentOn()

{

GroupDTO.IsActive = true;

isActiveMessage = $"{Localizer["Group"]} {Localizer["Active"]}";

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Creamos el **GroupForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit" OnInvalidSubmit="OnInvalidSubmit">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Group"]"

@bind-Value="@GroupDTO.Name"

For="@(() => GroupDTO.Name)"

Class="mb-4" />

@if(GroupDTO.Id == 0)

{

<MudAutocomplete T="Tournament"

Label=@Localizer["Tournament"]

Placeholder=@Localizer["SelectATournament"]

SearchFunc="SearchTournament"

Value="selectedTournament"

ValueChanged="TournamentChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

}

<MudTextField Label="@Localizer["Remarks"]"

@bind-Value="@GroupDTO.Remarks"

For="@(() => GroupDTO.Remarks)"

Class="mb-4"

Lines="5" />

<MudGrid Justify="Justify.SpaceBetween">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@isActiveMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (GroupDTO.IsActive)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="SetTournamentOff">

@Localizer["Deactivate"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="SetTournamentOn">

@Localizer["Activate"]

</MudButton>

}

</MudItem>

</MudGrid>

<div class="my-2">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **GroupCreate.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Pages.Teams;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupCreate

{

private GroupForm? groupForm;

private GroupDTO groupDTO = new() { IsActive = true };

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private async Task CreateAsync()

{

groupDTO.Code = "123456";

groupDTO.AdminId = "123456";

var responseHttp = await Repository.PostAsync<GroupDTO, Group>("/api/groups/full", groupDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

var group = responseHttp.Response;

Return();

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = string.Format(Localizer["GroupCreated"], group!.Name, group.Code),

Icon = SweetAlertIcon.Info,

});

}

private void Return()

{

groupForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/groups");

}

}

1. Modificamos el **GroupCreate.razor**:

<MudDialog>

<DialogContent>

<GroupForm @ref="groupForm" GroupDTO="groupDTO" OnValidSubmit="CreateAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

1. Probamos.
2. Ahora vamos a completar la edición de grupos. Modificamos el **GroupEdit.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupEdit

{

private GroupDTO? groupDTO;

private GroupForm? tournamentForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Group>($"api/groups/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

NavigationManager.NavigateTo("groups");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

else

{

var group = responseHttp.Response;

groupDTO = new GroupDTO()

{

AdminId = group.AdminId,

Name = group.Name,

Code = group.Code,

Id = group.Id,

Image = group.Image,

IsActive = group.IsActive,

Remarks = group.Remarks,

TournamentId = group.TournamentId

};

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/groups/full", groupDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

tournamentForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("groups");

}

}

1. Modificamos el **GroupEdit.razor**:

@if(groupDTO is null)

{

<Loading/>

}

else

{

<MudDialog>

<DialogContent>

<GroupForm @ref="tournamentForm" GroupDTO="groupDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Probamos y hacemos el **commit**.

## Unirme a un grupo existente

1. Adicionamos los siguientes literales:

| UserAddedToGroupOk | User added to the group. The user can now enter and enter their predictions. | Usuario agregado al grupo. Ya el usuario puede entrar e ingresar sus predicciones. |
| --- | --- | --- |
| ERR017 | The group code is not valid. | El código del grupo no es válido. |
| JoinExistingGroup | Join an existing group | Unirse a un grupo existente |

1. Creamos el **JoinGroupDTO**:

using Fantasy.Shared.Resources;

using System.ComponentModel.DataAnnotations;

namespace Fantasy.Shared.DTOs;

public class JoinGroupDTO

{

[Display(Name = "Code", ResourceType = typeof(Literals))]

[MaxLength(6, ErrorMessageResourceName = "MaxLength", ErrorMessageResourceType = typeof(Literals))]

[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Literals))]

public string Code { get; set; } = null!;

public string? UserName { get; set; }

}

1. Modificamos el **IUserGroupRepository**:

Task<ActionResponse<UserGroup>> JoinAsync(JoinGroupDTO joinGroupDTO);

1. Modificamos el **IUserGroupUnitOfWok**:

Task<ActionResponse<UserGroup>> JoinAsync(JoinGroupDTO joinGroupDTO);

1. Modificamos el **UserGroupRepository**:

public async Task<ActionResponse<UserGroup>> JoinAsync(JoinGroupDTO joinGroupDTO)

{

var group = await \_context.Groups.FirstOrDefaultAsync(x => x.Code == joinGroupDTO.Code);

if (group == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR017"

};

}

var user = await \_usersRepository.GetUserAsync(joinGroupDTO.UserName);

if (user == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR013"

};

}

var userGroup = new UserGroup

{

Group = group,

User = user,

};

\_context.Add(userGroup);

try

{

await \_context.SaveChangesAsync();

return new ActionResponse<UserGroup>

{

WasSuccess = true,

Result = userGroup

};

}

catch (DbUpdateException)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR003"

};

}

catch (Exception exception)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = exception.Message

};

}

}

1. Modificamos el **UserGroupUnitOfWork**:

public async Task<ActionResponse<UserGroup>> JoinAsync(JoinGroupDTO joinGroupDTO) => await \_userGroupsRepository.JoinAsync(joinGroupDTO);

1. Modificamos el **UserGroupController**:

[HttpPost("join")]

public async Task<IActionResult> PostAsync(JoinGroupDTO joinGroupDTO)

{

joinGroupDTO.UserName = User.Identity!.Name!;

var action = await \_userGroupsUnitOfWork.JoinAsync(joinGroupDTO);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest(action.Message);

}

1. Creamos el **JoinGroupForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class JoinGroupForm

{

private string? imageUrl;

private string? isActiveMessage;

private EditContext editContext = null!;

protected override void OnInitialized()

{

editContext = new(JoinGroupDTO);

}

[EditorRequired, Parameter] public JoinGroupDTO JoinGroupDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

private void OnInvalidSubmit(EditContext editContext)

{

var messages = editContext.GetValidationMessages();

foreach (var message in messages)

{

Snackbar.Add(Localizer[message], Severity.Error);

}

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

}

1. Creamos el **JoinGroupForm.razor**:

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit" OnInvalidSubmit="OnInvalidSubmit">

<DataAnnotationsValidator />

<MudTextField Label="@Localizer["Code"]"

@bind-Value="@JoinGroupDTO.Code"

For="@(() => JoinGroupDTO.Code)"

Class="mb-4" />

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

1. Modificamos el **JoinGroup.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class JoinGroup

{

private JoinGroupForm? joinGroupForm;

private JoinGroupDTO joinGroupDTO = new();

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

private async Task CreateAsync()

{

var responseHttp = await Repository.PostAsync("/api/usergroups/join", joinGroupDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message!], Severity.Error);

return;

}

var group = responseHttp.Response;

Return();

Snackbar.Add(Localizer["UserAddedToGroupOk"], Severity.Success);

}

private void Return()

{

joinGroupForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo("/groups");

}

}

1. Modificamos el **JoinGroup.razor**:

<MudDialog>

<DialogContent>

<JoinGroupForm @ref="joinGroupForm" JoinGroupDTO="joinGroupDTO" OnValidSubmit="CreateAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

1. Modificamos el **GroupsIndex.razor.cs**:

private async Task ShowModalJoinAsync()

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

var dialog = DialogService.Show<JoinGroup>($"{Localizer["JoinExistingGroup"]}", options);

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadTotalRecordsAsync();

await table.ReloadServerData();

}

}

1. Probamos y hacemos el **commit**.

## Unirme a un grupo por URL

1. Adicionamos los siguientes literales:

| ConfirmGroupMessage | To join the group: {0}, click the button. | Para unirse al grupo: {0}, haga click en el botón. |
| --- | --- | --- |
| CopyInvitationURLTitle | Copy invitation | Copiar invitación |
| InvitationURLCopied | Group invitation link {0} copied to clipboard. Share it with your friends to join the group. | Link de invitación al grupo {0} copiado al portapapeles. Compartirlo con sus amigos para que se unan al grupo. |

1. Modificamos el **IGroupsRepository**:

Task<ActionResponse<Group>> GetAsync(string code);

1. Modificamos el **IGroupsUnitOfWork**:

Task<ActionResponse<Group>> GetAsync(string code);

1. Modificamos el **GroupsRepository**:

public async Task<ActionResponse<Group>> GetAsync(string code)

{

var group = await \_context.Groups.FirstOrDefaultAsync(x => x.Code == code);

if (group == null)

{

return new ActionResponse<Group>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<Group>

{

WasSuccess = true,

Result = group

};

}

1. Modificamos el **GroupsUnitOfWork**:

public async Task<ActionResponse<Group>> GetAsync(string code) => await \_groupsRepository.GetAsync(code);

1. Modificamos el **GroupsController**:

[HttpGet("code/{code}")]

public async Task<IActionResult> GetAsync(string code)

{

var response = await \_groupsUnitOfWork.GetAsync(code);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

1. Creamos el recurso **Parameters.resx** con modificador público y la siguiente clave (reemplace el puerto por el suyo):

| URLFront | https://localhost:7069 |
| --- | --- |

1. En la carpeta **wwwroot** creamos la carpeta **scripts** y dentro de esta creamos **el copyToClipboard.js**:

function copyToClipboard(text) {

navigator.clipboard.writeText(text).then(function () {

console.log('Texto copiado al portapapeles');

}).catch(function (error) {

console.error('Error al copiar al portapapeles: ', error);

});

}

1. Adicionamos el llamado al script en el **index.html**:

<script src="\_content/MudBlazor/MudBlazor.min.js"></script>

<script src="scripts/copyToClipboard.js"></script>

</body>

1. En **Helpers** creamos el **IClipboardService**:

namespace Fantasy.Frontend.Helpers;

public interface IClipboardService

{

Task CopyToClipboardAsync(string text);

}

1. En **Helpers** creamos el **ClipboardService**:

using Microsoft.JSInterop;

namespace Fantasy.Frontend.Helpers;

public class ClipboardService : IClipboardService

{

private readonly IJSRuntime \_jsRuntime;

public ClipboardService(IJSRuntime jsRuntime)

{

\_jsRuntime = jsRuntime;

}

public async Task CopyToClipboardAsync(string text)

{

await \_jsRuntime.InvokeVoidAsync("copyToClipboard", text);

}

}

1. Creamos el **JoinGroupByURL.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class JoinGroupByURL

{

private string? message;

private Group? group;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Code { get; set; } = string.Empty;

protected override async Task OnParametersSetAsync()

{

var responseHttp = await Repository.GetAsync<Group>($"api/groups/code/{Code}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)

{

Snackbar.Add(Localizer["ERR017"], Severity.Error);

NavigationManager.NavigateTo("groups");

}

else

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

NavigationManager.NavigateTo("/");

}

return;

}

group = responseHttp.Response;

}

protected async Task JoinGroupAsync()

{

var responseHttp = await Repository.PostAsync($"/api/usergroups/join?code={Code}", new JoinGroupDTO { Code = Code });

if (responseHttp.Error)

{

message = await responseHttp.GetErrorMessageAsync();

NavigationManager.NavigateTo("/");

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Snackbar.Add(Localizer["UserAddedToGroupOk"], Severity.Success);

var closeOnEscapeKey = new DialogOptions() { CloseOnEscapeKey = true };

NavigationManager.NavigateTo("/groups");

}

}

1. Modificamos el **JoinGroupByURL.razor**:

@page "/groups/join"

@if(group is null)

{

<Loading/>

}

else

{

<MudPaper Class="confirmation-container p-4 shadow-sm">

<MudGrid>

<MudItem xs="12" Class="text-center mb-4">

<MudText Typo="Typo.h3">@Localizer["JoinGroup"]</MudText>

</MudItem>

<MudItem xs="12" Class="text-center mb-4">

<MudText Typo="Typo.body1">@string.Format(Localizer["ConfirmGroupMessage"], group.Name)</MudText>

<MudImage Src="@group.ImageFull" Width="160" />

</MudItem>

<MudItem xs="12" Class="text-center">

<MudButton Variant="Variant.Filled" Color="Color.Primary" OnClick="JoinGroupAsync">@Localizer["JoinGroup"]</MudButton>

</MudItem>

</MudGrid>

</MudPaper>

}

1. Modificamos el **GroupCreate.razor.cs**:

[Inject] private IStringLocalizer<Parameters> Parameters { get; set; } = null!;

[Inject] private IClipboardService ClipboardService { get; set; } = null!;

…

private async Task CreateAsync()

{

groupDTO.Code = "123456";

groupDTO.AdminId = "123456";

var responseHttp = await Repository.PostAsync<GroupDTO, Group>("/api/groups/full", groupDTO);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

var group = responseHttp.Response;

var joinURL = $"{Parameters["URLFront"]}/groups/join/?code={group!.Code}";

await ClipboardService.CopyToClipboardAsync(joinURL);

Return();

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = string.Format(Localizer["GroupCreated"], group!.Name, group.Code, joinURL),

Icon = SweetAlertIcon.Info,

});

}

1. Modificamos el **GroupIndex.razor.cs**:

[Inject] private IClipboardService ClipboardService { get; set; } = null!;

[Inject] private IStringLocalizer<Parameters> Parameters { get; set; } = null!;

…

private async Task CopyInvitationAsync(Group group)

{

var joinURL = $"{Parameters["URLFront"]}/groups/join/?code={group!.Code}";

await ClipboardService.CopyToClipboardAsync(joinURL);

var text = string.Format(Localizer["InvitationURLCopied"], group!.Name);

Snackbar.Add(text, Severity.Success);

}

1. Modificamos el **GroupIndex.razor**:

<HeaderContent>

<MudTh>@Localizer["Grupo"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Admin"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Tournament"]</MudTh>

<MudTh style="width: 80px;">@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Code"]</MudTh>

<MudTh style="width: 80px;">@Localizer["IsActive"]</MudTh>

<MudTh># @Localizer["Members"]</MudTh>

<MudTh style="width: 80px;">@Localizer["CopyInvitationURLTitle"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>

<img src="@context.ImageFull" style="width:80px;" />

</MudTd>

<MudTd>@context.Admin.FullName</MudTd>

<MudTd>

<img src="@context.Admin.PhotoFull" width="80" height="80" style="border-radius:50%" />

</MudTd>

<MudTd>@context.Tournament.Name</MudTd>

<MudTd>

<img src="@context.Tournament.ImageFull" style="width:80px;" />

</MudTd>

<MudTd>@context.Code</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>@context.MembersCount</MudTd>

<MudTd>

<MudButton Variant="Variant.Filled"

Color="Color.Secondary"

OnClick="@(() => CopyInvitationAsync(@context))"

Disabled="@(!context.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.ContentCopy" />

</MudButton>

</MudTd>

<MudTd>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Info"

OnClick="@(() => TeamsAction(@context))"

Class="me-2"

Disabled="@(!context.IsActive)">

@Localizer["GroupDetails"]

</MudButton>

@if (context.Admin.UserName == username)

{

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Edit"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))"

Class="m-2">

@Localizer["Edit"]

</MudButton>

}

</MudTd>

</RowTemplate>

1. Probamos y hacemos el **commit**.

## Ver detalles del grupo - primera parte: verificando predicciones para todos los partidos

1. Adicionamos los siguientes literales:

| Predictions | Predictions | Predicciones |
| --- | --- | --- |
| Positions | Positions | Posiciones |

1. Modificamos el **IGroupsRepository**:

Task CheckPredictionsForAllMatchesAsync(int id);

1. Modificamos el **IGroupsUnitOfWork**:

Task CheckPredictionsForAllMatchesAsync(int id);

1. Modificamos el **GroupsRepository**:

public async Task CheckPredictionsForAllMatchesAsync(int id)

{

var group = await \_context.Groups

.Include(x => x.Members)

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

if (group == null)

{

return;

}

var tournament = await \_context.Tournaments

.Include(x => x.Matches)

.FirstOrDefaultAsync(x => x.Id == group.TournamentId);

if (group == null)

{

return;

}

var newPredictions = new List<Prediction>();

foreach (var userGroup in group.Members!)

{

foreach (var match in tournament!.Matches!)

{

var prediction = await \_context.Predictions.FirstOrDefaultAsync(x => x.GroupId == group.Id &&

x.Match.Id == match.Id &&

x.UserId == userGroup.UserId &&

x.TournamentId == tournament.Id);

if (prediction == null)

{

newPredictions.Add(new Prediction

{

Group = group,

Match = match,

Tournament = tournament,

User = userGroup.User,

UserId = userGroup.UserId,

});

}

}

}

if (newPredictions.Count > 0)

{

try

{

\_context.AddRange(newPredictions);

await \_context.SaveChangesAsync();

}

catch (Exception ex)

{

ex.ToString();

}

}

}

1. Modificamos el **GroupsUnitOfWork**:

public async Task CheckPredictionsForAllMatchesAsync(int id) => await \_groupsRepository.CheckPredictionsForAllMatchesAsync(id);

1. Modificamos el **GroupsController**:

[HttpGet("CheckPredictionsForAllMatches/{id}")]

public async Task<IActionResult> CheckPredictionsForAllMatchesAsync(int id)

{

await \_groupsUnitOfWork.CheckPredictionsForAllMatchesAsync(id);

return Ok();

}

1. Adicionamos el **Predictions.razor.cs** temporal:

using Microsoft.AspNetCore.Components;

namespace Fantasy.Frontend.Pages.Groups;

public partial class Predictions

{

[Parameter] public int GroupId { get; set; }

}

1. Modificamos el **Predictions.razor** temporal:

<h3>Predictions</h3>

1. Adicionamos el **Positions.razor.cs** temporal:

using Microsoft.AspNetCore.Components;

namespace Fantasy.Frontend.Pages.Groups;

public partial class Positions

{

[Parameter] public int GroupId { get; set; }

}

1. Modificamos el **Positions.razor** temporal:

<h3>Positions</h3>

1. Creamos el **GroupDetails.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class GroupDetails

{

private Group? group;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Parameter] public int GroupId { get; set; }

protected override async Task OnParametersSetAsync()

{

await LoadGroupAsync();

await CheckPredictionsForAllMatchesAsync();

}

private async Task CheckPredictionsForAllMatchesAsync()

{

var responseHttp = await Repository.GetAsync($"api/groups/CheckPredictionsForAllMatches/{GroupId}");

}

private async Task LoadGroupAsync()

{

var responseHttp = await Repository.GetAsync<Group>($"api/groups/{GroupId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

NavigationManager.NavigateTo("groups");

return;

}

group = responseHttp.Response;

}

}

1. Creamos el **GroupDetails.razor**:

@page "/groups/details/{GroupId:int}"

@if(group is null)

{

<Loading/>

}

else

{

<MudPaper Class="p-4 my-4">

<MudGrid AlignItems="Center" JustifyContent="Center">

<MudItem xs="4" Class="d-flex justify-center">

<MudImage Src="@group.ImageFull" Height="100" />

</MudItem>

<MudItem xs="4" Class="d-flex justify-center">

<MudText Typo="Typo.h4" Align="Align.Center">@group.Name</MudText>

</MudItem>

<MudItem xs="4" Class="d-flex justify-center">

<MudImage Src="@group.Tournament.ImageFull" Height="100" />

</MudItem>

</MudGrid>

</MudPaper>

<MudTabs>

<MudTabPanel Text="@Localizer["Predictions"]">

<MudContainer MaxWidth="MaxWidth.Large">

<Predictions GroupId="GroupId"/>

</MudContainer>

</MudTabPanel>

<MudTabPanel Text="@Localizer["Positions"]">

<MudContainer MaxWidth="MaxWidth.Large">

<Positions GroupId="GroupId" />

</MudContainer>

</MudTabPanel>

</MudTabs>

}

1. Modificamos el **GroupsIndex.razor.cs**:

private void GroupDetails(Group group)

{

NavigationManager.NavigateTo($"/groups/details/{group.Id}");

}

1. Modificamos el **GroupsIndex.razor**:

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Info"

OnClick="@(() => GroupDetails(@context))"

Class="me-2">

@Localizer["GroupDetails"]

</MudButton>

1. Probamos y hacemos el **commit**.

## Ver detalles del grupo - segunda parte: listando predicciones

1. Adicionamos los siguientes literales:

| Watch | Watch | Ver |
| --- | --- | --- |
| Prediction | Prediction | Predicción |
| WatchPredictions | Watch Predictions | Ver predicciones |

1. Modificamos el **Predictions.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

[Authorize(Roles = "Admin, User")]

public partial class Predictions

{

private List<Prediction>? predictions;

private MudTable<Prediction> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrlMatch = "api/predictions";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

[Parameter] public int GroupId { get; set; }

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrlMatch}/totalRecordsPaginated/?id={GroupId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<Prediction>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrlMatch}/paginated?id={GroupId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Prediction>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

return new TableData<Prediction>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo("/groups");

}

private async Task EditPredictionAsync(Prediction prediction)

{

//TODO: Pending

}

private async Task WatchPredictionAsync(Prediction prediction)

{

//TODO: Pending

}

private bool CanWatch(DateTime date)

{

var difference = DateTime.Now - date;

var minutes = difference.TotalMinutes;

return minutes >= 10;

}

}

1. Modificamos el **Predictions.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@predictions"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true"

Class="mt-4">

<ToolBarContent>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Date"]</MudTh>

<MudTh>@Localizer["Local"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["GoalsLocal"]</MudTh>

<MudTh>@Localizer["GoalsVisitor"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["Points"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Match.DateLocal</MudTd>

<MudTd>@context.Match.Local.Name</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

<MudImage Src="@context.Match.Local.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsLocal</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsVisitor</MudText>

</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

<MudImage Src="@context.Match.Visitor.ImageFull" Width="90" Height="60" />

</MudTd>

<MudTd>@context.Match.Visitor.Name</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.Points</MudText>

</MudTd>

<MudTd>

@if (CanWatch(context))

{

<MudTooltip Text="@Localizer["WatchPredictions"]">

<MudButton Variant="Variant.Filled"

Color="Color.Info"

OnClick="@(() => WatchPredictionsAsync(@context))"

Disabled="@(!userEnabledForGroup)">

<MudIcon Icon="@Icons.Material.Filled.Visibility" />

</MudButton>

</MudTooltip>

}

else

{

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => EditPredictionAsync(context.Id))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

}

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos y hacemos el **commit**.

## Editar predicciones

1. Agregamos el **PredictionForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class PredictionForm

{

private EditContext editContext = null!;

private Match? match;

[EditorRequired, Parameter] public PredictionDTO PredictionDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

protected override void OnInitialized()

{

base.OnInitialized();

editContext = new(PredictionDTO);

}

protected override async Task OnParametersSetAsync()

{

await base.OnParametersSetAsync();

await LoadMathAsync();

}

private async Task LoadMathAsync()

{

var responseHttp = await Repository.GetAsync<Match>($"api/Matches/{PredictionDTO.MatchId}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

NavigationManager.NavigateTo($"groups/details/{PredictionDTO!.GroupId}");

return;

}

match = responseHttp.Response;

}

private void OnInvalidSubmit(EditContext editContext)

{

var messages = editContext.GetValidationMessages();

foreach (var message in messages)

{

Snackbar.Add(Localizer[message], Severity.Error);

}

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

private void ValidateInput()

{

if (PredictionDTO.GoalsLocal < 0)

{

PredictionDTO.GoalsLocal = 0;

}

if (PredictionDTO.GoalsVisitor < 0)

{

PredictionDTO.GoalsVisitor = 0;

}

}

}

1. Modificamos el **PredictionForm.razor**:

@if(match is null)

{

<Loading/>

}

else

{

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit" OnInvalidSubmit="OnInvalidSubmit">

<DataAnnotationsValidator />

<MudStack Row="true" Spacing="2" Class="my-4">

<MudStack Spacing="2">

<MudImage Src="@match.Local.ImageFull" Width="100" Height="60"/>

<MudText Typo="Typo.h5" Align="Align.Center">@match.Local.Name</MudText>

</MudStack>

<MudTextField @bind-Value="@PredictionDTO.GoalsLocal"

For="@(() => PredictionDTO.GoalsLocal)"

InputType="InputType.Number"

Adornment="Adornment.Start"

Style="font-size: 40px; text-align: center;"

Class="mb-4 p-4"

@onblur="ValidateInput" />

<MudText Typo="Typo.h3"

Align="Align.Center"

Class="mt-4">

Vs.

</MudText>

<MudTextField @bind-Value="@PredictionDTO.GoalsVisitor"

For="@(() => PredictionDTO.GoalsVisitor)"

InputType="InputType.Number"

Adornment="Adornment.Start"

Style="font-size: 40px; text-align: center;"

Class="mb-4 p-4"

@onblur="ValidateInput" />

<MudStack Spacing="2">

<MudImage Src="@match.Visitor.ImageFull" Width="100" Height="60" />

<MudText Typo="Typo.h5" Align="Align.Center">@match.Visitor.Name</MudText>

</MudStack>

</MudStack>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

}

1. Agregamos el **PredictionEdit.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class PredictionEdit

{

private PredictionDTO? predictionDTO;

private PredictionForm? predictionForm;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Prediction>($"api/Predictions/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

NavigationManager.NavigateTo($"groups/details/{predictionDTO!.GroupId}");

}

else

{

var prediction = responseHttp.Response;

predictionDTO = new PredictionDTO()

{

GoalsLocal = prediction!.GoalsLocal,

GoalsVisitor = prediction!.GoalsVisitor,

GroupId = prediction!.GroupId,

Id = prediction!.Id,

MatchId = prediction!.MatchId,

Points = prediction!.Points,

TournamentId = prediction!.TournamentId,

UserId = prediction!.UserId,

};

}

}

private async Task EditAsync()

{

var responseHttp = await Repository.PutAsync("api/Predictions/full", predictionDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

predictionForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"groups/details/{predictionDTO!.GroupId}/false");

}

}

1. Modificamos el **PredictionEdit.razor**:

@if (predictionDTO is null)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<PredictionForm @ref="predictionForm" PredictionDTO="predictionDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Modificamos el **Prediction.razor.cs**:

private async Task EditPredictionAsync(int id)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

var parameters = new DialogParameters

{

{ "Id", id }

};

var dialog = DialogService.Show<PredictionEdit>($"{Localizer["Edit"]} {Localizer["Prediction"]}", parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadAsync();

await table.ReloadServerData();

}

}

1. Modificamos el **Prediction.razor**:

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Edit"

Color="Color.Warning"

OnClick=@(() => EditPredictionAsync(context.Id))>

@Localizer["Edit"] @Localizer["Prediction"]

</MudButton>

1. Probamos y hacemos el **commit**.

## Cerrar un partido

1. Adicionamos los siguientes literales:

| CloseMatch | Close Match | Cerrar Partido |
| --- | --- | --- |
| CloseMatchConfirmMessage | Are you sure you want to close the match {0} Vs. {1}? | ¿Estás seguro de cerrar el partido {0} Vs. {1}? |
| CloseMatchTitle | Close Match | Cerrar Partido |
| GoalsLocalError | You must enter a value greater than or equal to zero in local goals. | Debes ingresar un valor mayor o igual a cero en goles del local. |
| GoalsVisitorError | You must enter a value greater than or equal to zero in visitor goals. | Debes ingresar un valor mayor o igual a cero en goles del visitante. |
| ERR018 | This match is no longer open to predictions. | Este partido ya no admite predicciones. |

1. Creamos la enumeración **MatchStatus**:

namespace Fantasy.Shared.Enums;

public enum MatchStatus

{

LocalWin,

Tie,

VisitorWin

}

1. Modificamos el **MatchesRepository**:

\_context.Update(currentMatch);

try

{

await \_context.SaveChangesAsync();

if (currentMatch.GoalsLocal != null && currentMatch.GoalsVisitor != null)

{

await CloseMatchAsync(currentMatch);

}

return new ActionResponse<Match>

{

WasSuccess = true,

Result = currentMatch

};

}

…

private async Task CloseMatchAsync(Match match)

{

var predictions = await \_context.Predictions

.Where(x => x.MatchId == match.Id)

.ToListAsync();

foreach (var prediction in predictions)

{

var points = CalculatePoints(match, prediction);

prediction.Points = points;

\_context.Update(prediction);

}

await \_context.SaveChangesAsync();

}

private int CalculatePoints(Match match, Prediction prediction)

{

int points = 0;

var matchStatus = GetMatchStatus(match.GoalsLocal!.Value, match.GoalsVisitor!.Value);

var predictionStatus = GetMatchStatus(prediction.GoalsLocal!.Value, prediction.GoalsVisitor!.Value);

if (matchStatus == predictionStatus) points += 5;

if (match.GoalsLocal == prediction.GoalsLocal) points += 2;

if (match.GoalsVisitor == prediction.GoalsVisitor) points += 2;

if (Math.Abs((decimal)match.GoalsLocal! - (decimal)match.GoalsVisitor!) == Math.Abs((decimal)prediction.GoalsLocal! - (decimal)prediction.GoalsVisitor!)) points++;

return points;

}

private MatchStatus GetMatchStatus(int goalsLocal, int goalsVisitor)

{

if (goalsLocal > goalsVisitor) return MatchStatus.LocalWin;

if (goalsLocal < goalsVisitor) return MatchStatus.VisitorWin;

return MatchStatus.Tie;

}

1. Modificamos el **PredictionsRepository**:

public async Task<ActionResponse<Prediction>> UpdateAsync(PredictionDTO predictionDTO)

{

var currentPrediction = await \_context.Predictions

.Include(x => x.Match)

.FirstOrDefaultAsync(x => x.Id == predictionDTO.Id);

if (currentPrediction == null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR016"

};

}

if (currentPrediction.Match.GoalsLocal != null || currentPrediction.Match.GoalsVisitor != null)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR018"

};

}

var difference = currentPrediction.Match.Date - DateTime.UtcNow;

var minutes = difference.TotalMinutes;

if (minutes <= 10)

{

return new ActionResponse<Prediction>

{

WasSuccess = false,

Message = "ERR018"

};

}

currentPrediction.GoalsLocal = predictionDTO.GoalsLocal;

currentPrediction.GoalsVisitor = predictionDTO.GoalsVisitor;

currentPrediction.Points = predictionDTO.Points;

…

1. Adicionamos el **CloseForm.razor.cs**:

using CurrieTechnologies.Razor.SweetAlert2;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components.Forms;

using Microsoft.AspNetCore.Components.Routing;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

using Fantasy.Shared.Entities;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class CloseForm

{

private EditContext editContext = null!;

private Match? match;

[EditorRequired, Parameter] public MatchDTO MatchDTO { get; set; } = null!;

[EditorRequired, Parameter] public EventCallback OnValidSubmit { get; set; }

[EditorRequired, Parameter] public EventCallback ReturnAction { get; set; }

public bool FormPostedSuccessfully { get; set; } = false;

[Inject] private SweetAlertService SweetAlertService { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

protected override void OnInitialized()

{

base.OnInitialized();

editContext = new(MatchDTO);

}

protected override async Task OnParametersSetAsync()

{

await base.OnParametersSetAsync();

await LoadMathAsync();

}

private async Task LoadMathAsync()

{

var responseHttp = await Repository.GetAsync<Match>($"api/Matches/{MatchDTO.Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

NavigationManager.NavigateTo($"/tournament/matches/{MatchDTO.TournamentId}");

return;

}

match = responseHttp.Response;

}

private void OnInvalidSubmit(EditContext editContext)

{

var messages = editContext.GetValidationMessages();

foreach (var message in messages)

{

Snackbar.Add(Localizer[message], Severity.Error);

}

}

private async Task OnBeforeInternalNavigation(LocationChangingContext context)

{

var formWasEdited = editContext.IsModified();

if (!formWasEdited || FormPostedSuccessfully)

{

return;

}

var result = await SweetAlertService.FireAsync(new SweetAlertOptions

{

Title = Localizer["Confirmation"],

Text = Localizer["LeaveAndLoseChanges"],

Icon = SweetAlertIcon.Warning,

ShowCancelButton = true,

CancelButtonText = Localizer["Cancel"],

});

var confirm = !string.IsNullOrEmpty(result.Value);

if (confirm)

{

return;

}

context.PreventNavigation();

}

private void ValidateInput()

{

if (MatchDTO.GoalsLocal < 0)

{

MatchDTO.GoalsLocal = 0;

}

if (MatchDTO.GoalsVisitor < 0)

{

MatchDTO.GoalsVisitor = 0;

}

}

}

1. Modificamos el **CloseForm.razor**:

@if (match is null)

{

<Loading />

}

else

{

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation" />

<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit" OnInvalidSubmit="OnInvalidSubmit">

<DataAnnotationsValidator />

<MudStack Row="true" Spacing="2" Class="my-4">

<MudStack Spacing="2">

<MudImage Src="@match.Local.ImageFull" Width="100" Height="60" />

<MudText Typo="Typo.h5" Align="Align.Center">@match.Local.Name</MudText>

</MudStack>

<MudTextField @bind-Value="@MatchDTO.GoalsLocal"

For="@(() => MatchDTO.GoalsLocal)"

InputType="InputType.Number"

Adornment="Adornment.Start"

Style="font-size: 40px; text-align: center;"

Class="mb-4 p-4"

@onblur="ValidateInput" />

<MudText Typo="Typo.h3"

Align="Align.Center"

Class="mt-4">

Vs.

</MudText>

<MudTextField @bind-Value="@MatchDTO.GoalsVisitor"

For="@(() => MatchDTO.GoalsVisitor)"

InputType="InputType.Number"

Adornment="Adornment.Start"

Style="font-size: 40px; text-align: center;"

Class="mb-4 p-4"

@onblur="ValidateInput" />

<MudStack Spacing="2">

<MudImage Src="@match.Visitor.ImageFull" Width="100" Height="60" />

<MudText Typo="Typo.h5" Align="Align.Center">@match.Visitor.Name</MudText>

</MudStack>

</MudStack>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.Check"

Color="Color.Primary"

ButtonType="ButtonType.Submit">

@Localizer["SaveChanges"]

</MudButton>

</EditForm>

}

1. Adicionamos el **CloseMatch.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Frontend.Shared;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Tournaments;

public partial class CloseMatch

{

private MatchDTO? matchDTO;

private CloseForm? closeForm;

private Match? match;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

var responseHttp = await Repository.GetAsync<Match>($"api/Matches/{Id}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

NavigationManager.NavigateTo($"/tournament/matches/{matchDTO!.TournamentId}");

}

else

{

match = responseHttp.Response;

matchDTO = new MatchDTO()

{

GoalsLocal = match!.GoalsLocal,

GoalsVisitor = match!.GoalsVisitor,

Id = match!.Id,

TournamentId = match!.TournamentId,

Date = match!.Date,

IsActive = match!.IsActive,

LocalId = match!.LocalId,

VisitorId = match!.VisitorId,

};

}

}

private async Task EditAsync()

{

if (matchDTO!.GoalsLocal == null || matchDTO.GoalsLocal < 0)

{

Snackbar.Add(Localizer["GoalsLocalError"], Severity.Error);

return;

}

if (matchDTO!.GoalsVisitor == null || matchDTO.GoalsVisitor < 0)

{

Snackbar.Add(Localizer["GoalsVisitorError"], Severity.Error);

return;

}

var parameters = new DialogParameters

{

{ "Message", string.Format(Localizer["CloseMatchConfirmMessage"], match!.Local.Name, match.Visitor.Name) }

};

var options = new DialogOptions { CloseButton = true, MaxWidth = MaxWidth.ExtraSmall, CloseOnEscapeKey = true };

var dialog = DialogService.Show<ConfirmDialog>(Localizer["Confirmation"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

return;

}

var responseHttp = await Repository.PutAsync("api/Matches/full", matchDTO);

if (responseHttp.Error)

{

var mensajeError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[mensajeError!], Severity.Error);

return;

}

Return();

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

private void Return()

{

closeForm!.FormPostedSuccessfully = true;

NavigationManager.NavigateTo($"/tournament/matches/{matchDTO!.TournamentId}");

}

}

1. Modificamos el **CloseMatch.razor.cs**:

@if (matchDTO is null)

{

<Loading />

}

else

{

<MudDialog>

<DialogContent>

<CloseForm @ref="closeForm" MatchDTO="matchDTO" OnValidSubmit="EditAsync" ReturnAction="Return" />

</DialogContent>

</MudDialog>

}

1. Modificamos el **TournamentMatches.razor.cs**:

private async Task CloseMatchAsync(int id)

{

var options = new DialogOptions() { CloseOnEscapeKey = true, CloseButton = true };

var parameters = new DialogParameters

{

{ "Id", id }

};

var dialog = DialogService.Show<CloseMatch>(Localizer["CloseMatchTitle"], parameters, options);

var result = await dialog.Result;

if (result!.Canceled)

{

await LoadAsync();

await table.ReloadServerData();

}

}

1. Modificamos el **TournamentMatches.razor**:

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsLocal</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsVisitor</MudText>

</MudTd>

…

<MudTd>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Edit"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))"

Class="me-2"

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

@Localizer["Edit"]

</MudButton>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Close"

Color="Color.Info"

OnClick="@(() => CloseMatchAsync(context.Id))"

Class="me-2"

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

@Localizer["CloseMatch"]

</MudButton>

<MudButton Variant="Variant.Outlined"

EndIcon="@Icons.Material.Filled.Delete"

Color="Color.Error"

OnClick=@(() => DeleteAsync(@context))

Disabled="@(context.GoalsLocal != null || context.GoalsVisitor != null)">

@Localizer["Delete"]

</MudButton>

</MudTd>

1. Modificamos el **Predictions.razor.cs**:

private bool CanWatch(Prediction prediction)

{

if (prediction.Match.GoalsLocal != null || prediction.Match.GoalsVisitor != null)

{

return true;

}

var dateMatch = prediction.Match.Date.ToLocalTime();

var currentDate = DateTime.Now;

var minutesMatch = dateMatch.Subtract(DateTime.MinValue).TotalMinutes;

var minutesNow = currentDate.Subtract(DateTime.MinValue).TotalMinutes;

var difference = minutesNow - minutesMatch;

var canWatch = difference >= -10;

return canWatch;

}

1. Modificamos el **Predictions.razor**:

@if (CanWatch(context))

1. Probamos y hacemos el **commit**.

## Ver posiciones en un grupo

1. Creamos el **PositionDTO**:

using Fantasy.Shared.Entities;

namespace Fantasy.Shared.DTOs

{

public class PositionDTO

{

public User User { get; set; } = null!;

public int Points { get; set; }

}

}

1. Modificamos el **IPredictionsRepository**:

Task<ActionResponse<IEnumerable<PositionDTO>>> GetPositionsAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsForPositionsAsync(PaginationDTO pagination);

1. Modificamos el **IPredictionsUnitOfWork**:

Task<ActionResponse<IEnumerable<PositionDTO>>> GetPositionsAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsForPositionsAsync(PaginationDTO pagination);

1. Modificamos el **PredictionsRepository**:

public async Task<ActionResponse<IEnumerable<PositionDTO>>> GetPositionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Where(x => x.GroupId == pagination.Id && x.Points.HasValue)

.GroupBy(x => x.User)

.Select(g => new PositionDTO

{

User = g.Key,

Points = g.Sum(x => x.Points ?? 0)

})

.OrderByDescending(x => x.Points)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<PositionDTO>>

{

WasSuccess = true,

Result = await queryable

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsForPositionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Where(x => x.GroupId == pagination.Id && x.Points.HasValue)

.GroupBy(x => x.User)

.Select(g => new PositionDTO

{

User = g.Key,

Points = g.Sum(x => x.Points ?? 0)

})

.OrderByDescending(x => x.Points)

.AsQueryable();

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **PredictionsUnitOfWork**:

public async Task<ActionResponse<IEnumerable<PositionDTO>>> GetPositionsAsync(PaginationDTO pagination) => await \_predictionsRepository.GetPositionsAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsForPositionsAsync(PaginationDTO pagination) => await \_predictionsRepository.GetTotalRecordsForPositionsAsync(pagination);

1. Modificamos el **PredictionsController**:

[HttpGet("positions")]

public async Task<IActionResult> GetPositionsAsync([FromQuery] PaginationDTO pagination)

{

var response = await \_predictionsUnitOfWork.GetPositionsAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsForPositionsPaginated")]

public async Task<IActionResult> GetTotalRecordsForPositionsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_predictionsUnitOfWork.GetTotalRecordsForPositionsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Modificamos el **Positions.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

[Authorize(Roles = "Admin, User")]

public partial class Positions

{

private List<PositionDTO>? positions;

private MudTable<PositionDTO> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrlMatch = "api/predictions";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

[Parameter] public int GroupId { get; set; }

[Parameter] public bool IsAnonymouns { get; set; }

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrlMatch}/totalRecordsForPositionsPaginated/?id={GroupId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<PositionDTO>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrlMatch}/positions/?id={GroupId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<PositionDTO>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<PositionDTO> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<PositionDTO> { Items = [], TotalItems = 0 };

}

return new TableData<PositionDTO>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

if (IsAnonymouns)

{

NavigationManager.NavigateTo("/");

}

else

{

NavigationManager.NavigateTo("/groups");

}

}

private async Task WatchBalanceAsync(PositionDTO positionDTO)

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Medium,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", GroupId },

{ "Email", positionDTO.User.Email }

};

var dialog = DialogService.Show<Balance>(Localizer["PredictionsBalance"], parameters, options);

await dialog.Result;

}

}

1. Modificamos el **Positions.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@positions"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true"

Class="mt-4">

<ToolBarContent>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["User"]</MudTh>

<MudTh>@Localizer["Points"]</MudTh>

<MudTh style="width: 170px;">@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudImage Src="@context.User.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h4" Align="Align.Center">@context.User.FullName</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.Points</MudText>

</MudTd>

<MudTd>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Visibility"

Color="Color.Info"

OnClick=@(() => WatchPredictionAsync(@context))>

@Localizer["Watch"] @Localizer["Predictions"]

</MudButton>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Probamos y hacemos el **commit**.

## Ver las otras predicciones

1. Adicionamos los siguientes literales:

| RealScore | Real Score | Marcador Real |
| --- | --- | --- |
| PredictedScore | Predicted Score | Marcador Predecido |

1. Modificamos el **PaginationDTO**:

public int Id2 { get; set; }

1. Modificamos el **IPredictionsRepository**:

Task<ActionResponse<IEnumerable<Prediction>>> GetAllPredictionsAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAllPredictionsAsync(PaginationDTO pagination);

1. Modificamos el **IPredictionsUnitOfWork**:

Task<ActionResponse<IEnumerable<Prediction>>> GetAllPredictionsAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsAllPredictionsAsync(PaginationDTO pagination);

1. Modificamos el **PredictionsRepository**:

public async Task<ActionResponse<IEnumerable<Prediction>>> GetAllPredictionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

.Include(x => x.User)

.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.MatchId == pagination.Id2);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Prediction>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.User.FirstName)

.ThenBy(x => x.User.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsAllPredictionsAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.MatchId == pagination.Id2);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.User.FirstName.ToLower().Contains(pagination.Filter.ToLower()) ||

x.User.LastName.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **PredictionsUnitOfWork**:

public async Task<ActionResponse<IEnumerable<Prediction>>> GetAllPredictionsAsync(PaginationDTO pagination) => await \_predictionsRepository.GetAllPredictionsAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsAllPredictionsAsync(PaginationDTO pagination) => await \_predictionsRepository.GetTotalRecordsAllPredictionsAsync(pagination);

1. Modificamos el **PredictionsController**:

[HttpGet("paginatedAllPredictions")]

public async Task<IActionResult> GetAllPredictionsAsync([FromQuery] PaginationDTO pagination)

{

var response = await \_predictionsUnitOfWork.GetAllPredictionsAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsPaginatedAllPredictions")]

public async Task<IActionResult> GetTotalRecordsAllPredictionsAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_predictionsUnitOfWork.GetTotalRecordsAllPredictionsAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Adicionamos el **WatchPredictions.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class WatchPredictions

{

private List<Prediction>? predictions;

private MudTable<Prediction> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/predictions";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

private Match? match;

[Parameter] public int GroupId { get; set; }

[Parameter] public int MatchId { get; set; }

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginatedAllPredictions/?id={GroupId}&id2={MatchId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<Prediction>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginatedAllPredictions/?id={GroupId}&id2={MatchId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Prediction>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

return new TableData<Prediction>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo($"/groups/details/{GroupId}/false");

}

}

1. Modificamos el **WatchPredictions.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@predictions"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true"

Class="mt-4">

<ToolBarContent>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["User"]</MudTh>

<MudTh>@Localizer["Local"]</MudTh>

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["RealScore"]</MudTh>

<MudTh>@Localizer["PredictedScore"]</MudTh>

<MudTh>@Localizer["Points"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudImage Src="@context.User.PhotoFull" Width="60" Height="60" Style="border-radius: 50%;" />

</MudTd>

<MudTd>@context.User.FullName</MudTd>

<MudTd>

<MudImage Src="@context.Match.Local.ImageFull" Width="60" Height="40" />

</MudTd>

<MudTd>

<MudImage Src="@context.Match.Visitor.ImageFull" Width="60" Height="40" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.Match.GoalsLocal - @context.Match.GoalsVisitor</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.GoalsLocal - @context.GoalsVisitor</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.Points</MudText>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **Predictions.razor.cs**:

private async Task WatchPredictionsAsync(Prediction prediction)

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Medium,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", prediction.GroupId },

{ "MatchId", prediction.MatchId }

};

var dialog = DialogService.Show<WatchPredictions>($"{Localizer["Watch"]} {Localizer["Predictions"]}", parameters, options);

await dialog.Result;

}

1. Modificamos el **Predictions.razor**:

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Visibility"

Color="Color.Info"

OnClick=@(() => WatchPredictionsAsync(@context))>

@Localizer["Watch"] @Localizer["Predictions"]

</MudButton>

1. Probamos y hacemos el **commit**.

## Ver mi balance de puntos

1. Adicionamos el siguiente literal:

| PredictionsBalance | Predictions Balance | Resumen de Predicciones |
| --- | --- | --- |

1. Modificamos el **IPredictionsRepository**:

Task<ActionResponse<IEnumerable<Prediction>>> GetBalanceAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsBalanceAsync(PaginationDTO pagination);

1. Modificamos el **IPredictionsUnitOfWork**:

Task<ActionResponse<IEnumerable<Prediction>>> GetBalanceAsync(PaginationDTO pagination);

Task<ActionResponse<int>> GetTotalRecordsBalanceAsync(PaginationDTO pagination);

1. Modificamos el **PredictionsRepository**:

public async Task<ActionResponse<IEnumerable<Prediction>>> GetBalanceAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions

.Include(x => x.Match)

.ThenInclude(x => x.Local)

.Include(x => x.Match)

.ThenInclude(x => x.Visitor)

.Include(x => x.User)

.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

return new ActionResponse<IEnumerable<Prediction>>

{

WasSuccess = true,

Result = await queryable

.OrderBy(x => x.User.FirstName)

.ThenBy(x => x.User.LastName)

.Paginate(pagination)

.ToListAsync()

};

}

public async Task<ActionResponse<int>> GetTotalRecordsBalanceAsync(PaginationDTO pagination)

{

var queryable = \_context.Predictions.AsQueryable();

queryable = queryable.Where(x => x.GroupId == pagination.Id);

queryable = queryable.Where(x => x.User.Email == pagination.Email);

if (!string.IsNullOrWhiteSpace(pagination.Filter))

{

queryable = queryable.Where(x => x.Match.Local.Name.ToLower().Contains(pagination.Filter.ToLower()) ||

x.Match.Visitor.Name.ToLower().Contains(pagination.Filter.ToLower()));

}

double count = await queryable.CountAsync();

return new ActionResponse<int>

{

WasSuccess = true,

Result = (int)count

};

}

1. Modificamos el **PredictionsUnitOfWork**:

public async Task<ActionResponse<IEnumerable<Prediction>>> GetBalanceAsync(PaginationDTO pagination) => await \_predictionsRepository.GetBalanceAsync(pagination);

public async Task<ActionResponse<int>> GetTotalRecordsBalanceAsync(PaginationDTO pagination) => await \_predictionsRepository.GetTotalRecordsBalanceAsync(pagination);

1. Modificamos el **PredictionsController**:

[HttpGet("paginatedBalance")]

public async Task<IActionResult> GetBalanceAsync([FromQuery] PaginationDTO pagination)

{

var response = await \_predictionsUnitOfWork.GetBalanceAsync(pagination);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

[HttpGet("totalRecordsBalance")]

public async Task<IActionResult> GetTotalRecordsBalanceAsync([FromQuery] PaginationDTO pagination)

{

var action = await \_predictionsUnitOfWork.GetTotalRecordsBalanceAsync(pagination);

if (action.WasSuccess)

{

return Ok(action.Result);

}

return BadRequest();

}

1. Adicionamos el **Balance.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class Balance

{

private List<Prediction>? predictions;

private MudTable<Prediction> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/predictions";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

private Match? match;

[Parameter] public int GroupId { get; set; }

[Parameter] public string Email { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrl}/totalRecordsBalance/?id={GroupId}&email={Email}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<Prediction>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginatedBalance/?id={GroupId}&email={Email}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<Prediction>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<Prediction> { Items = [], TotalItems = 0 };

}

return new TableData<Prediction>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo($"/groups/details/{GroupId}/false");

}

}

1. Modificamos el **Balance.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@predictions"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true"

Class="mt-4">

<ToolBarContent>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Local"]</MudTh>

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["Local"]</MudTh>

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["RealScore"]</MudTh>

<MudTh>@Localizer["PredictedScore"]</MudTh>

<MudTh>@Localizer["Points"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Center">@context.Match.Local.Name</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Center">@context.Match.Visitor.Name</MudText>

</MudTd>

<MudTd>

<MudImage Src="@context.Match.Local.ImageFull" Width="60" Height="40" />

</MudTd>

<MudTd>

<MudImage Src="@context.Match.Visitor.ImageFull" Width="60" Height="40" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.Match.GoalsLocal - @context.Match.GoalsVisitor</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.GoalsLocal - @context.GoalsVisitor</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.Points</MudText>

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **Positions.razor.cs**:

private async Task WatchBalanceAsync(PositionDTO positionDTO)

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Medium,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", GroupId },

{ "Email", positionDTO.User.Email }

};

var dialog = DialogService.Show<Balance>(Localizer["PredictionsBalance"], parameters, options);

await dialog.Result;

}

1. Modificamos el **Positions.razor**:

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Visibility"

Color="Color.Info"

OnClick=@(() => WatchBalanceAsync(@context))>

@Localizer["PredictionsBalance"]

</MudButton>

1. Probamos y hacemos el **commit**.

## Administrando los usuarios de mis grupos

1. Adicionamos el siguiente literal:

| AdminUsersGroup | Admin users group | Administrar usuarios de un grupo |
| --- | --- | --- |

1. Modificamos el **IUserGroupsRepository**:

Task<ActionResponse<UserGroup>> GetAsync(int groupId, string email);

1. Modificamos el **IUserGroupsUnitOfWork**:

Task<ActionResponse<UserGroup>> GetAsync(int groupId, string email);

1. Modificamos el **UserGroupsRepository**:

public async Task<ActionResponse<UserGroup>> GetAsync(int groupId, string email)

{

var userGroup = await \_context.UserGroups

.Include(x => x.User)

.FirstOrDefaultAsync(x => x.GroupId == groupId && x.User.Email == email);

if (userGroup == null)

{

return new ActionResponse<UserGroup>

{

WasSuccess = false,

Message = "ERR001"

};

}

return new ActionResponse<UserGroup>

{

WasSuccess = true,

Result = userGroup

};

}

1. Modificamos el **UserGroupsUnitOfWork**:

public async Task<ActionResponse<UserGroup>> GetAsync(int groupId, string email) => await \_userGroupsRepository.GetAsync(groupId, email);

1. Modificamos el **UserGroupsController**:

[HttpGet("{groupId}/{email}")]

public async Task<IActionResult> GetAsync(int groupId, string email)

{

var response = await \_userGroupsUnitOfWork.GetAsync(groupId, email);

if (response.WasSuccess)

{

return Ok(response.Result);

}

return NotFound(response.Message);

}

1. Agregamos el **UsersGroup.razor.cs**:

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages.Groups;

public partial class UsersGroup

{

private List<UserGroup>? userGroups;

private MudTable<UserGroup> table = new();

private readonly int[] pageSizeOptions = { 10, 25, 50, int.MaxValue };

private int totalRecords = 0;

private bool loading;

private const string baseUrl = "api/userGroups";

private string infoFormat = "{first\_item}-{last\_item} de {all\_items}";

[Parameter] public int GroupId { get; set; }

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Parameter, SupplyParameterFromQuery] public string Filter { get; set; } = string.Empty;

protected override async Task OnInitializedAsync()

{

await LoadAsync();

}

private async Task LoadAsync()

{

await LoadTotalRecords();

}

private async Task<bool> LoadTotalRecords()

{

loading = true;

var url = $"{baseUrl}/totalRecordsPaginated/?id={GroupId}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<int>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return false;

}

totalRecords = responseHttp.Response;

loading = false;

return true;

}

private async Task<TableData<UserGroup>> LoadListAsync(TableState state, CancellationToken cancellationToken)

{

int page = state.Page + 1;

int pageSize = state.PageSize;

var url = $"{baseUrl}/paginated/?id={GroupId}&page={page}&recordsnumber={pageSize}";

if (!string.IsNullOrWhiteSpace(Filter))

{

url += $"&filter={Filter}";

}

var responseHttp = await Repository.GetAsync<List<UserGroup>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return new TableData<UserGroup> { Items = [], TotalItems = 0 };

}

if (responseHttp.Response == null)

{

return new TableData<UserGroup> { Items = [], TotalItems = 0 };

}

return new TableData<UserGroup>

{

Items = responseHttp.Response,

TotalItems = totalRecords

};

}

private async Task SetFilterValue(string value)

{

Filter = value;

await LoadAsync();

await table.ReloadServerData();

}

private void ReturnAction()

{

NavigationManager.NavigateTo($"/groups");

}

private async Task ActivateUserGroupAsync(UserGroup userGroup)

{

userGroup.IsActive = true;

await UpdateUserGroupAsync(userGroup);

}

private async Task DectivateUserGroupAsync(UserGroup userGroup)

{

userGroup.IsActive = false;

await UpdateUserGroupAsync(userGroup);

}

private async Task UpdateUserGroupAsync(UserGroup userGroup)

{

var userGroupDTO = new UserGroupDTO

{

IsActive = userGroup.IsActive,

Id = userGroup.Id,

GroupId = userGroup.Id,

UserId = userGroup.UserId,

};

var responseHttp = await Repository.PutAsync($"{baseUrl}/full", userGroupDTO);

if (responseHttp.Error)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

return;

}

Snackbar.Add(Localizer["RecordSavedOk"], Severity.Success);

}

}

1. Modificamos el **UsersGroup.razor**:

@if (loading)

{

<Loading />

}

else

{

<MudTable Items="@userGroups"

@ref="table"

ServerData="LoadListAsync"

Dense="true"

Hover="true"

Striped="true"

FixedHeader="true"

FixedFooter="true"

Class="mt-4">

<ToolBarContent>

<MudButton Variant="Variant.Outlined"

Class="mr-4"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Tertiary"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

<MudSpacer />

<FilterComponent ApplyFilter="SetFilterValue" />

</ToolBarContent>

<HeaderContent>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["FirstName"]</MudTh>

<MudTh>@Localizer["LastName"]</MudTh>

<MudTh>@Localizer["Email"]</MudTh>

<MudTh>@Localizer["PhoneNumber"]</MudTh>

<MudTh style="width: 80px;">@Localizer["IsActive"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>

<MudImage Src="@context.User.PhotoFull" Width="60" Height="60" Style="border-radius: 50%;" />

</MudTd>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Start">@context.User.FirstName</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Start">@context.User.LastName</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Start">@context.User.Email</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.body1" Align="Align.Start">@context.User.PhoneNumber</MudText>

</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>

@if (context.IsActive)

{

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="@(() => DectivateUserGroupAsync(context))">

@Localizer["Deactivate"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="@(() => ActivateUserGroupAsync(context))">

@Localizer["Activate"]

</MudButton>

}

</MudTd>

</RowTemplate>

<NoRecordsContent>

<MudText>@Localizer["NoRecords"]</MudText>

</NoRecordsContent>

<PagerContent>

<MudTablePager RowsPerPageString=@Localizer["RecordsNumber"]

PageSizeOptions="pageSizeOptions"

AllItemsText=@Localizer["All"]

InfoFormat="@infoFormat" />

</PagerContent>

</MudTable>

}

1. Modificamos el **GroupsIndex.razor.cs**:

private async Task AdminUsersGroupAsync(Group group)

{

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Medium,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", group.Id },

};

var dialog = DialogService.Show<UsersGroup>(@Localizer["AdminUsersGroup"], parameters, options);

await dialog.Result;

}

}

1. Modificamos el **GroupsIndex.razor**:

<MudStack Row="true" Spacing="2">

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => ShowModalAsync(context.Id, true))">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["CopyInvitationURLTitle"]">

<MudButton Variant="Variant.Filled"

Color="Color.Secondary"

OnClick="@(() => CopyInvitationAsync(@context))"

Disabled="@(!context.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.ContentCopy" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["AdminUsersGroup"]">

<MudButton Variant="Variant.Filled"

Color="Color.Primary"

OnClick="@(() => AdminUsersGroupAsync(@context))"

Disabled="@(!context.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.People" />

</MudButton>

</MudTooltip>

</MudStack>

1. Modificamos el **Predictions.razor.cs**:

…

private bool userEnabledForGroup;

private string username = string.Empty;

…

[Inject] private AuthenticationStateProvider AuthenticationStateProvider { get; set; } = null!;

…

protected override async Task OnInitializedAsync()

{

await LoadAsync();

await LoadUserNameAsync();

await CheckUserEnabledAsync();

}

private async Task CheckUserEnabledAsync()

{

var responseHttp = await Repository.GetAsync<UserGroup>($"api/userGroups/{GroupId}/{username}");

if (responseHttp.Error)

{

if (responseHttp.HttpResponseMessage.StatusCode != System.Net.HttpStatusCode.NotFound)

{

var messageError = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(messageError, Severity.Error);

}

}

var userGroup = responseHttp.Response;

userEnabledForGroup = userGroup!.IsActive;

}

private async Task LoadUserNameAsync()

{

var authState = await AuthenticationStateProvider.GetAuthenticationStateAsync();

var user = authState.User;

if (user.Identity != null && user.Identity.IsAuthenticated)

{

username = user.Identity.Name!;

}

}

1. Modificamos el **Predictions.razor**:

<MudTooltip Text="@Localizer["Edit"]">

<MudButton Variant="Variant.Filled"

Color="Color.Warning"

OnClick="@(() => EditPredictionAsync(context.Id))"

Disabled="@(!userEnabledForGroup)">

<MudIcon Icon="@Icons.Material.Filled.Edit" />

</MudButton>

</MudTooltip>

1. Probamos y hacemos el **commit**.

## Creando el Home de la App

1. Adicionamos el archivo de fondo, para mi ejemplo **61gpbM.jpg**.
2. Modifico el **app.css**:

html, body {

font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;

background-image: url('/images/61gpbM.jpg');

background-size: cover;

background-position: center;

background-repeat: no-repeat;

background-attachment: fixed;

}

1. Modifico el **IGroupsRepository**:

Task<ActionResponse<IEnumerable<Group>>> GetAllAsync();

1. Modifico el **IGroupsUnitOfWork**:

Task<ActionResponse<IEnumerable<Group>>> GetAllAsync();

1. Modifico el **GroupsUnitOfWork**:

public async Task<ActionResponse<IEnumerable<Group>>> GetAllAsync() => await \_groupsRepository.GetAllAsync();

1. Modifico el **GroupsController**:

[AllowAnonymous]

[HttpGet("all")]

public async Task<IActionResult> GetAllAsync()

{

var response = await \_groupsUnitOfWork.GetAllAsync();

if (response.WasSuccess)

{

return Ok(response.Result);

}

return BadRequest();

}

1. Modifico el **Home.razor.cs**:

using Fantasy.Frontend.Helpers;

using Fantasy.Frontend.Repositories;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

using MudBlazor;

namespace Fantasy.Frontend.Pages;

public partial class Home

{

private const string baseUrl = "api/groups";

private List<Group>? Groups { get; set; }

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

[Inject] private IRepository Repository { get; set; } = null!;

[Inject] private ISnackbar Snackbar { get; set; } = null!;

[Inject] private IClipboardService ClipboardService { get; set; } = null!;

[Inject] private IStringLocalizer<Parameters> Parameters { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await base.OnInitializedAsync();

await LoadGroupsAsync();

}

private async Task LoadGroupsAsync()

{

var url = $"{baseUrl}/all";

var responseHttp = await Repository.GetAsync<List<Group>>(url);

if (responseHttp.Error)

{

var message = await responseHttp.GetErrorMessageAsync();

Snackbar.Add(Localizer[message], Severity.Error);

return;

}

Groups = responseHttp.Response;

}

private async Task CopyInvitationAsync(Group group)

{

var joinURL = $"{Parameters["URLFront"]}/groups/join/?code={group!.Code}";

await ClipboardService.CopyToClipboardAsync(joinURL);

var text = string.Format(Localizer["InvitationURLCopied"], group!.Name);

Snackbar.Add(text, Severity.Success);

}

}

1. Modifico el **Home.razor**:

@page "/"

<PageTitle>@Localizer["Home"]</PageTitle>

<MudPaper Class="p-4 my-4">

<MudText Typo="Typo.h3" Align="Align.Center">@Localizer["Title"]</MudText>

<MudText Typo="Typo.h5" Align="Align.Center">@Localizer["Subtitle"]</MudText>

</MudPaper>

<MudContainer MaxWidth="MaxWidth.Large">

@if (Groups == null)

{

<Loading />

}

else if (Groups.Count != 0)

{

<MudCarousel TData="Group" Style="height: 620px; display: flex; justify-content: center; align-items: center;">

@foreach (var group in Groups)

{

<MudCarouselItem>

<MudCard Style="height: 100%; display: flex; flex-direction: column; justify-content: space-between; align-items: center;">

<div style="padding: 2rem;">

<MudCardMedia Image="@group.ImageFull" Alt="Group image" Style="width: 250px; height: 250px; object-fit: cover;" />

</div>

<MudCardContent Style="text-align: center;" Class="mt-8">

<MudStack Spacing="2">

<MudText Color="Color.Secondary" Typo="Typo.h3">@group.Name</MudText>

<MudText>@group.Remarks</MudText>

<MudStack Row="true" Spacing="2" AlignItems="AlignItems.Center">

<MudImage Src="@group.Admin.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

<MudText Color="Color.Secondary" Typo="Typo.h5">@group.Admin.FullName</MudText>

<MudText Color="Color.Info" Typo="Typo.h5">@group.Admin.Email</MudText>

<MudTooltip Text="@Localizer["CopyInvitationURLTitle"]">

<MudButton Variant="Variant.Filled"

Color="Color.Secondary"

OnClick="@(() => CopyInvitationAsync(group))"

Disabled="@(!group.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.ContentCopy" />

</MudButton>

</MudTooltip>

</MudStack>

</MudStack>

</MudCardContent>

<MudCardActions Style="width: 100%; justify-content: space-around;">

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

Color="Color.Primary">

@Localizer["Predictions"]

</MudButton>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Stars"

Color="Color.Secondary">

@Localizer["Positions"]

</MudButton>

</MudCardActions>

</MudCard>

</MudCarouselItem>

}

</MudCarousel>

}

</MudContainer>

1. Probamos y hacemos el **commit**.

## Viendo las posiciones como usuario anónimo

1. Modificamos el **GroupsController**:

[AllowAnonymous]

[HttpGet("CheckPredictionsForAllMatches/{id}")]

public async Task<IActionResult> CheckPredictionsForAllMatchesAsync(int id)

…

[AllowAnonymous]

[HttpGet("{id}")]

public override async Task<IActionResult> GetAsync(int id)

1. Modificamos el **PredictionsController**:

[AllowAnonymous]

[HttpGet("positions")]

public async Task<IActionResult> GetPositionsAsync([FromQuery] PaginationDTO pagination)

…

[AllowAnonymous]

[HttpGet("totalRecordsForPositionsPaginated")]

public async Task<IActionResult> GetTotalRecordsForPositionsAsync([FromQuery] PaginationDTO pagination)

1. Modificamos el **GroupDetails.razor.cs**:

[Parameter] public bool IsAnonymouns { get; set; }

1. Modificamos el **GroupDetails.razor**:

<MudTabs>

@if (!IsAnonymouns)

{

<MudTabPanel Text="@Localizer["Predictions"]">

<MudContainer MaxWidth="MaxWidth.Large">

<Predictions GroupId="GroupId" />

</MudContainer>

</MudTabPanel>

}

<MudTabPanel Text="@Localizer["Positions"]">

<MudContainer MaxWidth="MaxWidth.Large">

<Positions GroupId="GroupId" IsAnonymouns="@IsAnonymouns" />

</MudContainer>

</MudTabPanel>

</MudTabs>

1. Modificamos el **GroupsIndex.razor.cs**:

private void GroupDetails(Group group)

{

NavigationManager.NavigateTo($"/groups/details/{group.Id}/false");

}

1. Modificamos el **Positions.razor.cs**:

[Parameter] public int GroupId { get; set; }

[Parameter] public bool IsAnonymouns { get; set; }

…

private void ReturnAction()

{

if (IsAnonymouns)

{

NavigationManager.NavigateTo("/");

}

else

{

NavigationManager.NavigateTo("/groups");

}

}

1. Modificamos el **Positions.razor**:

<HeaderContent>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["User"]</MudTh>

<MudTh>@Localizer["Points"]</MudTh>

@if (!IsAnonymouns)

{

<MudTh style="width: 170px;">@Localizer["Actions"]</MudTh>

}

</HeaderContent>

<RowTemplate>

<MudTd>

<MudImage Src="@context.User.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.User.FullName</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h5" Align="Align.Center">@context.Points</MudText>

</MudTd>

@if (!IsAnonymouns)

{

<MudTd>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.Visibility"

Color="Color.Info"

OnClick=@(() => WatchBalanceAsync(@context))>

@Localizer["PredictionsBalance"]

</MudButton>

</MudTd>

}

</RowTemplate>

1. Modificamos el **Home.razor.cs**:

[Inject] private IClipboardService ClipboardService { get; set; } = null!;

[Inject] private IStringLocalizer<Parameters> Parameters { get; set; } = null!;

[Inject] private IDialogService DialogService { get; set; } = null!;

…

private async Task CopyInvitationAsync(Group group)

{

var joinURL = $"{Parameters["URLFront"]}/groups/join/?code={group!.Code}";

await ClipboardService.CopyToClipboardAsync(joinURL);

var text = string.Format(Localizer["InvitationURLCopied"], group!.Name);

Snackbar.Add(text, Severity.Success);

}

private async Task GroupDetailsAsync(Group group)

{

{

var options = new DialogOptions()

{

CloseOnEscapeKey = true,

CloseButton = true,

MaxWidth = MaxWidth.Medium,

FullWidth = true

};

var parameters = new DialogParameters

{

{ "GroupId", group.Id },

{ "IsAnonymouns", true }

};

var dialog = DialogService.Show<GroupDetails>(@Localizer["GroupDetails"], parameters, options);

await dialog.Result;

}

}

1. Modificamos el **Home.razor**:

@page "/"

<PageTitle>@Localizer["Home"]</PageTitle>

<MudPaper Class="p-4 my-4">

<MudText Typo="Typo.h3" Align="Align.Center">@Localizer["Title"]</MudText>

<MudText Typo="Typo.h5" Align="Align.Center">@Localizer["Subtitle"]</MudText>

</MudPaper>

<MudContainer MaxWidth="MaxWidth.Large">

@if (Groups == null)

{

<Loading />

}

else if (Groups.Count != 0)

{

<MudCarousel TData="Group" Style="height: 620px; display: flex; justify-content: center; align-items: center;">

@foreach (var group in Groups)

{

<MudCarouselItem>

<MudCard Style="height: 100%; display: flex; flex-direction: column; justify-content: space-between; align-items: center;">

<div style="padding: 2rem;">

<MudCardMedia Image="@group.ImageFull" Alt="Group image" Style="width: 250px; height: 250px; object-fit: cover;" />

</div>

<MudCardContent Style="text-align: center;" Class="mt-8">

<MudStack Spacing="2">

<MudText Color="Color.Secondary" Typo="Typo.h3">@group.Name</MudText>

<MudText>@group.Remarks</MudText>

<MudStack Row="true" Spacing="2" AlignItems="AlignItems.Center">

<MudImage Src="@group.Admin.PhotoFull" Width="80" Height="80" Style="border-radius: 50%;" />

<MudText Color="Color.Secondary" Typo="Typo.h5">@group.Admin.FullName</MudText>

<MudText Color="Color.Info" Typo="Typo.h5">@group.Admin.Email</MudText>

<MudTooltip Text="@Localizer["CopyInvitationURLTitle"]">

<MudButton Variant="Variant.Filled"

Color="Color.Secondary"

OnClick="@(() => CopyInvitationAsync(group))"

Disabled="@(!group.IsActive)">

<MudIcon Icon="@Icons.Material.Filled.ContentCopy" />

</MudButton>

</MudTooltip>

</MudStack>

</MudStack>

</MudCardContent>

<MudCardActions>

<MudButton Variant="Variant.Filled"

EndIcon="@Icons.Material.Filled.SportsSoccer"

OnClick="@(() => GroupDetailsAsync(group))"

Color="Color.Primary"

Class="m-2">

@Localizer["GroupDetails"]

</MudButton>

</MudCardActions>

</MudCard>

</MudCarouselItem>

}

</MudCarousel>

}

</MudContainer>

1. Probamos y hacemos el **commit**.

## Creando el acerca de

1. Adicionamos los siguientes literales:

| Author | Author | Autor |
| --- | --- | --- |
| AboutText | <p>A system where different groups of friends can make predictions about football tournaments. In Colombia, it’s called “Polla”; in Argentina, “Prode”; and in the United States, “Fantasy.” The idea is that any number of football tournaments, such as the Copa América, the World Cup, the Euro Cup, the Champions League, or the Colombian League, among others, can be registered. Groups of friends will be able to form their own “Pollas” and make predictions about the matches. Once the matches are completed and business rules are applied, the participant who accumulates the most points will win the “Fantasy,” the “Polla,” or whatever it’s called in their country.</p>    <p>Each user can create multiple groups or join existing groups to participate in any football tournament enabled by the administrator. The group creator will be considered the group administrator and will be able to define the conditions for distributing the prize, for example:</p>  <ul>  <li>70% for first place.</li>  <li>20% for second place.</li>  <li>10% for third place.</li>  </ul>  <br/>  <p>The administrator will also have the ability to activate or deactivate members of their group. For example, if a member has not paid the corresponding amount for the “Polla,” the administrator can deactivate them, and an inactive user will not be able to enter predictions.</p>  <p>Points are awarded as follows:</p>  <ul>  <li>5 points for guess the winner or a draw.</li>  <li>2 points for guess the goals of the home team.</li>  <li>2 points for guess the goals of the away team.</li>  <li>1 point for guess the goal difference.</li>  </ul>  <br/>  <p>The maximum points per match is 10, in the case of a perfect prediction. Please consider the following rules:</p>  <ul>  <li>Predictions can only be entered or modified up to 10 minutes before the match starts.</li>  <li>The result is based on the 90 minutes of regulation time plus any added time. Goals scored during extra time or penalties are not counted.</li>  <li>Matches from the second round onwards will award double points.</li>  </ul> | <p>Sistema donde diferentes grupos de amigos pueden hacer predicciones sobre torneos de fútbol. En Colombia, se le llama “Polla”; en Argentina, “Prode”; y en Estados Unidos, “Fantasy”. La idea es que cualquier número de torneos de fútbol, como la Copa América, el Mundial, la Eurocopa, la Champions League, o el Torneo Colombiano, entre otros, pueda ser registrado. Los grupos de amigos podrán formar sus propias “Pollas” y realizar predicciones sobre los partidos. Una vez completados los partidos y aplicadas las reglas de negocio, el participante que acumule más puntos ganará la “Fantasy”, la “Polla” o como se le denomine en su país.</p>    <p>Cada usuario podrá crear múltiples grupos o unirse a grupos existentes para participar en cualquier torneo de fútbol habilitado por el administrador. El creador del grupo será considerado el administrador de dicho grupo y podrá definir las condiciones para repartir el premio, por ejemplo:</p>  <ul>  <li>70% para el primer puesto.</li>  <li>20% para el segundo puesto.</li>  <li>10% para el tercer puesto.</li>  </ul>  <br/>  <p>El administrador también tendrá la facultad de activar o desactivar a los miembros de su grupo. Por ejemplo, si un miembro no ha pagado el valor correspondiente a la polla, el administrador podrá desactivarlo, y un usuario inactivo no podrá ingresar predicciones.</p>  <p>La forma de obtener puntos es la siguiente:</p>  <ul>  <li>5 puntos por acertar el ganador o acertar un empate.</li>  <li>2 puntos por acertar los goles del equipo local.</li>  <li>2 puntos por acertar los goles del equipo visitante.</li>  <li>1 punto por acertar la diferencia de goles.</li>  </ul>  <br/>  <p>El máximo de puntos por partido será 10, en caso de acertar el resultado perfecto. Ten en cuenta las siguientes consideraciones:</p>  <ul>  <li>Solo se podrán ingresar o modificar predicciones hasta 10 minutos antes de iniciar un partido.</li>  <li>El resultado se basará en los 90 minutos de tiempo reglamentario más las adiciones. No se tendrán en cuenta los goles en tiempos extra o penales.</li>  <li>Los partidos de segunda ronda en adelante otorgarán el doble de puntos.</li>  </ul> |

1. Adicionamos a **wwwroot/images** la imagén de logo de la aplicación y la imagen o imágenes de los autores de la App, para mi caso usaré **JuanZuluaga.jpg** y **Logo.png**.
2. Creamos en **Pages** el **About.razor.cs**:

using Fantasy.Shared.Resources;

using Microsoft.AspNetCore.Components;

using Microsoft.Extensions.Localization;

namespace Fantasy.Frontend.Pages;

public partial class About

{

[Inject] private IStringLocalizer<Literals> Localizer { get; set; } = null!;

}

1. Modificamos el **About.razor.cs**:

@page "/about"

<MudStack AlignItems="AlignItems.Center" JustifyContent="JustifyContent.Center" Spacing="3">

<MudCard Class="p-4">

<MudStack AlignItems="AlignItems.Center" JustifyContent="JustifyContent.Center">

<MudText Typo="Typo.h2" Class="me-4" Color="Color.Primary">@Localizer["Title"]</MudText>

<MudImage Src="images/Logo.png" Width="200" Class="p-3" />

<MudText Typo="Typo.input">

@((MarkupString)Localizer["AboutText"].ToString())

</MudText>

<MudText Typo="Typo.h3" Color="Color.Secondary" Class="my-3">@Localizer["Author"]</MudText>

</MudStack>

<MudStack AlignItems="AlignItems.Center" JustifyContent="JustifyContent.Center" Spacing="2">

<MudCard Class="card">

<MudImage Src="images/JuanZuluaga.jpg" Width="250" Height="250" />

<MudText Typo="Typo.h5" Class="centered-text p-2">Juan Zuluaga</MudText>

</MudCard>

</MudStack>

</MudCard>

</MudStack>

1. Modificamos el **NavMenu.razor**:

<MudNavMenu>

<MudStack AlignItems="AlignItems.Center">

<MudImage Src="images/Logo.png" Width="200" Class="p-3" />

</MudStack>

<MudDivider />

<MudNavLink Href="/" Match="NavLinkMatch.All" Icon="@Icons.Material.Rounded.Home">@Localizer["Home"]</MudNavLink>

<MudDivider />

1. Probamos y hacemos el **commit**.

## Regla de negocio, “partidos de segunda ronda tendrán doble puntaje”

1. Adicionamos los siguientes literales:

| DoublePoints | Double Points | Puntaje Doble |
| --- | --- | --- |
| DoublePointsMatchMessage | Double Points Match | Partido de Doble Puntaje |
| SinglePointsMatchMessage | Single Points Match | Partido de Puntaje Sencillo |

1. Modificamos la entidad **Match**:

[Display(Name = "DoublePoints", ResourceType = typeof(Literals))]

public bool DoublePoints { get; set; }

1. Modificamos el **MatchDTO**:

[Display(Name = "DoublePoints", ResourceType = typeof(Literals))]

public bool DoublePoints { get; set; }

1. Creamos la migración y la aplicamos.
2. Modificamos el **MatchesRepository**:

En Add…

var match = new Match

{

IsActive = matchDTO.IsActive,

Date = matchDTO.Date,

Tournament = tournament,

Local = local,

Visitor = visitor,

DoublePoints = matchDTO.DoublePoints,

};

En Update…

currentMatch.Local = local;

currentMatch.Visitor = visitor;

currentMatch.GoalsVisitor = matchDTO.GoalsVisitor;

currentMatch.GoalsLocal = matchDTO.GoalsLocal;

currentMatch.Date = matchDTO.Date;

currentMatch.IsActive = matchDTO.IsActive;

currentMatch.DoublePoints = matchDTO.DoublePoints;

En CalculatePoints…

if (Math.Abs((decimal)match.GoalsLocal! - (decimal)match.GoalsVisitor!) == Math.Abs((decimal)prediction.GoalsLocal! - (decimal)prediction.GoalsVisitor!)) points++;

if (match.DoublePoints) points \*= 2;

return points;

1. Modificamos el **MatchForm.razor.cs**:

private string? doublePointsMessage;

…

protected override async Task OnParametersSetAsync()

{

base.OnParametersSet();

await LoadMatchesAsync();

isActiveMessage = MatchDTO.IsActive ? Localizer["MatchActive"] : Localizer["MatchInactive"];

doublePointsMessage = MatchDTO.DoublePoints ? Localizer["DoublePointsMatchMessage"] : Localizer["SinglePointsMatchMessage"];

if (MatchDTO.Id != 0)

{

LoadInitialValues();

}

else

{

MatchDTO.Date = DateTime.Now;

}

}

private void SetDoublePointsOff()

{

MatchDTO.DoublePoints = false;

doublePointsMessage = Localizer["SinglePointsMatchMessage"];

}

private void SetDoublePointsOn()

{

MatchDTO.DoublePoints = true;

doublePointsMessage = Localizer["DoublePointsMatchMessage"];

}

1. Modificamos el **MatchForm.razor**:

<MudGrid Justify="Justify.SpaceBetween" Class="mb-2">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@doublePointsMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (MatchDTO.DoublePoints)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="SetDoublePointsOff">

@Localizer["SinglePointsMatchMessage"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="SetDoublePointsOn">

@Localizer["DoublePointsMatchMessage"]

</MudButton>

}

</MudItem>

</MudGrid>

<MudGrid Justify="Justify.SpaceBetween" Class="mb-2">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@isActiveMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (MatchDTO.IsActive)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Cancel"

Color="Color.Error"

OnClick="SetTournamentOff">

@Localizer["Deactivate"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.CheckCircle"

Color="Color.Success"

OnClick="SetTournamentOn">

@Localizer["Activate"]

</MudButton>

}

</MudItem>

</MudGrid>

1. Modificamos el **EditMatch.razor.cs**:

matchDTO = new MatchDTO()

{

Id = match!.Id,

IsActive = match!.IsActive,

Date = match!.Date,

GoalsLocal = match!.GoalsLocal,

GoalsVisitor = match!.GoalsVisitor,

LocalId = match!.LocalId,

TournamentId = match!.TournamentId,

VisitorId = match!.VisitorId,

DoublePoints = match!.DoublePoints,

};

1. Modificamos el **CloseMatch.razor.cs**:

matchDTO = new MatchDTO()

{

GoalsLocal = match!.GoalsLocal,

GoalsVisitor = match!.GoalsVisitor,

Id = match!.Id,

TournamentId = match!.TournamentId,

Date = match!.Date,

IsActive = match!.IsActive,

LocalId = match!.LocalId,

VisitorId = match!.VisitorId,

DoublePoints = match!.DoublePoints,

};

1. Modificamos el **TournamentMatches.razor.cs**:

<MudTh>@Localizer["Visitor"]</MudTh>

<MudTh>@Localizer["DoublePoints"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

…

<MudTd>@context.Visitor.Name</MudTd>

<MudTd>

@if (context.DoublePoints)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>

1. Probamos y hacemos el **commit**.

## Mejora para que se vean mejor las imágenes de selecciones y equipos profesionales

1. Adicionamos los siguientes literales:

| IsImageSquare | Is Image Square? | ¿La imagen es cuadrada? |
| --- | --- | --- |
| ImageIsSquare | The image is square | La imagén es cuadrada |
| ImageIsRectangular | The image is rectangular | La imagén es rectangular |
| Square | Square | Cuadrada |
| Rectangular | Rectangular | Rectangular |

1. Agregamos esta propiedad a la entidad **Team**:

[Display(Name = "IsImageSquare", ResourceType = typeof(Literals))]

public bool IsImageSquare { get; set; }

1. Modificamos el **TeamDTO**:

[Display(Name = "IsImageSquare", ResourceType = typeof(Literals))]

public bool IsImageSquare { get; set; }

1. Modificamos el **TeamRepository**:

En el Add:

var team = new Team

{

Country = country,

Name = teamDTO.Name,

IsImageSquare = teamDTO.IsImageSquare,

};

En el Update:

currentTeam.Country = country;

currentTeam.Name = teamDTO.Name;

currentTeam.IsImageSquare = teamDTO.IsImageSquare;

1. Modificamos el **Predictions.razor**:

<MudTd style="text-align:center; vertical-align:middle;">

@if (context.Match.Local.IsImageSquare)

{

<MudImage Src="@context.Match.Local.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.Match.Local.ImageFull" Width="90" Height="60" />

}

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsLocal</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsVisitor</MudText>

</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

@if (context.Match.Visitor.IsImageSquare)

{

<MudImage Src="@context.Match.Visitor.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.Match.Visitor.ImageFull" Width="90" Height="60" />

}

</MudTd>

1. Modificamos el **TeamEdit.razor.cs**:

teamDTO = new TeamDTO()

{

Id = team!.Id,

Name = team!.Name,

Image = team.Image,

CountryId = team.CountryId,

IsImageSquare = team.IsImageSquare

};

1. Modificamos el **TeamForm.razor.cs**:

private string? shapeImageMessage;

…

protected override void OnParametersSet()

{

base.OnParametersSet();

if (!string.IsNullOrEmpty(TeamDTO.Image))

{

imageUrl = TeamDTO.Image;

TeamDTO.Image = null;

}

shapeImageMessage = TeamDTO.IsImageSquare ? Localizer["ImageIsSquare"] : Localizer["ImageIsRectangular"];

}

…

private void SetImageSquare()

{

TeamDTO.IsImageSquare = true;

shapeImageMessage = Localizer["ImageIsSquare"];

}

private void SetImageRectangular()

{

TeamDTO.IsImageSquare = false;

shapeImageMessage = Localizer["ImageIsRectangular"];

}

1. Modificamos el **TeamForm.razor**:

<MudAutocomplete T="Country"

Label=@Localizer["Country"]

Placeholder=@Localizer["SelectACountry"]

SearchFunc="SearchCountry"

Value="selectedCountry"

ValueChanged="CountryChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<MudGrid Justify="Justify.SpaceBetween" Class="my-2">

<MudItem xs="6">

<MudText Typo="Typo.input" Align="Align.Left">@shapeImageMessage</MudText>

</MudItem>

<MudItem xs="6" class="d-flex justify-content-end">

@if (TeamDTO.IsImageSquare)

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Square"

Color="Color.Primary"

OnClick="SetImageRectangular">

@Localizer["Rectangular"]

</MudButton>

}

else

{

<MudButton Variant="Variant.Filled"

StartIcon="@Icons.Material.Filled.Rectangle"

Color="Color.Secondary"

OnClick="SetImageSquare">

@Localizer["Square"]

</MudButton>

}

</MudItem>

</MudGrid>

<div class="my-2">

<InputImg Label=@Localizer["Image"] ImageSelected="ImageSelected" ImageURL="@imageUrl" />

</div>

1. Modificamos el **TeamsIndex.razor**:

<HeaderContent>

<MudTh>@Localizer["Team"]</MudTh>

<MudTh>@Localizer["Image"]</MudTh>

<MudTh>@Localizer["IsImageSquare"]</MudTh>

<MudTh>@Localizer["Country"]</MudTh>

<MudTh>@Localizer["Actions"]</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd>@context.Name</MudTd>

<MudTd>

@if (context.IsImageSquare)

{

<MudImage Src="@context.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.ImageFull" Width="90" Height="60" />

}

</MudTd>

<MudTd>

@if (context.IsImageSquare)

{

<MudIcon Icon="@Icons.Material.Filled.CheckCircle" Color="Color.Success" />

}

else

{

<MudIcon Icon="@Icons.Material.Filled.Cancel" Color="Color.Error" />

}

</MudTd>

<MudTd>@context.Country.Name</MudTd>

1. Modificamos el **AddTeamForm.razor**:

<MudAutocomplete T="Team"

Label=@Localizer["Team"]

Placeholder=@Localizer["SelectATeam"]

SearchFunc="SearchTeam"

Value="selectedTeam"

ValueChanged="TeamChanged"

ToStringFunc="@(e=> e==null?null : $"{e.Name}")"

Class="mb-2">

<ItemTemplate Context="itemContext">

@itemContext.Name

</ItemTemplate>

</MudAutocomplete>

<div class="my-2">

@if (selectedTeam.Id != 0)

{

@if (selectedTeam.IsImageSquare)

{

<MudImage Src="@imageUrl" Width="120" Height="120" />

}

else

{

<MudImage Src="@imageUrl" Width="120" Height="80" />

}

}

</div>

<MudButton Variant="Variant.Outlined"

StartIcon="@Icons.Material.Filled.ArrowBack"

Color="Color.Info"

OnClick="ReturnAction">

@Localizer["Return"]

</MudButton>

1. Modificamos el **MatchForm.razor**:

</MudGrid>

<div style="display: flex; align-items: center; justify-content: center; margin-top: 30px; margin-bottom: 30px;">

<div class="mb-2" style="margin-right: 10px;">

@if(selectedLocal.Id != 0)

{

@if (selectedLocal.IsImageSquare)

{

<MudImage Src="@imageUrlLocal" Width="120" Height="120" />

}

else

{

<MudImage Src="@imageUrlLocal" Width="120" Height="80" />

}

}

</div>

<MudText Typo="Typo.h3" Align="Align.Center" Class="mx-2">Vs</MudText>

<div class="mb-2" style="margin-left: 10px;">

@if (selectedVisitor.Id != 0)

{

@if (selectedVisitor.IsImageSquare)

{

<MudImage Src="@imageUrlVisitor" Width="120" Height="120" />

}

else

{

<MudImage Src="@imageUrlVisitor" Width="120" Height="80" />

}

}

</div>

</div>

<MudButton Variant="Variant.Outlined"

1. Modificamos el **TournamentMatches.razor**:

<MudTd style="text-align:center; vertical-align:middle;">

@if (context.Local.IsImageSquare)

{

<MudImage Src="@context.Local.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.Local.ImageFull" Width="90" Height="60" />

}

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsLocal</MudText>

</MudTd>

<MudTd>

<MudText Typo="Typo.h3" Align="Align.Center">@context.GoalsVisitor</MudText>

</MudTd>

<MudTd style="text-align:center; vertical-align:middle;">

@if (context.Visitor.IsImageSquare)

{

<MudImage Src="@context.Visitor.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.Visitor.ImageFull" Width="90" Height="60" />

}

</MudTd>

1. Modificamos el **TournamentTeams.razor**:

<MudTd>@context.Team.Name</MudTd>

<MudTd>

@if (context.Team.IsImageSquare)

{

<MudImage Src="@context.Team.ImageFull" Width="60" Height="60" />

}

else

{

<MudImage Src="@context.Team.ImageFull" Width="90" Height="60" />

}

</MudTd>

1. Probamos y hacemos el **commit**.

## Cambiando el idioma a gusto del usuario

1. Adicionamos los siguientes literales:

| Spanish | Spanish | Español |
| --- | --- | --- |
| English | English | Inglés |

1. Creamos el **LocalStorageService**:

using Microsoft.JSInterop;

namespace Fantasy.Frontend.Helpers;

public class LocalStorageService

{

private readonly IJSRuntime \_jsRuntime;

public LocalStorageService(IJSRuntime jsRuntime)

{

\_jsRuntime = jsRuntime;

}

// Save an item in the browser's localStorage

public async Task SetItemAsync(string key, string value)

{

await \_jsRuntime.InvokeVoidAsync("localStorage.setItem", key, value);

}

// Retrieve an item from the browser's localStorage

public async Task<string> GetItemAsync(string key)

{

return await \_jsRuntime.InvokeAsync<string>("localStorage.getItem", key);

}

}

1. Creamos el **LanguageService**:

using Fantasy.Frontend.Helpers;

using Fantasy.Shared.Resources;

using Microsoft.Extensions.Localization;

using System.Globalization;

public class LanguageService

{

private readonly IStringLocalizer<Literals> \_localizer;

private readonly LocalStorageService \_localStorageService;

private const string LanguageKey = "preferredLanguage";

public string CurrentLanguage { get; private set; }

public LanguageService(IStringLocalizer<Literals> localizer, LocalStorageService localStorageService)

{

\_localizer = localizer;

\_localStorageService = localStorageService;

CurrentLanguage = CultureInfo.CurrentCulture.TwoLetterISOLanguageName;

}

public async Task InitializeLanguageAsync()

{

var savedLanguage = await \_localStorageService.GetItemAsync(LanguageKey);

if (!string.IsNullOrEmpty(savedLanguage))

{

SetLanguage(savedLanguage);

}

}

public async void SetLanguage(string languageCode)

{

var culture = new CultureInfo(languageCode);

CultureInfo.DefaultThreadCurrentCulture = culture;

CultureInfo.DefaultThreadCurrentUICulture = culture;

CurrentLanguage = languageCode;

await \_localStorageService.SetItemAsync(LanguageKey, languageCode);

}

}

1. Modificamos el **Program**:

builder.Services.AddScoped<AuthenticationProviderJWT>();

builder.Services.AddScoped<AuthenticationStateProvider, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

builder.Services.AddScoped<ILoginService, AuthenticationProviderJWT>(x => x.GetRequiredService<AuthenticationProviderJWT>());

builder.Services.AddScoped<IClipboardService, ClipboardService>();

// Register language service and localStorage service for managing language preferences

builder.Services.AddScoped<LanguageService>();

builder.Services.AddScoped<LocalStorageService>();

// Build the application

var host = builder.Build();

// Retrieve the language service to set the initial language based on user preferences or browser language

var languageService = host.Services.GetRequiredService<LanguageService>();

// Initialize the language preference (from localStorage or browser)

await languageService.InitializeLanguageAsync(); // This will set the initial culture based on local storage or browser

// Run the application

await host.RunAsync();

1. Modificamos el **Fantasy.Frontend**:

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

<BlazorWebAssemblyLoadAllGlobalizationData>true</BlazorWebAssemblyLoadAllGlobalizationData>

</PropertyGroup>

1. Modificamos el **App.razor.cs**:

[Inject] private LanguageService LanguageService { get; set; } = null!;

protected override async Task OnInitializedAsync()

{

await LanguageService.InitializeLanguageAsync();

}

1. Copiar las banderas de **España** y **Reino Unido** en **wwroot/images**.
2. Modificar el **Home.razor.cs**:

[Inject] private LanguageService LanguageService { get; set; } = null!;

[Inject] private NavigationManager NavigationManager { get; set; } = null!;

private string selectedLanguage = "es"; // Default to Spanish

…

protected override async Task OnInitializedAsync()

{

await base.OnInitializedAsync();

await LoadGroupsAsync();

selectedLanguage = LanguageService.CurrentLanguage;

}

private void ChangeLanguage(string language)

{

LanguageService.SetLanguage(language);

NavigationManager.NavigateTo(NavigationManager.Uri, forceLoad: true);

}

1. Modificar el **Home.razor**:

<MudPaper Class="p-4 my-4">

<MudStack Row Justify="Justify.SpaceBetween">

<MudStack>

<MudText Typo="Typo.h3">@Localizer["Title"]</MudText>

<MudText Typo="Typo.h5">@Localizer["Subtitle"]</MudText>

</MudStack>

<MudStack Row Justify="Justify.FlexEnd">

<MudTooltip Text="@Localizer["Spanish"]">

<MudButton OnClick="@(() => ChangeLanguage("es"))">

<MudImage Src="/images/Spain.png" Width="50" Height="50" Style="border-radius: 50%;" />

</MudButton>

</MudTooltip>

<MudTooltip Text="@Localizer["English"]">

<MudButton OnClick="@(() => ChangeLanguage("en"))">

<MudImage Src="/images/United Kingdom.png" Width="50" Height="50" Style="border-radius: 50%;" />

</MudButton>

</MudTooltip>

</MudStack>

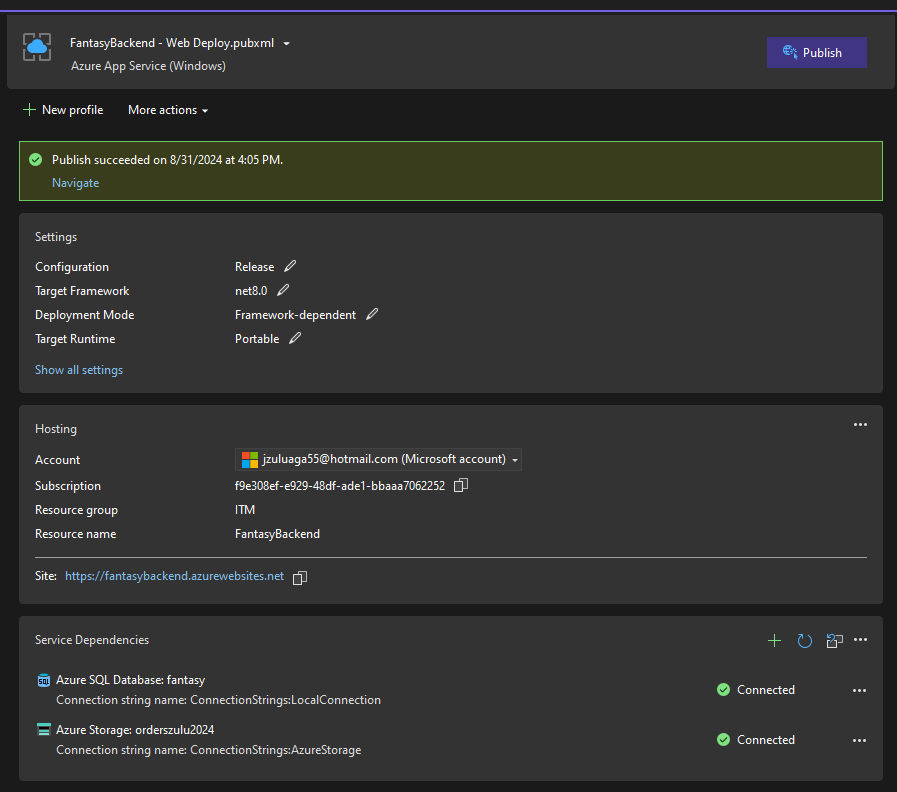
</MudStack>

</MudPaper>

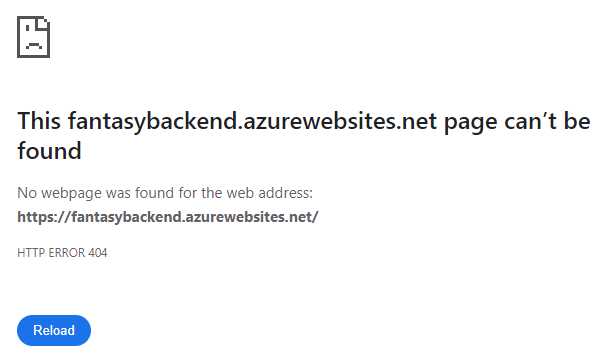
1. Probamos y hacemos el **commit**.

# Publicando en Azure

1. Entramos a portal Azure y cremos una nueva base de datos SQL Server vacía.
2. Nos aseguramos que tenemos acceso a esa base de datos desde el SQL Management Studio, posiblemente te toque agregar la dirección IP púbica de tu máquina local.
3. Copiamos el string de conexión y ponemos a correr nuestro backend local contra la base de datos de Azure, de esta manera asegurarno que se corran bien las migraciones y el alimentador de la base de datos.
4. Retornamos nuestros string de conexión a la base de datos local y comentamos todos los string de conexión que no estemos usando activamente.
5. Publicar el backend en Azure, ver video para poder configurar todos los pasos correctamente:



1. Si todo estuvo bien te debe salir una pantalla similar a esta:



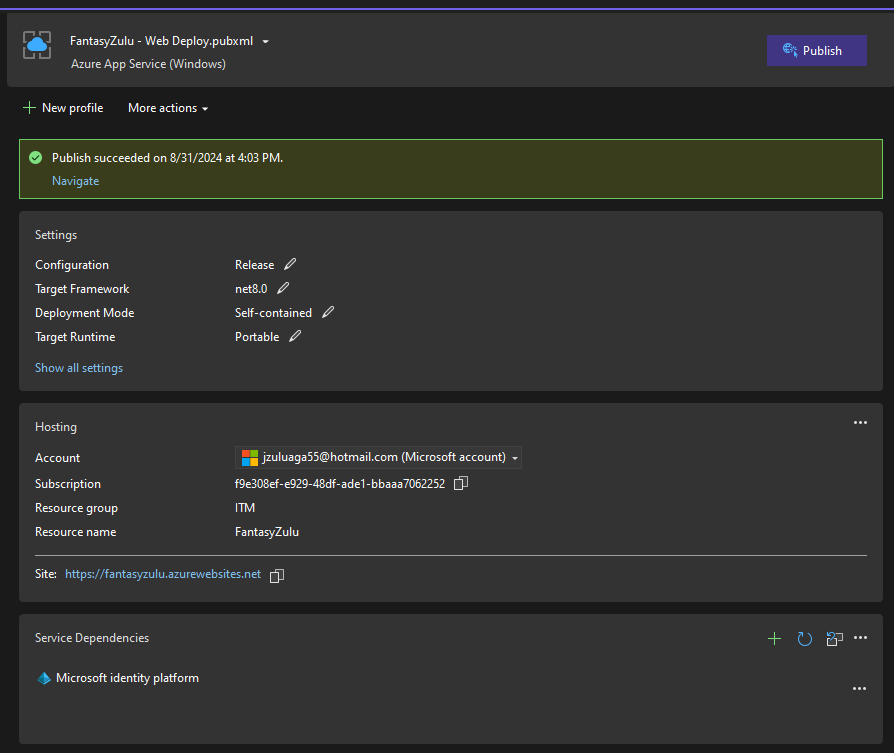
1. Tome la dirección de publicación del Backend (según mi ejemplo es: <https://fantasybackend.azurewebsites.net>) y modifique el **Program** del Frontend. **Nota**: reemplace las URL por las suyas:

var uriBack = "https://fantasybackend.azurewebsites.net";

//var uriBack = https://localhost:7232;

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri(uriBack) });

1. Publicar el frontend en Azure, ver video para poder configurar todos los pasos correctamente:



1. Tome la dirección de publicación del Frontend (según mi ejemplo es: <https://fantasyzulu.azurewebsites.net>) y modifique el **appsettings** del Backend. **Nota**: reemplace las URL por las suyas:

"Url Frontend": "fantasyzulu.azurewebsites.net",

//"Url Frontend": "localhost:7069",

1. Cambie el parámetro en el archivo de recursos **Parameters**:

| URLFront | https://fantasyzulu.azurewebsites.net |
| --- | --- |

1. Publique de nuevo el **backend** y luego el **frontend**.

# Creando pruebas unitarias

## Generales

1. Agreguele estos paquetes al nuevo proyecto **Fantasy.Test**:

**Microsoft.EntityFrameworkCore.InMemory**

**Moq**

1. Y actualizamos los paquetes del proyecto.
2. Instalamos las extensiones **Fine Code Coverage** y **Run Coverlet Report VS2022**. Para poder medir la cobertura de nuestras pruebas unitarias.

## Paises

### Controlador

1. Cree la carpeta **Controllers** y dentro de este adicione la clase **CountriesControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Moq;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class CountriesControllerTests

{

private Mock<ICountriesUnitOfWork> \_mockCountriesUnitOfWork = null!;

private CountriesController \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockCountriesUnitOfWork = new Mock<ICountriesUnitOfWork>();

\_controller = new CountriesController(null!, \_mockCountriesUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsOkResult\_WithListOfCountries()

{

// Arrange

var mockData = new List<Country>

{

new() { Id = 1, Name = "Country 1" },

new() { Id = 2, Name = "Country 2" }

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetComboAsync()).ReturnsAsync(mockData);

// Act

var result = await \_controller.GetComboAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(List<Country>));

Assert.AreEqual(2, ((List<Country>)okResult.Value).Count);

}

[TestMethod]

public async Task GetAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = [new() { Id = 1, Name = "Country 1" }]

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<Country>));

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<Country>> { WasSuccess = false };

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = new List<Country> { new Country { Id = 1, Name = "Country 1" } }

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(List<Country>));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<Country>> { WasSuccess = false };

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 10

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetTotalRecordsAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(10, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<int> { WasSuccess = false };

\_mockCountriesUnitOfWork.Setup(uow => uow.GetTotalRecordsAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<Country>

{

WasSuccess = true,

Result = new Country { Id = 1, Name = "Country 1" }

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync(1)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(1);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(Country));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsNotFound\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<Country>

{

WasSuccess = false,

Message = "Country not found"

};

\_mockCountriesUnitOfWork.Setup(uow => uow.GetAsync(1)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(1);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual("Country not found", notFoundResult.Value);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Creamos la carpeta **UnitsOfWork** y dentro de esta adicione la clase **CountriesUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class CountriesUnitOfWorkTests

{

private Mock<ICountriesRepository> \_mockCountriesRepository = null!;

private CountriesUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockCountriesRepository = new Mock<ICountriesRepository>();

\_unitOfWork = new CountriesUnitOfWork(null!, \_mockCountriesRepository.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsActionResponse\_WithListOfCountries()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = new List<Country> { new Country { Id = 1, Name = "Country 1" } }

};

\_mockCountriesRepository.Setup(repo => repo.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsInstanceOfType(result.Result, typeof(IEnumerable<Country>));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsActionResponse\_WithListOfCountries()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<Country>>

{

WasSuccess = true,

Result = new List<Country> { new Country { Id = 1, Name = "Country 1" } }

};

\_mockCountriesRepository.Setup(repo => repo.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsInstanceOfType(result.Result, typeof(IEnumerable<Country>));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsActionResponse\_WithTotalRecords()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 10

};

\_mockCountriesRepository.Setup(repo => repo.GetTotalRecordsAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(10, result.Result);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsActionResponse\_WithCountry()

{

// Arrange

var mockResponse = new ActionResponse<Country>

{

WasSuccess = true,

Result = new Country { Id = 1, Name = "Country 1" }

};

\_mockCountriesRepository.Setup(repo => repo.GetAsync(1)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsInstanceOfType(result.Result, typeof(Country));

}

[TestMethod]

public async Task GetComboAsync\_ReturnsListOfCountries()

{

// Arrange

var mockData = new List<Country>

{

new Country { Id = 1, Name = "Country 1" },

new Country { Id = 2, Name = "Country 2" }

};

\_mockCountriesRepository.Setup(repo => repo.GetComboAsync()).ReturnsAsync(mockData);

// Act

var result = await \_unitOfWork.GetComboAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(IEnumerable<Country>));

Assert.AreEqual(2, ((List<Country>)result).Count);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Cree la carpeta **Repositories** y dentro de esta adicione la clase **CountriesRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class CountriesRepositoryTests

{

private DataContext \_context = null!;

private CountriesRepository \_repository = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

\_context = new DataContext(options);

\_repository = new CountriesRepository(\_context);

// Seed the in-memory database

\_context.Countries.AddRange(new List<Country>

{

new Country { Id = 1, Name = "Country B", Teams = [], Users = [] },

new Country { Id = 2, Name = "Country A", Teams = [], Users = [] }

});

\_context.SaveChanges();

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_ReturnsCountriesOrderedByName()

{

// Act

var result = await \_repository.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

Assert.AreEqual("Country A", result.Result!.First().Name);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsPaginatedCountries()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 1 };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

Assert.AreEqual("Country A", result.Result!.First().Name);

}

[TestMethod]

public async Task GetAsync\_WithPaginationAndFilter\_ReturnsFilteredCountries()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 1, Filter = "B" };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

Assert.AreEqual("Country B", result.Result!.First().Name);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecordCount()

{

// Arrange

var pagination = new PaginationDTO();

// Act

var result = await \_repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_WithFilter\_ReturnsFilteredRecordCount()

{

// Arrange

var pagination = new PaginationDTO { Filter = "A" };

// Act

var result = await \_repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsCountry\_WhenFound()

{

// Act

var result = await \_repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Country B", result.Result!.Name);

Assert.AreEqual(0, result.Result!.TeamsCount);

Assert.AreEqual(0, result.Result!.UsersCount);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsNotFound\_WhenCountryNotFound()

{

// Act

var result = await \_repository.GetAsync(999); // ID that doesn't exist

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsCountriesOrderedByName()

{

// Act

var result = await \_repository.GetComboAsync();

// Assert

Assert.AreEqual(2, result.Count());

Assert.AreEqual("Country A", result.First().Name);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Genérico

### Controlador

1. Adicione la clase **GenericControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Moq;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class GenericControllerTests

{

private Mock<IGenericUnitOfWork<SampleEntity>> \_mockUnitOfWork = null!;

private GenericController<SampleEntity> \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockUnitOfWork = new Mock<IGenericUnitOfWork<SampleEntity>>();

\_controller = new GenericController<SampleEntity>(\_mockUnitOfWork.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>>

{

WasSuccess = true,

Result = new List<SampleEntity> { new SampleEntity { Id = 1, Name = "Entity 1" } }

};

\_mockUnitOfWork.Setup(uow => uow.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<SampleEntity>));

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>> { WasSuccess = false };

\_mockUnitOfWork.Setup(uow => uow.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>>

{

WasSuccess = true,

Result = new List<SampleEntity> { new SampleEntity { Id = 1, Name = "Entity 1" } }

};

\_mockUnitOfWork.Setup(uow => uow.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<SampleEntity>));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>> { WasSuccess = false };

\_mockUnitOfWork.Setup(uow => uow.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 10

};

\_mockUnitOfWork.Setup(uow => uow.GetTotalRecordsAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(10, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<int> { WasSuccess = false };

\_mockUnitOfWork.Setup(uow => uow.GetTotalRecordsAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = new SampleEntity { Id = 1, Name = "Entity 1" }

};

\_mockUnitOfWork.Setup(uow => uow.GetAsync(1)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(1);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(SampleEntity));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsNotFound\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity> { WasSuccess = false };

\_mockUnitOfWork.Setup(uow => uow.GetAsync(1)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.GetAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundResult));

}

[TestMethod]

public async Task PostAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Entity 1" };

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = model

};

\_mockUnitOfWork.Setup(uow => uow.AddAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.PostAsync(model);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(SampleEntity));

}

[TestMethod]

public async Task PostAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Entity 1" };

var mockResponse = new ActionResponse<SampleEntity> { WasSuccess = false, Message = "Error" };

\_mockUnitOfWork.Setup(uow => uow.AddAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.PostAsync(model);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Error", badRequestResult.Value);

}

[TestMethod]

public async Task PutAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Entity 1" };

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = model

};

\_mockUnitOfWork.Setup(uow => uow.UpdateAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.PutAsync(model);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.IsInstanceOfType(okResult.Value, typeof(SampleEntity));

}

[TestMethod]

public async Task PutAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Entity 1" };

var mockResponse = new ActionResponse<SampleEntity> { WasSuccess = false, Message = "Error" };

\_mockUnitOfWork.Setup(uow => uow.UpdateAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.PutAsync(model);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Error", badRequestResult.Value);

}

[TestMethod]

public async Task DeleteAsync\_ReturnsNoContent\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity> { WasSuccess = true };

// Configura el mock para que el tipo genérico sea `SampleEntity`

\_mockUnitOfWork.Setup(uow => uow.DeleteAsync(It.IsAny<int>())).ReturnsAsync(mockResponse);

// Act

var result = await \_controller.DeleteAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(NoContentResult));

}

[TestMethod]

public async Task DeleteAsync\_ReturnsBadRequest\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = false,

Message = "Error occurred while deleting the entity."

};

\_mockUnitOfWork.Setup(uow => uow.DeleteAsync(It.IsAny<int>())).ReturnsAsync(mockResponse as ActionResponse<SampleEntity>);

// Act

var result = await \_controller.DeleteAsync(1);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Error occurred while deleting the entity.", badRequestResult.Value);

}

}

public class SampleEntity

{

public int Id { get; set; }

public string Name { get; set; } = null!;

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **GenericUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class GenericUnitOfWorkTests

{

private Mock<IGenericRepository<SampleEntity>> \_mockRepository = null!;

private GenericUnitOfWork<SampleEntity> \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockRepository = new Mock<IGenericRepository<SampleEntity>>();

\_unitOfWork = new GenericUnitOfWork<SampleEntity>(\_mockRepository.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsAddedEntity\_WhenSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Test Entity" };

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = model

};

\_mockRepository.Setup(repo => repo.AddAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.AddAsync(model);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(model, result.Result);

}

[TestMethod]

public async Task DeleteAsync\_ReturnsDeletedEntity\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = new SampleEntity { Id = 1, Name = "Test Entity" }

};

\_mockRepository.Setup(repo => repo.DeleteAsync(It.IsAny<int>())).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.DeleteAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

}

[TestMethod]

public async Task DeleteAsync\_ReturnsError\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = false,

Message = "Error occurred while deleting the entity."

};

\_mockRepository.Setup(repo => repo.DeleteAsync(It.IsAny<int>())).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.DeleteAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error occurred while deleting the entity.", result.Message);

}

[TestMethod]

public async Task GetAsync\_ReturnsEntities\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>>

{

WasSuccess = true,

Result = [new() { Id = 1, Name = "Test Entity" }]

};

\_mockRepository.Setup(repo => repo.GetAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsEntity\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = new SampleEntity { Id = 1, Name = "Test Entity" }

};

\_mockRepository.Setup(repo => repo.GetAsync(It.IsAny<int>())).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsError\_WhenNotSuccess()

{

// Arrange

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = false,

Message = "Entity not found."

};

\_mockRepository.Setup(repo => repo.GetAsync(It.IsAny<int>())).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Entity not found.", result.Message);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsPaginatedEntities\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO();

var mockResponse = new ActionResponse<IEnumerable<SampleEntity>>

{

WasSuccess = true,

Result = [new SampleEntity { Id = 1, Name = "Test Entity" }]

};

\_mockRepository.Setup(repo => repo.GetAsync(pagination)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalCount\_WhenSuccess()

{

// Arrange

var mockResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 10

};

\_mockRepository.Setup(repo => repo.GetTotalRecordsAsync()).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(10, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsUpdatedEntity\_WhenSuccess()

{

// Arrange

var model = new SampleEntity { Id = 1, Name = "Updated Entity" };

var mockResponse = new ActionResponse<SampleEntity>

{

WasSuccess = true,

Result = model

};

\_mockRepository.Setup(repo => repo.UpdateAsync(model)).ReturnsAsync(mockResponse);

// Act

var result = await \_unitOfWork.UpdateAsync(model);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(model, result.Result);

}

}

public class SampleEntity

{

public int Id { get; set; }

public string Name { get; set; } = null!;

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **GenericRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.EntityFrameworkCore;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class GenericRepositoryTests

{

private DataContext \_context = null!;

private GenericRepository<Country> \_repository = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

\_context = new DataContext(options);

\_repository = new GenericRepository<Country>(\_context);

\_context.Countries.AddRange(new List<Country>

{

new Country { Id = 1, Name = "Country 1" },

new Country { Id = 2, Name = "Country 2" }

});

\_context.SaveChanges();

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task AddAsync\_ReturnsAddedEntity\_WhenSuccess()

{

// Arrange

var newCountry = new Country { Id = 3, Name = "New Country" };

// Act

var result = await \_repository.AddAsync(newCountry);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(3, result.Result.Id);

}

[TestMethod]

public async Task DeleteAsync\_ReturnsSuccess\_WhenEntityExists()

{

// Act

var result = await \_repository.DeleteAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNull(await \_context.Countries.FindAsync(1));

}

[TestMethod]

public async Task DeleteAsync\_ReturnsError\_WhenEntityDoesNotExist()

{

// Act

var result = await \_repository.DeleteAsync(999); // Non-existent ID

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsEntity\_WhenEntityExists()

{

// Act

var result = await \_repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsError\_WhenEntityDoesNotExist()

{

// Act

var result = await \_repository.GetAsync(999); // Non-existent ID

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task GetAsync\_ReturnsAllEntities()

{

// Act

var result = await \_repository.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result.Count());

}

[TestMethod]

public async Task UpdateAsync\_ReturnsUpdatedEntity\_WhenSuccess()

{

// Arrange

var countryToUpdate = await \_context.Countries.FindAsync(1);

countryToUpdate!.Name = "Updated Country";

// Act

var result = await \_repository.UpdateAsync(countryToUpdate);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Updated Country", result.Result!.Name);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs()

{

// Arrange

var countryToUpdate = new Country { Id = 999, Name = "Non-existent Country" };

// Act

var result = await \_repository.UpdateAsync(countryToUpdate);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsPaginatedEntities()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 1 };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecordsCount()

{

// Act

var result = await \_repository.GetTotalRecordsAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs()

{

// Arrange

var newCountry = new Country { Id = 3, Name = "New Country" };

// Mock the DbContext and simulate DbUpdateException when SaveChangesAsync is called

var mockContext = new Mock<DataContext>(

new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options);

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new DbUpdateException());

\_repository = new GenericRepository<Country>(mockContext.Object);

// Act

var result = await \_repository.AddAsync(newCountry);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var newCountry = new Country { Id = 3, Name = "New Country" };

// Mock the DbContext and simulate a general exception when SaveChangesAsync is called

var mockContext = new Mock<DataContext>(

new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options);

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new Exception("General exception occurred"));

\_repository = new GenericRepository<Country>(mockContext.Object);

// Act

var result = await \_repository.AddAsync(newCountry);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task DeleteAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var mockContext = new Mock<DataContext>(

new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options);

// Simulate the entity to be deleted

var countryToDelete = new Country { Id = 1, Name = "Country 1" };

// Configure the DbSet to return the simulated entity

var mockDbSet = new Mock<DbSet<Country>>();

mockDbSet.Setup(m => m.FindAsync(1)).ReturnsAsync(countryToDelete);

mockContext.Setup(c => c.Set<Country>()).Returns(mockDbSet.Object);

// Simulate a general exception when trying to save changes

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new Exception("General exception occurred"));

\_repository = new GenericRepository<Country>(mockContext.Object);

// Act

var result = await \_repository.DeleteAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR002", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var mockContext = new Mock<DataContext>(

new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options);

// Simulate the entity to be updated

var countryToUpdate = new Country { Id = 1, Name = "Country 1" };

// Configure the DbSet to simulate the Update operation

var mockDbSet = new Mock<DbSet<Country>>();

mockDbSet.Setup(m => m.Update(It.IsAny<Country>()));

mockContext.Setup(c => c.Set<Country>()).Returns(mockDbSet.Object);

// Simulate a general exception when trying to save changes

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new Exception("General exception occurred"));

\_repository = new GenericRepository<Country>(mockContext.Object);

// Act

var result = await \_repository.UpdateAsync(countryToUpdate);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Equipos

### Controlador

1. Adicione la clase **TeamsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Moq;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class TeamsControllerTests

{

private Mock<ITeamsUnitOfWork> \_mockTeamsUnitOfWork = null!;

private Mock<IGenericUnitOfWork<Team>> \_mockGenericUnitOfWork = null!;

private TeamsController \_teamsController = null!;

[TestInitialize]

public void Setup()

{

// Initialize mock objects and controller

\_mockTeamsUnitOfWork = new Mock<ITeamsUnitOfWork>();

\_mockGenericUnitOfWork = new Mock<IGenericUnitOfWork<Team>>();

\_teamsController = new TeamsController(\_mockGenericUnitOfWork.Object, \_mockTeamsUnitOfWork.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock GetAsync to return a successful response

var teams = new List<Team> { new() { Id = 1, Name = "Team A" }, new() { Id = 2, Name = "Team B" } };

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = true, Result = teams };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync()).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method

var result = await \_teamsController.GetAsync();

// Assert: Verify that the result is an OkObjectResult with the expected data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(teams, okResult!.Value);

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange: Mock GetAsync to return a failed response

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = false };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync()).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method

var result = await \_teamsController.GetAsync();

// Assert: Verify that the result is a BadRequestResult

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_Paginated\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock paginated GetAsync

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var teams = new List<Team> { new Team { Id = 1, Name = "Team A" } };

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = true, Result = teams };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method with pagination

var result = await \_teamsController.GetAsync(pagination);

// Assert: Verify that the result is an OkObjectResult with the expected data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(teams, okResult!.Value);

}

[TestMethod]

public async Task GetAsync\_Paginated\_ReturnsBadRequest\_WhenFailure()

{

// Arrange: Mock paginated GetAsync to return a failed response

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = false };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method with pagination

var result = await \_teamsController.GetAsync(pagination);

// Assert: Verify that the result is a BadRequestResult

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock GetTotalRecordsAsync

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_mockTeamsUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetTotalRecordsAsync method

var result = await \_teamsController.GetTotalRecordsAsync(pagination);

// Assert: Verify that the result is an OkObjectResult with the total records

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(100, okResult!.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange: Mock GetTotalRecordsAsync to return a failed response

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int> { WasSuccess = false };

\_mockTeamsUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetTotalRecordsAsync method

var result = await \_teamsController.GetTotalRecordsAsync(pagination);

// Assert: Verify that the result is a BadRequestResult

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock GetAsync by ID

var team = new Team { Id = 1, Name = "Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync(1)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method by ID

var result = await \_teamsController.GetAsync(1);

// Assert: Verify that the result is an OkObjectResult with the expected data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(team, okResult!.Value);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsNotFound\_WhenFailure()

{

// Arrange: Mock GetAsync by ID to return a failed response

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Team not found" };

\_mockTeamsUnitOfWork.Setup(u => u.GetAsync(1)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method by ID

var result = await \_teamsController.GetAsync(1);

// Assert: Verify that the result is a NotFoundObjectResult

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

var notFoundResult = result as NotFoundObjectResult;

Assert.AreEqual("Team not found", notFoundResult!.Value);

}

[TestMethod]

public async Task PostAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock AddAsync

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1 };

var team = new Team { Id = 1, Name = "Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsUnitOfWork.Setup(u => u.AddAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the PostAsync method

var result = await \_teamsController.PostAsync(teamDTO);

// Assert: Verify that the result is an OkObjectResult with the expected data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(team, okResult!.Value);

}

[TestMethod]

public async Task PostAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange: Mock AddAsync to return a failed response

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1 };

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Error adding team" };

\_mockTeamsUnitOfWork.Setup(u => u.AddAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the PostAsync method

var result = await \_teamsController.PostAsync(teamDTO);

// Assert: Verify that the result is a BadRequestObjectResult

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

var badRequestResult = result as BadRequestObjectResult;

Assert.AreEqual("Error adding team", badRequestResult!.Value);

}

[TestMethod]

public async Task PutAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock UpdateAsync

var teamDTO = new TeamDTO { Id = 1, Name = "Team A", CountryId = 1 };

var team = new Team { Id = 1, Name = "Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsUnitOfWork.Setup(u => u.UpdateAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the PutAsync method

var result = await \_teamsController.PutAsync(teamDTO);

// Assert: Verify that the result is an OkObjectResult with the expected data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(team, okResult!.Value);

}

[TestMethod]

public async Task PutAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange: Mock UpdateAsync to return a failed response

var teamDTO = new TeamDTO { Id = 1, Name = "Team A", CountryId = 1 };

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Error updating team" };

\_mockTeamsUnitOfWork.Setup(u => u.UpdateAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the PutAsync method

var result = await \_teamsController.PutAsync(teamDTO);

// Assert: Verify that the result is a BadRequestObjectResult

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

var badRequestResult = result as BadRequestObjectResult;

Assert.AreEqual("Error updating team", badRequestResult!.Value);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange: Mock GetComboAsync to return a list of teams

var comboData = new List<Team>

{

new Team { Id = 1, Name = "Team A" },

new Team { Id = 2, Name = "Team B" }

};

\_mockTeamsUnitOfWork.Setup(u => u.GetComboAsync(It.IsAny<int>()))

.ReturnsAsync(comboData);

// Act: Call the GetComboAsync method

var result = await \_teamsController.GetComboAsync(1);

// Assert: Verify that the result is an OkObjectResult with the expected combo data

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(comboData, okResult!.Value);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsEmptyOk\_WhenNoData()

{

// Arrange: Mock GetComboAsync to return an empty list of teams

var comboData = new List<Team>(); // Empty list

\_mockTeamsUnitOfWork.Setup(u => u.GetComboAsync(It.IsAny<int>()))

.ReturnsAsync(comboData);

// Act: Call the GetComboAsync method

var result = await \_teamsController.GetComboAsync(1);

// Assert: Verify that the result is an OkObjectResult with an empty list

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

var okResult = result as OkObjectResult;

Assert.AreEqual(comboData, okResult!.Value); // Should be empty

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **TeamsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class TeamsUnitOfWorkTests

{

private Mock<ITeamsRepository> \_mockTeamsRepository = null!;

private Mock<IGenericRepository<Team>> \_mockGenericRepository = null!;

private TeamsUnitOfWork \_teamsUnitOfWork = null!;

[TestInitialize]

public void Setup()

{

// Initialize mocks and the unit of work

\_mockTeamsRepository = new Mock<ITeamsRepository>();

\_mockGenericRepository = new Mock<IGenericRepository<Team>>();

\_teamsUnitOfWork = new TeamsUnitOfWork(\_mockGenericRepository.Object, \_mockTeamsRepository.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange: Mock AddAsync

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1 };

var team = new Team { Id = 1, Name = "Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsRepository.Setup(r => r.AddAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the AddAsync method

var result = await \_teamsUnitOfWork.AddAsync(teamDTO);

// Assert: Verify the action response is returned

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(team, result.Result);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenFailure()

{

// Arrange: Mock AddAsync to return an error response

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1 };

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Error adding team" };

\_mockTeamsRepository.Setup(r => r.AddAsync(It.IsAny<TeamDTO>())).ReturnsAsync(actionResponse);

// Act: Call the AddAsync method

var result = await \_teamsUnitOfWork.AddAsync(teamDTO);

// Assert: Verify the error response

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error adding team", result.Message);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsTeams\_WhenSuccess()

{

// Arrange: Mock GetComboAsync

var comboData = new List<Team> { new Team { Id = 1, Name = "Team A" }, new Team { Id = 2, Name = "Team B" } };

\_mockTeamsRepository.Setup(r => r.GetComboAsync(It.IsAny<int>())).ReturnsAsync(comboData);

// Act: Call the GetComboAsync method

var result = await \_teamsUnitOfWork.GetComboAsync(1);

// Assert: Verify the result is a list of teams

Assert.AreEqual(comboData, result);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange: Mock UpdateAsync

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team A", CountryId = 1 };

var team = new Team { Id = 1, Name = "Updated Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsRepository.Setup(r => r.UpdateAsync(teamDTO)).ReturnsAsync(actionResponse);

// Act: Call the UpdateAsync method

var result = await \_teamsUnitOfWork.UpdateAsync(teamDTO);

// Assert: Verify the action response is returned

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(team, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenFailure()

{

// Arrange: Mock UpdateAsync to return an error response

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team A", CountryId = 1 };

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Error updating team" };

\_mockTeamsRepository.Setup(r => r.UpdateAsync(It.IsAny<TeamDTO>())).ReturnsAsync(actionResponse);

// Act: Call the UpdateAsync method

var result = await \_teamsUnitOfWork.UpdateAsync(teamDTO);

// Assert: Verify the error response

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error updating team", result.Message);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange: Mock GetAsync by ID

var team = new Team { Id = 1, Name = "Team A" };

var actionResponse = new ActionResponse<Team> { WasSuccess = true, Result = team };

\_mockTeamsRepository.Setup(r => r.GetAsync(1)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method

var result = await \_teamsUnitOfWork.GetAsync(1);

// Assert: Verify the action response

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(team, result.Result);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsError\_WhenFailure()

{

// Arrange: Mock GetAsync by ID to return an error response

var actionResponse = new ActionResponse<Team> { WasSuccess = false, Message = "Team not found" };

\_mockTeamsRepository.Setup(r => r.GetAsync(1)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method

var result = await \_teamsUnitOfWork.GetAsync(1);

// Assert: Verify the error response

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Team not found", result.Message);

}

[TestMethod]

public async Task GetAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange: Mock GetAsync to return a list of teams

var teams = new List<Team> { new Team { Id = 1, Name = "Team A" }, new Team { Id = 2, Name = "Team B" } };

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = true, Result = teams };

\_mockTeamsRepository.Setup(r => r.GetAsync()).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method

var result = await \_teamsUnitOfWork.GetAsync();

// Assert: Verify the action response

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(teams, result.Result);

}

[TestMethod]

public async Task GetAsync\_ReturnsPaginatedTeams\_WhenSuccess()

{

// Arrange: Mock GetAsync with pagination

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var teams = new List<Team> { new Team { Id = 1, Name = "Team A" }, new Team { Id = 2, Name = "Team B" } };

var actionResponse = new ActionResponse<IEnumerable<Team>> { WasSuccess = true, Result = teams };

\_mockTeamsRepository.Setup(r => r.GetAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetAsync method with pagination

var result = await \_teamsUnitOfWork.GetAsync(pagination);

// Assert: Verify the action response

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(teams, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange: Mock GetTotalRecordsAsync

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_mockTeamsRepository.Setup(r => r.GetTotalRecordsAsync(pagination)).ReturnsAsync(actionResponse);

// Act: Call the GetTotalRecordsAsync method

var result = await \_teamsUnitOfWork.GetTotalRecordsAsync(pagination);

// Assert: Verify the action response

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(100, result.Result);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **TeamsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class TeamsRepositoryTests

{

private TeamsRepository \_repository = null!;

private Mock<IFileStorage> \_mockFileStorage = null!;

private DataContext \_context = null!;

[TestInitialize]

public void Setup()

{

// Set up the In-Memory Database

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(options);

\_mockFileStorage = new Mock<IFileStorage>();

// Initialize the repository

\_repository = new TeamsRepository(\_context, \_mockFileStorage.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsSuccess\_WhenTeamIsAdded()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

\_context.Countries.Add(country);

await \_context.SaveChangesAsync();

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1 };

// Act

var result = await \_repository.AddAsync(teamDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Team A", result.Result!.Name);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenCountryNotFound()

{

// Arrange

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 999 }; // Non-existent country

// Act

var result = await \_repository.AddAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR004", result.Message); // Country not found error code

}

[TestMethod]

public async Task GetComboAsync\_ReturnsTeams\_WhenTeamsExist()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team1 = new Team { Id = 1, Name = "Team A", CountryId = 1 };

var team2 = new Team { Id = 2, Name = "Team B", CountryId = 1 };

\_context.Countries.Add(country);

\_context.Teams.AddRange(team1, team2);

await \_context.SaveChangesAsync();

// Act

var result = await \_repository.GetComboAsync(1);

// Assert

Assert.AreEqual(2, result.Count());

Assert.IsTrue(result.Any(t => t.Name == "Team A"));

Assert.IsTrue(result.Any(t => t.Name == "Team B"));

}

[TestMethod]

public async Task UpdateAsync\_ReturnsSuccess\_WhenTeamIsUpdated()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team = new Team { Id = 1, Name = "Old Team", Country = country };

\_context.Countries.Add(country);

\_context.Teams.Add(team);

await \_context.SaveChangesAsync();

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team", CountryId = 1 };

// Act

var result = await \_repository.UpdateAsync(teamDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Updated Team", result.Result!.Name);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenTeamNotFound()

{

// Arrange

var teamDTO = new TeamDTO { Id = 999, Name = "Non-existent Team", CountryId = 1 };

// Act

var result = await \_repository.UpdateAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR005", result.Message); // Team not found error code

}

[TestMethod]

public async Task GetAsync\_ReturnsTeams\_WhenTeamsExist()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team1 = new Team { Id = 1, Name = "Team A", Country = country };

var team2 = new Team { Id = 2, Name = "Team B", Country = country };

\_context.Countries.Add(country);

\_context.Teams.AddRange(team1, team2);

await \_context.SaveChangesAsync();

// Act

var result = await \_repository.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsTeam\_WhenTeamExists()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team = new Team { Id = 1, Name = "Team A", Country = country };

\_context.Countries.Add(country);

\_context.Teams.Add(team);

await \_context.SaveChangesAsync();

// Act

var result = await \_repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Team A", result.Result!.Name);

Assert.AreEqual("/images/NoImage.png", result.Result!.ImageFull);

Assert.AreEqual(0, result.Result!.TournamentsCount);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsError\_WhenTeamNotFound()

{

// Act

var result = await \_repository.GetAsync(999); // Non-existent team ID

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message); // Team not found error code

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsCount\_WhenFilterApplied()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team1 = new Team { Id = 1, Name = "Team A", Country = country };

var team2 = new Team { Id = 2, Name = "Team B", Country = country };

\_context.Countries.Add(country);

\_context.Teams.AddRange(team1, team2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "Team" };

// Act

var result = await \_repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetAsync\_Paginated\_ReturnsPaginatedTeams\_WhenTeamsExist()

{

// Arrange

var country = new Country { Id = 1, Name = "Country A" };

var team1 = new Team { Id = 1, Name = "Team A", Country = country };

var team2 = new Team { Id = 2, Name = "Team B", Country = country };

\_context.Countries.Add(country);

\_context.Teams.AddRange(team1, team2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Act

var result = await \_repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task AddAsync\_ReturnsSuccess\_WhenTeamIsAddedWithImage()

{

// Arrange: Add a country to the in-memory database to avoid "ERR004"

var country = new Country { Id = 1, Name = "Country A" };

\_context.Countries.Add(country);

await \_context.SaveChangesAsync();

// Create a TeamDTO with a Base64 image string

var imageBase64 = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }); // Example Base64 image

var teamDTO = new TeamDTO { Name = "Team A", CountryId = 1, Image = imageBase64 };

// Mock the SaveFileAsync method to return a fake image URL

\_mockFileStorage.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "teams"))

.ReturnsAsync("http://example.com/teamimage.jpg");

// Act: Call the AddAsync method

var result = await \_repository.AddAsync(teamDTO);

// Assert: Ensure that the team was added successfully and the image was saved

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Team A", result.Result!.Name);

Assert.AreEqual("http://example.com/teamimage.jpg", result.Result.Image);

// Verify that SaveFileAsync was called with the correct parameters

\_mockFileStorage.Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "teams"), Times.Once);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenCountryNotFound()

{

// Arrange: Add a team to the in-memory database but do not add the country to simulate "ERR004"

var team = new Team { Id = 1, Name = "Team A", CountryId = 1 };

\_context.Teams.Add(team);

await \_context.SaveChangesAsync();

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team A", CountryId = 999 }; // Non-existent country ID

// Act: Call the UpdateAsync method

var result = await \_repository.UpdateAsync(teamDTO);

// Assert: Ensure the response indicates failure and returns the correct error message

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR004", result.Message); // Country not found error code

}

[TestMethod]

public async Task UpdateAsync\_ReturnsSuccess\_WhenTeamIsUpdatedWithImage()

{

// Arrange: Add a country and a team to the in-memory database

var country = new Country { Id = 1, Name = "Country A" };

var team = new Team { Id = 1, Name = "Team A", Country = country, CountryId = 1 };

\_context.Countries.Add(country);

\_context.Teams.Add(team);

await \_context.SaveChangesAsync();

// Create a TeamDTO with a Base64 image string

var imageBase64 = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }); // Example Base64 image

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team A", CountryId = 1, Image = imageBase64 };

// Mock the SaveFileAsync method to return a fake image URL

\_mockFileStorage.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "teams"))

.ReturnsAsync("http://example.com/teamimage.jpg");

// Act: Call the UpdateAsync method

var result = await \_repository.UpdateAsync(teamDTO);

// Assert: Ensure the team was updated successfully and the image was saved

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Updated Team A", result.Result!.Name);

Assert.AreEqual("http://example.com/teamimage.jpg", result.Result.Image);

// Verify that SaveFileAsync was called with the correct parameters

\_mockFileStorage.Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "teams"), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ReturnsFilteredTeams\_WhenFilterIsApplied()

{

// Arrange: Add countries and teams to the in-memory database

var country1 = new Country { Id = 1, Name = "Country A" };

var country2 = new Country { Id = 2, Name = "Country B" };

var team1 = new Team { Id = 1, Name = "Team Alpha", Country = country1 };

var team2 = new Team { Id = 2, Name = "Team Beta", Country = country2 };

var team3 = new Team { Id = 3, Name = "Team Gamma", Country = country1 };

\_context.Countries.AddRange(country1, country2);

\_context.Teams.AddRange(team1, team2, team3);

await \_context.SaveChangesAsync();

// Create a PaginationDTO with a filter for teams with "Alpha" in their name

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "Alpha" };

// Act: Call the GetAsync method with the filter

var result = await \_repository.GetAsync(pagination);

// Assert: Ensure only the team with "Alpha" in the name is returned

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

Assert.AreEqual("Team Alpha", result.Result!.First().Name);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add a team to the in-memory database

var country = new Country { Id = 1, Name = "Country A" };

var team = new Team { Id = 1, Name = "Original Team", Country = country };

context.Countries.Add(country);

context.Teams.Add(team);

await context.SaveChangesAsync();

// Create a fake context to simulate a DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new TeamsRepository(fakeContext, \_mockFileStorage.Object);

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team", CountryId = 1 };

// Act

var result = await repository.UpdateAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Assert that the error message matches ERR003

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add a team to the in-memory database

var country = new Country { Id = 1, Name = "Country A" };

var team = new Team { Id = 1, Name = "Original Team", Country = country };

context.Countries.Add(country);

context.Teams.Add(team);

await context.SaveChangesAsync();

// Create a fake context to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new TeamsRepository(fakeContext, \_mockFileStorage.Object);

var teamDTO = new TeamDTO { Id = 1, Name = "Updated Team", CountryId = 1 };

// Act

var result = await repository.UpdateAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Assert that the error message matches the simulated general exception message

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add a country to the in-memory database

var country = new Country { Id = 1, Name = "Country A" };

context.Countries.Add(country);

await context.SaveChangesAsync();

// Create a fake context to simulate a DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new TeamsRepository(fakeContext, \_mockFileStorage.Object);

var teamDTO = new TeamDTO { Name = "New Team", CountryId = 1 };

// Act

var result = await repository.AddAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Assert that the error message matches ERR003

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add a country to the in-memory database

var country = new Country { Id = 1, Name = "Country A" };

context.Countries.Add(country);

await context.SaveChangesAsync();

// Create a fake context to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new TeamsRepository(fakeContext, \_mockFileStorage.Object);

var teamDTO = new TeamDTO { Name = "New Team", CountryId = 1 };

// Act

var result = await repository.AddAsync(teamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Assert that the error message matches the simulated general exception message

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Torneo

### Controlador

1. Adicione la clase **TournamentsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Moq;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class TournamentsControllerTests

{

private Mock<ITournamentsUnitOfWork> \_mockUnitOfWork = null!;

private TournamentsController \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockUnitOfWork = new Mock<ITournamentsUnitOfWork>();

\_controller = new TournamentsController(null!, \_mockUnitOfWork.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange

var tournaments = new List<Tournament> { new() { Id = 1, Name = "Tournament 1" } };

\_mockUnitOfWork.Setup(u => u.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<Tournament>> { WasSuccess = true, Result = tournaments });

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenFailed()

{

// Arrange

\_mockUnitOfWork.Setup(u => u.GetAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<Tournament>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var tournaments = new List<Tournament> { new() { Id = 1, Name = "Tournament 1" } };

\_mockUnitOfWork.Setup(u => u.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<Tournament>> { WasSuccess = true, Result = tournaments });

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

\_mockUnitOfWork.Setup(u => u.GetAsync(pagination))

.ReturnsAsync(new ActionResponse<IEnumerable<Tournament>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

\_mockUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 5 });

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

\_mockUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false });

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsOk\_WhenSuccess()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament 1" };

\_mockUnitOfWork.Setup(u => u.GetAsync(1))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = true, Result = tournament });

// Act

var result = await \_controller.GetAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task GetAsync\_WithId\_ReturnsNotFound\_WhenFailed()

{

// Arrange

\_mockUnitOfWork.Setup(u => u.GetAsync(1))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = false, Message = "Not found" });

// Act

var result = await \_controller.GetAsync(1);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

}

[TestMethod]

public async Task GetComboAsync\_ReturnsOk()

{

// Arrange

var comboList = new List<Tournament> { new Tournament { Id = 1, Name = "Combo 1" } };

\_mockUnitOfWork.Setup(u => u.GetComboAsync())

.ReturnsAsync(comboList);

// Act

var result = await \_controller.GetComboAsync();

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task PostAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange

var tournamentDTO = new TournamentDTO { };

\_mockUnitOfWork.Setup(u => u.AddAsync(tournamentDTO))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = true, Result = new Tournament() });

// Act

var result = await \_controller.PostAsync(tournamentDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task PostAsync\_ReturnsBadRequest\_WhenFailed()

{

// Arrange

var tournamentDTO = new TournamentDTO { };

\_mockUnitOfWork.Setup(u => u.AddAsync(tournamentDTO))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = false, Message = "Error" });

// Act

var result = await \_controller.PostAsync(tournamentDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestObjectResult));

}

[TestMethod]

public async Task PutAsync\_ReturnsOk\_WhenSuccess()

{

// Arrange

var tournamentDTO = new TournamentDTO { };

\_mockUnitOfWork.Setup(u => u.UpdateAsync(tournamentDTO))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = true, Result = new Tournament() });

// Act

var result = await \_controller.PutAsync(tournamentDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(OkObjectResult));

}

[TestMethod]

public async Task PutAsync\_ReturnsNotFound\_WhenFailed()

{

// Arrange

var tournamentDTO = new TournamentDTO { };

\_mockUnitOfWork.Setup(u => u.UpdateAsync(tournamentDTO))

.ReturnsAsync(new ActionResponse<Tournament> { WasSuccess = false, Message = "Not found" });

// Act

var result = await \_controller.PutAsync(tournamentDTO);

// Assert

Assert.IsInstanceOfType(result, typeof(NotFoundObjectResult));

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **TournamentsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class TournamentsUnitOfWorkTests

{

private Mock<IGenericRepository<Tournament>> \_mockGenericRepository = null!;

private Mock<ITournamentsRepository> \_mockTournamentsRepository = null!;

private TournamentsUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockGenericRepository = new Mock<IGenericRepository<Tournament>>();

\_mockTournamentsRepository = new Mock<ITournamentsRepository>();

\_unitOfWork = new TournamentsUnitOfWork(\_mockGenericRepository.Object, \_mockTournamentsRepository.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange

var tournamentDTO = new TournamentDTO { /\* Tournament properties \*/ };

var response = new ActionResponse<Tournament> { WasSuccess = true };

\_mockTournamentsRepository.Setup(r => r.AddAsync(tournamentDTO))

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.AddAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsTournamentList\_WhenSuccess()

{

// Arrange

var tournaments = new List<Tournament> { new() { Id = 1, Name = "Tournament 1" } };

\_mockTournamentsRepository.Setup(r => r.GetComboAsync())

.ReturnsAsync(tournaments);

// Act

var result = await \_unitOfWork.GetComboAsync();

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Count());

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecords\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 5 };

\_mockTournamentsRepository.Setup(r => r.GetTotalRecordsAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(5, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange

var tournamentDTO = new TournamentDTO { /\* Tournament properties \*/ };

var response = new ActionResponse<Tournament> { WasSuccess = true };

\_mockTournamentsRepository.Setup(r => r.UpdateAsync(tournamentDTO))

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.UpdateAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament 1" };

var response = new ActionResponse<Tournament> { WasSuccess = true, Result = tournament };

\_mockTournamentsRepository.Setup(r => r.GetAsync(1))

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Id);

}

[TestMethod]

public async Task GetAsync\_ReturnsActionResponse\_WithAllTournaments()

{

// Arrange

var tournaments = new List<Tournament> { new() { Id = 1, Name = "Tournament 1" } };

var response = new ActionResponse<IEnumerable<Tournament>> { WasSuccess = true, Result = tournaments };

\_mockTournamentsRepository.Setup(r => r.GetAsync())

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsActionResponse\_WithPaginatedTournaments()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var tournaments = new List<Tournament> { new() { Id = 1, Name = "Tournament 1" } };

var response = new ActionResponse<IEnumerable<Tournament>> { WasSuccess = true, Result = tournaments };

\_mockTournamentsRepository.Setup(r => r.GetAsync(pagination))

.ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. En el proyecto de **Tests** cree la carpeta **General** y dentro de esta la clase **FakeDbContext**:

using Fantasy.Backend.Data;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Tests.General;

public class FakeDbContext : DataContext

{

public FakeDbContext(DbContextOptions<DataContext> options)

: base(options)

{

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

throw new DbUpdateException();

}

}

1. En la misma carpeta cree el **FakeDbContextWithGeneralException**:

using Fantasy.Backend.Data;

using Microsoft.EntityFrameworkCore;

namespace Fantasy.Tests.General;

public class FakeDbContextWithGeneralException : DataContext

{

public FakeDbContextWithGeneralException(DbContextOptions<DataContext> options)

: base(options)

{

}

public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

throw new Exception("General exception occurred");

}

}

1. Adicione la clase **TournamentsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class TournamentsRepositoryTests

{

private Mock<IFileStorage> \_mockFileStorage = null!;

private TournamentsRepository \_repository = null!;

[TestInitialize]

public void Setup()

{

\_mockFileStorage = new Mock<IFileStorage>();

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WhenSuccess\_WithoutImage()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

\_repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Name = "Test Tournament", Image = null };

\_mockFileStorage.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "tournaments"))

.ReturnsAsync("imagePath");

// Act

var result = await \_repository.AddAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual("Test Tournament", result.Result.Name);

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WhenSuccess\_WithImage()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

\_repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var validBase64Image = "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAwCAAAAC0lEQVR42mP8/wcAAwAB/ebQjH0AAAAASUVORK5CYII=";

var tournamentDTO = new TournamentDTO { Name = "Test Tournament", Image = validBase64Image };

\_mockFileStorage.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "tournaments"))

.ReturnsAsync("imagePath");

// Act

var result = await \_repository.AddAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual("Test Tournament", result.Result.Name);

Assert.AreEqual("imagePath", result.Result.Image);

\_mockFileStorage.Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "tournaments"), Times.Once);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var mockContext = new Mock<DataContext>(options);

\_repository = new TournamentsRepository(mockContext.Object, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Name = "Test Tournament" };

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new DbUpdateException());

// Act

var result = await \_repository.AddAsync(tournamentDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var mockContext = new Mock<DataContext>(options);

\_repository = new TournamentsRepository(mockContext.Object, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Name = "Test Tournament" };

mockContext.Setup(c => c.SaveChangesAsync(It.IsAny<CancellationToken>()))

.ThrowsAsync(new Exception("General exception occurred"));

// Act

var result = await \_repository.AddAsync(tournamentDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsActiveTournaments()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.AddRange(

new Tournament { Id = 1, Name = "Tournament 1", IsActive = true },

new Tournament { Id = 2, Name = "Tournament 2", IsActive = false }

);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

// Act

var result = await repository.GetComboAsync();

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(1, result.Count());

Assert.AreEqual("Tournament 1", result.First().Name);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsPaginatedTournaments()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.AddRange(

new Tournament { Id = 1, Name = "Tournament 1" },

new Tournament { Id = 2, Name = "Tournament 2" }

);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Act

var result = await repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsTournament\_WhenExists()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var tournament = new Tournament { Id = 1, Name = "Tournament 1" };

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

// Act

var result = await repository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

Assert.AreEqual("Tournament 1", result.Result.Name);

Assert.AreEqual("/images/NoImage.png", result.Result.ImageFull);

Assert.AreEqual(0, result.Result.TeamsCount);

Assert.AreEqual(0, result.Result.MatchesCount);

Assert.AreEqual(0, result.Result.GroupsCount);

Assert.AreEqual(0, result.Result.PredictionsCount);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsError\_WhenNotExists()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

// Act

var result = await repository.GetAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecordCount()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.AddRange(

new Tournament { Id = 1, Name = "Test Tournament 1" },

new Tournament { Id = 2, Name = "Other Tournament 2" }

);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var pagination = new PaginationDTO { Filter = "Test" };

// Act

var result = await repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsActionResponse\_WhenSuccess()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var tournament = new Tournament { Id = 1, Name = "Original Tournament", IsActive = false };

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Id = 1, Name = "Updated Tournament", IsActive = true };

// Act

var result = await repository.UpdateAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual("Updated Tournament", result.Result!.Name);

Assert.IsTrue(result.Result.IsActive);

context.Entry(result.Result).Reload();

Assert.AreEqual("Updated Tournament", context.Tournaments.Find(1)!.Name);

Assert.IsTrue(context.Tournaments.Find(1)!.IsActive);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenNotExists()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Id = 1, Name = "Updated Tournament" };

// Act

var result = await repository.UpdateAsync(tournamentDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR005", result.Message);

}

[TestMethod]

public async Task

UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var tournament = new Tournament { Id = 1, Name = "Original Tournament" };

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

var fakeContext = new FakeDbContext(options);

var repository = new TournamentsRepository(fakeContext, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Id = 1, Name = "Updated Tournament" };

// Act

var result = await repository.UpdateAsync(tournamentDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var tournament = new Tournament { Id = 1, Name = "Original Tournament" };

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new TournamentsRepository(fakeContext, \_mockFileStorage.Object);

var tournamentDTO = new TournamentDTO { Id = 1, Name = "Updated Tournament" };

// Act

var result = await repository.UpdateAsync(tournamentDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task GetAsync\_WithPaginationAndFilter\_ReturnsFilteredTournaments()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.AddRange(

new Tournament { Id = 1, Name = "Test Tournament 1" },

new Tournament { Id = 2, Name = "Another Tournament" },

new Tournament { Id = 3, Name = "Test Tournament 2" }

);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "Test" };

// Act

var result = await repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

Assert.IsTrue(result.Result!.All(t => t.Name.Contains("Test")));

}

[TestMethod]

public async Task UpdateAsync\_ReturnsSuccess\_WhenTournamentDTOHasImage()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var tournament = new Tournament { Id = 1, Name = "Original Tournament", IsActive = false, Remarks = "Original Remarks" };

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

var repository = new TournamentsRepository(context, \_mockFileStorage.Object);

var validBase64Image = "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAwCAAAAC0lEQVR42mP8/wcAAwAB/ebQjH0AAAAASUVORK5CYII=";

var tournamentDTO = new TournamentDTO

{

Id = 1,

Name = "Updated Tournament",

IsActive = true,

Remarks = "Updated Remarks",

Image = validBase64Image

};

\_mockFileStorage.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "tournaments"))

.ReturnsAsync("newImagePath");

// Act

var result = await repository.UpdateAsync(tournamentDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual("Updated Tournament", result.Result.Name);

Assert.AreEqual(true, result.Result.IsActive);

Assert.AreEqual("Updated Remarks", result.Result.Remarks);

Assert.AreEqual("newImagePath", result.Result.Image);

\_mockFileStorage.Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "tournaments"), Times.Once);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Torneo/Equipos

### Controlador

1. Adicione la clase **TournamentTeamsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Mvc;

using Moq;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class TournamentTeamsControllerTests

{

private Mock<ITournamentTeamsUnitOfWork> \_mockUnitOfWork = null!;

private TournamentTeamsController \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockUnitOfWork = new Mock<ITournamentTeamsUnitOfWork>();

\_controller = new TournamentTeamsController(null!, \_mockUnitOfWork.Object);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsOkResult\_WithTournamentTeams()

{

// Arrange

var tournamentId = 1;

var mockTeams = new List<TournamentTeam>

{

new() { Id = 1, TournamentId = tournamentId },

new() { Id = 2, TournamentId = tournamentId }

};

\_mockUnitOfWork.Setup(u => u.GetComboAsync(tournamentId))

.ReturnsAsync(mockTeams);

// Act

var result = await \_controller.GetComboAsync(tournamentId);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

var teams = okResult.Value as IEnumerable<TournamentTeam>;

Assert.IsNotNull(teams);

Assert.AreEqual(2, teams.Count());

}

[TestMethod]

public async Task PostAsync\_ReturnsOkResult\_WhenSuccess()

{

// Arrange

var tournamentTeamDTO = new TournamentTeamDTO { Id = 1, TournamentId = 1, TeamId = 1 };

var actionResponse = new ActionResponse<TournamentTeam>

{

WasSuccess = true,

Result = new TournamentTeam { Id = 1, TournamentId = 1, TeamId = 1 }

};

\_mockUnitOfWork.Setup(u => u.AddAsync(tournamentTeamDTO))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.PostAsync(tournamentTeamDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

var team = okResult.Value as TournamentTeam;

Assert.IsNotNull(team);

Assert.AreEqual(1, team.Id);

}

[TestMethod]

public async Task PostAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange

var tournamentTeamDTO = new TournamentTeamDTO { Id = 1, TournamentId = 1, TeamId = 1 };

var actionResponse = new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = "Error occurred"

};

\_mockUnitOfWork.Setup(u => u.AddAsync(tournamentTeamDTO))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.PostAsync(tournamentTeamDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Error occurred", badRequestResult.Value);

}

[TestMethod]

public async Task GetAsync\_ReturnsOkResult\_WithPaginatedTournamentTeams()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var mockTeams = new List<TournamentTeam>

{

new TournamentTeam { Id = 1, TournamentId = 1 },

new TournamentTeam { Id = 2, TournamentId = 1 }

};

var actionResponse = new ActionResponse<IEnumerable<TournamentTeam>>

{

WasSuccess = true,

Result = mockTeams

};

\_mockUnitOfWork.Setup(u => u.GetAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

var teams = okResult.Value as IEnumerable<TournamentTeam>;

Assert.IsNotNull(teams);

Assert.AreEqual(2, teams.Count());

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<IEnumerable<TournamentTeam>>

{

WasSuccess = false

};

\_mockUnitOfWork.Setup(u => u.GetAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOkResult\_WithTotalRecords()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 5

};

\_mockUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(5, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenFailure()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int>

{

WasSuccess = false

};

\_mockUnitOfWork.Setup(u => u.GetTotalRecordsAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **TournamentTeamsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class TournamentTeamsUnitOfWorkTests

{

private Mock<ITournamentTeamsRepository> \_mockRepository = null!;

private TournamentTeamsUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockRepository = new Mock<ITournamentTeamsRepository>();

\_unitOfWork = new TournamentTeamsUnitOfWork(null!, \_mockRepository.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WithTournamentTeam\_WhenSuccess()

{

// Arrange

var tournamentTeamDTO = new TournamentTeamDTO { Id = 1, TournamentId = 1, TeamId = 1 };

var actionResponse = new ActionResponse<TournamentTeam>

{

WasSuccess = true,

Result = new TournamentTeam { Id = 1, TournamentId = 1, TeamId = 1 }

};

\_mockRepository.Setup(r => r.AddAsync(tournamentTeamDTO))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Id);

Assert.AreEqual(1, result.Result.TournamentId);

Assert.AreEqual(1, result.Result.TeamId);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsTournamentTeams()

{

// Arrange

var tournamentId = 1;

var mockTeams = new List<TournamentTeam>

{

new TournamentTeam { Id = 1, TournamentId = tournamentId },

new TournamentTeam { Id = 2, TournamentId = tournamentId }

};

\_mockRepository.Setup(r => r.GetComboAsync(tournamentId))

.ReturnsAsync(mockTeams);

// Act

var result = await \_unitOfWork.GetComboAsync(tournamentId);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(2, result.Count());

Assert.AreEqual(tournamentId, result.First().TournamentId);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecordCount\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int>

{

WasSuccess = true,

Result = 5

};

\_mockRepository.Setup(r => r.GetTotalRecordsAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(5, result.Result);

}

[TestMethod]

public async Task GetAsync\_ReturnsPaginatedTournamentTeams\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var mockTeams = new List<TournamentTeam>

{

new TournamentTeam { Id = 1, TournamentId = 1 },

new TournamentTeam { Id = 2, TournamentId = 1 }

};

var actionResponse = new ActionResponse<IEnumerable<TournamentTeam>>

{

WasSuccess = true,

Result = mockTeams

};

\_mockRepository.Setup(r => r.GetAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(2, result.Result.Count());

}

[TestMethod]

public async Task AddAsync\_ReturnsActionResponse\_WithError\_WhenFailure()

{

// Arrange

var tournamentTeamDTO = new TournamentTeamDTO { Id = 1, TournamentId = 1, TeamId = 1 };

var actionResponse = new ActionResponse<TournamentTeam>

{

WasSuccess = false,

Message = "Error occurred"

};

\_mockRepository.Setup(r => r.AddAsync(tournamentTeamDTO))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error occurred", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsError\_WhenFailure()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<int>

{

WasSuccess = false,

Message = "Error occurred"

};

\_mockRepository.Setup(r => r.GetTotalRecordsAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error occurred", result.Message);

}

[TestMethod]

public async Task GetAsync\_ReturnsError\_WhenFailure()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var actionResponse = new ActionResponse<IEnumerable<TournamentTeam>>

{

WasSuccess = false,

Message = "Error occurred"

};

\_mockRepository.Setup(r => r.GetAsync(pagination))

.ReturnsAsync(actionResponse);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error occurred", result.Message);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **TournamentTeamsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class TournamentTeamsRepositoryTests

{

private Mock<DataContext> \_mockContext = null!;

private TournamentTeamsRepository \_repository = null!;

[TestInitialize]

public void Setup()

{

\_mockContext = new Mock<DataContext>(new DbContextOptions<DataContext>());

\_repository = new TournamentTeamsRepository(\_mockContext.Object);

}

[TestMethod]

public async Task AddAsync\_ReturnsSuccess\_WhenTournamentAndTeamExist()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.Add(new Tournament { Id = 1, Name = "Test Tournament" });

context.Teams.Add(new Team { Id = 1, Name = "Test Team" });

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var tournamentTeamDTO = new TournamentTeamDTO

{

TournamentId = 1,

TeamId = 1

};

// Act

var result = await repository.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(1, result.Result.Tournament.Id);

Assert.AreEqual(1, result.Result.Team.Id);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenTournamentDoesNotExist()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Teams.Add(new Team { Id = 1, Name = "Test Team" });

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var tournamentTeamDTO = new TournamentTeamDTO

{

TournamentId = 99, // Non-existent Tournament

TeamId = 1

};

// Act

var result = await repository.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR009", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenTeamDoesNotExist()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.Tournaments.Add(new Tournament { Id = 1, Name = "Test Tournament" });

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var tournamentTeamDTO = new TournamentTeamDTO

{

TournamentId = 1,

TeamId = 99 // Non-existent Team

};

// Act

var result = await repository.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR005", result.Message);

}

[TestMethod]

public async Task GetComboAsync\_ReturnsTournamentTeams()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.TournamentTeams.AddRange(

new TournamentTeam { Id = 1, TournamentId = 1, Team = new Team { Id = 1, Name = "Team A" } },

new TournamentTeam { Id = 2, TournamentId = 1, Team = new Team { Id = 2, Name = "Team B" } }

);

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

// Act

var result = await repository.GetComboAsync(1);

// Assert

Assert.IsNotNull(result);

Assert.AreEqual(2, result.Count());

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecordCount()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.TournamentTeams.AddRange(

new TournamentTeam { Id = 1, TournamentId = 1, Team = new Team { Id = 1, Name = "Team A" } },

new TournamentTeam { Id = 2, TournamentId = 1, Team = new Team { Id = 2, Name = "Team B" } }

);

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var pagination = new PaginationDTO { Id = 1 };

// Act

var result = await repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetAsync\_WithPaginationAndFilter\_ReturnsFilteredTournamentTeams()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.TournamentTeams.AddRange(

new TournamentTeam { Id = 1, TournamentId = 1, Team = new Team { Id = 1, Name = "Team Alpha" } },

new TournamentTeam { Id = 2, TournamentId = 1, Team = new Team { Id = 2, Name = "Team Beta" } }

);

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var pagination = new PaginationDTO { Id = 1, Filter = "Alpha", Page = 1, RecordsNumber = 10 };

// Act

var result = await repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count()); // Should return only the teams that match the filter

Assert.AreEqual("Team Alpha", result.Result!.First().Team.Name);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsCorrectCount\_WhenFilterIsApplied()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

context.TournamentTeams.AddRange(

new TournamentTeam { Id = 1, TournamentId = 1, Team = new Team { Id = 1, Name = "Team Alpha" } },

new TournamentTeam { Id = 2, TournamentId = 1, Team = new Team { Id = 2, Name = "Team Beta" } },

new TournamentTeam { Id = 3, TournamentId = 1, Team = new Team { Id = 3, Name = "Team Gamma" } }

);

await context.SaveChangesAsync();

var repository = new TournamentTeamsRepository(context);

var pagination = new PaginationDTO { Id = 1, Filter = "Alpha" };

// Act

var result = await repository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only "Team Alpha" should match the filter

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForTournamentTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create related entities

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team = new Team { Id = 1, Name = "Team A" };

// Add the entities to the context

context.Tournaments.Add(tournament);

context.Teams.Add(team);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new TournamentTeamsRepository(fakeContext);

var tournamentTeamDTO = new TournamentTeamDTO

{

TournamentId = tournament.Id,

TeamId = team.Id

};

// Act

var result = await repository.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Verify that DbUpdateException is caught and handled

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForTournamentTeam()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create related entities

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team = new Team { Id = 1, Name = "Team A" };

// Add the entities to the context

context.Tournaments.Add(tournament);

context.Teams.Add(team);

await context.SaveChangesAsync();

// Use FakeDbContextWithGeneralException to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new TournamentTeamsRepository(fakeContext);

var tournamentTeamDTO = new TournamentTeamDTO

{

TournamentId = tournament.Id,

TeamId = team.Id

};

// Act

var result = await repository.AddAsync(tournamentTeamDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Verify that a general exception is caught and handled

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Grupos

### Controlador

1. Adicione la clase **GroupsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Moq;

using System.Security.Claims;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class GroupsControllerTests

{

private Mock<IGroupsUnitOfWork> \_mockGroupsUnitOfWork = null!;

private GroupsController \_controller = null!;

[TestInitialize]

public void Setup()

{

\_mockGroupsUnitOfWork = new Mock<IGroupsUnitOfWork>();

\_controller = new GroupsController(Mock.Of<IGenericUnitOfWork<Group>>(), \_mockGroupsUnitOfWork.Object);

}

[TestMethod]

public async Task GetAllAsync\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var groups = new List<Group> { new Group { Id = 1, Name = "Test Group" } };

\_mockGroupsUnitOfWork.Setup(u => u.GetAllAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = true, Result = groups });

// Act

var result = await \_controller.GetAllAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

}

[TestMethod]

public async Task GetAllAsync\_ReturnsBadRequest\_WhenWasSuccessIsFalse()

{

// Arrange

\_mockGroupsUnitOfWork.Setup(u => u.GetAllAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = false });

// Act

var result = await \_controller.GetAllAsync();

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var group = new Group { Id = 1, Name = "Test Group" };

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(1))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_controller.GetAsync(1);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(group, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsNotFound\_WhenWasSuccessIsFalse()

{

// Arrange

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(1))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = false, Message = "Group not found" });

// Act

var result = await \_controller.GetAsync(1);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode);

Assert.AreEqual("Group not found", notFoundResult.Value);

}

[TestMethod]

public async Task PostAsync\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var groupDTO = new GroupDTO { Name = "New Group" };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(

[

new Claim(ClaimTypes.Name, "testuser")

], "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

\_mockGroupsUnitOfWork.Setup(u => u.AddAsync(It.IsAny<GroupDTO>()))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = new Group { Id = 1, Name = "New Group" } });

// Act

var result = await \_controller.PostAsync(groupDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

}

[TestMethod]

public async Task PostAsync\_ReturnsBadRequest\_WhenWasSuccessIsFalse()

{

// Arrange

var groupDTO = new GroupDTO { Name = "New Group" };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, "testuser")

}, "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

\_mockGroupsUnitOfWork.Setup(u => u.AddAsync(It.IsAny<GroupDTO>()))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = false, Message = "Error occurred" });

// Act

var result = await \_controller.PostAsync(groupDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("Error occurred", badRequestResult.Value);

}

[TestMethod]

public async Task PutAsync\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var groupDTO = new GroupDTO { Id = 1, Name = "Updated Group" };

\_mockGroupsUnitOfWork.Setup(u => u.UpdateAsync(It.IsAny<GroupDTO>()))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = new Group { Id = 1, Name = "Updated Group" } });

// Act

var result = await \_controller.PutAsync(groupDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

}

[TestMethod]

public async Task PutAsync\_ReturnsBadRequest\_WhenWasSuccessIsFalse()

{

// Arrange

var groupDTO = new GroupDTO { Id = 1, Name = "Updated Group" };

\_mockGroupsUnitOfWork.Setup(u => u.UpdateAsync(It.IsAny<GroupDTO>()))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = false, Message = "Error occurred" });

// Act

var result = await \_controller.PutAsync(groupDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("Error occurred", badRequestResult.Value);

}

[TestMethod]

public async Task GetAsync\_WithCode\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var groupCode = "test-code";

var group = new Group { Id = 1, Name = "Test Group", Code = groupCode };

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(groupCode))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_controller.GetAsync(groupCode);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(group, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_WithCode\_ReturnsNotFound\_WhenWasSuccessIsFalse()

{

// Arrange

var groupCode = "test-code";

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(groupCode))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = false, Message = "Group not found" });

// Act

var result = await \_controller.GetAsync(groupCode);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode);

Assert.AreEqual("Group not found", notFoundResult.Value);

}

[TestMethod]

public async Task GetAsync\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(

[

new Claim(ClaimTypes.Name, "testuser")

], "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

var groups = new List<Group> { new() { Id = 1, Name = "Test Group" } };

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = true, Result = groups });

// Act

var result = await \_controller.GetAsync(paginationDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(groups, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenWasSuccessIsFalse()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(

[

new Claim(ClaimTypes.Name, "testuser")

], "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

\_mockGroupsUnitOfWork.Setup(u => u.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(paginationDTO);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsOk\_WhenWasSuccessIsTrue()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(

[

new Claim(ClaimTypes.Name, "testuser")

], "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

\_mockGroupsUnitOfWork.Setup(u => u.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 100 });

// Act

var result = await \_controller.GetTotalRecordsAsync(paginationDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(100, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsBadRequest\_WhenWasSuccessIsFalse()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Mocking the User.Identity.Name property

var user = new ClaimsPrincipal(new ClaimsIdentity(

[

new Claim(ClaimTypes.Name, "testuser")

], "mock"));

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = user }

};

\_mockGroupsUnitOfWork.Setup(u => u.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false });

// Act

var result = await \_controller.GetTotalRecordsAsync(paginationDTO);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_ReturnsOk\_WhenCalled()

{

// Arrange

int groupId = 1;

\_mockGroupsUnitOfWork.Setup(u => u.CheckPredictionsForAllMatchesAsync(groupId))

.Returns(Task.CompletedTask);

// Act

var result = await \_controller.CheckPredictionsForAllMatchesAsync(groupId);

// Assert

var okResult = result as OkResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

\_mockGroupsUnitOfWork.Verify(u => u.CheckPredictionsForAllMatchesAsync(groupId), Times.Once);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **GroupsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class GroupsUnitOfWorkTests

{

private Mock<IGroupsRepository> \_mockGroupsRepository = null!;

private GroupsUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockGroupsRepository = new Mock<IGroupsRepository>();

\_unitOfWork = new GroupsUnitOfWork(Mock.Of<IGenericRepository<Group>>(), \_mockGroupsRepository.Object);

}

[TestMethod]

public async Task GetAsync\_WithPagination\_ReturnsGroups()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var groups = new List<Group> { new Group { Id = 1, Name = "Test Group" } };

\_mockGroupsRepository.Setup(r => r.GetAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = true, Result = groups });

// Act

var result = await \_unitOfWork.GetAsync(paginationDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(groups, result.Result);

\_mockGroupsRepository.Verify(r => r.GetAsync(paginationDTO), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ById\_ReturnsGroup()

{

// Arrange

var group = new Group { Id = 1, Name = "Test Group" };

\_mockGroupsRepository.Setup(r => r.GetAsync(1))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(group, result.Result);

\_mockGroupsRepository.Verify(r => r.GetAsync(1), Times.Once);

}

[TestMethod]

public async Task AddAsync\_ReturnsAddedGroup()

{

// Arrange

var groupDTO = new GroupDTO { Name = "New Group" };

var group = new Group { Id = 1, Name = "New Group" };

\_mockGroupsRepository.Setup(r => r.AddAsync(groupDTO))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_unitOfWork.AddAsync(groupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(group, result.Result);

\_mockGroupsRepository.Verify(r => r.AddAsync(groupDTO), Times.Once);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecords()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

\_mockGroupsRepository.Setup(r => r.GetTotalRecordsAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 100 });

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(paginationDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(100, result.Result);

\_mockGroupsRepository.Verify(r => r.GetTotalRecordsAsync(paginationDTO), Times.Once);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsUpdatedGroup()

{

// Arrange

var groupDTO = new GroupDTO { Id = 1, Name = "Updated Group" };

var group = new Group { Id = 1, Name = "Updated Group" };

\_mockGroupsRepository.Setup(r => r.UpdateAsync(groupDTO))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_unitOfWork.UpdateAsync(groupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(group, result.Result);

\_mockGroupsRepository.Verify(r => r.UpdateAsync(groupDTO), Times.Once);

}

[TestMethod]

public async Task GetAsync\_ByCode\_ReturnsGroup()

{

// Arrange

var groupCode = "test-code";

var group = new Group { Id = 1, Name = "Test Group", Code = groupCode };

\_mockGroupsRepository.Setup(r => r.GetAsync(groupCode))

.ReturnsAsync(new ActionResponse<Group> { WasSuccess = true, Result = group });

// Act

var result = await \_unitOfWork.GetAsync(groupCode);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(group, result.Result);

\_mockGroupsRepository.Verify(r => r.GetAsync(groupCode), Times.Once);

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_CallsRepositoryMethod()

{

// Arrange

int groupId = 1;

\_mockGroupsRepository.Setup(r => r.CheckPredictionsForAllMatchesAsync(groupId))

.Returns(Task.CompletedTask);

// Act

await \_unitOfWork.CheckPredictionsForAllMatchesAsync(groupId);

// Assert

\_mockGroupsRepository.Verify(r => r.CheckPredictionsForAllMatchesAsync(groupId), Times.Once);

}

[TestMethod]

public async Task GetAllAsync\_ReturnsAllGroups()

{

// Arrange

var groups = new List<Group> { new Group { Id = 1, Name = "Test Group" } };

\_mockGroupsRepository.Setup(r => r.GetAllAsync())

.ReturnsAsync(new ActionResponse<IEnumerable<Group>> { WasSuccess = true, Result = groups });

// Act

var result = await \_unitOfWork.GetAllAsync();

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(groups, result.Result);

\_mockGroupsRepository.Verify(r => r.GetAllAsync(), Times.Once);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **GroupsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

using Match = Fantasy.Shared.Entities.Match;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class GroupsRepositoryTests

{

private DataContext \_context = null!;

private IFileStorage \_fileStorageMock = null!;

private IUsersRepository \_usersRepositoryMock = null!;

private GroupsRepository \_groupsRepository = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "GroupsTestDb")

.Options;

\_context = new DataContext(options);

\_fileStorageMock = Mock.Of<IFileStorage>();

\_usersRepositoryMock = Mock.Of<IUsersRepository>();

\_groupsRepository = new GroupsRepository(\_context, \_fileStorageMock, \_usersRepositoryMock);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task AddAsync\_ShouldAddGroupSuccessfully()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

var tournament = new Tournament { Id = 1, Name = "Test Tournament" };

var groupDTO = new GroupDTO

{

AdminId = admin.Id,

TournamentId = tournament.Id,

Name = "Test Group",

Remarks = "Test Remarks",

Image = null

};

// Add the admin to the in-memory context (this is necessary to simulate the real DB behavior)

\_context.Users.Add(admin);

\_context.Tournaments.Add(tournament);

await \_context.SaveChangesAsync();

// Mock the admin retrieval

Mock.Get(\_usersRepositoryMock)

.Setup(repo => repo.GetUserAsync(admin.Id))

.ReturnsAsync(admin);

// Verify that the admin and tournament were added correctly

var adminInDb = await \_context.Users.FirstOrDefaultAsync(x => x.Id == admin.Id);

Assert.IsNotNull(adminInDb);

var tournamentInDb = await \_context.Tournaments.FirstOrDefaultAsync(x => x.Id == tournament.Id);

Assert.IsNotNull(tournamentInDb);

// Act

var response = await \_groupsRepository.AddAsync(groupDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("Test Group", response.Result.Name);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnErrorWhenTournamentNotFound()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

var groupDTO = new GroupDTO

{

AdminId = admin.Id,

TournamentId = 999, // ID de torneo inexistente

Name = "Test Group",

Remarks = "Test Remarks",

Image = null

};

// Add the admin to the in-memory context

\_context.Users.Add(admin);

await \_context.SaveChangesAsync();

// Mock the admin retrieval

Mock.Get(\_usersRepositoryMock)

.Setup(repo => repo.GetUserAsync(admin.Id))

.ReturnsAsync(admin);

// Act

var response = await \_groupsRepository.AddAsync(groupDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("ERR009", response.Message);

Assert.IsNull(response.Result);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnErrorWhenAdminNotFound()

{

// Arrange

var groupDTO = new GroupDTO

{

AdminId = Guid.NewGuid().ToString(),

TournamentId = 1,

Name = "Test Group",

Remarks = "Test Remarks",

Image = null

};

Mock.Get(\_usersRepositoryMock)

.Setup(repo => repo.GetUserAsync(groupDTO.AdminId))

.ReturnsAsync((User)null!);

// Act

var response = await \_groupsRepository.AddAsync(groupDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("ERR013", response.Message);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnGroup\_WhenGroupExists()

{

// Arrange

// Create an admin user

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

// Create a tournament

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

// Create a group with required fields (Admin, Code, and Tournament)

var group = new Group

{

Id = 1,

Name = "Test Group",

Admin = admin, // Assign the required Admin

Code = "ABC123", // Provide a unique code for the group

IsActive = true,

Tournament = tournament, // Assign the required Tournament

Members = new List<UserGroup> { new UserGroup { User = admin } }

};

// Add the admin, tournament, and group to the in-memory database

\_context.Users.Add(admin);

\_context.Tournaments.Add(tournament);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Act

// Call the method to retrieve the group by its ID

var response = await \_groupsRepository.GetAsync(group.Id);

// Assert

// Check if the result was successful and the group ID matches

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(group.Id, response.Result!.Id);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError\_WhenGroupDoesNotExist()

{

// Arrange

var nonExistentGroupId = 999; // This ID does not exist in the in-memory database

// Act

var response = await \_groupsRepository.GetAsync(nonExistentGroupId);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("ERR001", response.Message);

Assert.IsNull(response.Result); // The result should be null since the group doesn't exist

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdateGroupSuccessfully()

{

// Arrange

// Create an admin user

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

// Create a group with the required fields (AdminId and Code)

var group = new Group

{

Id = 1,

Name = "Old Name",

Remarks = "Old Remarks",

Admin = admin, // Assign the required Admin

Code = "ABC123", // Provide a valid Code

IsActive = true

};

// Add the admin and group to the in-memory database

\_context.Users.Add(admin);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Prepare the DTO with updated values

var groupDTO = new GroupDTO

{

Id = group.Id,

Name = "New Name",

Remarks = "New Remarks",

IsActive = true

};

// Act

var response = await \_groupsRepository.UpdateAsync(groupDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual("New Name", response.Result!.Name);

Assert.AreEqual("New Remarks", response.Result.Remarks);

}

[TestMethod]

public async Task UpdateAsync\_ShouldSaveImage\_WhenImageIsProvided()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

var group = new Group

{

Id = 1,

Name = "Old Name",

Remarks = "Old Remarks",

Admin = admin,

Code = "ABC123",

IsActive = true,

Image = null // Initially, the group has no image

};

// Add the admin and group to the in-memory database

\_context.Users.Add(admin);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Mock the file storage to simulate saving the image

var imageBase64 = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }); // Example base64-encoded image

var savedImagePath = "saved-image-path.jpg"; // The path returned by the mock

Mock.Get(\_fileStorageMock)

.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "groups"))

.ReturnsAsync(savedImagePath);

var groupDTO = new GroupDTO

{

Id = group.Id,

Name = "New Name",

Remarks = "New Remarks",

IsActive = true,

Image = imageBase64 // Provide a base64-encoded image

};

// Act

var response = await \_groupsRepository.UpdateAsync(groupDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual("New Name", response.Result!.Name);

Assert.AreEqual("New Remarks", response.Result.Remarks);

Assert.AreEqual(savedImagePath, response.Result.Image); // Ensure the image path was updated

// Verify that the SaveFileAsync method was called with the correct arguments

Mock.Get(\_fileStorageMock).Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "groups"), Times.Once);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenGroupNotFound()

{

// Arrange

var groupDTO = new GroupDTO { Id = 999, Name = "New Name", Remarks = "New Remarks" };

// Act

var response = await \_groupsRepository.UpdateAsync(groupDTO);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("ERR014", response.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnCorrectCount()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "John",

LastName = "Doe"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user@example.com",

FirstName = "Jane",

LastName = "Doe"

};

\_context.Users.Add(admin);

\_context.Users.Add(user);

var group1 = new Group

{

Name = "Group 1",

Admin = admin,

Code = "ABC123", // Provide a valid code

IsActive = true,

Members = new List<UserGroup> { new() { User = user } }

};

var group2 = new Group

{

Name = "Group 2",

Admin = admin,

Code = "DEF456", // Provide a valid code

IsActive = true,

Members = new List<UserGroup> { new() { User = user } }

};

\_context.Groups.Add(group1);

\_context.Groups.Add(group2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO { Email = user.Email, Filter = "G" };

// Act

var response = await \_groupsRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result);

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_ShouldAddNewPredictions\_WhenNoPredictionsExist()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user1@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "Admin"

};

var localTeam = new Team { Id = 1, Name = "Local Team" };

var visitorTeam = new Team { Id = 2, Name = "Visitor Team" };

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

TournamentId = 1,

Admin = admin,

Members = [new UserGroup { User = user }]

};

var tournament = new Tournament

{

Id = group.TournamentId,

Name = "Test Tournament"

};

var match = new Match

{

Id = 1,

Tournament = tournament,

TournamentId = tournament.Id,

Date = DateTime.UtcNow,

IsActive = true,

Local = localTeam,

LocalId = localTeam.Id,

Visitor = visitorTeam,

VisitorId = visitorTeam.Id

};

\_context.Users.Add(user);

\_context.Users.Add(admin);

\_context.Teams.Add(localTeam);

\_context.Teams.Add(visitorTeam);

\_context.Groups.Add(group);

\_context.Tournaments.Add(tournament);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

// Act

await \_groupsRepository.CheckPredictionsForAllMatchesAsync(group.Id);

// Assert

var predictions = await \_context.Predictions.Where(p => p.GroupId == group.Id).ToListAsync();

Assert.AreEqual(1, predictions.Count); // Ensure a prediction was added

Assert.AreEqual(match.Id, predictions.First().Match.Id); // Ensure the match ID is correct

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_ShouldDoNothing\_WhenGroupDoesNotExist()

{

// Arrange

var nonExistentGroupId = 999;

// Act

await \_groupsRepository.CheckPredictionsForAllMatchesAsync(nonExistentGroupId);

// Assert

// No exception should be thrown and no predictions should be added

var predictions = await \_context.Predictions.ToListAsync();

Assert.AreEqual(0, predictions.Count);

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_ShouldDoNothing\_WhenTournamentHasNoMatches()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user1@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "Admin"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament",

Matches = [] // Tournament has no matches

};

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

TournamentId = tournament.Id,

Admin = admin,

Members = [new() { User = user }]

};

\_context.Users.Add(user);

\_context.Users.Add(admin);

\_context.Groups.Add(group);

\_context.Tournaments.Add(tournament); // Add tournament without matches

await \_context.SaveChangesAsync();

// Act

await \_groupsRepository.CheckPredictionsForAllMatchesAsync(group.Id);

// Assert

// No predictions should be added since the tournament has no matches

var predictions = await \_context.Predictions.ToListAsync();

Assert.AreEqual(0, predictions.Count);

}

[TestMethod]

public async Task CheckPredictionsForAllMatchesAsync\_ShouldReturn\_WhenTournamentDoesNotExist()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user1@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "Admin"

};

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

TournamentId = 1, // Non-existent tournament

Admin = admin,

Members = new List<UserGroup> { new UserGroup { User = user } }

};

\_context.Users.Add(user);

\_context.Users.Add(admin);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Act

await \_groupsRepository.CheckPredictionsForAllMatchesAsync(group.Id);

// Assert

// Since the tournament does not exist, no predictions should be added

var predictions = await \_context.Predictions.ToListAsync();

Assert.AreEqual(0, predictions.Count);

}

[TestMethod]

public async Task GetAllAsync\_ShouldReturnAllActiveGroups()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user1@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user2@example.com",

FirstName = "Jane",

LastName = "Doe"

};

var admin1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin1@example.com",

FirstName = "Admin",

LastName = "One"

};

var admin2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin2@example.com",

FirstName = "Admin",

LastName = "Two"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

var group1 = new Group

{

Id = 1,

Code = "000001",

Name = "Group 1",

IsActive = true,

Tournament = tournament,

Admin = admin1, // Set the required Admin

Members = [new UserGroup { User = user1 }]

};

var group2 = new Group

{

Id = 2,

Code = "000002",

Name = "Group 2",

IsActive = true,

Tournament = tournament,

Admin = admin2, // Set the required Admin

Members = [new UserGroup { User = user2 }]

};

var group3 = new Group

{

Id = 3,

Code = "000003",

Name = "Group 3",

IsActive = false, // Inactive group

Tournament = tournament,

Admin = admin1, // Set the required Admin

Members = [new UserGroup { User = user1 }]

};

\_context.Users.AddRange(user1, user2, admin1, admin2);

\_context.Tournaments.Add(tournament);

\_context.Groups.AddRange(group1, group2, group3);

await \_context.SaveChangesAsync();

// Act

var response = await \_groupsRepository.GetAllAsync();

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result!.Count()); // Only 2 active groups should be returned

Assert.IsTrue(response.Result!.Any(g => g.Name == "Group 1"));

Assert.IsTrue(response.Result!.Any(g => g.Name == "Group 2"));

Assert.IsFalse(response.Result!.Any(g => g.Name == "Group 3")); // Inactive group should not be included

}

[TestMethod]

public async Task GetAsyncByCode\_ShouldReturnGroup\_WhenGroupExists()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "User"

};

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

IsActive = true,

Admin = admin // Set the required Admin

};

\_context.Users.Add(admin);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Act

var response = await \_groupsRepository.GetAsync(group.Code);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(group.Id, response.Result.Id);

Assert.AreEqual(group.Code, response.Result.Code);

Assert.AreEqual(group.Name, response.Result.Name);

Assert.AreEqual(0, response.Result.PredictionsCount);

Assert.AreEqual(0, response.Result.MembersCount);

Assert.AreEqual("/images/NoImage.png", response.Result.ImageFull);

}

[TestMethod]

public async Task GetAsyncByCode\_ShouldReturnError\_WhenGroupDoesNotExist()

{

// Arrange

var nonExistentCode = "XYZ789"; // Code for a group that doesn't exist

// Act

var response = await \_groupsRepository.GetAsync(nonExistentCode);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("ERR001", response.Message);

Assert.IsNull(response.Result);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnGroups\_WhenUserHasGroupsWithoutFilter()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "User"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

var group1 = new Group

{

Id = 1,

Code = "000001",

Name = "Group 1",

IsActive = true,

Tournament = tournament,

Admin = admin,

Members = [new UserGroup { User = user }]

};

var group2 = new Group

{

Id = 2,

Code = "000002",

Name = "Group 2",

IsActive = true,

Tournament = tournament,

Admin = admin,

Members = [new UserGroup { User = user }]

};

\_context.Users.AddRange(user, admin);

\_context.Tournaments.Add(tournament);

\_context.Groups.AddRange(group1, group2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Email = user.Email,

Page = 1,

RecordsNumber = 10

};

// Act

var response = await \_groupsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(2, response.Result!.Count());

Assert.IsTrue(response.Result!.Any(g => g.Name == "Group 1"));

Assert.IsTrue(response.Result!.Any(g => g.Name == "Group 2"));

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredGroups\_WhenFilterIsApplied()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "User"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

var group1 = new Group

{

Id = 1,

Code = "000001",

Name = "Group 1",

IsActive = true,

Tournament = tournament,

Admin = admin,

Members = [new UserGroup { User = user }]

};

var group2 = new Group

{

Id = 2,

Code = "000002",

Name = "Another Group",

IsActive = true,

Tournament = tournament,

Admin = admin,

Members = [new UserGroup { User = user }]

};

\_context.Users.AddRange(user, admin);

\_context.Tournaments.Add(tournament);

\_context.Groups.AddRange(group1, group2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Email = user.Email,

Filter = "Group 1", // Apply filter for "Group 1"

Page = 1,

RecordsNumber = 10

};

// Act

var response = await \_groupsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(1, response.Result!.Count());

Assert.IsTrue(response.Result!.Any(g => g.Name == "Group 1"));

Assert.IsFalse(response.Result!.Any(g => g.Name == "Another Group"));

}

[TestMethod]

public async Task GetAsync\_ShouldReturnEmptyList\_WhenUserHasNoGroups()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "user@example.com",

FirstName = "John",

LastName = "Doe"

};

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "User"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

var group1 = new Group

{

Id = 1,

Code = "000001",

Name = "Group 1",

IsActive = true,

Tournament = tournament,

Admin = admin,

Members = new List<UserGroup> { new() { User = admin } } // Different user

};

\_context.Users.AddRange(user, admin);

\_context.Tournaments.Add(tournament);

\_context.Groups.Add(group1);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Email = user.Email,

Page = 1,

RecordsNumber = 10

};

// Act

var response = await \_groupsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.AreEqual(0, response.Result!.Count()); // User has no groups

}

[TestMethod]

public async Task AddAsync\_ShouldSaveImage\_WhenImageIsProvided()

{

// Arrange

var admin = new User

{

Id = Guid.NewGuid().ToString(),

Email = "admin@example.com",

FirstName = "Admin",

LastName = "User"

};

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament"

};

var groupDTO = new GroupDTO

{

AdminId = admin.Id,

TournamentId = tournament.Id,

Name = "Test Group",

Remarks = "Test Remarks",

Image = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }) // Example base64-encoded image

};

// Mock the GetUserAsync to return the admin user

Mock.Get(\_usersRepositoryMock)

.Setup(repo => repo.GetUserAsync(admin.Id))

.ReturnsAsync(admin);

// Add tournament to the in-memory context

\_context.Tournaments.Add(tournament);

await \_context.SaveChangesAsync();

// Mock the file storage to simulate saving the image

var savedImagePath = "saved-image-path.jpg"; // The path returned by the mock

Mock.Get(\_fileStorageMock)

.Setup(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "groups"))

.ReturnsAsync(savedImagePath);

// Act

var response = await \_groupsRepository.AddAsync(groupDTO);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual("Test Group", response.Result.Name);

Assert.AreEqual(savedImagePath, response.Result.Image); // Ensure the image path was saved

// Verify that SaveFileAsync was called with the correct parameters

Mock.Get(\_fileStorageMock).Verify(f => f.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "groups"), Times.Once);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var group = new Group

{

Id = 1,

Name = "Original Group",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

context.Groups.Add(group);

await context.SaveChangesAsync();

var fakeContext = new FakeDbContext(options);

var repository = new GroupsRepository(fakeContext, \_fileStorageMock, \_usersRepositoryMock);

var groupDTO = new GroupDTO { Id = 1, Name = "Updated Group" };

// Act

var result = await repository.UpdateAsync(groupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

var group = new Group { Id = 1, Name = "Original Group", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

context.Groups.Add(group);

await context.SaveChangesAsync();

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new GroupsRepository(fakeContext, \_fileStorageMock, \_usersRepositoryMock);

var groupDTO = new GroupDTO { Id = 1, Name = "Updated Group" };

// Act

var result = await repository.UpdateAsync(groupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Check for the expected exception message

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the user repository and file storage

var mockUsersRepository = new Mock<IUsersRepository>();

var mockFileStorage = new Mock<IFileStorage>();

// Mocking GetUserAsync to return a valid admin user

var adminUser = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(adminUser); // Return a valid admin user

// Adding a valid tournament to the context with required properties

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament", // Set the Name to avoid the required property issue

Remarks = "Tournament Remarks"

};

context.Tournaments.Add(tournament);

await context.SaveChangesAsync(); // Save initial tournament data

var fakeContext = new FakeDbContext(options); // Fake context to simulate DbUpdateException

var repository = new GroupsRepository(fakeContext, mockFileStorage.Object, mockUsersRepository.Object);

var groupDTO = new GroupDTO { AdminId = adminUser.Id, TournamentId = 1, Name = "Test Group" };

// Act

var result = await repository.AddAsync(groupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the user repository and file storage

var mockUsersRepository = new Mock<IUsersRepository>();

var mockFileStorage = new Mock<IFileStorage>();

// Mocking GetUserAsync to return a valid admin user

var adminUser = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(adminUser); // Return a valid admin user

// Adding a valid tournament to the context with required properties

var tournament = new Tournament

{

Id = 1,

Name = "Test Tournament", // Set the Name to avoid the required property issue

Remarks = "Tournament Remarks"

};

context.Tournaments.Add(tournament);

await context.SaveChangesAsync(); // Save initial tournament data

// Use FakeDbContextWithGeneralException to simulate general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new GroupsRepository(fakeContext, mockFileStorage.Object, mockUsersRepository.Object);

var groupDTO = new GroupDTO { AdminId = adminUser.Id, TournamentId = 1, Name = "Test Group" };

// Act

var result = await repository.AddAsync(groupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Usuarios/Grupos

### Controlador

1. Adicione la clase **UserGroupsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Moq;

using System.Security.Claims;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class UserGroupsControllerTests

{

private Mock<IUserGroupsUnitOfWork> \_userGroupsUnitOfWorkMock = null!;

private Mock<IGenericUnitOfWork<UserGroup>> \_genericUnitOfWorkMock = null!;

private UserGroupsController \_controller = null!;

[TestInitialize]

public void SetUp()

{

\_userGroupsUnitOfWorkMock = new Mock<IUserGroupsUnitOfWork>();

\_genericUnitOfWorkMock = new Mock<IGenericUnitOfWork<UserGroup>>();

\_controller = new UserGroupsController(\_genericUnitOfWorkMock.Object, \_userGroupsUnitOfWorkMock.Object);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk\_WhenPaginationSucceeds()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var groups = new List<UserGroup> { new() { Id = 1 }, new() { Id = 2 } };

var response = new ActionResponse<IEnumerable<UserGroup>> { WasSuccess = true, Result = groups };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(groups, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest\_WhenPaginationFails()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<UserGroup>> { WasSuccess = false };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnOk\_WhenTotalRecordsSucceeds()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var totalRecords = 100;

var response = new ActionResponse<int> { WasSuccess = true, Result = totalRecords };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(totalRecords, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnBadRequest\_WhenTotalRecordsFails()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = false };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_controller.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsInstanceOfType(result, typeof(BadRequestResult));

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnOk\_WhenGroupExists()

{

// Arrange

var groupId = 1;

var group = new UserGroup { Id = groupId };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = group };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(groupId)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(groupId);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(group, okResult.Value);

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnNotFound\_WhenGroupDoesNotExist()

{

// Arrange

var groupId = 1;

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Group not found" };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(groupId)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(groupId);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual("Group not found", notFoundResult.Value);

}

[TestMethod]

public async Task PostAsync\_ShouldReturnOk\_WhenGroupAddedSuccessfully()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsUnitOfWorkMock.Setup(u => u.AddAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_controller.PostAsync(userGroupDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(response.Result, okResult.Value);

}

[TestMethod]

public async Task PostAsync\_ShouldReturnBadRequest\_WhenAddFails()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Add failed" };

\_userGroupsUnitOfWorkMock.Setup(u => u.AddAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_controller.PostAsync(userGroupDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Add failed", badRequestResult.Value);

}

[TestMethod]

public async Task PutAsync\_ShouldReturnOk\_WhenGroupUpdatedSuccessfully()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsUnitOfWorkMock.Setup(u => u.UpdateAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_controller.PutAsync(userGroupDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(response.Result, okResult.Value);

}

[TestMethod]

public async Task PutAsync\_ShouldReturnNotFound\_WhenUpdateFails()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Update failed" };

\_userGroupsUnitOfWorkMock.Setup(u => u.UpdateAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_controller.PutAsync(userGroupDTO);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual("Update failed", notFoundResult.Value);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk\_WhenGroupIsFoundByEmail()

{

// Arrange

int groupId = 1;

string email = "user@example.com";

var group = new UserGroup { Id = groupId };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = group };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(groupId, email)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(groupId, email);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(group, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnNotFound\_WhenGroupIsNotFoundByEmail()

{

// Arrange

int groupId = 1;

string email = "user@example.com";

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Group not found" };

\_userGroupsUnitOfWorkMock.Setup(u => u.GetAsync(groupId, email)).ReturnsAsync(response);

// Act

var result = await \_controller.GetAsync(groupId, email);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual("Group not found", notFoundResult.Value);

}

[TestMethod]

public async Task PostAsync\_ShouldReturnOk\_WhenJoinGroupIsSuccessful()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123" };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

// Mock HttpContext for User.Identity.Name

var httpContext = new DefaultHttpContext();

httpContext.User = new ClaimsPrincipal(new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, "testUser") // Simulate logged-in user

}));

\_controller.ControllerContext = new ControllerContext()

{

HttpContext = httpContext

};

\_userGroupsUnitOfWorkMock.Setup(u => u.JoinAsync(It.Is<JoinGroupDTO>(j => j.UserName == "testUser")))

.ReturnsAsync(response);

// Act

var result = await \_controller.PostAsync(joinGroupDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(response.Result, okResult.Value);

}

[TestMethod]

public async Task PostAsync\_ShouldReturnBadRequest\_WhenJoinGroupFails()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123" };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Join group failed" };

// Mock HttpContext for User.Identity.Name

var httpContext = new DefaultHttpContext

{

User = new ClaimsPrincipal(new ClaimsIdentity(

[

new(ClaimTypes.Name, "testUser") // Simulate a logged-in user with a Name claim

]))

};

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContext

};

// Mock the JoinAsync method to simulate a failed join operation

\_userGroupsUnitOfWorkMock.Setup(u => u.JoinAsync(It.Is<JoinGroupDTO>(j => j.UserName == "testUser")))

.ReturnsAsync(response);

// Act

var result = await \_controller.PostAsync(joinGroupDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual("Join group failed", badRequestResult.Value);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **UserGroupsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class UserGroupsUnitOfWorkTests

{

private Mock<IUserGroupsRepository> \_userGroupsRepositoryMock = null!;

private Mock<IGenericRepository<UserGroup>> \_genericRepositoryMock = null!;

private UserGroupsUnitOfWork \_unitOfWork = null!;

[TestInitialize]

public void SetUp()

{

\_userGroupsRepositoryMock = new Mock<IUserGroupsRepository>();

\_genericRepositoryMock = new Mock<IGenericRepository<UserGroup>>();

\_unitOfWork = new UserGroupsUnitOfWork(\_genericRepositoryMock.Object, \_userGroupsRepositoryMock.Object);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnGroups\_WhenPaginationIsSuccessful()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var userGroups = new List<UserGroup> { new() { Id = 1 }, new UserGroup { Id = 2 } };

var response = new ActionResponse<IEnumerable<UserGroup>> { WasSuccess = true, Result = userGroups };

\_userGroupsRepositoryMock.Setup(repo => repo.GetAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(userGroups, result.Result);

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnGroup\_WhenIdExists()

{

// Arrange

var userGroup = new UserGroup { Id = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = userGroup };

\_userGroupsRepositoryMock.Setup(repo => repo.GetAsync(1)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(userGroup, result.Result);

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnError\_WhenIdDoesNotExist()

{

// Arrange

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Group not found" };

\_userGroupsRepositoryMock.Setup(repo => repo.GetAsync(1)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Group not found", result.Message);

}

[TestMethod]

public async Task AddAsync\_ShouldAddGroup\_WhenSuccessful()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsRepositoryMock.Setup(repo => repo.AddAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.AddAsync(userGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(response.Result, result.Result);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenAddFails()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Add failed" };

\_userGroupsRepositoryMock.Setup(repo => repo.AddAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.AddAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Add failed", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnTotalRecords\_WhenSuccessful()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_userGroupsRepositoryMock.Setup(repo => repo.GetTotalRecordsAsync(pagination)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(100, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdateGroup\_WhenSuccessful()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsRepositoryMock.Setup(repo => repo.UpdateAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.UpdateAsync(userGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(response.Result, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenUpdateFails()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Update failed" };

\_userGroupsRepositoryMock.Setup(repo => repo.UpdateAsync(userGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.UpdateAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Update failed", result.Message);

}

[TestMethod]

public async Task JoinAsync\_ShouldJoinGroup\_WhenSuccessful()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123", UserName = "testUser" };

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsRepositoryMock.Setup(repo => repo.JoinAsync(joinGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.JoinAsync(joinGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(response.Result, result.Result);

}

[TestMethod]

public async Task JoinAsync\_ShouldReturnError\_WhenJoinFails()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123", UserName = "testUser" };

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Join failed" };

\_userGroupsRepositoryMock.Setup(repo => repo.JoinAsync(joinGroupDTO)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.JoinAsync(joinGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Join failed", result.Message);

}

[TestMethod]

public async Task GetAsyncByGroupIdAndEmail\_ShouldReturnGroup\_WhenSuccessful()

{

// Arrange

int groupId = 1;

string email = "test@example.com";

var response = new ActionResponse<UserGroup> { WasSuccess = true, Result = new UserGroup { Id = 1 } };

\_userGroupsRepositoryMock.Setup(repo => repo.GetAsync(groupId, email)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(groupId, email);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(response.Result, result.Result);

}

[TestMethod]

public async Task GetAsyncByGroupIdAndEmail\_ShouldReturnError\_WhenGroupNotFound()

{

// Arrange

int groupId = 1;

string email = "test@example.com";

var response = new ActionResponse<UserGroup> { WasSuccess = false, Message = "Group not found" };

\_userGroupsRepositoryMock.Setup(repo => repo.GetAsync(groupId, email)).ReturnsAsync(response);

// Act

var result = await \_unitOfWork.GetAsync(groupId, email);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Group not found", result.Message);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **UserGroupsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class UserGroupsRepositoryTests

{

private DataContext \_context = null!;

private UserGroupsRepository \_userGroupsRepository = null!;

private Mock<IUsersRepository> \_usersRepositoryMock = null!;

[TestInitialize]

public void SetUp()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new DataContext(options);

\_usersRepositoryMock = new Mock<IUsersRepository>();

\_userGroupsRepository = new UserGroupsRepository(\_context, \_usersRepositoryMock.Object);

}

[TestCleanup]

public void CleanUp()

{

\_context.Database.EnsureDeleted();

}

[TestMethod]

public async Task JoinAsync\_ShouldReturnError\_WhenGroupNotFound()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123", UserName = "testUser" };

// Act

var result = await \_userGroupsRepository.JoinAsync(joinGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR017", result.Message);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredUserGroups\_WhenFilterIsApplied()

{

// Arrange

var pagination = new PaginationDTO

{

Id = 1,

Page = 1,

RecordsNumber = 10,

Filter = "john"

};

var userGroups = new List<UserGroup>

{

new() {

Id = 1,

GroupId = 1,

User = new User { FirstName = "John", LastName = "Doe" }

},

new() {

Id = 2,

GroupId = 1,

User = new User { FirstName = "Jane", LastName = "Smith" }

},

new() {

Id = 3,

GroupId = 1,

User = new User { FirstName = "Johnny", LastName = "Appleseed" }

}

};

// Add the user groups to the in-memory database

\_context.UserGroups.AddRange(userGroups);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

var resultList = result.Result!.ToList();

// Only "John Doe" and "Johnny Appleseed" should match the filter "john"

Assert.AreEqual(2, resultList.Count);

Assert.IsTrue(resultList.Any(ug => ug.User.FirstName == "John" && ug.User.LastName == "Doe"));

Assert.IsTrue(resultList.Any(ug => ug.User.FirstName == "Johnny" && ug.User.LastName == "Appleseed"));

Assert.IsFalse(resultList.Any(ug => ug.User.FirstName == "Jane" && ug.User.LastName == "Smith"));

}

[TestMethod]

public async Task JoinAsync\_ShouldReturnError\_WhenUserNotFound()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123", UserName = "testUser" };

// Creating a valid Group with required properties

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

AdminId = Guid.NewGuid().ToString(),

IsActive = true

};

// Add the group to the in-memory context

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Mock the user repository to return null (user not found)

\_usersRepositoryMock.Setup(u => u.GetUserAsync(joinGroupDTO.UserName)).ReturnsAsync((User)null!);

// Act

var result = await \_userGroupsRepository.JoinAsync(joinGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR013", result.Message);

}

[TestMethod]

public async Task JoinAsync\_ShouldAddUserGroup\_WhenSuccessful()

{

// Arrange

var joinGroupDTO = new JoinGroupDTO { Code = "ABC123", UserName = "testUser" };

// Creating a valid Group with required properties

var group = new Group

{

Id = 1,

Code = "ABC123",

Name = "Test Group",

AdminId = Guid.NewGuid().ToString(),

IsActive = true

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

UserName = "testUser",

FirstName = "John",

LastName = "Doe"

};

// Add the group to the in-memory context

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Mock the user repository to return the user

\_usersRepositoryMock.Setup(u => u.GetUserAsync(joinGroupDTO.UserName)).ReturnsAsync(user);

// Act

var result = await \_userGroupsRepository.JoinAsync(joinGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(user, result.Result.User);

Assert.AreEqual(group, result.Result.Group);

}

[TestMethod]

public async Task AddAsync\_ShouldAddUserGroup\_WhenSuccessful()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

// Creating a valid Group with required properties

var group = new Group

{

Id = 1,

Name = "Group 1",

AdminId = Guid.NewGuid().ToString(),

Code = "ABC123",

IsActive = true

};

var user = new User { Id = userGroupDTO.UserId, FirstName = "John", LastName = "Doe" };

// Add the group to the in-memory context

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

// Mock the user repository to return the user

\_usersRepositoryMock.Setup(u => u.GetUserAsync(Guid.Parse(userGroupDTO.UserId))).ReturnsAsync(user);

// Act

var result = await \_userGroupsRepository.AddAsync(userGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(group, result.Result!.Group);

Assert.AreEqual(user, result.Result.User);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenUserNotFound()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(Guid.Parse(userGroupDTO.UserId))).ReturnsAsync((User)null!);

// Act

var result = await \_userGroupsRepository.AddAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR013", result.Message);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenGroupNotFound()

{

// Arrange

var userGroupDTO = new UserGroupDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1 };

var user = new User { Id = userGroupDTO.UserId, FirstName = "John", LastName = "Doe" };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(Guid.Parse(userGroupDTO.UserId))).ReturnsAsync(user);

// Act

var result = await \_userGroupsRepository.AddAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR014", result.Message);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnPaginatedUserGroups\_WhenSuccessful()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var userGroups = new List<UserGroup>

{

new UserGroup { Id = 1, User = new User { FirstName = "John", LastName = "Doe" }, GroupId = 1 },

new UserGroup { Id = 2, User = new User { FirstName = "Jane", LastName = "Smith" }, GroupId = 1 }

};

\_context.UserGroups.AddRange(userGroups);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnUserGroup\_WhenSuccessful()

{

// Arrange

var userGroup = new UserGroup

{

Id = 1,

User = new User

{

FirstName = "John",

LastName = "Doe"

}

};

// Add the userGroup to the in-memory database

\_context.UserGroups.Add(userGroup);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(userGroup, result.Result);

}

[TestMethod]

public async Task GetAsyncById\_ShouldReturnError\_WhenUserGroupNotFound()

{

// Act

var result = await \_userGroupsRepository.GetAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnTotalRecordCount()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var userGroups = new List<UserGroup>

{

new() { Id = 1, User = new User { FirstName = "John", LastName = "Doe" }, GroupId = 1 },

new() { Id = 2, User = new User { FirstName = "Jane", LastName = "Smith" }, GroupId = 1 }

};

\_context.UserGroups.AddRange(userGroups);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnFilteredRecordCount\_WhenFilterIsApplied()

{

// Arrange

var pagination = new PaginationDTO

{

Id = 1,

Filter = "john"

};

var userGroups = new List<UserGroup>

{

new() {

Id = 1,

GroupId = 1,

User = new User { FirstName = "John", LastName = "Doe" }

},

new() {

Id = 2,

GroupId = 1,

User = new User { FirstName = "Jane", LastName = "Smith" }

},

new() {

Id = 3,

GroupId = 1,

User = new User { FirstName = "Johnny", LastName = "Appleseed" }

}

};

// Add the user groups to the in-memory database

\_context.UserGroups.AddRange(userGroups);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdateUserGroup\_WhenSuccessful()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var userGroupDTO = new UserGroupDTO { Id = 1, IsActive = false };

var userGroup = new UserGroup

{

Id = 1,

IsActive = true,

UserId = user.Id,

User = user

};

// Add the user and userGroup to the in-memory database

\_context.Users.Add(user);

\_context.UserGroups.Add(userGroup);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.UpdateAsync(userGroupDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsFalse(userGroup.IsActive); // The userGroup should now be inactive

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenUserGroupNotFound()

{

// Arrange

var userGroupDTO = new UserGroupDTO { Id = 1 };

// Act

var result = await \_userGroupsRepository.UpdateAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR015", result.Message);

}

[TestMethod]

public async Task GetAsyncByGroupIdAndEmail\_ShouldReturnUserGroup\_WhenSuccessful()

{

// Arrange

var userGroup = new UserGroup

{

Id = 1,

GroupId = 1,

User = new User

{

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

}

};

// Add the userGroup to the in-memory database

\_context.UserGroups.Add(userGroup);

await \_context.SaveChangesAsync();

// Act

var result = await \_userGroupsRepository.GetAsync(1, "test@example.com");

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(userGroup, result.Result);

}

[TestMethod]

public async Task GetAsyncByGroupIdAndEmail\_ShouldReturnError\_WhenUserGroupNotFound()

{

// Act

var result = await \_userGroupsRepository.GetAsync(1, "test@example.com");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create a UserGroup entity with required fields (UserId is required)

var userGroup = new UserGroup

{

Id = 1,

UserId = Guid.NewGuid().ToString(), // Ensure UserId is set

IsActive = true

};

context.UserGroups.Add(userGroup);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new UserGroupsRepository(fakeContext, \_usersRepositoryMock.Object);

var userGroupDTO = new UserGroupDTO

{

Id = 1,

IsActive = false // Update some value

};

// Act

var result = await repository.UpdateAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Ensure the correct error message for DbUpdateException

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create and add entities directly to the context

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var userGroup = new UserGroup { Id = 1, User = user, Group = group, IsActive = true };

context.Users.Add(user);

context.Groups.Add(group);

context.UserGroups.Add(userGroup);

await context.SaveChangesAsync();

// Use the FakeDbContextWithGeneralException to simulate an exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new UserGroupsRepository(fakeContext, \_usersRepositoryMock.Object);

var userGroupDTO = new UserGroupDTO

{

Id = 1,

IsActive = false

};

// Act

var result = await repository.UpdateAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

// Mock GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

// Add group to the context

context.Groups.Add(group);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new UserGroupsRepository(fakeContext, mockUsersRepository.Object);

var userGroupDTO = new UserGroupDTO

{

UserId = user.Id,

GroupId = group.Id

};

// Act

var result = await repository.AddAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Verify that DbUpdateException is caught and handled

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

// Mock GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

// Add group to the context

context.Groups.Add(group);

await context.SaveChangesAsync();

// Use FakeDbContextWithGeneralException to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new UserGroupsRepository(fakeContext, mockUsersRepository.Object);

var userGroupDTO = new UserGroupDTO

{

UserId = user.Id,

GroupId = group.Id

};

// Act

var result = await repository.AddAsync(userGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Verify that a general exception is caught and handled

}

[TestMethod]

public async Task JoinAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

// Mock GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(user);

// Add group to the context

context.Groups.Add(group);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new UserGroupsRepository(fakeContext, mockUsersRepository.Object);

var joinGroupDTO = new JoinGroupDTO

{

UserName = user.Id,

Code = group.Code

};

// Act

var result = await repository.JoinAsync(joinGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Verify that DbUpdateException is caught and handled

}

[TestMethod]

public async Task JoinAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForUserGroup()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

// Mock GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(user);

// Add group to the context

context.Groups.Add(group);

await context.SaveChangesAsync();

// Use FakeDbContextWithGeneralException to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new UserGroupsRepository(fakeContext, mockUsersRepository.Object);

var joinGroupDTO = new JoinGroupDTO

{

UserName = user.Id,

Code = group.Code

};

// Act

var result = await repository.JoinAsync(joinGroupDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Verify that a general exception is caught and handled

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **MatchesUnitOfWorkTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.EntityFrameworkCore;

using Moq;

using Match = Fantasy.Shared.Entities.Match;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class MatchesUnitOfWorkTests

{

private Mock<IMatchesRepository> \_matchesRepositoryMock = null!;

private MatchesUnitOfWork \_matchesUnitOfWork = null!;

[TestInitialize]

public void SetUp()

{

\_matchesRepositoryMock = new Mock<IMatchesRepository>();

\_matchesUnitOfWork = new MatchesUnitOfWork(null!, \_matchesRepositoryMock.Object);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnMatch\_WhenMatchExists()

{

// Arrange

var match = new Match { Id = 1, Local = new Team { Name = "Team A" }, Visitor = new Team { Name = "Team B" } };

\_matchesRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = true, Result = match });

// Act

var result = await \_matchesUnitOfWork.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(match.Id, result.Result!.Id);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError\_WhenMatchDoesNotExist()

{

// Arrange

\_matchesRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<int>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = false, Message = "Match not found" });

// Act

var result = await \_matchesUnitOfWork.GetAsync(1);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Match not found", result.Message);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnMatch\_WhenAddedSuccessfully()

{

// Arrange

var matchDTO = new MatchDTO { Id = 1, LocalId = 1, VisitorId = 2 };

var match = new Match { Id = 1, LocalId = 1, VisitorId = 2 };

\_matchesRepositoryMock.Setup(repo => repo.AddAsync(It.IsAny<MatchDTO>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = true, Result = match });

// Act

var result = await \_matchesUnitOfWork.AddAsync(matchDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(match.Id, result.Result!.Id);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenAddingFails()

{

// Arrange

var matchDTO = new MatchDTO { Id = 1, LocalId = 1, VisitorId = 2 };

\_matchesRepositoryMock.Setup(repo => repo.AddAsync(It.IsAny<MatchDTO>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = false, Message = "Error adding match" });

// Act

var result = await \_matchesUnitOfWork.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error adding match", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnMatch\_WhenUpdatedSuccessfully()

{

// Arrange

var matchDTO = new MatchDTO { Id = 1, LocalId = 1, VisitorId = 2 };

var match = new Match { Id = 1, LocalId = 1, VisitorId = 2 };

\_matchesRepositoryMock.Setup(repo => repo.UpdateAsync(It.IsAny<MatchDTO>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = true, Result = match });

// Act

var result = await \_matchesUnitOfWork.UpdateAsync(matchDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(match.Id, result.Result!.Id);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenUpdatingFails()

{

// Arrange

var matchDTO = new MatchDTO { Id = 1, LocalId = 1, VisitorId = 2 };

\_matchesRepositoryMock.Setup(repo => repo.UpdateAsync(It.IsAny<MatchDTO>()))

.ReturnsAsync(new ActionResponse<Match> { WasSuccess = false, Message = "Error updating match" });

// Act

var result = await \_matchesUnitOfWork.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error updating match", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnTotalRecords\_WhenSuccessful()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

\_matchesRepositoryMock.Setup(repo => repo.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 5 });

// Act

var result = await \_matchesUnitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(5, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnFilteredCount\_WhenFilterIsApplied\_InMemoryDb()

{

// Arrange: Set up the in-memory database context

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "FantasyTestDb")

.Options;

using var context = new DataContext(options);

// Create sample data

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match1 = new Match { Id = 1, TournamentId = tournament.Id, Local = team1, Visitor = team2, Date = DateTime.Now };

var match2 = new Match { Id = 2, TournamentId = tournament.Id, Local = team2, Visitor = team1, Date = DateTime.Now };

context.Tournaments.Add(tournament);

context.Teams.AddRange(team1, team2);

context.Matches.AddRange(match1, match2);

await context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = tournament.Id,

Filter = "Team A", // Applying filter for "Team A"

Page = 1,

RecordsNumber = 10

};

// Create the repository instance

var matchesRepository = new MatchesRepository(context);

// Act: Execute the method to be tested

var result = await matchesRepository.GetTotalRecordsAsync(pagination);

// Assert: Verify the result

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result); // Both matches involve "Team A"

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnError\_WhenRequestFails()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

\_matchesRepositoryMock.Setup(repo => repo.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false, Message = "Error retrieving total records" });

// Act

var result = await \_matchesUnitOfWork.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Error retrieving total records", result.Message);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnMatches\_WhenMatchesExist()

{

// Arrange

var pagination = new PaginationDTO

{

Page = 1,

RecordsNumber = 10

};

var matches = new List<Match>

{

new() { Id = 1, Local = new Team { Name = "Team A" }, Visitor = new Team { Name = "Team B" } },

new() { Id = 2, Local = new Team { Name = "Team C" }, Visitor = new Team { Name = "Team D" } }

};

\_matchesRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<Match>> { WasSuccess = true, Result = matches });

// Act

var response = await \_matchesUnitOfWork.GetAsync(pagination);

// Assert

Assert.IsTrue(response.WasSuccess);

Assert.IsNotNull(response.Result);

Assert.AreEqual(2, response.Result.Count());

Assert.AreEqual(matches, response.Result);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError\_WhenNoMatchesExist()

{

// Arrange

var pagination = new PaginationDTO

{

Page = 1,

RecordsNumber = 10

};

\_matchesRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<Match>> { WasSuccess = false, Message = "No matches found" });

// Act

var response = await \_matchesUnitOfWork.GetAsync(pagination);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual("No matches found", response.Message);

Assert.IsNull(response.Result);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **MatchesRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Enums;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Match = Fantasy.Shared.Entities.Match;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class MatchesRepositoryTests

{

private DataContext \_context = null!;

private MatchesRepository \_matchesRepository = null!;

[TestInitialize]

public void SetUp()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "FantasyTestDb")

.Options;

\_context = new DataContext(options);

\_matchesRepository = new MatchesRepository(\_context);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredMatches\_WhenFilterIsApplied\_InMemoryDb()

{

// Arrange: Set up the in-memory database context

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "FantasyTestDb")

.Options;

using var context = new DataContext(options);

// Create sample data: tournament, teams, and matches

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match1 = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team2, Date = DateTime.Now };

var match2 = new Match { Id = 2, Tournament = tournament, Local = team2, Visitor = team1, Date = DateTime.Now.AddDays(1) };

context.Tournaments.Add(tournament);

context.Teams.AddRange(team1, team2);

context.Matches.AddRange(match1, match2);

await context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = tournament.Id,

Filter = "Team A", // Applying filter for "Team A"

Page = 1,

RecordsNumber = 10

};

// Create the repository instance

var matchesRepository = new MatchesRepository(context);

// Act: Execute the method to be tested

var result = await matchesRepository.GetAsync(pagination);

// Assert: Verify the result

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count()); // Both matches involve "Team A"

Assert.IsTrue(result.Result!.All(m => m.Local.Name == "Team A" || m.Visitor.Name == "Team A")); // Matches should have "Team A"

Assert.IsTrue(result.Result!.First().Date < result.Result!.Last().Date); // Matches should be ordered by Date

}

[TestMethod]

public async Task GetAsync\_ShouldReturnAllMatches\_WhenNoFilterIsApplied()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match1 = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team2, Date = DateTime.Now };

var match2 = new Match { Id = 2, Tournament = tournament, Local = team2, Visitor = team1, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.AddRange(team1, team2);

\_context.Matches.AddRange(match1, match2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1, // Tournament Id

Page = 1,

RecordsNumber = 10

};

// Act

var result = await \_matchesRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

}

[TestMethod]

public async Task AddAsync\_ShouldAddMatch\_WhenValidDataIsProvided()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

\_context.Tournaments.Add(tournament);

\_context.Teams.AddRange(team1, team2);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

TournamentId = tournament.Id,

LocalId = team1.Id,

VisitorId = team2.Id,

Date = DateTime.Now,

IsActive = true,

DoublePoints = false

};

// Act

var result = await \_matchesRepository.AddAsync(matchDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(team1.Id, result.Result.Local.Id);

Assert.AreEqual(team2.Id, result.Result.Visitor.Id);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenLocalTeamNotFound()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team2 = new Team { Id = 2, Name = "Team B" }; // Only the visitor team is provided

\_context.Tournaments.Add(tournament);

\_context.Teams.Add(team2); // Only adding visitor team, no local team

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

TournamentId = tournament.Id,

LocalId = 999, // Invalid LocalId (local team not found)

VisitorId = team2.Id,

Date = DateTime.Now,

IsActive = true,

DoublePoints = false

};

// Act

var result = await \_matchesRepository.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR010", result.Message); // Error for missing local team

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenVisitorTeamNotFound()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" }; // Only the local team is provided

\_context.Tournaments.Add(tournament);

\_context.Teams.Add(team1); // Only adding local team, no visitor team

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

TournamentId = tournament.Id,

LocalId = team1.Id,

VisitorId = 999, // Invalid VisitorId (visitor team not found)

Date = DateTime.Now,

IsActive = true,

DoublePoints = false

};

// Act

var result = await \_matchesRepository.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR011", result.Message); // Error for missing visitor team

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenTournamentIsNotFound()

{

// Arrange

var matchDTO = new MatchDTO

{

TournamentId = 999, // Invalid TournamentId

LocalId = 1,

VisitorId = 2,

Date = DateTime.Now,

IsActive = true

};

// Act

var result = await \_matchesRepository.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR009", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdateMatch\_WhenValidDataIsProvided()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team2, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.AddRange(team1, team2);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

Id = match.Id,

TournamentId = tournament.Id,

LocalId = team1.Id,

VisitorId = team2.Id,

Date = DateTime.Now.AddDays(1),

IsActive = false,

DoublePoints = true

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(false, result.Result!.IsActive);

Assert.AreEqual(true, result.Result.DoublePoints);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenMatchIsNotFound()

{

// Arrange

var matchDTO = new MatchDTO

{

Id = 999, // Invalid MatchId

TournamentId = 1,

LocalId = 1,

VisitorId = 2,

Date = DateTime.Now

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR012", result.Message);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnTotalRecords\_WhenMatchesExist()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match1 = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team2, Date = DateTime.Now };

var match2 = new Match { Id = 2, Tournament = tournament, Local = team2, Visitor = team1, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.AddRange(team1, team2);

\_context.Matches.AddRange(match1, match2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Page = 1,

RecordsNumber = 10

};

// Act

var result = await \_matchesRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError\_WhenMatchDoesNotExist()

{

// Act

var result = await \_matchesRepository.GetAsync(999); // Non-existent match id

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenMatchNotFound()

{

// Arrange

var matchDTO = new MatchDTO

{

Id = 999, // Non-existent match

TournamentId = 1,

LocalId = 1,

VisitorId = 2,

Date = DateTime.Now

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR012", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenTournamentNotFound()

{

// Arrange

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match = new Match { Id = 1, Local = team1, Visitor = team2, Date = DateTime.Now };

\_context.Teams.AddRange(team1, team2);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

Id = match.Id,

TournamentId = 999, // Non-existent tournament

LocalId = team1.Id,

VisitorId = team2.Id,

Date = DateTime.Now

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR009", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenLocalTeamNotFound()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match = new Match { Id = 1, Tournament = tournament, Local = team2, Visitor = team2, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.Add(team2);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

Id = match.Id,

TournamentId = tournament.Id,

LocalId = 999, // Non-existent local team

VisitorId = team2.Id,

Date = DateTime.Now

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR010", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenVisitorTeamNotFound()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var match = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team1, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.Add(team1);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

Id = match.Id,

TournamentId = tournament.Id,

LocalId = team1.Id,

VisitorId = 999, // Non-existent visitor team

Date = DateTime.Now

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR011", result.Message);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnSuccess\_WhenUpdateIsValid()

{

// Arrange

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match = new Match { Id = 1, Tournament = tournament, Local = team1, Visitor = team2, Date = DateTime.Now };

\_context.Tournaments.Add(tournament);

\_context.Teams.AddRange(team1, team2);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var matchDTO = new MatchDTO

{

Id = match.Id,

TournamentId = tournament.Id,

LocalId = team1.Id,

VisitorId = team2.Id,

Date = DateTime.Now.AddDays(1),

GoalsLocal = 2,

GoalsVisitor = 1,

IsActive = false,

DoublePoints = true

};

// Act

var result = await \_matchesRepository.UpdateAsync(matchDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(false, result.Result!.IsActive);

Assert.AreEqual(true, result.Result.DoublePoints);

Assert.AreEqual(2, result.Result.GoalsLocal);

Assert.AreEqual(1, result.Result.GoalsVisitor);

}

[TestMethod]

public async Task CloseMatchAsync\_ShouldUpdatePredictions\_WhenMatchIsClosed()

{

// Arrange

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

var match = new Match { Id = 1, GoalsLocal = 2, GoalsVisitor = 1 };

var prediction1 = new Prediction

{

Id = 1,

TournamentId = 1,

GroupId = 1,

UserId = userId,

MatchId = 1,

GoalsLocal = 2,

GoalsVisitor = 1

}; // Exact prediction

var prediction2 = new Prediction

{

Id = 2,

TournamentId = 1,

GroupId = 1,

UserId = userId,

MatchId = 1,

GoalsLocal = 2,

GoalsVisitor = 0

}; // Partially correct

\_context.Matches.Add(match);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

// Act

await \_matchesRepository.CloseMatchAsync(match);

// Assert

var updatedPrediction1 = await \_context.Predictions.FindAsync(1);

var updatedPrediction2 = await \_context.Predictions.FindAsync(2);

Assert.AreEqual(10, updatedPrediction1!.Points); // Exact match should have more points

Assert.AreEqual(7, updatedPrediction2!.Points); // Partially correct prediction

}

[TestMethod]

public void CalculatePoints\_ShouldReturnCorrectPoints\_WhenPredictionMatchesExactScore()

{

// Arrange

var match = new Match { GoalsLocal = 3, GoalsVisitor = 1, DoublePoints = false };

var prediction = new Prediction { GoalsLocal = 3, GoalsVisitor = 1 };

// Act

var points = \_matchesRepository.CalculatePoints(match, prediction);

// Assert

Assert.AreEqual(10, points); // 5 points for correct outcome, 2 + 2 for exact score

}

[TestMethod]

public void CalculatePoints\_ShouldReturnHalfPoints\_WhenPredictionMatchesOutcomeOnly()

{

// Arrange

var match = new Match { GoalsLocal = 3, GoalsVisitor = 1, DoublePoints = false };

var prediction = new Prediction { GoalsLocal = 2, GoalsVisitor = 0 }; // Correct outcome but different score

// Act

var points = \_matchesRepository.CalculatePoints(match, prediction);

// Assert

Assert.AreEqual(6, points); // 5 points for correct outcome

}

[TestMethod]

public void CalculatePoints\_ShouldReturnZeroPoints\_WhenPredictionIsIncorrect()

{

// Arrange

var match = new Match { GoalsLocal = 1, GoalsVisitor = 3, DoublePoints = false };

var prediction = new Prediction { GoalsLocal = 2, GoalsVisitor = 1 }; // Completely incorrect

// Act

var points = \_matchesRepository.CalculatePoints(match, prediction);

// Assert

Assert.AreEqual(0, points); // No points for incorrect prediction

}

[TestMethod]

public void CalculatePoints\_ShouldDoublePoints\_WhenDoublePointsIsEnabled()

{

// Arrange

var match = new Match { GoalsLocal = 3, GoalsVisitor = 1, DoublePoints = true };

var prediction = new Prediction { GoalsLocal = 3, GoalsVisitor = 1 }; // Exact match

// Act

var points = \_matchesRepository.CalculatePoints(match, prediction);

// Assert

Assert.AreEqual(20, points); // 10 points doubled

}

[TestMethod]

public void GetMatchStatus\_ShouldReturnLocalWin\_WhenLocalTeamScoresMoreGoals()

{

// Arrange

var goalsLocal = 3;

var goalsVisitor = 1;

// Act

var status = \_matchesRepository.GetMatchStatus(goalsLocal, goalsVisitor);

// Assert

Assert.AreEqual(MatchStatus.LocalWin, status);

}

[TestMethod]

public void GetMatchStatus\_ShouldReturnVisitorWin\_WhenVisitorTeamScoresMoreGoals()

{

// Arrange

var goalsLocal = 1;

var goalsVisitor = 3;

// Act

var status = \_matchesRepository.GetMatchStatus(goalsLocal, goalsVisitor);

// Assert

Assert.AreEqual(MatchStatus.VisitorWin, status);

}

[TestMethod]

public void GetMatchStatus\_ShouldReturnTie\_WhenBothTeamsScoreEqualGoals()

{

// Arrange

var goalsLocal = 2;

var goalsVisitor = 2;

// Act

var status = \_matchesRepository.GetMatchStatus(goalsLocal, goalsVisitor);

// Assert

Assert.AreEqual(MatchStatus.Tie, status);

}

[TestMethod]

public void CalculatePoints\_ShouldReturnZero\_WhenGoalsInPredictionAreNull()

{

// Arrange

var match = new Match { GoalsLocal = 2, GoalsVisitor = 1, DoublePoints = false };

var prediction = new Prediction { GoalsLocal = null, GoalsVisitor = null }; // Both goals are null

// Act

var points = \_matchesRepository.CalculatePoints(match, prediction);

// Assert

Assert.AreEqual(0, points); // Should return 0 because goals in the prediction are null

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForMatch()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add related entities to ensure the UpdateAsync process doesn't fail early

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var localTeam = new Team { Id = 1, Name = "Team A" };

var visitorTeam = new Team { Id = 2, Name = "Team B" };

context.Tournaments.Add(tournament);

context.Teams.Add(localTeam);

context.Teams.Add(visitorTeam);

var match = new Match { Id = 1, Tournament = tournament, Local = localTeam, Visitor = visitorTeam, Date = DateTime.Now };

context.Matches.Add(match);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new MatchesRepository(fakeContext);

var matchDTO = new MatchDTO

{

Id = 1,

TournamentId = tournament.Id,

LocalId = localTeam.Id,

VisitorId = visitorTeam.Id,

Date = DateTime.Now.AddDays(1),

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await repository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Check for the correct error message for DbUpdateException

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForMatch()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create and add entities directly to the context (no need to re-add them later).

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var localTeam = new Team { Id = 1, Name = "Team A" };

var visitorTeam = new Team { Id = 2, Name = "Team B" };

var match = new Match

{

Id = 1,

Date = DateTime.Now,

Local = localTeam,

Visitor = visitorTeam,

Tournament = tournament

};

context.Tournaments.Add(tournament);

context.Teams.AddRange(localTeam, visitorTeam);

context.Matches.Add(match);

await context.SaveChangesAsync();

// Use the FakeDbContextWithGeneralException to simulate an exception.

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new MatchesRepository(fakeContext);

var matchDTO = new MatchDTO

{

Id = 1,

TournamentId = 1,

LocalId = 1,

VisitorId = 2,

GoalsLocal = 2,

GoalsVisitor = 1,

Date = DateTime.Now,

IsActive = true,

DoublePoints = false

};

// Act

var result = await repository.UpdateAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForMatch()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add related entities to ensure the AddAsync process doesn't fail early

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var localTeam = new Team { Id = 1, Name = "Team A" };

var visitorTeam = new Team { Id = 2, Name = "Team B" };

context.Tournaments.Add(tournament);

context.Teams.Add(localTeam);

context.Teams.Add(visitorTeam);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new MatchesRepository(fakeContext);

var matchDTO = new MatchDTO

{

TournamentId = tournament.Id,

LocalId = localTeam.Id,

VisitorId = visitorTeam.Id,

Date = DateTime.Now.AddDays(1),

DoublePoints = true

};

// Act

var result = await repository.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Check for the correct error message for DbUpdateException

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForMatch()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Add related entities to ensure the AddAsync process doesn't fail early

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var localTeam = new Team { Id = 1, Name = "Team A" };

var visitorTeam = new Team { Id = 2, Name = "Team B" };

context.Tournaments.Add(tournament);

context.Teams.Add(localTeam);

context.Teams.Add(visitorTeam);

await context.SaveChangesAsync();

// Use FakeDbContextWithGeneralException to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new MatchesRepository(fakeContext);

var matchDTO = new MatchDTO

{

TournamentId = tournament.Id,

LocalId = localTeam.Id,

VisitorId = visitorTeam.Id,

Date = DateTime.Now.AddDays(1),

DoublePoints = true

};

// Act

var result = await repository.AddAsync(matchDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Check for the correct error message for general exception

}

[TestMethod]

public async Task GetAsync\_ReturnsFilteredMatches\_WhenFilterIsApplied()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create and add some test data

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var localTeam1 = new Team { Id = 1, Name = "Team A" };

var visitorTeam1 = new Team { Id = 2, Name = "Team B" };

var localTeam2 = new Team { Id = 3, Name = "Team C" };

var visitorTeam2 = new Team { Id = 4, Name = "Team D" };

context.Tournaments.Add(tournament);

context.Teams.AddRange(localTeam1, visitorTeam1, localTeam2, visitorTeam2);

var match1 = new Match

{

Id = 1,

TournamentId = 1,

Local = localTeam1,

Visitor = visitorTeam1,

Date = DateTime.Now.AddDays(1),

IsActive = true

};

var match2 = new Match

{

Id = 2,

TournamentId = 1,

Local = localTeam2,

Visitor = visitorTeam2,

Date = DateTime.Now.AddDays(2),

IsActive = true

};

context.Matches.AddRange(match1, match2);

await context.SaveChangesAsync();

var repository = new MatchesRepository(context);

// Create a PaginationDTO with a filter that matches "Team A" (local) or "Team B" (visitor)

var pagination = new PaginationDTO

{

Id = 1, // Tournament ID to filter by

Filter = "Team A",

Page = 1,

RecordsNumber = 10

};

// Act

var result = await repository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count()); // Only match1 should match the filter

Assert.AreEqual("Team A", result.Result!.First().Local.Name); // Ensure match1 is returned

}

[TestMethod]

public async Task GetAsync\_ShouldReturnMatch\_WhenMatchExists()

{

// Create a unique in-memory database for this test

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString()) // Use a unique database name

.Options;

// Create the data context

using var context = new DataContext(options);

var matchesRepository = new MatchesRepository(context);

// Arrange: Set up the data

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var team1 = new Team { Id = 1, Name = "Team A" };

var team2 = new Team { Id = 2, Name = "Team B" };

var match = new Match

{

Id = 1,

Tournament = tournament,

Local = team1,

Visitor = team2,

Date = DateTime.Now

};

// Add the data to the context and save changes

context.Tournaments.Add(tournament);

context.Teams.AddRange(team1, team2);

context.Matches.Add(match);

await context.SaveChangesAsync();

// Act: Execute the method being tested

var result = await matchesRepository.GetAsync(match.Id);

// Assert: Verify the result

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(match.Id, result.Result!.Id);

Assert.AreEqual(match.Local.Name, result.Result.Local.Name);

Assert.AreEqual(match.Visitor.Name, result.Result.Visitor.Name);

Assert.AreEqual(0, result.Result.PredictionsCount);

Assert.AreEqual(match.Date.ToLocalTime(), result.Result.DateLocal);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Predicciones

### Controlador

1. Adicione la clase **PredictionsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Moq;

using System.Security.Claims;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class PredictionsControllerTests

{

private Mock<IPredictionsUnitOfWork> \_predictionsUnitOfWorkMock = null!;

private PredictionsController \_predictionsController = null!;

private Mock<ClaimsPrincipal> \_mockUser = null!;

[TestInitialize]

public void SetUp()

{

// Initialize the mock for IPredictionsUnitOfWork

\_predictionsUnitOfWorkMock = new Mock<IPredictionsUnitOfWork>();

// Mock the User Identity

\_mockUser = new Mock<ClaimsPrincipal>();

\_mockUser.Setup(u => u.Identity!.Name).Returns("testuser@example.com");

\_mockUser.Setup(u => u.Identity!.IsAuthenticated).Returns(true); // Make sure user is authenticated

// Create the controller with the mocked unit of work

\_predictionsController = new PredictionsController(

new Mock<IGenericUnitOfWork<Prediction>>().Object,

\_predictionsUnitOfWorkMock.Object)

{

ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = \_mockUser.Object }

}

};

}

[TestMethod]

public async Task GetBalanceAsync\_ShouldReturnOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = new List<Prediction>() };

\_predictionsUnitOfWorkMock.Setup(u => u.GetBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetBalanceAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<Prediction>));

}

[TestMethod]

public async Task GetBalanceAsync\_ShouldReturnBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetBalanceAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetTotalRecordsBalanceAsync\_ShouldReturnOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 10 };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsBalanceAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(10, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsBalanceAsync\_ShouldReturnBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsBalanceAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetAllPredictionsAsync\_ShouldReturnOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = new List<Prediction>() };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAllPredictionsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<Prediction>));

}

[TestMethod]

public async Task GetAllPredictionsAsync\_ShouldReturnBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAllPredictionsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetTotalRecordsAllPredictionsAsync\_ShouldReturnOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 50 };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsAllPredictionsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(50, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsAllPredictionsAsync\_ShouldReturnBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsAllPredictionsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetTotalRecordsForPositionsAsync\_ShouldReturnOk\_WhenSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 30 };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsForPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsForPositionsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(30, okResult.Value);

}

[TestMethod]

public async Task GetTotalRecordsForPositionsAsync\_ShouldReturnBadRequest\_WhenFailed()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsForPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsForPositionsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnOk\_WhenPredictionIsFound()

{

// Arrange

var prediction = new Prediction { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = prediction };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAsync(It.IsAny<int>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAsync(1);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(prediction, okResult.Value);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnNotFound\_WhenPredictionIsNotFound()

{

// Arrange

var response = new ActionResponse<Prediction> { WasSuccess = false, Message = "Not Found" };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAsync(It.IsAny<int>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAsync(1);

// Assert

var notFoundResult = result as NotFoundObjectResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode);

Assert.AreEqual("Not Found", notFoundResult.Value);

}

[TestMethod]

public async Task PutAsync\_ShouldReturnOk\_WhenUpdateIsSuccessful()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = new Prediction() };

\_predictionsUnitOfWorkMock.Setup(u => u.UpdateAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.PutAsync(predictionDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.IsInstanceOfType(okResult.Value, typeof(Prediction));

}

[TestMethod]

public async Task PutAsync\_ShouldReturnBadRequest\_WhenUpdateFails()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = false, Message = "Error" };

\_predictionsUnitOfWorkMock.Setup(u => u.UpdateAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.PutAsync(predictionDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("Error", badRequestResult.Value);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnOk\_WhenResponseIsSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPredictions = new List<Prediction> { new Prediction { Id = 1 }, new Prediction { Id = 2 } };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = mockPredictions };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult); // Ensure that the result is OkObjectResult

Assert.AreEqual(200, okResult.StatusCode); // Ensure that the status code is 200 (OK)

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<Prediction>)); // Ensure the value is of the correct type

Assert.AreEqual(mockPredictions, okResult.Value); // Ensure that the returned value matches the mocked result

}

[TestMethod]

public async Task GetAsync\_ShouldReturnBadRequest\_WhenResponseIsFailure()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult); // Ensure that the result is BadRequestResult

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure that the status code is 400 (Bad Request)

}

[TestMethod]

public async Task GetAsync\_ShouldSetPaginationEmailToUserIdentityName()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = new List<Prediction>() };

\_predictionsUnitOfWorkMock.Setup(u => u.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

await \_predictionsController.GetAsync(pagination);

// Assert

\_predictionsUnitOfWorkMock.Verify(u => u.GetAsync(It.Is<PaginationDTO>(p => p.Email == "testuser@example.com")), Times.Once);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnOk\_WhenResponseIsSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult); // Ensure that the result is OkObjectResult

Assert.AreEqual(200, okResult.StatusCode); // Ensure that the status code is 200 (OK)

Assert.AreEqual(100, okResult.Value); // Ensure the returned value matches the mocked result

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnBadRequest\_WhenResponseIsFailure()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetTotalRecordsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult); // Ensure that the result is BadRequestResult

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure that the status code is 400 (Bad Request)

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldSetPaginationEmailToUserIdentityName()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_predictionsUnitOfWorkMock.Setup(u => u.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

await \_predictionsController.GetTotalRecordsAsync(pagination);

// Assert

\_predictionsUnitOfWorkMock.Verify(u => u.GetTotalRecordsAsync(It.Is<PaginationDTO>(p => p.Email == "testuser@example.com")), Times.Once);

}

[TestMethod]

public async Task PostAsync\_ShouldReturnOk\_WhenPredictionIsAddedSuccessfully()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var mockPrediction = new Prediction { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = mockPrediction };

\_predictionsUnitOfWorkMock.Setup(u => u.AddAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.PostAsync(predictionDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult); // Ensure that the result is OkObjectResult

Assert.AreEqual(200, okResult.StatusCode); // Ensure that the status code is 200 (OK)

Assert.IsInstanceOfType(okResult.Value, typeof(Prediction)); // Ensure the value is of the correct type

Assert.AreEqual(mockPrediction, okResult.Value); // Ensure that the returned value matches the mocked result

}

[TestMethod]

public async Task PostAsync\_ShouldReturnBadRequest\_WhenPredictionAdditionFails()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = false, Message = "Error adding prediction" };

\_predictionsUnitOfWorkMock.Setup(u => u.AddAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.PostAsync(predictionDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult); // Ensure that the result is BadRequestObjectResult

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure that the status code is 400 (Bad Request)

Assert.AreEqual("Error adding prediction", badRequestResult.Value); // Ensure the message matches the expected error message

}

[TestMethod]

public async Task PostAsync\_ShouldCallAddAsyncWithCorrectPredictionDTO()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = new Prediction { Id = 1 } };

\_predictionsUnitOfWorkMock.Setup(u => u.AddAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

await \_predictionsController.PostAsync(predictionDTO);

// Assert

\_predictionsUnitOfWorkMock.Verify(u => u.AddAsync(It.Is<PredictionDTO>(p => p.Id == 1)), Times.Once);

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldReturnOk\_WhenResponseIsSuccess()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPositions = new List<PositionDTO> { new() { User = new User(), Points = 20 }, new PositionDTO { User = new User(), Points = 10 } }; // Mocked list of positions

var response = new ActionResponse<IEnumerable<PositionDTO>> { WasSuccess = true, Result = mockPositions };

\_predictionsUnitOfWorkMock.Setup(u => u.GetPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetPositionsAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult); // Ensure that the result is OkObjectResult

Assert.AreEqual(200, okResult.StatusCode); // Ensure that the status code is 200 (OK)

Assert.IsInstanceOfType(okResult.Value, typeof(IEnumerable<PositionDTO>)); // Ensure the value is of the correct type

Assert.AreEqual(mockPositions, okResult.Value); // Ensure that the returned value matches the mocked result

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldReturnBadRequest\_WhenResponseIsFailure()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<PositionDTO>> { WasSuccess = false };

\_predictionsUnitOfWorkMock.Setup(u => u.GetPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsController.GetPositionsAsync(pagination);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult); // Ensure that the result is BadRequestResult

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure that the status code is 400 (Bad Request)

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldCallGetPositionsAsyncWithCorrectPaginationDTO()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var response = new ActionResponse<IEnumerable<PositionDTO>> { WasSuccess = true, Result = new List<PositionDTO>() };

\_predictionsUnitOfWorkMock.Setup(u => u.GetPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

await \_predictionsController.GetPositionsAsync(pagination);

// Assert

\_predictionsUnitOfWorkMock.Verify(u => u.GetPositionsAsync(It.Is<PaginationDTO>(p => p.Id == 1)), Times.Once);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **PredictionsUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Moq;

namespace Fantasy.Tests.UnitsOfWork

{

[TestClass]

public class PredictionsUnitOfWorkTests

{

private Mock<IPredictionsRepository> \_predictionsRepositoryMock = null!;

private PredictionsUnitOfWork \_predictionsUnitOfWork = null!;

[TestInitialize]

public void SetUp()

{

// Initialize the mock for IPredictionsRepository

\_predictionsRepositoryMock = new Mock<IPredictionsRepository>();

// Initialize the unit of work with the mocked repository

\_predictionsUnitOfWork = new PredictionsUnitOfWork(

new Mock<IGenericRepository<Prediction>>().Object,

\_predictionsRepositoryMock.Object);

}

[TestMethod]

public async Task GetAsync\_ByPagination\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPredictions = new List<Prediction> { new() { Id = 1 }, new() { Id = 2 } };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = mockPredictions };

\_predictionsRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPredictions, result.Result);

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnCorrectResponse()

{

// Arrange

var mockPrediction = new Prediction { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = mockPrediction };

\_predictionsRepositoryMock.Setup(repo => repo.GetAsync(It.IsAny<int>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetAsync(1);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPrediction, result.Result);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var mockPrediction = new Prediction { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = mockPrediction };

\_predictionsRepositoryMock.Setup(repo => repo.AddAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.AddAsync(predictionDTO);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPrediction, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var paginationDTO = new PaginationDTO { Id = 1 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 100 };

\_predictionsRepositoryMock.Setup(repo => repo.GetTotalRecordsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetTotalRecordsAsync(paginationDTO);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(100, result.Result);

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var predictionDTO = new PredictionDTO { Id = 1 };

var mockPrediction = new Prediction { Id = 1 };

var response = new ActionResponse<Prediction> { WasSuccess = true, Result = mockPrediction };

\_predictionsRepositoryMock.Setup(repo => repo.UpdateAsync(It.IsAny<PredictionDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.UpdateAsync(predictionDTO);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPrediction, result.Result);

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPositions = new List<PositionDTO> { new() { User = new User(), Points = 20 }, new() { User = new User(), Points = 10 } };

var response = new ActionResponse<IEnumerable<PositionDTO>> { WasSuccess = true, Result = mockPositions };

\_predictionsRepositoryMock.Setup(repo => repo.GetPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetPositionsAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPositions, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsForPositionsAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 50 };

\_predictionsRepositoryMock.Setup(repo => repo.GetTotalRecordsForPositionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetTotalRecordsForPositionsAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(50, result.Result);

}

[TestMethod]

public async Task GetAllPredictionsAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPredictions = new List<Prediction> { new Prediction { Id = 1 }, new Prediction { Id = 2 } };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = mockPredictions };

\_predictionsRepositoryMock.Setup(repo => repo.GetAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetAllPredictionsAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPredictions, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAllPredictionsAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 80 };

\_predictionsRepositoryMock.Setup(repo => repo.GetTotalRecordsAllPredictionsAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetTotalRecordsAllPredictionsAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(80, result.Result);

}

[TestMethod]

public async Task GetBalanceAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1, Page = 1, RecordsNumber = 10 };

var mockPredictions = new List<Prediction> { new Prediction { Id = 1 }, new Prediction { Id = 2 } };

var response = new ActionResponse<IEnumerable<Prediction>> { WasSuccess = true, Result = mockPredictions };

\_predictionsRepositoryMock.Setup(repo => repo.GetBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetBalanceAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(mockPredictions, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsBalanceAsync\_ShouldReturnCorrectResponse()

{

// Arrange

var pagination = new PaginationDTO { Id = 1 };

var response = new ActionResponse<int> { WasSuccess = true, Result = 60 };

\_predictionsRepositoryMock.Setup(repo => repo.GetTotalRecordsBalanceAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(response);

// Act

var result = await \_predictionsUnitOfWork.GetTotalRecordsBalanceAsync(pagination);

// Assert

Assert.IsNotNull(result);

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(60, result.Result);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Modificamos el **PredictionsRepository**:

public virtual bool CanWatch(Prediction prediction)

1. En **Fantasy.Tests.General** creamos el **TestablePredictionsRepository**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.Entities;

namespace Fantasy.Tests.General

{

public class TestablePredictionsRepository : PredictionsRepository

{

private readonly bool \_canWatchResult;

public TestablePredictionsRepository(DataContext context, IUsersRepository usersRepository, bool canWatchResult)

: base(context, usersRepository)

{

\_canWatchResult = canWatchResult;

}

public override bool CanWatch(Prediction prediction)

{

return \_canWatchResult;

}

}

}

1. Adicione la clase **PredictionsRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Tests.General;

using Microsoft.EntityFrameworkCore;

using Moq;

using Match = Fantasy.Shared.Entities.Match;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class PredictionsRepositoryTests

{

private DataContext \_context = null!;

private PredictionsRepository \_predictionsRepository = null!;

private Mock<IUsersRepository> \_usersRepositoryMock = null!;

[TestInitialize]

public void SetUp()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "PredictionsTestDb")

.Options;

\_context = new DataContext(options);

\_usersRepositoryMock = new Mock<IUsersRepository>();

\_predictionsRepository = new PredictionsRepository(\_context, \_usersRepositoryMock.Object);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

[TestMethod]

public async Task GetAsync\_ByPagination\_ShouldReturnFilteredPredictions()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match1 = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now.AddDays(1)

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match1

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Predictions.Add(prediction1);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com",

Page = 1,

RecordsNumber = 10

};

// Act

var result = await \_predictionsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetAsync\_ById\_ShouldReturnPrediction\_WhenExists()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

// Act

var result = await \_predictionsRepository.GetAsync(1);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Id);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenUserNotFound()

{

// Arrange

var predictionDTO = new PredictionDTO { UserId = Guid.NewGuid().ToString(), GroupId = 1, MatchId = 1 };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(It.IsAny<Guid>())).ReturnsAsync((User)null!);

// Act

var result = await \_predictionsRepository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR013", result.Message); // Error for user not found

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenGroupNotFound()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(It.IsAny<Guid>())).ReturnsAsync(user);

var predictionDTO = new PredictionDTO { UserId = user.Id.ToString(), GroupId = 999, MatchId = 1 };

// Act

var result = await \_predictionsRepository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR014", result.Message); // Error for group not found

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenPredictionIsLocked()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await \_predictionsRepository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR018", result.Message); // Error for locked prediction

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnCorrectCount()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com",

Page = 1,

RecordsNumber = 10

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldReturnCorrectPositions()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Points = 10

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Page = 1,

RecordsNumber = 10

};

// Act

var result = await \_predictionsRepository.GetPositionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

Assert.AreEqual(10, result.Result!.First().Points);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnFilteredPredictions\_WhenFilterIsApplied()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match1 = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now.AddDays(1)

};

var match2 = new Match

{

Id = 2,

Local = new Team { Name = "Team C" },

Visitor = new Team { Name = "Team D" },

Date = DateTime.Now.AddDays(2)

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match1

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user,

Match = match2

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.AddRange(match1, match2);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com",

Page = 1,

RecordsNumber = 10,

Filter = "Team A"

};

// Act

var result = await \_predictionsRepository.GetAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

Assert.AreEqual("Team A", result.Result!.First().Match.Local.Name);

}

[TestMethod]

public async Task GetAsync\_ShouldReturnError\_WhenPredictionIsNull()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

await \_context.SaveChangesAsync();

// Act

var result = await \_predictionsRepository.GetAsync(999);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR001", result.Message);

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenTournamentIsNotFound()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(It.IsAny<Guid>())).ReturnsAsync(user);

\_context.Groups.Add(group);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

UserId = user.Id.ToString(),

GroupId = 1,

TournamentId = 999, // Invalid TournamentId

MatchId = 1

};

// Act

var result = await \_predictionsRepository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR009", result.Message); // Error for missing tournament

}

[TestMethod]

public async Task AddAsync\_ShouldReturnError\_WhenMatchIsNotFound()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(It.IsAny<Guid>())).ReturnsAsync(user);

\_context.Groups.Add(group);

\_context.Tournaments.Add(tournament);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

UserId = user.Id.ToString(),

GroupId = 1,

TournamentId = 1, // Valid TournamentId

MatchId = 999 // Invalid MatchId

};

// Act

var result = await \_predictionsRepository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR012", result.Message); // Error for missing match

}

[TestMethod]

public async Task AddAsync\_ShouldCreatePrediction\_WhenAllDataIsValid()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

var match = new Match { Id = 1, Local = new Team { Name = "Team A" }, Visitor = new Team { Name = "Team B" }, Date = DateTime.Now };

\_usersRepositoryMock.Setup(u => u.GetUserAsync(It.IsAny<Guid>())).ReturnsAsync(user);

\_context.Groups.Add(group);

\_context.Tournaments.Add(tournament);

\_context.Matches.Add(match);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

UserId = user.Id.ToString(),

GroupId = 1,

TournamentId = 1,

MatchId = 1,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await \_predictionsRepository.AddAsync(predictionDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.IsNotNull(result.Result);

Assert.AreEqual(2, result.Result.GoalsLocal);

Assert.AreEqual(1, result.Result.GoalsVisitor);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ShouldReturnCorrectCount\_WhenFilterIsApplied()

{

// Arrange

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match1 = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var match2 = new Match

{

Id = 2,

Local = new Team { Name = "Team C" },

Visitor = new Team { Name = "Team D" },

Date = DateTime.Now

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match1

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user,

Match = match2

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.AddRange(match1, match2);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com",

Filter = "Team A" // Applying filter for "Team A"

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only one prediction should match the filter

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenPredictionIsNotFound()

{

// Arrange

var predictionDTO = new PredictionDTO

{

Id = 999, // Invalid Id

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await \_predictionsRepository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR016", result.Message); // Error for missing prediction

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenMatchHasGoalsAlreadySet()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // FirstName is required

LastName = "Doe" // LastName is required

};

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1 // Goals already set

};

var prediction = new Prediction

{

Id = 1,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await \_predictionsRepository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR018", result.Message); // Error for prediction being locked

}

[TestMethod]

public async Task UpdateAsync\_ShouldReturnError\_WhenCanWatchReturnsTrue()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var match = new Match

{

Id = 1,

GoalsLocal = null,

GoalsVisitor = null

};

var prediction = new Prediction

{

Id = 1,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Create a repository with CanWatch returning true

var testRepository = new TestablePredictionsRepository(\_context, \_usersRepositoryMock.Object, true);

// Act

var result = await testRepository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR018", result.Message); // Error for CanWatch returning true

}

[TestMethod]

public async Task UpdateAsync\_ShouldUpdatePrediction\_WhenDataIsValid()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // FirstName is required

LastName = "Doe" // LastName is required

};

var match = new Match

{

Id = 1,

GoalsLocal = null,

GoalsVisitor = null // No goals set

};

var prediction = new Prediction

{

Id = 1,

User = user,

Match = match

};

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Points = 5

};

// Create a repository with CanWatch returning false

var testRepository = new TestablePredictionsRepository(\_context, \_usersRepositoryMock.Object, false);

// Act

var result = await testRepository.UpdateAsync(predictionDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.GoalsLocal);

Assert.AreEqual(1, result.Result!.GoalsVisitor);

Assert.AreEqual(5, result.Result!.Points);

}

[TestMethod]

public async Task GetPositionsAsync\_ShouldReturnFilteredPositions\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Points = 10

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Points = 5

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Page = 1,

RecordsNumber = 10,

Filter = "John" // Applying filter for "John"

};

// Act

var result = await \_predictionsRepository.GetPositionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count()); // Only one user should match the filter

Assert.AreEqual("John", result.Result!.First().User.FirstName); // Ensure the filtered result is correct

}

[TestMethod]

public async Task GetTotalRecordsForPositions2Async\_ShouldReturnCorrectCount\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Points = 10

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Points = 5

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Filter = "John" // Applying filter

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsForPositionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only one user should match the filter

}

[TestMethod]

public async Task GetTotalRecordsForPositionsAsync\_ShouldReturnCorrectCount\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Points = 10

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Points = 5

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Filter = "John" // Applying filter

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsForPositionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only one user should match the filter

}

[TestMethod]

public async Task GetAllPredictionsAsync\_ShouldReturnAllPredictions\_WhenNoFilterIsApplied()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.Add(match);

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Id2 = 1

};

// Act

var result = await \_predictionsRepository.GetAllPredictionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetAllPredictionsAsync\_ShouldReturnFilteredPredictions\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match

{

Id = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Match = match

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Matches.Add(match);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Id2 = 1,

Filter = "John"

};

// Act

var result = await \_predictionsRepository.GetAllPredictionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count()); // Only one user should match the filter

Assert.AreEqual("John", result.Result!.First().User.FirstName);

}

[TestMethod]

public async Task GetTotalRecordsAllPredictionsAsync\_ShouldReturnCorrectCount\_WhenNoFilterIsApplied()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Providing required FirstName

LastName = "Doe" // Providing required LastName

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match { Id = 1 };

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.Add(match);

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Id2 = 1

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsAllPredictionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsAllPredictionsAsync\_ShouldReturnFilteredCount\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match { Id = 1 };

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Match = match

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Matches.Add(match);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Id2 = 1,

Filter = "John"

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsAllPredictionsAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only one prediction should match the filter

}

[TestMethod]

public async Task GetBalanceAsync\_ShouldReturnAllPredictions\_WhenNoFilterIsApplied()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John",

LastName = "Doe"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.Add(match);

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com"

};

// Act

var result = await \_predictionsRepository.GetBalanceAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count());

}

[TestMethod]

public async Task GetBalanceAsync\_ShouldReturnFilteredPredictions\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Local = new Team { Name = "Team A" },

Visitor = new Team { Name = "Team B" },

Date = DateTime.Now

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Match = match

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Matches.Add(match);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "john@example.com",

Filter = "Team A"

};

// Act

var result = await \_predictionsRepository.GetBalanceAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result!.Count()); // Only one match should match the filter

Assert.AreEqual("Team A", result.Result!.First().Match.Local.Name);

}

[TestMethod]

public async Task GetTotalRecordsBalanceAsync\_ShouldReturnCorrectCount\_WhenNoFilterIsApplied()

{

// Arrange

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Providing required FirstName

LastName = "Doe" // Providing required LastName

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(), // Providing required AdminId

Code = "GRP123" // Providing required Code

};

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1

};

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.Add(user);

\_context.Matches.Add(match);

\_context.Predictions.Add(prediction);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "test@example.com"

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsBalanceAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GetTotalRecordsBalanceAsync\_ShouldReturnFilteredCount\_WhenFilterIsApplied()

{

// Arrange

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "john@example.com",

FirstName = "John",

LastName = "Doe"

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

Email = "jane@example.com",

FirstName = "Jane",

LastName = "Smith"

};

var group = new Group

{

Id = 1,

Name = "Group A",

AdminId = Guid.NewGuid().ToString(),

Code = "GRP123"

};

// Set up the teams and the match

var teamA = new Team { Id = 1, Name = "Team A" };

var teamB = new Team { Id = 2, Name = "Team B" };

var match = new Match

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Local = teamA, // Ensure Local team matches the filter "Team A"

Visitor = teamB

};

var prediction1 = new Prediction

{

Id = 1,

Group = group,

User = user1,

Match = match

};

var prediction2 = new Prediction

{

Id = 2,

Group = group,

User = user2,

Match = match

};

\_context.Groups.Add(group);

\_context.Users.AddRange(user1, user2);

\_context.Teams.AddRange(teamA, teamB); // Add the teams to the context

\_context.Matches.Add(match);

\_context.Predictions.AddRange(prediction1, prediction2);

await \_context.SaveChangesAsync();

var pagination = new PaginationDTO

{

Id = 1,

Email = "john@example.com",

Filter = "Team A" // Ensure the filter matches the team name

};

// Act

var result = await \_predictionsRepository.GetTotalRecordsBalanceAsync(pagination);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result); // Only one prediction should match the filter

}

[TestMethod]

public void CanWatch\_ShouldReturnTrue\_WhenGoalsAreSet()

{

// Arrange

var match = new Match

{

GoalsLocal = 2,

GoalsVisitor = 1

};

var prediction = new Prediction

{

Match = match

};

// Act

var result = \_predictionsRepository.CanWatch(prediction);

// Assert

Assert.IsTrue(result); // Goals are set, so the match is completed, should return true.

}

[TestMethod]

public void CanWatch\_ShouldReturnTrue\_WhenMatchIsAboutToStart()

{

// Arrange

var match = new Match

{

Date = DateTime.Now.AddMinutes(5) // Match starting in 5 minutes

};

var prediction = new Prediction

{

Match = match

};

// Act

var result = \_predictionsRepository.CanWatch(prediction);

// Assert

Assert.IsTrue(result); // Match is starting within 10 minutes, should return true.

}

[TestMethod]

public void CanWatch\_ShouldReturnFalse\_WhenMatchIsMoreThan10MinutesAway()

{

// Arrange

var match = new Match

{

Date = DateTime.Now.AddMinutes(15) // Match starting in 15 minutes

};

var prediction = new Prediction

{

Match = match

};

// Act

var result = \_predictionsRepository.CanWatch(prediction);

// Assert

Assert.IsFalse(result); // Match is more than 10 minutes away, should return false.

}

[TestMethod]

public void CanWatch\_ShouldReturnTrue\_WhenMatchHasStarted()

{

// Arrange

var match = new Match

{

Date = DateTime.Now.AddMinutes(-5) // Match started 5 minutes ago

};

var prediction = new Prediction

{

Match = match

};

// Act

var result = \_predictionsRepository.CanWatch(prediction);

// Assert

Assert.IsTrue(result); // Match has already started, should return true.

}

[TestMethod]

public void CanWatch\_ShouldReturnTrue\_WhenMatchStartedMoreThan10MinutesAgo()

{

// Arrange

var match = new Match

{

Date = DateTime.Now.AddMinutes(-15) // Match started 15 minutes ago

};

var prediction = new Prediction

{

Match = match

};

// Act

var result = \_predictionsRepository.CanWatch(prediction);

// Assert

Assert.IsTrue(result); // Match was more than 10 minutes ago, but the current logic allows watching even if the match has started.

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForPrediction()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create the required user entity with FirstName and LastName properties

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Ensure FirstName is set

LastName = "Doe" // Ensure LastName is set

};

// Add the match and prediction entities, ensuring match has no goals set and is not "watchable"

var match = new Match

{

Id = 1,

GoalsLocal = null, // No goals set

GoalsVisitor = null, // No goals set

Date = DateTime.Now.AddHours(1) // Match is in the future to avoid CanWatch logic returning true

};

var prediction = new Prediction

{

Id = 1,

Match = match,

User = user,

GoalsLocal = 2,

GoalsVisitor = 1,

Points = 10

};

context.Users.Add(user);

context.Matches.Add(match);

context.Predictions.Add(prediction);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new PredictionsRepository(fakeContext, \_usersRepositoryMock.Object);

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Points = 5

};

// Act

var result = await repository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Check for the correct error message for DbUpdateException

}

[TestMethod]

public async Task UpdateAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForPrediction()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Create and add entities directly to the context.

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var match = new Match { Id = 1, Local = new Team { Id = 1, Name = "Team A" }, Visitor = new Team { Id = 2, Name = "Team B" }, Date = DateTime.Now.AddMinutes(30) }; // Future date to bypass CanWatch

var prediction = new Prediction

{

Id = 1,

Group = group,

User = user,

Match = match,

GoalsLocal = null,

GoalsVisitor = null,

Points = null

};

context.Users.Add(user);

context.Groups.Add(group);

context.Matches.Add(match);

context.Predictions.Add(prediction);

await context.SaveChangesAsync();

// Use the FakeDbContextWithGeneralException to simulate an exception.

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new PredictionsRepository(fakeContext, \_usersRepositoryMock.Object);

var predictionDTO = new PredictionDTO

{

Id = 1,

GoalsLocal = 2,

GoalsVisitor = 1,

Points = 5

};

// Act

var result = await repository.UpdateAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message);

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenDbUpdateExceptionOccurs\_ForPrediction()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var match = new Match { Id = 1, Local = new Team { Id = 1, Name = "Team A" }, Visitor = new Team { Id = 2, Name = "Team B" }, Date = DateTime.Now.AddMinutes(30) };

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

// Mocking GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

// Add the other entities to the context

context.Groups.Add(group);

context.Matches.Add(match);

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

// Use FakeDbContext to simulate DbUpdateException

var fakeContext = new FakeDbContext(options);

var repository = new PredictionsRepository(fakeContext, mockUsersRepository.Object);

var predictionDTO = new PredictionDTO

{

UserId = user.Id,

GroupId = group.Id,

TournamentId = tournament.Id,

MatchId = match.Id,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await repository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("ERR003", result.Message); // Verify that DbUpdateException is caught and handled

}

[TestMethod]

public async Task AddAsync\_ReturnsError\_WhenGeneralExceptionOccurs\_ForPrediction()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

using var context = new DataContext(options);

// Mocking the IUsersRepository

var mockUsersRepository = new Mock<IUsersRepository>();

// Create related entities

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var group = new Group { Id = 1, Name = "Group A", AdminId = Guid.NewGuid().ToString(), Code = "GRP123" };

var match = new Match { Id = 1, Local = new Team { Id = 1, Name = "Team A" }, Visitor = new Team { Id = 2, Name = "Team B" }, Date = DateTime.Now.AddMinutes(30) };

var tournament = new Tournament { Id = 1, Name = "Tournament A" };

// Mocking GetUserAsync to return a valid user

mockUsersRepository.Setup(repo => repo.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

// Add the other entities to the context

context.Groups.Add(group);

context.Matches.Add(match);

context.Tournaments.Add(tournament);

await context.SaveChangesAsync();

// Use FakeDbContextWithGeneralException to simulate a general exception

var fakeContext = new FakeDbContextWithGeneralException(options);

var repository = new PredictionsRepository(fakeContext, mockUsersRepository.Object);

var predictionDTO = new PredictionDTO

{

UserId = user.Id,

GroupId = group.Id,

TournamentId = tournament.Id,

MatchId = match.Id,

GoalsLocal = 2,

GoalsVisitor = 1

};

// Act

var result = await repository.AddAsync(predictionDTO);

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("General exception occurred", result.Message); // Verify that a general exception is caught and handled

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Usuarios

### Controlador

1. Adicione la clase **AccountsControllerTests**:

using Fantasy.Backend.Controllers;

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.UnitsOfWork.Interfaces;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Routing;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Moq;

using System.Security.Claims;

using System.Security.Principal;

using SignInResult = Microsoft.AspNetCore.Identity.SignInResult;

namespace Fantasy.Tests.Controllers;

[TestClass]

public class AccountsControllerTests

{

private Mock<IUsersUnitOfWork> \_mockUsersUnitOfWork = null!;

private Mock<IConfiguration> \_mockConfiguration = null!;

private Mock<IMailHelper> \_mockMailHelper = null!;

private Mock<IFileStorage> \_mockFileStorage = null!;

private Mock<ClaimsIdentity> \_mockClaimsIdentity = null!;

private Mock<ClaimsPrincipal> \_mockClaimsPrincipal = null!;

private Mock<DataContext> \_mockContext = null!;

private AccountsController \_controller = null!;

private DataContext \_context = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new DataContext(options);

\_mockUsersUnitOfWork = new Mock<IUsersUnitOfWork>();

\_mockClaimsIdentity = new Mock<ClaimsIdentity>();

\_mockClaimsPrincipal = new Mock<ClaimsPrincipal>();

\_mockContext = new Mock<DataContext>();

\_mockConfiguration = new Mock<IConfiguration>();

\_mockMailHelper = new Mock<IMailHelper>();

\_mockFileStorage = new Mock<IFileStorage>();

\_controller = new AccountsController(

\_mockUsersUnitOfWork.Object,

\_mockConfiguration.Object,

\_mockMailHelper.Object,

\_context,

\_mockFileStorage.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsOk\_WhenUserIsFound()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

// Mock User.Identity.Name to simulate the authenticated user email

\_mockClaimsIdentity.Setup(x => x.Name).Returns(user.Email);

\_mockClaimsPrincipal.Setup(x => x.Identity).Returns(\_mockClaimsIdentity.Object);

// Assign the mocked ClaimsPrincipal to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = \_mockClaimsPrincipal.Object }

};

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(user.Email))

.ReturnsAsync(user);

// Act

var result = await \_controller.GetAsync();

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode); // Check for 200 OK status code

Assert.AreEqual(user, okResult.Value); // Verify that the returned value is the mock user

}

[TestMethod]

public async Task GetAsync\_ReturnsOk\_WhenPaginationIsSuccessful()

{

// Arrange

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var users = new List<User> { new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" } };

\_mockUsersUnitOfWork.Setup(x => x.GetAsync(It.IsAny<PaginationDTO>()))

.ReturnsAsync(new ActionResponse<IEnumerable<User>>

{

WasSuccess = true,

Result = users

});

// Act

var result = await \_controller.GetAsync(pagination);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.AreEqual(users, okResult.Value);

}

[TestMethod]

public async Task RecoverPasswordAsync\_ReturnsNoContent\_WhenEmailIsSentSuccessfully()

{

// Arrange

var emailDTO = new EmailDTO { Email = "test@example.com", Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = emailDTO.Email, FirstName = "John", LastName = "Doe" };

// Mock the User retrieval and token generation

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

// Mock the configuration values for email subjects, bodies, and URL

\_mockConfiguration.Setup(x => x["Mail:SubjectRecoveryEn"]).Returns("Password Recovery");

\_mockConfiguration.Setup(x => x["Mail:BodyRecoveryEn"]).Returns("Please reset your password using this link: {0}");

\_mockConfiguration.Setup(x => x["Url Frontend"]).Returns("http://example.com");

// Mock the Url.Action to return a valid URL

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/reset\_password\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock the email sending process

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Act

var result = await \_controller.RecoverPasswordAsync(emailDTO);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode);

}

[TestMethod]

public async Task LoginAsync\_ReturnsOk\_WhenLoginIsSuccessful()

{

// Arrange

var loginDTO = new LoginDTO { Email = "test@example.com", Password = "password" };

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = loginDTO.Email,

FirstName = "John",

LastName = "Doe",

Photo = "some\_photo\_url",

Country = new Country { Id = 1, Name = "Test Country" }

};

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDTO))

.ReturnsAsync(SignInResult.Success);

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(loginDTO.Email))

.ReturnsAsync(user);

// Provide a valid 256-bit (32-byte) key for JWT signing

\_mockConfiguration.Setup(x => x["jwtKey"]).Returns("this\_is\_a\_very\_secure\_and\_long\_key\_32\_characters");

// Act

var result = await \_controller.LoginAsync(loginDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode);

Assert.IsNotNull(okResult.Value); // Token should be generated

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsNoContent\_WhenPasswordChangeIsSuccessful()

{

// Arrange

var changePasswordDTO = new ChangePasswordDTO { CurrentPassword = "current", NewPassword = "newPassword" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

// Mock the User.Identity.Name to simulate an authenticated user

var userIdentity = new GenericIdentity(user.Email);

var principal = new GenericPrincipal(userIdentity, roles: null);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = principal }

};

// Mock the GetUserAsync method to return the user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(user);

// Mock the ChangePasswordAsync method to return success

\_mockUsersUnitOfWork.Setup(x => x.ChangePasswordAsync(It.IsAny<User>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_controller.ChangePasswordAsync(changePasswordDTO);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode);

}

[TestMethod]

public async Task CreateUser\_ReturnsNoContent\_WhenUserIsCreatedSuccessfully()

{

// Arrange

var userDTO = new UserDTO { Email = "test@example.com", Password = "password", CountryId = 1, Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = userDTO.Email };

var country = new Country { Id = 1, Name = "Country A" };

\_context.Countries.Add(country);

await \_context.SaveChangesAsync();

// Mock AddUserAsync to return a successful identity result

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Success);

// Mock AddUserToRoleAsync to complete successfully

\_mockUsersUnitOfWork.Setup(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()))

.Returns(Task.CompletedTask);

// Mock SendMail for confirmation email to return a successful response

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock the configuration values for confirmation email

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email using this link: {0}");

\_mockConfiguration.Setup(x => x["Url Frontend"]).Returns("http://example.com");

// Mock Url.Action to return a valid confirmation URL

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/confirm\_email\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode);

}

[TestMethod]

public async Task GetAsync\_ReturnsBadRequest\_WhenGetAsyncFails()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Simulate a failed response from the unit of work

\_mockUsersUnitOfWork.Setup(x => x.GetAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<IEnumerable<User>> { WasSuccess = false });

// Act

var result = await \_controller.GetAsync(paginationDTO);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

}

[TestMethod]

public async Task GetPagesAsync\_ReturnsOk\_WhenGetTotalRecordsIsSuccessful()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var totalRecords = 100;

// Simulate a successful response from the unit of work

\_mockUsersUnitOfWork.Setup(x => x.GetTotalRecordsAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = totalRecords });

// Act

var result = await \_controller.GetPagesAsync(paginationDTO);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult);

Assert.AreEqual(200, okResult.StatusCode); // Ensure the status code is 200

Assert.AreEqual(totalRecords, okResult.Value); // Ensure the correct number of total records is returned

}

[TestMethod]

public async Task GetPagesAsync\_ReturnsBadRequest\_WhenGetTotalRecordsFails()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Simulate a failed response from the unit of work

\_mockUsersUnitOfWork.Setup(x => x.GetTotalRecordsAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = false });

// Act

var result = await \_controller.GetPagesAsync(paginationDTO);

// Assert

var badRequestResult = result as BadRequestResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

}

[TestMethod]

public async Task RecoverPasswordAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var emailDTO = new EmailDTO { Email = "nonexistent@example.com", Language = "en" };

// Simulate GetUserAsync returning null (user not found)

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.RecoverPasswordAsync(emailDTO);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Ensure the status code is 404

}

[TestMethod]

public async Task RecoverPasswordAsync\_ReturnsBadRequest\_WhenSendRecoverEmailFails()

{

// Arrange

var emailDTO = new EmailDTO { Email = "test@example.com", Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = emailDTO.Email };

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync(user);

// Simulate SendMail returning a failure response

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = false, Message = "Failed to send email" });

// Mock Url.Action to return a valid URL for the recovery email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/reset\_password\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock configuration values used in the email

\_mockConfiguration.Setup(x => x["Mail:SubjectRecoveryEn"]).Returns("Password Recovery");

\_mockConfiguration.Setup(x => x["Mail:BodyRecoveryEn"]).Returns("Click the link to reset your password: {0}");

\_mockConfiguration.Setup(x => x["Url Frontend"]).Returns("http://example.com");

// Mock HttpContext and Request.Scheme

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http"); // Mock the Request.Scheme to avoid null references

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

// Set HttpContext for the controller

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Act

var result = await \_controller.RecoverPasswordAsync(emailDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

Assert.AreEqual("Failed to send email", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var resetPasswordDTO = new ResetPasswordDTO { Email = "nonexistent@example.com", Token = "token", NewPassword = "newPassword123" };

// Simulate GetUserAsync returning null (user not found)

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(resetPasswordDTO.Email))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.ResetPasswordAsync(resetPasswordDTO);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Ensure the status code is 404

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsNoContent\_WhenPasswordResetIsSuccessful()

{

// Arrange

var resetPasswordDTO = new ResetPasswordDTO { Email = "test@example.com", Token = "token", NewPassword = "newPassword123" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = resetPasswordDTO.Email };

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(resetPasswordDTO.Email))

.ReturnsAsync(user);

// Simulate ResetPasswordAsync returning a successful result

\_mockUsersUnitOfWork.Setup(x => x.ResetPasswordAsync(user, resetPasswordDTO.Token, resetPasswordDTO.NewPassword))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_controller.ResetPasswordAsync(resetPasswordDTO);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode); // Ensure the status code is 204

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsBadRequest\_WhenPasswordResetFails()

{

// Arrange

var resetPasswordDTO = new ResetPasswordDTO { Email = "test@example.com", Token = "invalid\_token", NewPassword = "newPassword123" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = resetPasswordDTO.Email };

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(resetPasswordDTO.Email))

.ReturnsAsync(user);

// Simulate ResetPasswordAsync returning a failed result with an error message

var identityResult = IdentityResult.Failed(new IdentityError { Description = "Invalid token" });

\_mockUsersUnitOfWork.Setup(x => x.ResetPasswordAsync(user, resetPasswordDTO.Token, resetPasswordDTO.NewPassword))

.ReturnsAsync(identityResult);

// Act

var result = await \_controller.ResetPasswordAsync(resetPasswordDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

Assert.AreEqual("Invalid token", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsBadRequest\_WhenModelStateIsInvalid()

{

// Arrange

var changePasswordDTO = new ChangePasswordDTO { CurrentPassword = "current", NewPassword = "newPassword" };

// Mark ModelState as invalid

\_controller.ModelState.AddModelError("CurrentPassword", "Required");

// Act

var result = await \_controller.ChangePasswordAsync(changePasswordDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var changePasswordDTO = new ChangePasswordDTO { CurrentPassword = "current", NewPassword = "newPassword" };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns("test@example.com");

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning null (user not found)

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.ChangePasswordAsync(changePasswordDTO);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Ensure the status code is 404

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsBadRequest\_WhenPasswordChangeFails()

{

// Arrange

var changePasswordDTO = new ChangePasswordDTO { CurrentPassword = "current", NewPassword = "newPassword" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns(user.Email);

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(user);

// Simulate ChangePasswordAsync returning a failed result with an error message

var identityResult = IdentityResult.Failed(new IdentityError { Description = "Invalid password" });

\_mockUsersUnitOfWork.Setup(x => x.ChangePasswordAsync(user, changePasswordDTO.CurrentPassword, changePasswordDTO.NewPassword))

.ReturnsAsync(identityResult);

// Act

var result = await \_controller.ChangePasswordAsync(changePasswordDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

Assert.AreEqual("Invalid password", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task PutAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns("test@example.com");

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning null (user not found)

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.PutAsync(user);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Ensure the status code is 404

}

[TestMethod]

public async Task PutAsync\_ReturnsOk\_WhenUserUpdateIsSuccessful()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe", Photo = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }) };

var currentUser = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", Photo = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 }), Country = new Country { Id = 1, Name = "USA" } };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns(currentUser.Email);

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning the current user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(currentUser);

// Simulate a successful photo upload

\_mockFileStorage.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), It.IsAny<string>(), It.IsAny<string>()))

.ReturnsAsync("new\_photo\_url");

// Simulate a successful user update

\_mockUsersUnitOfWork.Setup(x => x.UpdateUserAsync(It.IsAny<User>()))

.ReturnsAsync(IdentityResult.Success);

// Provide a long enough JWT key for HS256

\_mockConfiguration.Setup(x => x["jwtKey"]).Returns("32CharSecureKeyThatIsLongEnoughForHS256");

// Act

var result = await \_controller.PutAsync(user);

// Assert

var okResult = result as OkObjectResult;

Assert.IsNotNull(okResult); // Ensure the result is OkObjectResult

Assert.AreEqual(200, okResult.StatusCode); // Ensure the status code is 200

Assert.IsNotNull(okResult.Value); // Ensure the token was returned

}

[TestMethod]

public async Task PutAsync\_ReturnsBadRequest\_WhenUserUpdateFails()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var currentUser = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns(currentUser.Email);

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning the current user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(currentUser);

// Simulate UpdateUserAsync returning a failed result

var identityResult = IdentityResult.Failed(new IdentityError { Description = "Update failed" });

\_mockUsersUnitOfWork.Setup(x => x.UpdateUserAsync(It.IsAny<User>()))

.ReturnsAsync(identityResult);

// Act

var result = await \_controller.PutAsync(user);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

// Extract the IdentityError from the BadRequestObjectResult and check its description

var identityError = badRequestResult.Value as IdentityError;

Assert.IsNotNull(identityError);

Assert.AreEqual("Update failed", identityError.Description); // Ensure the correct error message is returned

}

[TestMethod]

public async Task PutAsync\_ReturnsBadRequest\_WhenExceptionIsThrown()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var currentUser = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

// Mock the User.Identity.Name to return a specific email (simulating an authenticated user)

var userIdentityMock = new Mock<ClaimsIdentity>();

userIdentityMock.Setup(x => x.Name).Returns(currentUser.Email);

var claimsPrincipalMock = new Mock<ClaimsPrincipal>();

claimsPrincipalMock.Setup(x => x.Identity).Returns(userIdentityMock.Object);

// Assign the mocked User to the controller's HttpContext

\_controller.ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext { User = claimsPrincipalMock.Object }

};

// Simulate GetUserAsync returning the current user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<string>()))

.ReturnsAsync(currentUser);

// Simulate an exception being thrown when trying to update the user

\_mockUsersUnitOfWork.Setup(x => x.UpdateUserAsync(It.IsAny<User>()))

.ThrowsAsync(new Exception("An error occurred"));

// Act

var result = await \_controller.PutAsync(user);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

Assert.AreEqual("An error occurred", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task ResedTokenAsync\_ReturnsNoContent\_WhenEmailIsSentSuccessfully()

{

// Arrange

var emailDTO = new EmailDTO { Email = "test@example.com", Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = emailDTO.Email };

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync(user);

// Simulate SendConfirmationEmailAsync returning a success response

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock Url.Action to return a valid URL

\_controller.ControllerContext = new ControllerContext();

\_controller.ControllerContext.HttpContext = new DefaultHttpContext();

\_controller.Url = Mock.Of<IUrlHelper>(x => x.Action(It.IsAny<UrlActionContext>()) == "https://example.com/confirm");

// Mock configuration to return non-null values for email subject and body

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email by clicking the link: {0}");

// Act

var result = await \_controller.ResedTokenAsync(emailDTO);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode); // Ensure the status code is 204 No Content

}

[TestMethod]

public async Task ResedTokenAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var emailDTO = new EmailDTO { Email = "test@example.com", Language = "en" };

// Simulate GetUserAsync returning null (user not found)

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.ResedTokenAsync(emailDTO);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Ensure the status code is 404 Not Found

}

[TestMethod]

public async Task ResedTokenAsync\_ReturnsBadRequest\_WhenEmailSendFails()

{

// Arrange

var emailDTO = new EmailDTO { Email = "test@example.com", Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = emailDTO.Email };

// Simulate GetUserAsync returning a valid user

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(emailDTO.Email))

.ReturnsAsync(user);

// Simulate SendConfirmationEmailAsync returning a failure response

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = false, Message = "Failed to send email" });

// Mock Url.Action to return a valid URL

\_controller.ControllerContext = new ControllerContext();

\_controller.ControllerContext.HttpContext = new DefaultHttpContext();

\_controller.Url = Mock.Of<IUrlHelper>(x => x.Action(It.IsAny<UrlActionContext>()) == "https://example.com/confirm");

// Mock configuration to return non-null values for email subject and body

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email by clicking the link: {0}");

// Act

var result = await \_controller.ResedTokenAsync(emailDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400 Bad Request

Assert.AreEqual("Failed to send email", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsNoContent\_WhenEmailConfirmationIsSuccessful()

{

// Arrange

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

var token = "valid\_token";

var user = new User { Id = userId };

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.ConfirmEmailAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_controller.ConfirmEmailAsync(userId, token);

// Assert

var noContentResult = result as NoContentResult;

Assert.IsNotNull(noContentResult);

Assert.AreEqual(204, noContentResult.StatusCode);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsBadRequest\_WhenEmailConfirmationFails()

{

// Arrange

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

var token = "valid\_token";

var user = new User { Id = userId };

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync(user);

\_mockUsersUnitOfWork.Setup(x => x.ConfirmEmailAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Failed(new IdentityError { Description = "Invalid token" }));

// Act

var result = await \_controller.ConfirmEmailAsync(userId, token);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

// Verify that the error message is correct

var identityError = badRequestResult.Value as IdentityError;

Assert.IsNotNull(identityError);

Assert.AreEqual("Invalid token", identityError.Description);

}

[TestMethod]

public async Task CreateUser\_ReturnsBadRequest\_WhenCountryNotFound()

{

// Arrange

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

using var context = new DataContext(options);

var userDTO = new UserDTO { Email = "test@example.com", Password = "password", CountryId = 999, Language = "en" };

// Create controller with the in-memory context

\_controller = new AccountsController(

\_mockUsersUnitOfWork.Object,

\_mockConfiguration.Object,

\_mockMailHelper.Object,

context,

\_mockFileStorage.Object);

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("ERR004", badRequestResult.Value);

}

[TestMethod]

public async Task LoginAsync\_ReturnsBadRequest\_WhenLoginFails()

{

// Arrange

var loginDTO = new LoginDTO { Email = "test@example.com", Password = "wrong\_password" };

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDTO))

.ReturnsAsync(SignInResult.Failed);

// Act

var result = await \_controller.LoginAsync(loginDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("ERR006", badRequestResult.Value);

}

[TestMethod]

public async Task SendRecoverEmailAsync\_ReturnsSuccess\_WhenEmailIsSent()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock Url.Action to return a valid URL for the password reset email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/reset\_password\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme to avoid NullReferenceException

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for email subject and body to avoid null references

\_mockConfiguration.Setup(x => x["Mail:SubjectRecoveryEn"]).Returns("Password Recovery");

\_mockConfiguration.Setup(x => x["Mail:BodyRecoveryEn"]).Returns("Please reset your password using this link: {0}");

// Act

var result = await \_controller.SendRecoverEmailAsync(user, "en");

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task SendRecoverEmailAsync\_ReturnsFailure\_WhenEmailFails()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = false, Message = "Failed to send email" });

// Mock Url.Action to return a valid URL for the password reset email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/reset\_password\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme to avoid NullReferenceException

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for email subject and body to avoid null references

\_mockConfiguration.Setup(x => x["Mail:SubjectRecoveryEn"]).Returns("Password Recovery");

\_mockConfiguration.Setup(x => x["Mail:BodyRecoveryEn"]).Returns("Please reset your password using this link: {0}");

// Act

var result = await \_controller.SendRecoverEmailAsync(user, "en");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Failed to send email", result.Message);

}

[TestMethod]

public async Task SendConfirmationEmailAsync\_ReturnsSuccess\_WhenEmailIsSent()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("confirmation\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock Url.Action to return a valid URL for the confirmation email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/confirm\_email\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme to avoid NullReferenceException

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for email subject and body to avoid null references

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email using this link: {0}");

// Act

var result = await \_controller.SendConfirmationEmailAsync(user, "en");

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task SendConfirmationEmailAsync\_ReturnsFailure\_WhenEmailFails()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("confirmation\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = false, Message = "Failed to send email" });

// Mock Url.Action to return a valid URL for the confirmation email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/confirm\_email\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme to avoid NullReferenceException

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for email subject and body to avoid null references

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email using this link: {0}");

// Act

var result = await \_controller.SendConfirmationEmailAsync(user, "en");

// Assert

Assert.IsFalse(result.WasSuccess);

Assert.AreEqual("Failed to send email", result.Message);

}

[TestMethod]

public async Task SendRecoverEmailAsync\_ReturnsSuccess\_WhenEmailIsSent\_InSpanish()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

\_mockUsersUnitOfWork.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock Url.Action to return a valid URL for the password reset email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/reset\_password\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for Spanish email subject and body

\_mockConfiguration.Setup(x => x["Mail:SubjectRecoveryEs"]).Returns("Recuperar Contraseña");

\_mockConfiguration.Setup(x => x["Mail:BodyRecoveryEs"]).Returns("Restablece tu contraseña usando este enlace: {0}");

// Act

var result = await \_controller.SendRecoverEmailAsync(user, "es");

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task SendConfirmationEmailAsync\_ReturnsSuccess\_WhenEmailIsSent\_InSpanish()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

\_mockUsersUnitOfWork.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("confirmation\_token");

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = true });

// Mock Url.Action to return a valid URL for the confirmation email link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/confirm\_email\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Mock configuration for Spanish email subject and body

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEs"]).Returns("Confirma tu correo");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEs"]).Returns("Confirma tu correo usando este enlace: {0}");

// Act

var result = await \_controller.SendConfirmationEmailAsync(user, "es");

// Assert

Assert.IsTrue(result.WasSuccess);

}

[TestMethod]

public async Task LoginAsync\_ReturnsBadRequest\_WhenUserIsLockedOut()

{

// Arrange

var loginDTO = new LoginDTO { Email = "lockedout@example.com", Password = "password" };

// Simulate that the login attempt results in a locked-out state

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDTO))

.ReturnsAsync(SignInResult.LockedOut);

// Act

var result = await \_controller.LoginAsync(loginDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("ERR007", badRequestResult.Value); // Verify the correct error message

}

[TestMethod]

public async Task LoginAsync\_ReturnsBadRequest\_WhenLoginIsNotAllowed()

{

// Arrange

var loginDTO = new LoginDTO { Email = "notallowed@example.com", Password = "password" };

// Simulate that the login attempt results in a not allowed state

\_mockUsersUnitOfWork.Setup(x => x.LoginAsync(loginDTO))

.ReturnsAsync(SignInResult.NotAllowed);

// Act

var result = await \_controller.LoginAsync(loginDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode);

Assert.AreEqual("ERR008", badRequestResult.Value); // Verify the correct error message

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsNotFound\_WhenUserDoesNotExist()

{

// Arrange

var userId = Guid.NewGuid().ToString(); // Simulate a valid user ID

var token = "valid\_token";

// Simulate GetUserAsync returning null, indicating the user does not exist

\_mockUsersUnitOfWork.Setup(x => x.GetUserAsync(It.IsAny<Guid>()))

.ReturnsAsync((User)null!);

// Act

var result = await \_controller.ConfirmEmailAsync(userId, token);

// Assert

var notFoundResult = result as NotFoundResult;

Assert.IsNotNull(notFoundResult);

Assert.AreEqual(404, notFoundResult.StatusCode); // Verify the status code is 404 Not Found

}

[TestMethod]

public async Task CreateUser\_ReturnsBadRequest\_WhenSendConfirmationEmailFails()

{

// Arrange

var userDTO = new UserDTO { Email = "test@example.com", Password = "password", CountryId = 1, Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = userDTO.Email };

var country = new Country { Id = 1, Name = "Country A" };

\_context.Countries.Add(country);

await \_context.SaveChangesAsync();

// Simulate AddUserAsync to return a successful identity result

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(IdentityResult.Success);

// Simulate AddUserToRoleAsync completing successfully

\_mockUsersUnitOfWork.Setup(x => x.AddUserToRoleAsync(It.IsAny<User>(), It.IsAny<string>()))

.Returns(Task.CompletedTask);

// Simulate the SendConfirmationEmailAsync failing to send the email

\_mockMailHelper.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

.Returns(new ActionResponse<string> { WasSuccess = false, Message = "Failed to send email" });

// Mock configuration to avoid null references for email subject and body

\_mockConfiguration.Setup(x => x["Mail:SubjectConfirmationEn"]).Returns("Confirm your email");

\_mockConfiguration.Setup(x => x["Mail:BodyConfirmationEn"]).Returns("Please confirm your email using this link: {0}");

\_mockConfiguration.Setup(x => x["Url Frontend"]).Returns("http://example.com");

// Mock Url.Action to return a valid confirmation link

var mockUrlHelper = new Mock<IUrlHelper>();

mockUrlHelper.Setup(x => x.Action(It.IsAny<UrlActionContext>()))

.Returns("http://example.com/confirm\_email\_link");

\_controller.Url = mockUrlHelper.Object;

// Mock HttpContext and Request.Scheme to avoid NullReferenceException

var httpContextMock = new Mock<HttpContext>();

var requestMock = new Mock<HttpRequest>();

requestMock.Setup(x => x.Scheme).Returns("http");

httpContextMock.Setup(x => x.Request).Returns(requestMock.Object);

\_controller.ControllerContext = new ControllerContext

{

HttpContext = httpContextMock.Object

};

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Verify the status code is 400

Assert.AreEqual("Failed to send email", badRequestResult.Value); // Ensure the correct error message is returned

}

[TestMethod]

public async Task CreateUser\_ReturnsBadRequest\_WhenAddUserAsyncFails()

{

// Arrange

var userDTO = new UserDTO { Email = "test@example.com", Password = "password", CountryId = 1, Language = "en" };

var user = new User { Id = Guid.NewGuid().ToString(), Email = userDTO.Email };

var country = new Country { Id = 1, Name = "Country A" };

\_context.Countries.Add(country);

await \_context.SaveChangesAsync();

// Simulate AddUserAsync returning a failed result with an error message

var identityResult = IdentityResult.Failed(new IdentityError { Description = "User creation failed" });

\_mockUsersUnitOfWork.Setup(x => x.AddUserAsync(It.IsAny<User>(), It.IsAny<string>()))

.ReturnsAsync(identityResult);

// Act

var result = await \_controller.CreateUser(userDTO);

// Assert

var badRequestResult = result as BadRequestObjectResult;

Assert.IsNotNull(badRequestResult);

Assert.AreEqual(400, badRequestResult.StatusCode); // Ensure the status code is 400

// Verify the correct error message is returned

var identityError = badRequestResult.Value as IdentityError;

Assert.IsNotNull(identityError);

Assert.AreEqual("User creation failed", identityError.Description);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Unidad de Trabajo

1. Adicione la clase **UsersUnitOfWorkTests**:

using Fantasy.Backend.Repositories.Interfaces;

using Fantasy.Backend.UnitsOfWork.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Fantasy.Shared.Responses;

using Microsoft.AspNetCore.Identity;

using Moq;

namespace Fantasy.Tests.UnitsOfWork;

[TestClass]

public class UsersUnitOfWorkTests

{

private Mock<IUsersRepository> \_mockUsersRepository = null!;

private UsersUnitOfWork \_usersUnitOfWork = null!;

[TestInitialize]

public void Setup()

{

\_mockUsersRepository = new Mock<IUsersRepository>();

\_usersUnitOfWork = new UsersUnitOfWork(\_mockUsersRepository.Object);

}

[TestMethod]

public async Task GetAsync\_ReturnsUserList\_WhenPaginationIsProvided()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

var users = new List<User> { new User { Email = "test@example.com" } };

\_mockUsersRepository.Setup(x => x.GetAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<IEnumerable<User>> { WasSuccess = true, Result = users });

// Act

var result = await \_usersUnitOfWork.GetAsync(paginationDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(users, result.Result);

\_mockUsersRepository.Verify(x => x.GetAsync(paginationDTO), Times.Once);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsTotalRecords()

{

// Arrange

var paginationDTO = new PaginationDTO { Page = 1, RecordsNumber = 10 };

\_mockUsersRepository.Setup(x => x.GetTotalRecordsAsync(paginationDTO))

.ReturnsAsync(new ActionResponse<int> { WasSuccess = true, Result = 100 });

// Act

var result = await \_usersUnitOfWork.GetTotalRecordsAsync(paginationDTO);

// Assert

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(100, result.Result);

\_mockUsersRepository.Verify(x => x.GetTotalRecordsAsync(paginationDTO), Times.Once);

}

[TestMethod]

public async Task GeneratePasswordResetTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

// Act

var token = await \_usersUnitOfWork.GeneratePasswordResetTokenAsync(user);

// Assert

Assert.AreEqual("reset\_token", token);

\_mockUsersRepository.Verify(x => x.GeneratePasswordResetTokenAsync(user), Times.Once);

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.ResetPasswordAsync(user, "token", "new\_password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersUnitOfWork.ResetPasswordAsync(user, "token", "new\_password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

\_mockUsersRepository.Verify(x => x.ResetPasswordAsync(user, "token", "new\_password"), Times.Once);

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.ChangePasswordAsync(user, "old\_password", "new\_password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersUnitOfWork.ChangePasswordAsync(user, "old\_password", "new\_password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

\_mockUsersRepository.Verify(x => x.ChangePasswordAsync(user, "old\_password", "new\_password"), Times.Once);

}

[TestMethod]

public async Task UpdateUserAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.UpdateUserAsync(user))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersUnitOfWork.UpdateUserAsync(user);

// Assert

Assert.AreEqual(IdentityResult.Success, result);

\_mockUsersRepository.Verify(x => x.UpdateUserAsync(user), Times.Once);

}

[TestMethod]

public async Task GetUserAsync\_ById\_ReturnsUser()

{

// Arrange

var userId = Guid.NewGuid();

var user = new User { Id = userId.ToString(), Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.GetUserAsync(userId))

.ReturnsAsync(user);

// Act

var result = await \_usersUnitOfWork.GetUserAsync(userId);

// Assert

Assert.AreEqual(user, result);

\_mockUsersRepository.Verify(x => x.GetUserAsync(userId), Times.Once);

}

[TestMethod]

public async Task GenerateEmailConfirmationTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync("confirmation\_token");

// Act

var token = await \_usersUnitOfWork.GenerateEmailConfirmationTokenAsync(user);

// Assert

Assert.AreEqual("confirmation\_token", token);

\_mockUsersRepository.Verify(x => x.GenerateEmailConfirmationTokenAsync(user), Times.Once);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.ConfirmEmailAsync(user, "token"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersUnitOfWork.ConfirmEmailAsync(user, "token");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

\_mockUsersRepository.Verify(x => x.ConfirmEmailAsync(user, "token"), Times.Once);

}

[TestMethod]

public async Task AddUserAsync\_ReturnsIdentityResult()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.AddUserAsync(user, "password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersUnitOfWork.AddUserAsync(user, "password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

\_mockUsersRepository.Verify(x => x.AddUserAsync(user, "password"), Times.Once);

}

[TestMethod]

public async Task AddUserToRoleAsync\_SuccessfullyAddsUserToRole()

{

// Arrange

var user = new User { Email = "test@example.com" };

// Act

await \_usersUnitOfWork.AddUserToRoleAsync(user, "Admin");

// Assert

\_mockUsersRepository.Verify(x => x.AddUserToRoleAsync(user, "Admin"), Times.Once);

}

[TestMethod]

public async Task CheckRoleAsync\_VerifiesRoleCheck()

{

// Arrange

// Act

await \_usersUnitOfWork.CheckRoleAsync("Admin");

// Assert

\_mockUsersRepository.Verify(x => x.CheckRoleAsync("Admin"), Times.Once);

}

[TestMethod]

public async Task GetUserAsync\_ByEmail\_ReturnsUser()

{

// Arrange

var email = "test@example.com";

var user = new User { Email = email };

\_mockUsersRepository.Setup(x => x.GetUserAsync(email))

.ReturnsAsync(user);

// Act

var result = await \_usersUnitOfWork.GetUserAsync(email);

// Assert

Assert.AreEqual(user, result);

\_mockUsersRepository.Verify(x => x.GetUserAsync(email), Times.Once);

}

[TestMethod]

public async Task IsUserInRoleAsync\_ReturnsTrue\_WhenUserIsInRole()

{

// Arrange

var user = new User { Email = "test@example.com" };

\_mockUsersRepository.Setup(x => x.IsUserInRoleAsync(user, "Admin"))

.ReturnsAsync(true);

// Act

var result = await \_usersUnitOfWork.IsUserInRoleAsync(user, "Admin");

// Assert

Assert.IsTrue(result);

\_mockUsersRepository.Verify(x => x.IsUserInRoleAsync(user, "Admin"), Times.Once);

}

[TestMethod]

public async Task LoginAsync\_ReturnsSignInResult()

{

// Arrange

var loginDTO = new LoginDTO { Email = "test@example.com", Password = "password" };

\_mockUsersRepository.Setup(x => x.LoginAsync(loginDTO))

.ReturnsAsync(SignInResult.Success);

// Act

var result = await \_usersUnitOfWork.LoginAsync(loginDTO);

// Assert

Assert.AreEqual(SignInResult.Success, result);

\_mockUsersRepository.Verify(x => x.LoginAsync(loginDTO), Times.Once);

}

[TestMethod]

public async Task LogoutAsync\_CallsRepositoryLogout()

{

// Act

await \_usersUnitOfWork.LogoutAsync();

// Assert

\_mockUsersRepository.Verify(x => x.LogoutAsync(), Times.Once);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### Repositorio

1. Adicione la clase **UsersRepositoryTests**:

using Fantasy.Backend.Data;

using Fantasy.Backend.Helpers;

using Fantasy.Backend.Repositories.Implementations;

using Fantasy.Shared.DTOs;

using Fantasy.Shared.Entities;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

using Moq;

namespace Fantasy.Tests.Repositories;

[TestClass]

public class UsersRepositoryTests

{

private UsersRepository \_usersRepository = null!;

private DataContext \_context = null!;

private Mock<UserManager<User>> \_mockUserManager = null!;

private Mock<SignInManager<User>> \_mockSignInManager = null!;

private Mock<RoleManager<IdentityRole>> \_mockRoleManager = null!;

private Mock<IFileStorage> \_mockFileStorage = null!;

[TestInitialize]

public void Setup()

{

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

\_context = new DataContext(options);

// Setup mocks

\_mockUserManager = MockUserManager();

\_mockSignInManager = MockSignInManager();

\_mockRoleManager = MockRoleManager();

\_mockFileStorage = new Mock<IFileStorage>();

\_usersRepository = new UsersRepository(

\_context,

\_mockUserManager.Object,

\_mockRoleManager.Object,

\_mockSignInManager.Object,

\_mockFileStorage.Object);

}

[TestCleanup]

public void Cleanup()

{

\_context.Database.EnsureDeleted(); // Clean up the database after each test

\_context.Dispose();

}

private Mock<SignInManager<User>> MockSignInManager()

{

var httpContextAccessor = new Mock<IHttpContextAccessor>();

var claimsFactory = new Mock<IUserClaimsPrincipalFactory<User>>();

var options = new Mock<IOptions<IdentityOptions>>();

var logger = new Mock<ILogger<SignInManager<User>>>();

var schemes = new Mock<IAuthenticationSchemeProvider>();

var userConfirmation = new Mock<IUserConfirmation<User>>();

return new Mock<SignInManager<User>>(

\_mockUserManager.Object,

httpContextAccessor.Object,

claimsFactory.Object,

options.Object,

logger.Object,

schemes.Object,

userConfirmation.Object

);

}

// Helper methods to mock UserManager, SignInManager, RoleManager

private Mock<UserManager<User>> MockUserManager()

{

var store = new Mock<IUserStore<User>>();

return new Mock<UserManager<User>>(store.Object, null!, null!, null!, null!, null!, null!, null!, null!);

}

private Mock<RoleManager<IdentityRole>> MockRoleManager()

{

var store = new Mock<IRoleStore<IdentityRole>>();

return new Mock<RoleManager<IdentityRole>>(store.Object, null!, null!, null!, null!);

}

[TestMethod]

public async Task GetAsync\_ReturnsUsersWithPagination()

{

// Arrange: Set up the in-memory database options with a unique name for this test

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString()) // Unique database for each test

.Options;

using var context = new DataContext(options);

// Create and add the country once

var country = new Country { Id = 1, Name = "TestCountry" };

await context.Countries.AddAsync(country);

await context.SaveChangesAsync(); // Save the country to avoid conflicts

// Create users and associate them with the country

var user1 = new User

{

Id = Guid.NewGuid().ToString(),

FirstName = "John",

LastName = "Doe",

Country = country,

GroupsManaged = [],

GroupsBelong = [],

Predictions = []

};

var user2 = new User

{

Id = Guid.NewGuid().ToString(),

FirstName = "Jane",

LastName = "Doe",

Country = country,

GroupsManaged = [],

GroupsBelong = [],

Predictions = []

};

// Add users to the in-memory database

await context.Users.AddRangeAsync(user1, user2);

await context.SaveChangesAsync(); // Save the users

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Create the UsersRepository with the in-memory context

var repository = new UsersRepository(context, null!, null!, null!, null!);

// Act: Retrieve users with pagination

var result = await repository.GetAsync(pagination);

// Assert: Verify that the result was successful and contains 2 users

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result!.Count());

Assert.AreEqual(0, result.Result!.FirstOrDefault()!.PredictionsCount);

Assert.AreEqual("/images/NoImage.png", result.Result!.FirstOrDefault()!.PhotoFull);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_ReturnsCorrectCount()

{

// Arrange: Set up the in-memory database options with a unique name for this test

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString()) // Unique database for each test

.Options;

using var context = new DataContext(options);

// Create a user and add it to the in-memory database

var user = new User

{

Id = Guid.NewGuid().ToString(),

FirstName = "John",

LastName = "Doe"

};

await context.Users.AddAsync(user);

await context.SaveChangesAsync();

// Set up pagination parameters

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10 };

// Create the UsersRepository with the in-memory context

var repository = new UsersRepository(context, null!, null!, null!, null!);

// Act: Get the total records count

var result = await repository.GetTotalRecordsAsync(pagination);

// Assert: Verify that the total count is correct

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(1, result.Result);

}

[TestMethod]

public async Task GeneratePasswordResetTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUserManager.Setup(x => x.GeneratePasswordResetTokenAsync(user))

.ReturnsAsync("reset\_token");

// Act

var token = await \_usersRepository.GeneratePasswordResetTokenAsync(user);

// Assert

Assert.AreEqual("reset\_token", token);

}

[TestMethod]

public async Task ResetPasswordAsync\_ReturnsSuccessResult()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUserManager.Setup(x => x.ResetPasswordAsync(user, "token", "new\_password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersRepository.ResetPasswordAsync(user, "token", "new\_password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

}

[TestMethod]

public async Task ChangePasswordAsync\_ReturnsSuccessResult()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUserManager.Setup(x => x.ChangePasswordAsync(user, "old\_password", "new\_password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersRepository.ChangePasswordAsync(user, "old\_password", "new\_password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

}

[TestMethod]

public async Task AddUserAsync\_ReturnsSuccessResult\_WhenPhotoIsNotProvided()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

\_mockUserManager.Setup(x => x.CreateAsync(user, "password"))

.ReturnsAsync(IdentityResult.Success);

// Act

var result = await \_usersRepository.AddUserAsync(user, "password");

// Assert

Assert.AreEqual(IdentityResult.Success, result);

}

[TestMethod]

public async Task AddUserAsync\_StoresPhoto\_WhenProvided()

{

// Arrange: Set up the in-memory database options

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

// Create a new DataContext with the in-memory database

using var context = new DataContext(options);

// Create a valid Base64 string for the photo

var validBase64Photo = Convert.ToBase64String(new byte[] { 1, 2, 3, 4 });

// Create a user and associate the photo with it

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Required field

LastName = "Doe", // Required field

Photo = validBase64Photo

};

// Mock FileStorage to simulate saving the photo

var mockFileStorage = new Mock<IFileStorage>();

mockFileStorage.Setup(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "users"))

.ReturnsAsync("http://someurl.com/photo.jpg");

// Mock UserManager to simulate user creation

var mockUserManager = MockUserManager();

mockUserManager.Setup(x => x.CreateAsync(user, "password"))

.ReturnsAsync(IdentityResult.Success);

// Create UsersRepository using the in-memory database and mocked dependencies

var usersRepository = new UsersRepository(context, mockUserManager.Object, null!, null!, mockFileStorage.Object);

// Act: Add the user and store the photo

var result = await usersRepository.AddUserAsync(user, "password");

// Assert: Verify that the user creation was successful and the photo was saved

Assert.AreEqual(IdentityResult.Success, result);

// Verify that the photo was saved and the URL was updated

mockFileStorage.Verify(x => x.SaveFileAsync(It.IsAny<byte[]>(), ".jpg", "users"), Times.Once);

Assert.AreEqual("http://someurl.com/photo.jpg", user.Photo);

}

[TestMethod]

public async Task GetUserAsync\_ById\_ReturnsUser()

{

// Arrange: Set up the in-memory database options

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: "TestDb")

.Options;

// Create a new DataContext with the in-memory database

using var context = new DataContext(options);

// Create a country and add it to the in-memory database

var country = new Country { Id = 1, Name = "TestCountry" };

await context.Countries.AddAsync(country);

await context.SaveChangesAsync();

// Create a user and add it to the in-memory database

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Required property

LastName = "Doe", // Required property

Country = country // Set the Country

};

await context.Users.AddAsync(user);

await context.SaveChangesAsync(); // Ensure the user is saved

// Create the UsersRepository with the in-memory context

var repository = new UsersRepository(context, null!, null!, null!, null!);

// Act: Retrieve the user by ID

var result = await repository.GetUserAsync(Guid.Parse(user.Id));

// Assert: Verify that the user retrieved is the one added

Assert.IsNotNull(result, "User should not be null");

Assert.AreEqual(user.Email, result.Email);

Assert.AreEqual(user.FirstName, result.FirstName);

Assert.AreEqual(user.LastName, result.LastName);

Assert.IsNotNull(result.Country);

Assert.AreEqual(country.Name, result.Country.Name);

}

[TestMethod]

public async Task GetUserAsync\_ByEmail\_ReturnsUser()

{

// Arrange: Set up the in-memory database options with a unique name for each test

var options = new DbContextOptionsBuilder<DataContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString()) // Unique database for each test

.Options;

// Create a new DataContext with the in-memory database

using var context = new DataContext(options);

// Create a country and add it to the in-memory database with a unique Id

var country = new Country

{

Id = 1, // Ensure this Id is unique across your tests, or use Guid.NewGuid().ToString() if possible

Name = "TestCountry"

};

await context.Countries.AddAsync(country);

await context.SaveChangesAsync(); // Save the country to avoid conflicts

// Create a user and associate it with the country

var user = new User

{

Id = Guid.NewGuid().ToString(),

Email = "test@example.com",

FirstName = "John", // Required property

LastName = "Doe", // Required property

Country = country // Attach the country to the user

};

await context.Users.AddAsync(user);

await context.SaveChangesAsync(); // Ensure the user is saved

// Verify the user was saved in the database

var savedUser = await context.Users.Include(u => u.Country).FirstOrDefaultAsync(u => u.Email == user.Email);

Assert.IsNotNull(savedUser, "The user was not saved in the in-memory database.");

// Create the UsersRepository with the in-memory context

var repository = new UsersRepository(context, null!, null!, null!, null!);

// Act: Retrieve the user by email

var result = await repository.GetUserAsync(user.Email);

// Assert: Verify that the user retrieved is the one added

Assert.IsNotNull(result, "User should not be null");

Assert.AreEqual(user.Email, result.Email);

Assert.AreEqual(user.FirstName, result.FirstName);

Assert.AreEqual(user.LastName, result.LastName);

Assert.IsNotNull(result.Country);

Assert.AreEqual(country.Name, result.Country.Name);

}

[TestMethod]

public async Task LoginAsync\_ReturnsSignInResult()

{

// Arrange

var loginDTO = new LoginDTO { Email = "test@example.com", Password = "password" };

\_mockSignInManager.Setup(x => x.PasswordSignInAsync(loginDTO.Email, loginDTO.Password, false, true))

.ReturnsAsync(SignInResult.Success);

// Act

var result = await \_usersRepository.LoginAsync(loginDTO);

// Assert

Assert.AreEqual(SignInResult.Success, result);

}

[TestMethod]

public async Task LogoutAsync\_CallsSignOut()

{

// Act

await \_usersRepository.LogoutAsync();

// Assert

\_mockSignInManager.Verify(x => x.SignOutAsync(), Times.Once);

}

[TestMethod]

public async Task IsUserInRoleAsync\_ReturnsTrue\_WhenUserIsInRole()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var roleName = "Admin";

// Mock UserManager to return true for IsInRoleAsync

var mockUserManager = MockUserManager();

mockUserManager.Setup(x => x.IsInRoleAsync(user, roleName))

.ReturnsAsync(true);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, mockUserManager.Object, null!, null!, null!);

// Act: Call the IsUserInRoleAsync method

var result = await usersRepository.IsUserInRoleAsync(user, roleName);

// Assert: Verify that the result is true

Assert.IsTrue(result);

}

[TestMethod]

public async Task IsUserInRoleAsync\_ReturnsFalse\_WhenUserIsNotInRole()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var roleName = "Admin";

// Mock UserManager to return false for IsInRoleAsync

var mockUserManager = MockUserManager();

mockUserManager.Setup(x => x.IsInRoleAsync(user, roleName))

.ReturnsAsync(false);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, mockUserManager.Object, null!, null!, null!);

// Act: Call the IsUserInRoleAsync method

var result = await usersRepository.IsUserInRoleAsync(user, roleName);

// Assert: Verify that the result is false

Assert.IsFalse(result);

}

[TestMethod]

public async Task CheckRoleAsync\_DoesNotCreateRole\_WhenRoleExists()

{

// Arrange

var roleName = "Admin";

// Mock RoleManager to return true when checking if the role exists

\_mockRoleManager.Setup(x => x.RoleExistsAsync(roleName))

.ReturnsAsync(true);

// Create the UsersRepository with the mocked RoleManager

var usersRepository = new UsersRepository(null!, null!, \_mockRoleManager.Object, null!, null!);

// Act

await usersRepository.CheckRoleAsync(roleName);

// Assert: Verify that CreateAsync was never called since the role already exists

\_mockRoleManager.Verify(x => x.CreateAsync(It.IsAny<IdentityRole>()), Times.Never);

}

[TestMethod]

public async Task CheckRoleAsync\_CreatesRole\_WhenRoleDoesNotExist()

{

// Arrange

var roleName = "Admin";

// Mock RoleManager to return false when checking if the role exists

\_mockRoleManager.Setup(x => x.RoleExistsAsync(roleName))

.ReturnsAsync(false);

// Mock RoleManager to simulate successful role creation

\_mockRoleManager.Setup(x => x.CreateAsync(It.IsAny<IdentityRole>()))

.ReturnsAsync(IdentityResult.Success);

// Create the UsersRepository with the mocked RoleManager

var usersRepository = new UsersRepository(null!, null!, \_mockRoleManager.Object, null!, null!);

// Act

await usersRepository.CheckRoleAsync(roleName);

// Assert: Verify that CreateAsync was called once with the correct role

\_mockRoleManager.Verify(x => x.CreateAsync(It.Is<IdentityRole>(r => r.Name == roleName)), Times.Once);

}

[TestMethod]

public async Task AddUserToRoleAsync\_AddsUserToRoleSuccessfully()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var roleName = "Admin";

// Mock UserManager to simulate successful role addition

\_mockUserManager.Setup(x => x.AddToRoleAsync(user, roleName))

.ReturnsAsync(IdentityResult.Success);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call the AddUserToRoleAsync method

await usersRepository.AddUserToRoleAsync(user, roleName);

// Assert: Verify that AddToRoleAsync was called once with the correct parameters

\_mockUserManager.Verify(x => x.AddToRoleAsync(user, roleName), Times.Once);

}

[TestMethod]

public async Task GenerateEmailConfirmationTokenAsync\_ReturnsToken()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var expectedToken = "testToken";

// Mock UserManager to return a token when GenerateEmailConfirmationTokenAsync is called

\_mockUserManager.Setup(x => x.GenerateEmailConfirmationTokenAsync(user))

.ReturnsAsync(expectedToken);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call GenerateEmailConfirmationTokenAsync

var result = await usersRepository.GenerateEmailConfirmationTokenAsync(user);

// Assert: Verify that the returned token matches the expected value

Assert.AreEqual(expectedToken, result);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsSuccess()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var token = "validToken";

// Mock UserManager to simulate a successful confirmation

\_mockUserManager.Setup(x => x.ConfirmEmailAsync(user, token))

.ReturnsAsync(IdentityResult.Success);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call ConfirmEmailAsync

var result = await usersRepository.ConfirmEmailAsync(user, token);

// Assert: Verify that the confirmation was successful

Assert.AreEqual(IdentityResult.Success, result);

}

[TestMethod]

public async Task ConfirmEmailAsync\_ReturnsFailure\_WhenInvalidToken()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com" };

var token = "invalidToken";

// Mock UserManager to simulate a failed confirmation

var identityResult = IdentityResult.Failed(new IdentityError { Description = "Invalid token" });

\_mockUserManager.Setup(x => x.ConfirmEmailAsync(user, token))

.ReturnsAsync(identityResult);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call ConfirmEmailAsync with an invalid token

var result = await usersRepository.ConfirmEmailAsync(user, token);

// Assert: Verify that the confirmation failed

Assert.AreEqual(identityResult, result);

Assert.IsFalse(result.Succeeded);

Assert.AreEqual("Invalid token", result.Errors.First().Description);

}

[TestMethod]

public async Task UpdateUserAsync\_ReturnsSuccess\_WhenUserIsUpdatedSuccessfully()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

// Mock UserManager to simulate successful user update

\_mockUserManager.Setup(x => x.UpdateAsync(user))

.ReturnsAsync(IdentityResult.Success);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call UpdateUserAsync

var result = await usersRepository.UpdateUserAsync(user);

// Assert: Verify that the update was successful

Assert.AreEqual(IdentityResult.Success, result);

}

[TestMethod]

public async Task UpdateUserAsync\_ReturnsFailure\_WhenUserUpdateFails()

{

// Arrange

var user = new User { Id = Guid.NewGuid().ToString(), Email = "test@example.com", FirstName = "John", LastName = "Doe" };

// Mock UserManager to simulate failed user update

var identityResult = IdentityResult.Failed(new IdentityError { Description = "Update failed" });

\_mockUserManager.Setup(x => x.UpdateAsync(user))

.ReturnsAsync(identityResult);

// Create UsersRepository with the mocked UserManager

var usersRepository = new UsersRepository(null!, \_mockUserManager.Object, null!, null!, null!);

// Act: Call UpdateUserAsync

var result = await usersRepository.UpdateUserAsync(user);

// Assert: Verify that the update failed

Assert.AreEqual(identityResult, result);

Assert.IsFalse(result.Succeeded);

Assert.AreEqual("Update failed", result.Errors.First().Description);

}

[TestMethod]

public async Task GetTotalRecordsAsync\_WithFilter\_ReturnsFilteredCount()

{

// Arrange: Add users to the in-memory database

var user1 = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe" };

var user2 = new User { Id = Guid.NewGuid().ToString(), FirstName = "Jane", LastName = "Smith" };

var user3 = new User { Id = Guid.NewGuid().ToString(), FirstName = "Michael", LastName = "Johnson" };

await \_context.Users.AddRangeAsync(user1, user2, user3);

await \_context.SaveChangesAsync();

// Create a PaginationDTO with a filter that matches "John"

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "John" };

// Act: Call GetTotalRecordsAsync with the filter

var result = await \_usersRepository.GetTotalRecordsAsync(pagination);

// Assert: Verify that the result is correct (should match 2 users: "John Doe" and "Michael Johnson")

Assert.IsTrue(result.WasSuccess);

Assert.AreEqual(2, result.Result); // Expecting 2 users with "John" in either FirstName or LastName

}

[TestMethod]

public async Task GetAsync\_WithFilter\_ReturnsFilteredUsers()

{

// Arrange: Add users and a country to the in-memory database

var country = new Country { Id = 1, Name = "TestCountry" };

var user1 = new User { Id = Guid.NewGuid().ToString(), FirstName = "John", LastName = "Doe", Country = country };

var user2 = new User { Id = Guid.NewGuid().ToString(), FirstName = "Jane", LastName = "Smith", Country = country };

var user3 = new User { Id = Guid.NewGuid().ToString(), FirstName = "Michael", LastName = "Johnson", Country = country };

await \_context.Countries.AddAsync(country);

await \_context.Users.AddRangeAsync(user1, user2, user3);

await \_context.SaveChangesAsync();

// Create a PaginationDTO with a filter that matches "John"

var pagination = new PaginationDTO { Page = 1, RecordsNumber = 10, Filter = "John" };

// Act: Call GetAsync with the filter

var result = await \_usersRepository.GetAsync(pagination);

// Assert: Verify that the result contains the correct users (should match "John Doe" and "Michael Johnson")

Assert.IsTrue(result.WasSuccess);

var filteredUsers = result.Result!.ToList();

Assert.AreEqual(2, filteredUsers.Count); // Expecting 2 users

Assert.IsTrue(filteredUsers.Any(u => u.FirstName == "John" && u.LastName == "Doe"));

Assert.IsTrue(filteredUsers.Any(u => u.FirstName == "Michael" && u.LastName == "Johnson"));

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

## Otros

### MailHelperTest

1. Adicionamos el **ISmtpClient**:

using MimeKit;

namespace Fantasy.Backend.Helpers;

public interface ISmtpClient

{

void Connect(string host, int port, bool useSsl);

void Authenticate(string username, string password);

void Send(MimeMessage message);

void Disconnect(bool quit);

}

1. Adicione la clase **SmtpClientWrapper**:

using MailKit.Net.Smtp;

using MimeKit;

namespace Fantasy.Backend.Helpers;

public class SmtpClientWrapper : ISmtpClient

{

private readonly SmtpClient \_smtpClient = new SmtpClient();

public void Authenticate(string username, string password) => \_smtpClient.Authenticate(username, password);

public void Connect(string host, int port, bool useSsl) => \_smtpClient.Connect(host, port, useSsl);

public void Disconnect(bool quit) => \_smtpClient.Disconnect(quit);

public void Send(MimeMessage message) => \_smtpClient.Send(message);

}

1. Configuramos la nueva inyección en el **Program**:

builder.Services.AddScoped<ISmtpClient, SmtpClientWrapper>();

1. Modificamos el **MailHelper**, primero inyectamos el **ISmtpClient**:

public ActionResponse<string> SendMail(string toName, string toEmail, string subject, string body, string language)

{

try

{

var from = \_configuration["Mail:From"];

var name = \_configuration["Mail:NameEn"];

if (language == "es")

{

name = \_configuration["Mail:NameEs"];

}

var smtp = \_configuration["Mail:Smtp"];

var port = \_configuration["Mail:Port"];

var password = \_configuration["Mail:Password"];

var message = new MimeMessage();

message.From.Add(new MailboxAddress(name, from));

message.To.Add(new MailboxAddress(toName, toEmail));

message.Subject = subject;

BodyBuilder bodyBuilder = new BodyBuilder

{

HtmlBody = body

};

message.Body = bodyBuilder.ToMessageBody();

\_smtpClient.Connect(smtp!, int.Parse(port!), false);

\_smtpClient.Authenticate(from!, password!);

\_smtpClient.Send(message);

\_smtpClient.Disconnect(true);

return new ActionResponse<string> { WasSuccess = true };

}

catch (Exception ex)

{

return new ActionResponse<string>

{

WasSuccess = false,

Message = ex.Message,

};

}

}

1. Adicione la clase **MailHelperTests**:

using Fantasy.Backend.Helpers;

using Microsoft.Extensions.Configuration;

using MimeKit;

using Moq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Fantasy.Tests.Helpers;

[TestClass]

public class MailHelperTests

{

private Mock<IConfiguration> \_configurationMock = null!;

private Mock<ISmtpClient> \_smtpClientMock = null!;

private MailHelper \_mailHelper = null!;

[TestInitialize]

public void Initialize()

{

\_configurationMock = new Mock<IConfiguration>();

\_smtpClientMock = new Mock<ISmtpClient>();

\_configurationMock.SetupGet(x => x["Mail:From"]).Returns("From");

\_configurationMock.SetupGet(x => x["Mail:Name"]).Returns("Name");

\_configurationMock.SetupGet(x => x["Mail:Smtp"]).Returns("Smtp");

\_configurationMock.SetupGet(x => x["Mail:Port"]).Returns("123");

\_configurationMock.SetupGet(x => x["Mail:Password"]).Returns("Password");

\_mailHelper = new MailHelper(\_configurationMock.Object, \_smtpClientMock.Object);

}

[TestMethod]

public void SendMail\_ShouldReturnSuccessActionResponse()

{

// Arrange

var toName = "John Doe";

var toEmail = "john.doe@example.com";

var subject = "Test Subject";

var body = "Test Body";

var language = "es";

// Act

var response = \_mailHelper.SendMail(toName, toEmail, subject, body, language);

// Assert

Assert.IsTrue(response.WasSuccess);

\_smtpClientMock.Verify(x => x.Connect(It.IsAny<string>(), It.IsAny<int>(), It.IsAny<bool>()), Times.Once);

\_smtpClientMock.Verify(x => x.Authenticate(It.IsAny<string>(), It.IsAny<string>()), Times.Once);

\_smtpClientMock.Verify(x => x.Send(It.IsAny<MimeMessage>()), Times.Once);

\_smtpClientMock.Verify(x => x.Disconnect(It.IsAny<bool>()), Times.Once);

}

[TestMethod]

public void SendMail\_ShouldReturnErrorActionResponse\_WhenExceptionThrown()

{

// Arrange

var toName = "John Doe";

var toEmail = "john.doe@example.com";

var subject = "Test Subject";

var body = "Test Body";

var exceptionMessage = "SMTP error";

var language = "es";

\_smtpClientMock.Setup(x => x.Send(It.IsAny<MimeMessage>())).Throws(new Exception(exceptionMessage));

// Act

var response = \_mailHelper.SendMail(toName, toEmail, subject, body, language);

// Assert

Assert.IsFalse(response.WasSuccess);

Assert.AreEqual(exceptionMessage, response.Message);

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.
3. Hacemos commit.

### FileStorage

1. Adicionamos el **IBlobContainerClient**:

using Azure.Storage.Blobs.Models;

using Azure.Storage.Blobs;

namespace Fantasy.Backend.Helpers;

public interface IBlobContainerClient

{

Task<BlobClient> GetBlobClientAsync(string name);

Task CreateIfNotExistsAsync();

Task SetAccessPolicyAsync(PublicAccessType accessType);

}

1. Adicionamos el **BlobContainerClientWrapper**:

using Azure.Storage.Blobs.Models;

using Azure.Storage.Blobs;

namespace Fantasy.Backend.Helpers;

public class BlobContainerClientWrapper : IBlobContainerClient

{

private readonly BlobContainerClient \_blobContainerClient;

public BlobContainerClientWrapper(string connectionString, string containerName)

{

\_blobContainerClient = new BlobContainerClient(connectionString, containerName);

}

public Task<BlobClient> GetBlobClientAsync(string name) => Task.FromResult(\_blobContainerClient.GetBlobClient(name));

public Task CreateIfNotExistsAsync() => \_blobContainerClient.CreateIfNotExistsAsync();

public Task SetAccessPolicyAsync(PublicAccessType accessType) => \_blobContainerClient.SetAccessPolicyAsync(accessType);

}

1. Adicionamos el **IBlobContainerClientFactory**:

namespace Fantasy.Backend.Helpers;

public interface IBlobContainerClientFactory

{

IBlobContainerClient CreateBlobContainerClient(string connectionString, string containerName);

}

1. Adicionamos el **BlobContainerClientFactory**:

namespace Fantasy.Backend.Helpers;

public class BlobContainerClientFactory : IBlobContainerClientFactory

{

public IBlobContainerClient CreateBlobContainerClient(string connectionString, string containerName) => new BlobContainerClientWrapper(connectionString, containerName);

}

1. Configuramos la nueva inyección en el **Program** del **Backend**:

…

builder.Services.AddScoped<ISmtpClient, SmtpClientWrapper>();

builder.Services.AddScoped<IBlobContainerClientFactory, BlobContainerClientFactory>();

…

1. Modificamos el **FileStorage**:

using Azure.Storage.Blobs.Models;

namespace Fantasy.Backend.Helpers;

public class FileStorage : IFileStorage

{

private readonly string \_connectionString;

private readonly IBlobContainerClientFactory \_blobContainerClientFactory;

public FileStorage(IConfiguration configuration, IBlobContainerClientFactory blobContainerClientFactory)

{

\_connectionString = configuration["ConnectionStrings:AzureStorage"] ?? throw new InvalidOperationException("Connection string 'AzureStorage' not found.");

\_blobContainerClientFactory = blobContainerClientFactory;

}

public async Task RemoveFileAsync(string path, string containerName)

{

var client = \_blobContainerClientFactory.CreateBlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

var fileName = Path.GetFileName(path);

var blob = await client.GetBlobClientAsync(fileName);

await blob.DeleteIfExistsAsync();

}

public async Task<string> SaveFileAsync(byte[] content, string extension, string containerName)

{

var client = \_blobContainerClientFactory.CreateBlobContainerClient(\_connectionString, containerName);

await client.CreateIfNotExistsAsync();

await client.SetAccessPolicyAsync(PublicAccessType.Blob);

var fileName = $"{Guid.NewGuid()}{extension}";

var blob = await client.GetBlobClientAsync(fileName);

using (var ms = new MemoryStream(content))

{

await blob.UploadAsync(ms);

}

return blob.Uri.ToString();

}

}

1. Adicione la clase **FileStorageTests**:

using Azure;

using Azure.Storage.Blobs;

using Azure.Storage.Blobs.Models;

using Microsoft.Extensions.Configuration;

using Moq;

using Orders.Backend.Helpers;

namespace Orders.Tests.Helpers

{

[TestClass]

public class FileStorageTests

{

[TestMethod]

public async Task TestRemoveFileAsync()

{

// Arrange

var configurationMock = new Mock<IConfiguration>();

configurationMock.Setup(x => x["ConnectionStrings:AzureStorage"])

.Returns("fake\_connection\_string");

var blobClientMock = new Mock<BlobClient>();

blobClientMock.Setup(x => x.DeleteIfExistsAsync(It.IsAny<DeleteSnapshotsOption>(), It.IsAny<BlobRequestConditions>(), It.IsAny<CancellationToken>()))

.ReturnsAsync(Response.FromValue(true, Mock.Of<Response>()));

var blobContainerClientMock = new Mock<IBlobContainerClient>();

blobContainerClientMock.Setup(x => x.GetBlobClientAsync(It.IsAny<string>()))

.ReturnsAsync(blobClientMock.Object);

blobContainerClientMock.Setup(x => x.CreateIfNotExistsAsync())

.Returns(Task.CompletedTask);

var blobContainerClientFactoryMock = new Mock<IBlobContainerClientFactory>();

blobContainerClientFactoryMock.Setup(x => x.CreateBlobContainerClient(It.IsAny<string>(), It.IsAny<string>()))

.Returns(blobContainerClientMock.Object);

var fileStorage = new FileStorage(configurationMock.Object, blobContainerClientFactoryMock.Object);

// Act

await fileStorage.RemoveFileAsync("fake\_path", "fake\_container");

// Assert

blobClientMock.Verify(x => x.DeleteIfExistsAsync(It.IsAny<DeleteSnapshotsOption>(), It.IsAny<BlobRequestConditions>(), It.IsAny<CancellationToken>()), Times.Once);

}

[TestMethod]

public async Task TestSaveFileAsync\_Success()

{

// Arrange

var configurationMock = new Mock<IConfiguration>();

configurationMock.Setup(x => x["ConnectionStrings:AzureStorage"])

.Returns("fake\_connection\_string");

var blobClientMock = new Mock<BlobClient>();

var blobContentInfoMock = new Mock<BlobContentInfo>();

var responseMock = new Mock<Response<BlobContentInfo>>();

responseMock.Setup(x => x.Value)

.Returns(blobContentInfoMock.Object);

blobClientMock.Setup(x => x.UploadAsync(It.IsAny<Stream>(), true, default))

.ReturnsAsync(responseMock.Object);

blobClientMock.SetupGet(x => x.Uri)

.Returns(new Uri("http://fake.blob.url"));

var blobContainerClientMock = new Mock<IBlobContainerClient>();

blobContainerClientMock.Setup(x => x.GetBlobClientAsync(It.IsAny<string>()))

.ReturnsAsync(blobClientMock.Object);

blobContainerClientMock.Setup(x => x.CreateIfNotExistsAsync())

.Returns(Task.CompletedTask);

blobContainerClientMock.Setup(x => x.SetAccessPolicyAsync(PublicAccessType.Blob))

.Returns(Task.CompletedTask);

var blobContainerClientFactoryMock = new Mock<IBlobContainerClientFactory>();

blobContainerClientFactoryMock.Setup(x => x.CreateBlobContainerClient(It.IsAny<string>(), It.IsAny<string>()))

.Returns(blobContainerClientMock.Object);

var fileStorage = new FileStorage(configurationMock.Object, blobContainerClientFactoryMock.Object);

// Act

var result = await fileStorage.SaveFileAsync(new byte[] { }, ".txt", "fake\_container");

// Assert

Assert.AreEqual("http://fake.blob.url/", result);

}

}

}

1. Corra los test y verifique que todo está funcionando correctamente.
2. Verificamos la cobertura del código.

**Nota general**: para el resto de clases o métodos que no es posible probar, se puede colocar esta anotación:

[ExcludeFromCodeCoverage(Justification = "It is a wrapper used to test other classes. There is no way to prove it.")]

1. Hacemos commit.

Fin