TEST DESARROLLADOR BACKEND

TORNEO DE TENIS



Valdivia Ignacio Ariel

Febrero 2025

Contenido

[Consigna del Test: 2](#_Toc190089486)

[1. Planteamiento del Problema 3](#_Toc190089487)

[2. Arquitectura del Proyecto 3](#_Toc190089488)

[3. Diseño del Servicio y Lógica del Torneo 3](#_Toc190089489)

[4. Persistencia de Datos 4](#_Toc190089490)

[5. Pruebas y Testing 4](#_Toc190089491)

[6. API en .NET 4](#_Toc190089492)

Consigna del Test:

Se desea modelar el comportamiento de un torneo de tenis.

* La modalidad del torneo es por **eliminación directa**.
* Se puede asumir que la cantidad de jugadores es una **potencia de 2**.
* El torneo puede ser **Femenino o Masculino**.
* Cada jugador tiene un **nombre** y un **nivel de habilidad** (valor entre 0 y 100).
* En un enfrentamiento entre dos jugadores influyen el **nivel de habilidad** y la **suerte** para decidir al ganador. La suerte se define como el usuario lo desee en su diseño.
* En el **torneo masculino**, se deben considerar la **fuerza** y la **velocidad de desplazamiento** como parámetros adicionales para calcular el ganador.
* En el **torneo femenino**, se debe considerar el **tiempo de reacción** como parámetro adicional para calcular el ganador.
* **No existen empates**.
* Se requiere que, a partir de una lista de jugadores, se **simule el torneo** y se obtenga como resultado **el ganador del mismo**.
* Se recomienda realizar la solución en su IDE preferido.
* Se valorarán las **buenas prácticas de Programación Orientada a Objetos (POO)**.
* Se puede definir por parte del usuario cualquier **cuestión adicional** que considere no aclarada en la consigna.
* Se pueden agregar las **aclaraciones** que se consideren necesarias en la entrega del ejercicio.
* **Cualquier extra que aporte será bienvenido.**
* Se prefiere el **modelado en capas** o el uso de **arquitecturas limpias (Clean Architecture)**.
* Se prefiere la entrega del código en un **sistema de versionado** como GitHub, GitLab o Bitbucket.

**Nota sobre eliminación directa:**  
El sistema de eliminación directa implica que el perdedor de cada enfrentamiento es eliminado del torneo, mientras que el ganador avanza a la siguiente fase. El proceso continúa hasta que solo queda un **campeón**.

**Puntos extra (Opcionales)**

1. **Testing de la solución** (Unit Test).
2. **API REST** (Swagger + Integration Testing):
   * Dado una lista de jugadores, retorna el resultado del torneo.
   * Permite consultar el resultado de torneos finalizados exitosamente, con algunos filtros como:
     + Torneo Masculino y/o Femenino.
     + Otros criterios a definir.
3. **Uso de una base de datos** (en lugar de datos embebidos).
4. **Subir el código a un repositorio en GitHub/GitLab**.
5. **Desplegar la solución en un servicio como AWS/Azure**, usando **Docker o Kubernetes**.

1. Planteamiento del Problema

Se quiere desarrollar un sistema que simule un **torneo de tenis por eliminación directa**, donde:

* Los jugadores compiten en enfrentamientos 1 vs 1 hasta que quede un único campeón.
* Las reglas de victoria dependen de ciertos atributos y un factor de suerte.

Para esto, se aplicará un enfoque **orientado a objetos (POO)** con **principios SOLID**, arquitectura en **capas** y buenas prácticas de desarrollo.

2. Arquitectura del Proyecto

Se utilizará una arquitectura **en capas**, lo que permitirá:

* Separar las responsabilidades del código.
* Facilitar la **escalabilidad** y el mantenimiento.
* Posibilitar la **integración con bases de datos y APIs**.

Laravel sigue el patrón **MVC**, y para seguir buenas prácticas en **POO y SOLID**, organizaremos el código en:

**Capa de Dominio (app/Models/)**

* Modelos que representan el negocio (Jugador, Torneo, Partida).
* Contiene validaciones de datos y relaciones con otros modelos.

**Capa de Aplicación (app/Services/)**

* Implementa la lógica del torneo en una clase de servicio (TorneoService.php).
* Separa la lógica de negocio de los controladores.
* Facilita la reutilización y testing.

**Capa de Infraestructura (database/, routes/api.php)**

* Base de datos (migraciones y seeders para poblar jugadores).
* Rutas y controladores REST en app/Http/Controllers/.

**Capa de Presentación (resources/views/)**

* Laravel con plantillas Blade.
* Se devuelven respuestas JSON con controladores REST para la API.

3. Diseño del Servicio y Lógica del Torneo

Para manejar la lógica del torneo, seguiremos estos pasos:

**Inicializar un torneo con una lista de jugadores.**

* Validar que el número de jugadores es una potencia de 2.
* Dividir los jugadores en enfrentamientos iniciales.

**Realizar enfrentamientos y avanzar rondas.**

* Aplicar reglas para definir el ganador de cada partida.
* Eliminar jugadores perdedores.
* Repetir hasta obtener un campeón.

**Devolver el resultado final.**

* Mostrar el campeón y el historial de enfrentamientos.

Cada partido evaluará:

* **Nivel de habilidad del jugador.**
* **Parámetros adicionales (fuerza, velocidad, reacción).**
* **Factor de suerte.**

Laravel permite desacoplar la lógica de negocio en **services**, por lo que TorneoService.php manejará el torneo.

**Flujo de ejecución**

1. **El usuario envía una lista de jugadores.**
2. **El servicio organiza los enfrentamientos** según eliminación directa.
3. **Cada enfrentamiento aplica las reglas del torneo** (habilidad + factores adicionales).
4. **Se eliminan los perdedores y se avanza de ronda** hasta que haya un campeón.
5. **Se retorna el resultado final**.

4. Persistencia de Datos

Dado que Laravel usa **Eloquent ORM**, podemos definir modelos para persistir los datos en **MySQL**.

5. Pruebas y Testing

Laravel tiene soporte nativo para **PHPUnit**, lo que nos permite escribir pruebas unitarias y de integración para TorneoService.

6. API en .NET

Crear un **microservicio en .NET** que consuma la API de Laravel y exponga los datos a otros sistemas o usuarios externos.

* .NET manejaría **API pública para consultas externas**.