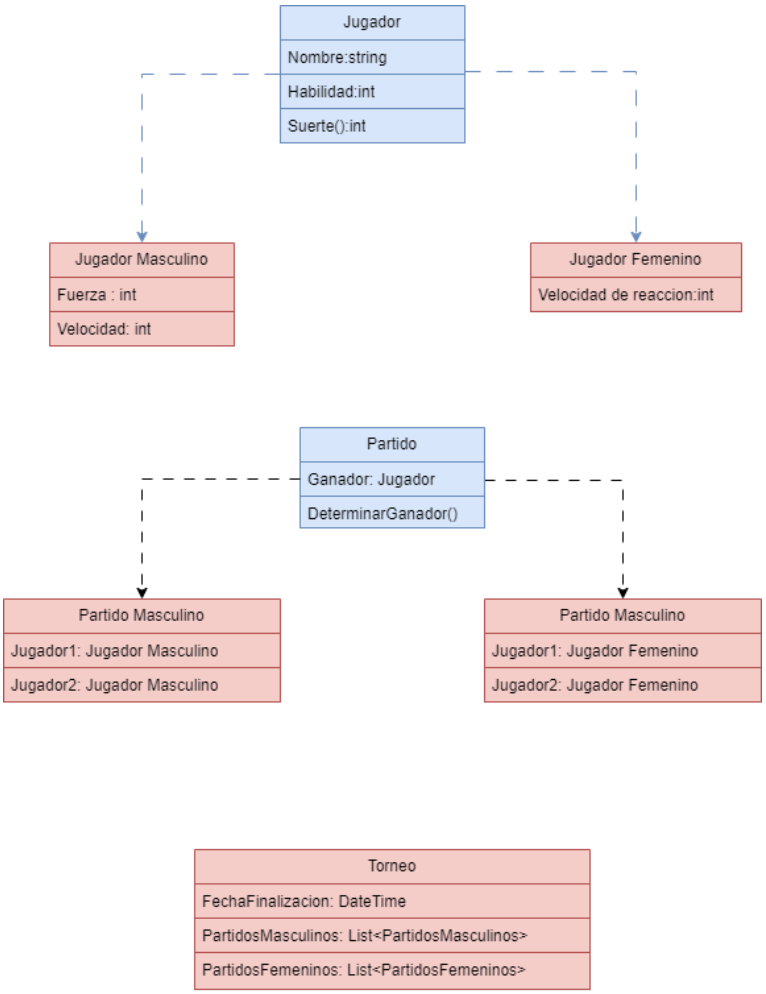
Challenge GeoPagos

Modelado de objetos

Para abordar el desafío propuesto, se implementó un modelo de objetos que se fundamenta en los principios SOLID. Este enfoque se evidencia en la aplicación del Principio de Responsabilidad Única, destacándose en los métodos "Suerte()" de la clase Jugadores y "DeterminarGanador()" de la clase Partidos.
Además, se incorporaron propiedades de herencia para fortalecer el concepto del Principio de Abierto/Cerrado en diversas secciones del código de la aplicación. En las secciones detalladas posteriormente, se observará la aplicación del Principio de Sustitución de Liskov. Cabe destacar que cada objeto cuenta con su propia interfaz asociada, lo que facilita una desagregación más efectiva, respaldada por el Principio de Segregación de Interfaces.



Para el modelo relacional derivado de la base de datos, se adoptó la estrategia "Table-per-Type" (TPT). Esto condujo a la creación de una tabla para jugadores masculinos y otra para jugadores femeninos, así como tablas separadas para los partidos. En el caso de los torneos, se optó por emplear una única tabla. Esta elección no solo simplifica la arquitectura, sino que también contribuye a una estructura más organizada y clara.

Test Realizados

Pruebas de sistema

Caso de prueba 1 - Alta Jugador femenino

Resultado esperado: “El sistema muestra que el nombre es requerido”

1. Ir a la pantalla de jugadores
2. Hacer click en el botón + que está al lado de el título de jugadores femeninos.
3. En la ventana emergente Hacer clic en Guardar

Caso de prueba 2 - Alta Jugador Masculino

Resultado esperado: “El sistema muestra que el nombre es requerido”

1. Ir a la pantalla de jugadores
2. Hacer click en el botón + que está al lado de el título de jugadores masculinos.
3. En la ventana emergente Hacer clic en Guardar

Caso de prueba 3 - Alta Jugador Masculino

Resultado esperado: “El sistema guarda el jugador y lo agrega a la lista”

1. Ir a la pantalla de jugadores
2. Hacer click en el botón + que está al lado de el título de jugadores masculinos.
3. Completar Nombre y establecer los otros valores en los sliders
4. En la ventana emergente Hacer clic en Guardar
5. Comprobar lista de jugadores

Caso de prueba 4 - Alta Jugador Masculino

Resultado esperado: “El sistema guarda el jugador y lo agrega a la lista”

1. Ir a la pantalla de jugadores
2. Hacer click en el botón + que está al lado de el título de jugadores masculinos.
3. Completar Nombre y establecer los otros valores en los sliders
4. En la ventana emergente Hacer clic en Guardar
5. Comprobar lista de jugadores

Caso de prueba 5 - Crear Torneo femenino

Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 6 - Crear Torneo femenino

Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Seleccionar un número impar de jugadores
4. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 7 - Crear Torneo femenino
Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

5. Ir a la pantalla de Torneos
6. Hacer click en el botón + que está al lado del título de la página.
7. Seleccionar un número par de jugadores que permita la creación de una cantidad impar de equipos
8. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 8 - Crear Torneo femenino
Resultado esperado: “El sistema avisa que se creó correctamente y redirige a la lista de torneos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Seleccionar un número de jugadores que permita una cantidad par de equipos.
4. Hacer clic en el Botón “Crear y finalizar Torneo”

Caso de prueba 9 - Crear y finalizar Torneo femenino
Resultado esperado:“El sistema avisa que se creó correctamente muestra al ganador y redirige a la lista de torneos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Seleccionar un número de jugadores que permita una cantidad par de equipos.
4. Hacer clic en el Botón “Crear y finalizar Torneo”

Caso de prueba 10 - Crear Torneo Masculino
Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Cambiar el Switch de tipo de partido que se encuentra en la esquina superior izquierda
4. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 11 - Crear Torneo masculino
Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Cambiar el Switch de tipo de partido que se encuentra en la esquina superior izquierda
4. Seleccionar un número impar de jugadores
5. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 12 - Crear Torneo masculino
Resultado esperado: “El sistema Pide seleccionar una cantidad de jugadores que permita un número par de equipos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Cambiar el Switch de tipo de partido que se encuentra en la esquina superior izquierda
4. Seleccionar un número par de jugadores que permita la creación de una cantidad impar de equipos
5. Hacer clic en el Botón “Crear Torneo”

Caso de prueba 13 - Crear Torneo Masculino
Resultado esperado: “El sistema avisa que se creó correctamente y redirige a la lista de torneos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Cambiar el Switch de tipo de partido que se encuentra en la esquina superior izquierda
4. Seleccionar un número de jugadores que permita una cantidad par de equipos.
5. Hacer clic en el Botón “Crear y finalizar Torneo”

Caso de prueba 14 - Crear y finalizar Torneo Masculino
Resultado esperado:“El sistema avisa que se creó correctamente muestra al ganador y redirige a la lista de torneos”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón + que está al lado del título de la página.
3. Seleccionar un número de jugadores que permita una cantidad par de equipos.
4. Hacer clic en el Botón “Crear y finalizar Torneo”

Caso de prueba 15 - Finalizar Torneo
Resultado esperado:“El sistema avisa la finalización del torneo y muestra el ganador”

1. Ir a la pantalla de Torneos
2. Hacer click en el botón “Finalizar” del torneo que se desea finalizar.

Caso de prueba 16 - Eliminar Torneo
Resultado esperado:“El sistema avisa que se eliminó correctamente el torneo y muestra su Id”

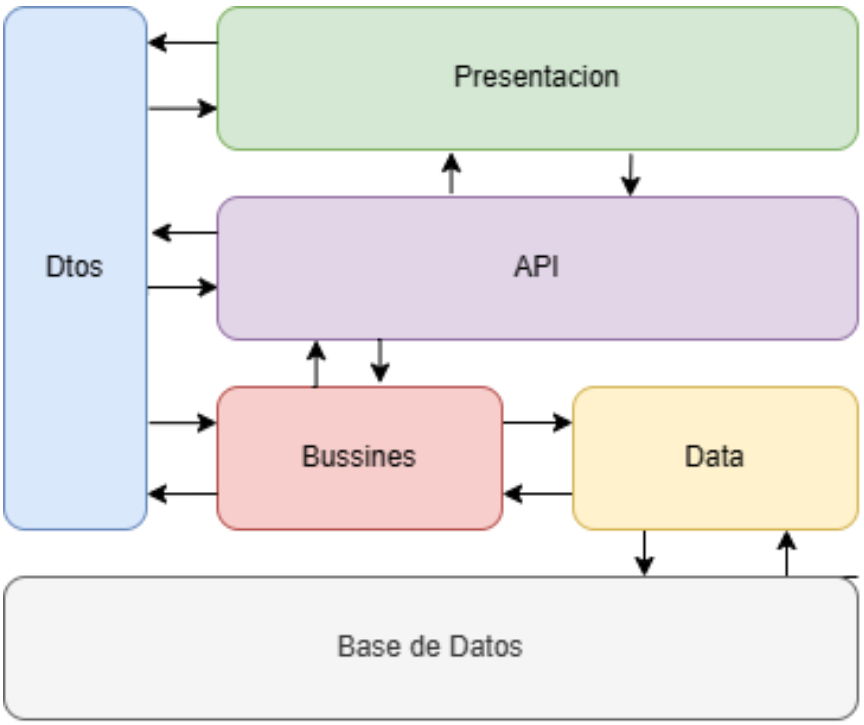
1. Ir a la pantalla de Torneos
2. Hacer click en el botón con el icono de eliminar del torneo que se desea eliminar.

Estilo de código

En relación al estilo de codificación, se aplicó la metodología "Code First", permitiendo un seguimiento dinámico de la evolución de la base de datos mediante la incorporación progresiva de migraciones. Con el fin de mantener un alto nivel de modularidad y flexibilidad, se implementó la inyección de dependencias en toda la aplicación. Para mejorar el rendimiento de las consultas que involucran información de múltiples tablas, se empleó SplitQuery (consultas divididas), optimizando así la sintaxis en el motor de búsqueda de la base de datos.
Además, se incorporaron diversos patrones de diseño. Por ejemplo, el Patrón Repositorio proporciona una abstracción entre la lógica del negocio y el acceso a los datos al centralizar las operaciones de almacenamiento y recuperación, facilitando la mantenibilidad y extensibilidad del código. El Patrón Service agrupa y encapsula la lógica de negocio en servicios independientes, fomentando la reutilización y prueba de dicha lógica al separarla de las capas de presentación y acceso a datos.
También se introdujo el Patrón DTO para transferir datos de manera eficiente entre componentes del sistema, agrupando campos en objetos simples para reducir la cantidad de llamadas entre capas y mejorar el rendimiento. Además, se utilizó el Patrón Mapper para mapear datos entre diferentes tipos, facilitando la conversión entre objetos de dominio y DTOs.
Este enfoque busca estructurar y organizar de manera eficaz el código. La adopción de las últimas sintaxis implementadas en .NET 8 asegura un código actualizado y compatible con las tecnologías más recientes. Asimismo, se incorporaron etiquetas en los atributos de las clases (Decorators), contribuyendo significativamente a una mejor comprensión y documentación del código.

Arquitectura

Se eligió adoptar una arquitectura dividida por capas, enfocándonos especialmente en las ventajas que ofrece, como la Separación de Responsabilidades, la Facilitación del Mantenimiento y la Reutilización de Código.



En el desarrollo de la API, se eligieron tecnologías avanzadas para garantizar un rendimiento óptimo y una implementación eficiente. Se utilizó .NET 8, con exportación a una imagen de Docker para facilitar el despliegue, alojada y operativa en Azure, ofreciendo así un entorno altamente escalable.
La capa de presentación se construyó mediante Blazor Web Assembly, brindando una integración suave con C# La misma también se encuentra desplegada en azure. Las capas Dtos, Data y Business son bibliotecas de clases desarrolladas en .NET 8, asegurando coherencia y eficacia en la estructura del código.
Como sistema de gestión de bases de datos, se optó por SQL Server, respaldado por Azure Services, proporcionando una implementación robusta y escalable que se alinea con las necesidades del proyecto. Esta elección de tecnologías se hizo con el objetivo de asegurar un desarrollo moderno y sólido para la API..

Conocimiento Funcional

A continuación se describen las funcionalidades dentro de la aplicación.

```
public void DeterminarGanador()
{
    if (Jugador1 == null || Jugador2 == null)
        return;

    var random = new Random();
    var probabilidadBase = 50;
    var probabilidadJugador1 = probabilidadBase + Jugador1.Suerte(Jugador2);

    var resultado = random.Next(0, 100);
     if (resultado <= probabilidadJugador1)
    {
        IdGanador = Jugador1.Id;
        Ganador = Jugador1;
     }
     else
    {
       IdGanador = Jugador2.Id;
        Ganador = Jugador2;
     }

}
```

En el siguiente método se describe el proceso para determinar al ganador de un partido. Inicialmente, se establece una probabilidad base del 50%, que representa las posibilidades de ganar en el caso de que ambos jugadores sean idénticos. Dado que cada jugador tiene atributos distintivos, se calcula la variable "Suerte" en función de estos atributos, la cual se suma o resta a la probabilidad base.
A continuación, se genera un número aleatorio entre 0 y 100. Si este número se encuentra dentro del rango de la probabilidad de ganar del jugador 1, se declara a este jugador como el ganador; de lo contrario, se declara ganador al jugador 2. Este enfoque considera las diferencias individuales entre los jugadores, incorporando la noción de "Suerte" como un factor dinámico en la determinación del resultado del partido.

```
public int Suerte(IJugadorDto contrincante)
{
    var suertePorHabilidad = NivelHabilidad - contrincante.NivelHabilidad;
    var suertePorTiempoDeReaccion = TiempoDeReaccion - ((JugadorFemeninoDto)contrincante).TiempoDeReaccion;
    return Math.Clamp(suertePorHabilidad + suertePorTiempoDeReaccion, -40, 40);
}
```

Para calcular la "Suerte" de una jugadora, en este caso, una jugadora femenina (análogo para el jugador masculino con sus respectivos atributos), se toma como parámetro a su contrincante. Se definen dos tipos de suerte: la suerte por habilidad, que depende de la diferencia entre los puntos de habilidad del jugador y su contrincante, y la suerte por tiempo de reacción, que sigue un proceso similar al de la suerte por habilidad.

Una vez calculadas todas las "Suertes" según los atributos correspondientes, se suman y se acotan en un rango máximo de variación entre (-40;40) para evitar variaciones extremas. El resultado se devuelve como un valor entero para su utilización en cálculos subsiguientes. Este enfoque permite incorporar de manera equilibrada diferentes aspectos de la habilidad y el tiempo de reacción de los jugadores en el cálculo de la "Suerte".

```
[NotMapped]
public string Ganador
{
    get
    {

        var partidosConGanadores = new List<IPartido>();
        partidosConGanadores.AddRange(PartidosMasculinos.Where(p => p.IdGanador != null));
        partidosConGanadores.AddRange(PartidosFemeninos.Where(p => p.IdGanador != null));


        var idGanador = partidosConGanadores.Find(p=>p.IPartidoSiguiente==null)?.IdGanador;

        if (idGanador == null)
            return "Aun no hay ganador";

        var ganadorMasculino = PartidosMasculinos.Find(p => p.IdGanador == idGanador)?.Ganador;
        var ganadorFemenino = PartidosFemeninos.Find(p => p.IdGanador == idGanador)?.Ganador;

        if (ganadorMasculino != null)
            return ganadorMasculino.Nombre;
        else if (ganadorFemenino != null)
            return ganadorFemenino.Nombre;
        else
            return "Aun no hay ganador";

    }
    set { }
}
```

Este atributo es un derivado de la clase "Torneo" y tiene la función de proporcionar el nombre del ganador del torneo, en caso de que exista. Para lograrlo, primero agrupa todos los partidos del torneo que cuentan con un ganador. A partir de esta agrupación, se selecciona el último partido, identificado por no tener asignado un partido siguiente. Finalmente, se devuelve el nombre del ganador de este último partido como resultado del atributo. Este enfoque asegura que se obtenga el nombre del ganador del torneo basándose en el último partido disputado sin un siguiente en la secuencia.

```
public ITorneoDto? FinalizarTorneo(int id)
{
    var torneo = RepositorioTorneos.GetById(id);
    if (torneo == null)
        return null;

    var torneoDto = Mapper.Map<Torneo, TorneoDto>(torneo as Torneo);
    var listaDePartidos = new List<IPartidoDto>(torneoDto.PartidosMasculinos);
    listaDePartidos.AddRange(torneoDto.PartidosFemeninos);

    while (listaDePartidos.Count >= 1)
    {
        listaDePartidos = ServicePartidos.DeterminarGanadores(listaDePartidos);
        if (listaDePartidos.Count % 2 == 0)
        {
            var nuevosPartidos = new List<IPartidoDto>();
            for (int i = 0; i < listaDePartidos.Count; i += 2)
            {
                var proximoPartido = ServicePartidos.CrearProximoPartido(listaDePartidos[i], listaDePartidos[i + 1]);
                if (proximoPartido != null)
                    nuevosPartidos.Add(proximoPartido);
            }
            listaDePartidos = nuevosPartidos;
        }
        else
        {
            ServicePartidos.DeterminarGanadores(listaDePartidos);
            listaDePartidos = new List<IPartidoDto>();
        }
    }
    RepositorioTorneos.FinalizarTorneo(torneoDto.Id);

    return torneoDto;
}
```

El método de finalización del torneo en el servicio recibe como parámetros el ID del torneo y con el cual solicita a la base de datos el correspondiente torneo a finalizar. En primer lugar, guarda la lista de los partidos actuales en una variable y luego itera sobre la lista hasta que solo quede un único partido.En la iteración, Utilizando el servicio de partidos, finaliza todos los partidos en la lista.
Posteriormente, verifica que la cantidad de partidos sea divisible por 2. En caso afirmativo, crea una nueva lista de partidos vacía y la llena mediante la iteración de un bucle que incrementa de dos en dos. De esta manera, se definen los siguientes partidos con la ayuda del servicio de partidos y se agregan a la lista recién creada. Luego, esta nueva lista reemplaza a la lista de partidos inicial, reduciendo a la mitad la cantidad de partidos.
Este proceso se repite hasta que se alcanza el último partido del torneo. Al salir del bucle while, se instruye al repositorio de torneos que marque el torneo como finalizado, asignándole la fecha de finalización correspondiente. Este enfoque estructurado asegura la progresiva conclusión del torneo, actualizando la información en el repositorio a medida que avanza el proceso.

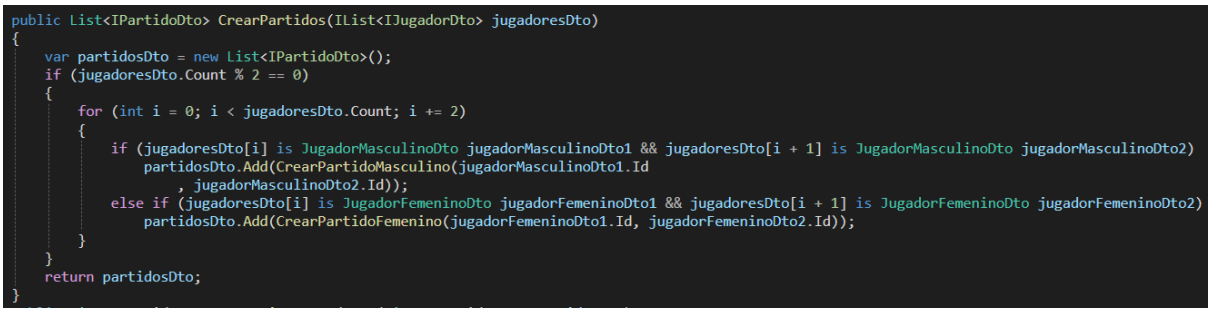
```
public IPartidoDto? CrearProximoPartido(IPartidoDto partidoDto1, IPartidoDto partidoDto2)
{
    if (partidoDto1.IdGanador == null
        || partidoDto2.IdGanador == null
        || partidoDto1.IdTorneo == null
        || partidoDto2.IdTorneo == null
        || partidoDto1.IdTorneo != partidoDto2.IdTorneo
        )
        return null;

    var IdGanador1 = (int)partidoDto1.IdGanador;
    var IdGanador2 = (int)partidoDto2.IdGanador;
    var IdTorneo = (int)partidoDto1.IdTorneo;

    if (partidoDto1 is PartidoMasculinoDto partidoMasculinoDto1 && partidoDto2 is PartidoMasculinoDto partidoMasculinoDto2)
        return CrearPartidoMasculino(IdGanador1, IdGanador2, IdTorneo, partidoMasculinoDto1, partidoMasculinoDto2);
    else if (partidoDto1 is PartidoFemeninoDto partidoFemeninoDto1 && partidoDto2 is PartidoFemeninoDto partidoFemeninoDto2)
        return CrearPartidoFemenino(IdGanador1, IdGanador2, IdTorneo, partidoFemeninoDto1, partidoFemeninoDto2);

    return null;
}
```

Este método, perteneciente al servicio de partidos, tiene como objetivo crear y devolver el próximo partido. Recibe como parámetros dos partidos, y realiza la verificación de la no nulidad tanto del ID del torneo como de los ganadores de los partidos. Posteriormente, se determina si el partido es de categoría masculina o femenina, y luego se procede a la creación del siguiente partido en la secuencia. Este enfoque asegura que el método pueda generar el próximo partido de manera adecuada, teniendo en cuenta las condiciones necesarias y la categoría correspondiente.

```
private IPartidoDto? CrearPartidoFemenino(int IdGanador1, int IdGanador2, int IdTorneo, PartidoFemeninoDto partidoFemeninoDto1, PartidoFemeninoDto partidoFemeninoDto2)
{
    var partidoFemenino = new PartidoFemenino();
    partidoFemenino.IdJugador1 = IdGanador1;
    partidoFemenino.IdJugador2 = IdGanador2;
    partidoFemenino.IdTorneo = IdTorneo;
    var nuevopartidoDb = RepositorioPartidosFemeninos.Add(partidoFemenino);
    PartidoFemenino nuevopartido = nuevopartidoDb as PartidoFemenino;
    partidoFemeninoDto1.IPartidoSiguiente = nuevopartido.Id;
    partidoFemeninoDto2.IPartidoSiguiente = nuevopartido.Id;
    RepositorioPartidosFemeninos.Update(Mapper.Map<PartidoFemeninoDto, PartidoFemenino>(partidoFemeninoDto1));
    RepositorioPartidosFemeninos.Update(Mapper.Map<PartidoFemeninoDto, PartidoFemenino>(partidoFemeninoDto2));
    return Mapper.Map<PartidoFemenino, PartidoFemeninoDto>(nuevopartido);
}
```

El siguiente método para la creación de un partido femenino (similar a la lógica para el masculino) comienza creando una nueva instancia de partido femenino. A esta instancia se le asigna el ID del torneo actual y los IDs de los jugadores ganadores de los partidos anteriores. Se guarda la instancia para obtener el ID generado por la base de datos. Posteriormente, se asigna este ID como el ID del partido siguiente a los partidos anteriores, y se actualiza la información en la base de datos.

Luego, se realiza el mapeo del partido creado y se devuelve. Este enfoque asegura la creación correcta de un nuevo partido femenino, manteniendo la coherencia con los resultados anteriores y actualizando la información en la base de datos de manera eficiente.



```
public List<IPartidoDto> CrearPartidos(IList<IJugadorDto> jugadoresDto)
{
    var partidosDto = new List<IPartidoDto>();
    if (jugadoresDto.Count % 2 == 0)
    {
        for (int i = 0; i < jugadoresDto.Count; i += 2)
        {
            if (jugadoresDto[i] is JugadorMasculinoDto jugadorMasculinoDto1 && jugadoresDto[i + 1] is JugadorMasculinoDto jugadorMasculinoDto2)
                partidosDto.Add(CrearPartidoMasculino(jugadorMasculinoDto1.Id
                    , jugadorMasculinoDto2.Id));
            else if (jugadoresDto[i] is JugadorFemeninoDto jugadorFemeninoDto1 && jugadoresDto[i + 1] is JugadorFemeninoDto jugadorFemeninoDto2)
                partidosDto.Add(CrearPartidoFemenino(jugadorFemeninoDto1.Id, jugadorFemeninoDto2.Id));
        }
    }
    return partidosDto;
}
```

El siguiente método es utilizado por el servicio de partidos para crear partidos basados en una lista dada de jugadores. Para ello, inicialmente se crea una lista de partidos vacía. Luego, se valida que la cantidad de jugadores sea divisible por dos. A continuación, mediante un bucle for, se itera sobre la lista de jugadores con incrementos de dos. Durante esta iteración, se clasifican a los jugadores según si son femeninos o masculinos para el partido correspondiente, y se agregan a la lista de partidos. Finalmente, la lista de partidos se devuelve como resultado. Este enfoque asegura la creación estructurada de partidos a partir de la lista de jugadores proporcionada, manteniendo la coherencia y eficiencia en el proceso.

Reglas de negocio y alcances considerados

- RN: La cantidad de equipos de un torneo siempre es par.
- RN: Un torneo se considera finalizado cuando hay un ganador y tiene establecida la fecha en que finalizó.
- Alcance: para simplicidad del objetivo se omitieron a nivel API y Presentación las funciones Sig. funciones siendo A: alta B: baja C:Consulta y M: Modificación
    - Partidos: ABMC
    - Jugadores: BM
    - Torneos:M