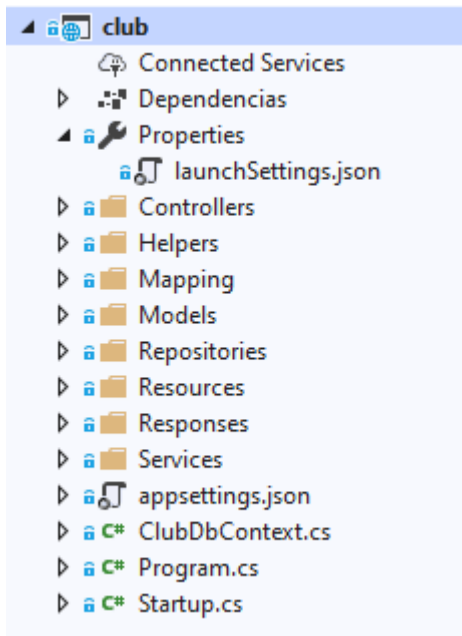


# Memoria Prueba ASP CORE 2.2

## Explicación del desarrollo.

### Estructura del proyecto



Controllors = Controladores de la aplicación, solo recogen y enrutan peticiones, les pasan los datos a los Servicios y devuelven las respuestas que estos les dan. El controller de Usuarios si que realiza un poco mas de funcionalidad para el acceso por JWT.

Helpers = Son un par de clases que usa el Servicio de Usuarios para crear el acceso por token JWT

Mapping = Se tratan de mapeos para convertir los Modelos a Objetos DTO y viceversa. Usan la librería AutoMapper.

Models = Las entidades que mapean las tablas de la BBDD

Repositories = He usado el Patrón Repositorio para conectar la capa de negocio y la de persistencia. Básicamente son un conjunto de funciones reutilizables por varios Services.

Resources = Son los Objetos DTO (Data transfer object) que usará la api para sus Requests y Respuestas al usuario.

Responses = Son los objetos que los Services devuelven a los controladores indicando si la respuesta es correcta o no y que DTO debe devolver la api.

Services = Son los servicios que realizan la lógica de negocio.

## Resources DTOs, Mapeo y Paginación

Los diferentes objetos de los Modelos se mapean como Objetos DTO para devolver mas información y ocultar algunos datos que son específicos de las Entidades de Entity Framework y que no queremos que vean los usuarios de nuestra API (password, ids, etc)

```
var hoursByCourt = todayBookings.GroupBy(b => b.CourtId)
    .ToDictionary(k => k.Key, v => v.Select(f => f.Reservation.ToString("HH:mm")).ToList());

var allCourts = await _courtRepository.ListAsync(pageNum, pageSize);
var resources = _mapper.Map<IEnumerable<Court>, IEnumerable<CourtAvailableResource>>(allCourts);
```

También se da el caso opuesto, que añadamos información. Como en este caso que se añade paginación.

```
7 referencias
public class BaseListResource<T>
{
    1 referencia | 0 excepciones
    public int PageNum { get; set; }
    1 referencia | 0 excepciones
    public int PageSize { get; set; }
    1 referencia | 0 excepciones
    public int TotalItems { get; set; } = 0;
    1 referencia | 0 excepciones
    public IEnumerable<T> Items { get; set; }

    /// <summary>
    /// Creates a List Resource.
    /// </summary>
    /// <param name="pageNum">pageNum.</param>
    /// <param name="pageSize">pageSize.</param>
    /// <param name="totalRecords">totalRecords.</param>
    /// <param name="resources">Items.</param>
    /// <returns>Response.</returns>
    6 referencias | 0 excepciones
    protected BaseListResource(IEnumerable<T> resources, int pageNum, int pageSize, int totalItems)
    {
        PageNum = pageNum;
        PageSize = pageSize;
        TotalItems = totalItems;
        Items = resources;
    }
}
```

Todos los endpoints que devuelven listas soportan paginación, se puede configurar usando añadiendo parámetros a la URL

GET



https://localhost:44318/api/bookings?pageNum=1&pageSize=2

## Resources DTOs de Error

He añadido DataAnnotations a los Objetos DTO, para lanzar la validación automática por parte de ASP Core

```
using System.ComponentModel.DataAnnotations;

namespace club.Resources
{
    8 referencias
    public class CourtSaveResource
    {
        [Required]
        [StringLength(20)]
        1 referencia | 0 excepciones
        public string Reference { get; set; }
        [Required]
        3 referencias | 0 excepciones
        public int SportId { get; set; }
    }
}
```

En mis mensajes de error he copiado el formato de la validación automática:

```
{
  "errors": {
    "MemberId": "Member id not found."
  },
  "title": "Item id not found",
  "status": 404
}
```

## Acceso con Token JWT

Todos los endpoints de la API están protegidos excepto:

POST	/api/users/authenticate
POST	/api/users

El resto requieren autenticación para poder interactuar con ellos.

Para empezar a trabajar con la API habría que:

1. Registrar un usuario con api/users
2. Autenticarse
3. Acceder al resto de endpoints

## Responses

Se trata de objetos que encapsulan las posibles respuestas de un endpoint de la API. Todos heredan de la misma clase base.

```
public abstract class BaseResponse<T>
{
    47 referencias | 0 excepciones
    public bool Success { get; private set; }
    64 referencias | 0 excepciones
    public ErrorResource Error { get; private set; }
    39 referencias | 0 excepciones
    public T Result { get; private set; }

    /// <summary>
    /// Creates a success response.
    /// </summary>
    /// <param name="resource">Item.</param>
    /// <returns>Response.</returns>
    11 referencias | 0 excepciones
    protected BaseResponse(T resource)
    {
        Success = true;
        Error = default;
        Result = resource;
    }

    /// <summary>
    /// Creates an error response.
    /// </summary>
    /// <param name="statusCode">Error status.</param>
    /// <param name="errorCode">Error title.</param>
    /// <param name="errorMessage">Error message.</param>
    /// <returns>Response.</returns>
    11 referencias | 0 excepciones
    protected BaseResponse(int statusCode, string errorTitle, string errorKey, string errorMessage)
    {
        Success = false;
        Error = new ErrorResource(statusCode, errorTitle, errorKey, errorMessage);
        Result = default;
    }
}
```

Por ejemplo si nos falla la validación podemos crear una Resource DTO de error y el controlador se encargara de parsearla a Json en la respuesta con el HTTP Code correcto.

```
if (member == null)
    return new CourtAvailableListResponse(404, "Item id not found", "MemberId", "Member id not found.");

DateTime dateTime = default;

if (!String.IsNullOrEmpty(dateString) && !DateTime.TryParse(dateString, out dateTime))
    return new CourtAvailableListResponse(400, "Wrong date format", "date", "Date format has to be: yyyy-MM-dd");
```

## Interfaces y Servicios

Por lo que he visto existen dos aproximaciones a donde se deben situar los Interfaces que definen los servicios. O bien se crea un fichero por cada clase Interfaz o bien se añaden al mismo fichero del Servicio. Yo me he decidido por la ultima opción.

## Esquema Entidad-Relación y descripción de cada tabla de la base de datos.

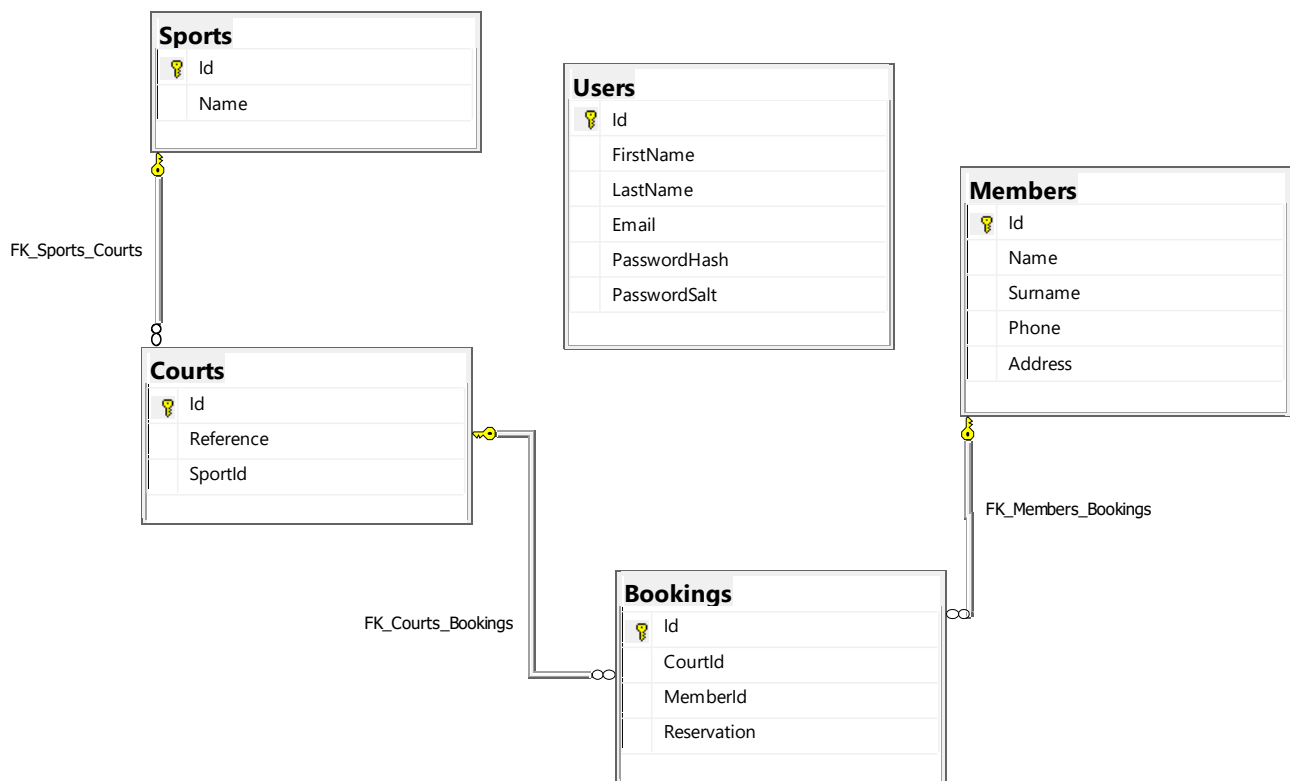


Tabla Users

	Column Name	Data Type	Allow Nulls
🔑	<b>Id</b>	int	<input type="checkbox"/>
	FirstName	varchar(255)	<input type="checkbox"/>
	LastName	varchar(255)	<input type="checkbox"/>
	Email	varchar(80)	<input type="checkbox"/>
	PasswordHash	varbinary(1024)	<input type="checkbox"/>
	PasswordSalt	varbinary(1024)	<input type="checkbox"/>

Tabla Sports

	Column Name	Data Type	Allow Nulls
🔑	<b>Id</b>	int	<input type="checkbox"/>
	Name	varchar(100)	<input type="checkbox"/>

### Tabla Members

	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Name	varchar(255)	<input type="checkbox"/>
	Surname	varchar(255)	<input type="checkbox"/>
	Phone	varchar(20)	<input type="checkbox"/>
	Address	varchar(255)	<input type="checkbox"/>

### Tabla Courts

	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Reference	varchar(20)	<input type="checkbox"/>
	SportId	int	<input type="checkbox"/>

### Tabla Bookings

	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	CourtId	int	<input type="checkbox"/>
	MemberId	int	<input type="checkbox"/>
	Reservation	datetime	<input type="checkbox"/>

## Requisitos del proyecto (versiones de .NET, SQL, Librerías Utilizadas etc. ...)

SQL Server 2017 Express Edition

ASP .NET Core 2.2

Entity Framework Core

Como IDE he usado **Visual Studio Community 2019**

Como gestor de BBDD he usado **SQL Server Management Studio v18.4**

Para probar la API Rest he usado **Postman**

También he usado la Extensión para Visual Studio de **SonarLint**



#### SonarLint for Visual Studio 2019





Roslyn based static code analysis: Find and instantly fix nasty bugs and code smells in C#, VB.Net, C, C+...

**Creado por:** SonarSource



**Fecha de instalación:** 07/12/2019

**Versión:** 4.14.0.12332

En el gestor de paquetes de NuGet podemos ver aquellos que están relacionados con ASP.NET Core


	<b>Microsoft.AspNetCore.App</b> por Microsoft Provides a default set of APIs for building an ASP.NET Core application.	v2.2.0
	<b>Microsoft.AspNetCore.Razor.Design</b> por Microsoft Razor is a markup syntax for adding server-side logic to web pages. This package contains MSBuild support for Razor.	v2.2.0
	<b>Microsoft.NETCore.App</b> por Microsoft A set of .NET API's that are included in the default .NET Core application model. 1249f08feda72b116c7e6e4e9a390671883c797d	v2.2.0
	<b>Microsoft.VisualStudio.Web.CodeGeneration.Design</b> por Microsoft Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	v2.2.4 v3.1.0

También he instalado AutoMapper:

	<b>AutoMapper</b> por Jimmy Bogard A convention-based object-object mapper.	v9.0.0
	<b>AutoMapper.Extensions.Microsoft.DependencyInjection</b> por Jimmy Bogard AutoMapper extensions for ASP.NET Core	v7.0.0

Se trata de un mapeador de objetos para convertir de las Entidades de los modelos a los Data Transfer Objects que devuelve la API.

Y he finalmente he instalado Nswag para hacer funcionar Swagger

	<b>Nswag.AspNetCore</b> por Rico Suter Nswag: The OpenAPI/Swagger API toolchain for .NET and TypeScript	v13.1.6
---	--	---------