## MARCO TEÓRICO

### Torneo:

Un torneo es un evento en el que un grupo de competidores participa en una serie de encuentros o rondas para determinar el ganador o un ranking final. Los torneos pueden tener un formato único o combinaciones de formatos, según el objetivo y las reglas establecidas. La finalidad de un torneo puede variar, desde reconocer la excelencia o habilidad en un área específica hasta fomentar el desarrollo social, cultural o económico a través de la competencia.

Existen múltiples formas de organizar un torneo, las más comunes son:

- **Torneos de eliminación simple**: En este formato, los competidores que pierden son eliminados y no pueden seguir participando. El torneo continúa hasta que solo queda un ganador.
- **Torneos de eliminación doble**: Similar a la eliminación simple, pero permite a los participantes que pierden una vez continuar compitiendo en una fase de consolación.
- **Torneos por grupos o liga (round-robin)**: Todos los participantes se enfrentan entre sí y el ganador es determinado por el número total de victorias, empates o puntos acumulados.
- **Torneos suizos**: En este formato, los competidores no juegan contra todos, sino que en cada ronda se enfrentan a oponentes con puntajes similares.

### Partido:

Un partido se refiere a una instancia de competencia entre dos o más partes en un evento específico. Este evento puede estar relacionado con deportes, juegos, política o cualquier otro campo que implique una forma de competencia o colaboración. En cada contexto, el término "partido" puede tener un significado y unas características distintas.

En deportes, un partido es una competición o enfrentamiento entre dos equipos o individuos que se lleva a cabo según un conjunto de reglas específicas del deporte en cuestión. Cada partido tiene un objetivo claro, que suele ser ganar el juego, acumular puntos o lograr una determinada meta.

Componentes:

- Equipos o Jugadores: Los participantes que compiten en el partido.

- Reglas: Normas que rigen el desarrollo del partido y aseguran un juego justo.
- Arbitraje: Oficiales o árbitros que supervisan el partido y aplican las reglas.
- Duración: Tiempo establecido para la realización del partido, que puede variar según el deporte.
- Resultados: El resultado del partido, que puede ser una victoria, derrota o empate.

Ejemplo: Un partido de fútbol entre el FC Barcelona y el Real Madrid, donde cada equipo compite para ganar el partido y sumar puntos en la liga.

**Jugador:**
Es una persona que participa en el deporte del fútbol (soccer en algunos países) como competidor, ya sea de manera profesional, semiprofesional o amateur. Su rol principal es desempeñarse en el campo de juego siguiendo las reglas del deporte con el objetivo de contribuir al éxito de su equipo.
**Árbitro:**
Es un oficial encargado de hacer cumplir las reglas del juego durante un partido de fútbol. Su rol principal es garantizar que el juego se desarrolle de acuerdo con las normas establecidas y resolver cualquier controversia que surja en el campo de juego.
**Tabla de posiciones:**
Es una representación organizada de los equipos en una liga o torneo deportivo, basada en su rendimiento y resultados durante la competición. Muestra cómo se ubican los equipos en función de una serie de criterios, generalmente relacionados con el número de victorias, derrotas y puntos obtenidos.

**Equipo de Fútbol:**
Es una agrupación de jugadores que compiten juntos en partidos y torneos de fútbol. Cada equipo está conformado por una serie de jugadores, cada uno con roles específicos en el campo, y tiene el objetivo de ganar partidos y, en última instancia, alcanzar el éxito en competiciones y ligas.

## Entradas:

1. Datos de Jugadores

- Información Personal: Nombre, fecha de nacimiento, nacionalidad, dirección, contacto.
- Historial Médico: Lesiones anteriores, estado de salud, informes médicos.
- Datos de Rendimiento: Estadísticas de partidos, métricas de entrenamiento, evaluaciones de habilidades.
- Contratos y Acuerdos: Términos del contrato, duración, salario, cláusulas especiales.

2. Datos de Entrenamientos

- Programas de Entrenamiento: Planes de entrenamiento, objetivos específicos, rutinas.
- Asistencia: Registro de asistencia a las sesiones de entrenamiento.
- Evaluaciones: Resultados de evaluaciones de habilidades y progreso.

3. Datos de Partidos

- Detalles del Partido: Fecha, hora, ubicación, equipo rival.
- Tácticas y Estrategias: Formaciones, tácticas utilizadas, planes de juego.
- Estadísticas del Partido: Resultados, goles, asistencias, tarjetas, estadísticas de posesión.

4. Datos de Torneos y Competiciones

- Calendario de Competiciones: Fechas, ubicaciones, formatos.
- Clasificaciones y Resultados: Posiciones en la liga, resultados de partidos, puntos obtenidos.
- Reglas y Normativas: Reglas específicas de cada torneo, criterios de clasificación.

5. Datos Administrativos

- Presupuesto y Finanzas: Información sobre ingresos, gastos, salarios.
- Gestión de Recursos: Equipamiento, instalaciones, logística.
- Relaciones con patrocinadores: Acuerdos de patrocinio, informes de patrocinio.

6. Datos de Cuerpo Técnico

- Información del Personal: Detalles sobre entrenadores, asistentes, preparadores físicos.

- Evaluaciones y Reportes: Informes sobre el desempeño del cuerpo técnico, estrategias utilizadas.

## Salidas:

1. Informes de Rendimiento

- Estadísticas de Jugadores: Goles, asistencias, tarjetas, minutos jugados, rendimiento en entrenamientos.
- Análisis de Partidos: Resultados, estadísticas detalladas, análisis de tácticas y estrategias.
- Evaluaciones: Informes sobre la evolución y progreso de jugadores.

2. Tablas de Posiciones

- Clasificación en Ligas y Torneos: Posición en la tabla, puntos obtenidos, diferencia de goles.
- Estadísticas de Competencias: Resultados de los partidos, puntos acumulados, ranking de equipos.

3. Informes Financieros

- Balance Presupuestario: Ingresos y gastos, presupuesto disponible, informes financieros periódicos.
- Datos de Contratos: Información sobre salarios, bonificaciones y otros acuerdos económicos.

4. Calendarios y Horarios

- Calendario de Partidos: Fechas y horarios de los partidos, ubicaciones de los encuentros.
- Programas de Entrenamiento: Horarios y detalles de las sesiones de entrenamiento.

5. Datos de Entrenamiento

- Resultados de Evaluaciones: Evaluaciones de rendimiento durante las sesiones de entrenamiento.

- Progresos: Informe sobre el progreso individual y colectivo en entrenamientos.

6. Análisis de Tácticas

- Informes de Estrategias: Evaluación de tácticas utilizadas en partidos, análisis de efectividad.
- Planificación Táctica: Recomendaciones para futuros partidos y entrenamientos.

7. Reportes de Lesiones y Estado de Salud

- Historial Médico: Informes sobre el estado de salud de jugadores, historial de lesiones y recuperaciones.
- Recomendaciones Médicas: Consejos y pautas de tratamiento y recuperación.

**Componentes:**

- Torneos
- Equipos
- Jugadores
- Tablas de posiciones
- árbitros
- Estadios
- Canchas

**POO:**

Programación Orientada a Objetos (POO) en PHP es un paradigma de programación que utiliza "objetos" y "clases" para organizar y estructurar el código. La POO permite modelar conceptos del mundo real en el software mediante el uso de objetos que interactúan entre sí. En PHP, la POO ofrece una forma de programar que facilita la reutilización de código, la escalabilidad y el mantenimiento del software.

**Clases**

- **Definición**: Una clase es una plantilla o modelo para crear objetos. Define atributos (propiedades) y métodos (funciones) que los objetos creados a partir de la clase tendrán.

**Ejemplo**:
php

```
class Coche {
    public $marca;
    public $modelo;

    public function arrancar() {
        echo "El coche está arrancando.";
    }
}
```

**Objetos**

- **Definición**: Un objeto es una instancia de una clase. Representa una entidad concreta con atributos y comportamientos definidos por su clase.

**Ejemplo**:
php

```
$miCoche = new Coche();
$miCoche->marca = "Toyota";
$miCoche->modelo = "Corolla";
$miCoche->arrancar();  // Imprime: El coche está arrancando.
```

**Encapsulamiento**

- **Definición**: El encapsulamiento oculta los detalles internos de los objetos y sólo expone lo necesario a través de métodos públicos. Esto protege los datos y reduce la complejidad.

**Ejemplo**:
php

```php
class CuentaBancaria {
    private $saldo;

    public function depositar($monto) {
        $this->saldo += $monto;
    }

    public function obtenerSaldo() {
        return $this->saldo;
    }
}
```

**Herencia**

- **Definición**: La herencia permite a una clase (subclase) heredar atributos y métodos de otra clase (superclase). Esto facilita la reutilización del código y la creación de jerarquías de clases.

**Ejemplo**:
php

```php
class Vehiculo {
    public function encender() {
        echo "El vehículo está encendido.";
    }
}

class Moto extends Vehiculo {
    public function hacerCaballito() {
        echo "La moto está haciendo un caballito.";
    }
}
```

**Polimorfismo**

- **Definición**: El polimorfismo permite que diferentes clases usen el mismo nombre de método pero con implementaciones diferentes. Esto facilita la sustitución de métodos y la interacción con diferentes tipos de objetos.

**Ejemplo**:
php

```
class Animal {
    public function hacerSonido() {
        echo "El animal hace un sonido.";
    }
}

class Perro extends Animal {
    public function hacerSonido() {
        echo "El perro ladra.";
    }
}

class Gato extends Animal {
    public function hacerSonido() {
        echo "El gato maúlla.";
    }
}

function reproducirSonido(Animal $animal) {
    $animal->hacerSonido();
}

reproducirSonido(new Perro()); // Imprime: El perro
ladra.
```

```
reproducirSonido(new Gato());   // Imprime: El gato maúlla.
```

**Abstracción**

- **Definición**: La abstracción permite definir una interfaz común para diferentes clases, mientras que cada clase proporciona su propia implementación. Esto se hace mediante clases abstractas y métodos abstractos.

  **Ejemplo**:

  php

```
abstract class Forma {
    abstract public function calcularArea();
}

class Circulo extends Forma {
    private $radio;

    public function __construct($radio) {
        $this->radio = $radio;
    }

    public function calcularArea() {
```

```
        return pi() * $this->radio *
$this->radio;

    }

}
```

## MVC:

El Modelo-Vista-Controlador (MVC) es un patrón de diseño arquitectónico utilizado para estructurar aplicaciones de software de manera que separes la lógica de negocio, la interfaz de usuario y el control de flujo en componentes distintos. Este enfoque ayuda a mejorar la organización del código y facilita el mantenimiento y escalabilidad de las aplicaciones. En PHP, MVC se usa comúnmente para desarrollar aplicaciones web.

Componentes del Patrón MVC

1. Modelo
    - Definición: El modelo es responsable de la lógica de negocio y la gestión de los datos de la aplicación. Interactúa con la base de datos y maneja las operaciones de creación, lectura, actualización y eliminación (CRUD) de datos.
    - Función: Gestionar los datos y la lógica de la aplicación. El modelo no tiene conocimiento sobre la presentación de los datos.

Ejemplo en PHP:
php

```
class Usuario {
    private $pdo;

    public function __construct($pdo) {
        $this->pdo = $pdo;
    }

    public function obtenerUsuarios() {
```

```
        $query = $this->pdo->prepare("SELECT * FROM
usuarios");
        $query->execute();
        return $query->fetchAll(PDO::FETCH_OBJ);
    }
}
```

2. Vista
    - Definición: La vista es responsable de la presentación de los datos al usuario. Define cómo se muestran los datos en la interfaz gráfica del usuario (GUI) y puede incluir HTML, CSS y JavaScript.
    - Función: Renderizar la interfaz de usuario y mostrar los datos proporcionados por el modelo. La vista no maneja la lógica de negocio.

Ejemplo en PHP:

```
// vista/usuarios.php
foreach ($usuarios as $usuario) {
    echo "<p>{$usuario->nombre}</p>";
}
```

3. Controlador
    - Definición: El controlador actúa como intermediario entre el modelo y la vista. Maneja las solicitudes del usuario, invoca las operaciones necesarias en el modelo y actualiza la vista con los datos obtenidos.
    - Función: Gestionar la entrada del usuario, procesar las solicitudes, actualizar el modelo y seleccionar la vista adecuada para mostrar los resultados.

Ejemplo en PHP:

```
class UsuarioControlador {
    private $modelo;

    public function __construct($modelo) {
```

```
        $this->modelo = $modelo;
    }

    public function listarUsuarios() {
        $usuarios = $this->modelo->obtenerUsuarios();
        require 'vista/usuarios.php';
    }
```

- }

**JS:** Es un lenguaje de programación interpretado y de alto nivel, que se utiliza principalmente para el desarrollo web. Es un lenguaje fundamental en el desarrollo de aplicaciones web y se ejecuta en el navegador del usuario, permitiendo a los desarrolladores crear interactividad y dinamismo en las páginas web.

Características Clave de JavaScript

1. Lenguaje de Script del Lado del Cliente: JavaScript se ejecuta en el navegador del usuario, lo que permite a las páginas web responder a eventos como clics, desplazamientos y entradas de usuario sin necesidad de recargar la página desde el servidor.
2. Lenguaje de Programación Orientado a Objetos y Basado en Prototipos: Aunque JavaScript admite programación orientada a objetos, su enfoque se basa en prototipos en lugar de clases tradicionales. Los objetos en JavaScript pueden heredar propiedades y métodos de otros objetos a través de prototipos.
3. Interactividad y Dinamismo: JavaScript se utiliza para crear interactividad en las páginas web, como formularios dinámicos, animaciones, validación de datos del lado del cliente y actualizaciones de contenido sin necesidad de recargar la página (mediante AJAX o Fetch API).
4. Lenguaje de Programación Event-Driven: JavaScript utiliza un modelo de eventos que permite a los desarrolladores escribir código que responde a eventos como clics del ratón, teclas presionadas y cambios en los datos.
5. Ejecución Asíncrona: JavaScript admite programación asíncrona, lo que significa que puede realizar tareas sin bloquear el hilo principal del

navegador. Esto se logra mediante el uso de callbacks, promesas y la nueva sintaxis `async/await`.

6. Compatibilidad con HTML y CSS: JavaScript se integra estrechamente con HTML y CSS. Puede manipular el DOM (Document Object Model) para cambiar el contenido, la estructura y el estilo de una página web en tiempo real.

**CSS:** que significa Cascading Style Sheets (Hojas de Estilo en Cascada), es un lenguaje de estilo utilizado para describir la presentación de un documento escrito en HTML o XML (incluidos los dialectos XML como SVG o XHTML). CSS permite a los desarrolladores controlar el diseño, el diseño y la apariencia de las páginas web.

Características Clave de CSS

1. Separación de Contenido y Presentación: CSS separa la estructura del contenido (definida por HTML) de su presentación (diseño y estilo). Esto hace que el mantenimiento del código sea más sencillo y que el diseño sea más consistente en toda la web.
2. Selección y Aplicación de Estilos: CSS usa selectores para aplicar estilos a elementos HTML específicos. Por ejemplo, se puede aplicar un color de fondo a todos los párrafos (`<p>`), cambiar el tamaño de fuente de los encabezados (`<h1>`, `<h2>`, etc.), o ajustar el margen y el relleno de los elementos.
3. Modelo de Caja: CSS se basa en el modelo de caja, que define cómo se renderizan los elementos en la página. Cada elemento HTML se considera una caja rectangular que tiene contenido, relleno, borde y margen.
4. Diseño Responsive: CSS permite el diseño responsive, lo que significa que las páginas web pueden adaptarse a diferentes tamaños de pantalla y dispositivos, como computadoras de escritorio, tabletas y teléfonos móviles. Esto se puede lograr mediante consultas de medios (media queries) y unidades flexibles.
5. Flexibilidad y Control: CSS ofrece un alto grado de control sobre el diseño de una página, incluyendo fuentes, colores, espacios, bordes,

alineación y más. Los desarrolladores pueden crear diseños complejos y adaptativos con facilidad.
6. Animaciones y Transiciones: CSS permite agregar animaciones y transiciones a los elementos, lo que ayuda a mejorar la experiencia del usuario y a hacer que las interfaces sean más interactivas.

**PHP:** Es un lenguaje de programación de código abierto especialmente diseñado para el desarrollo web. Su nombre original es "Personal Home Page", aunque actualmente se conoce como "PHP: Hypertext Preprocessor" (Preprocesador de Hipertexto).

Características Principales de PHP

1. Lenguaje del Lado del Servidor: PHP se ejecuta en el servidor web y genera contenido dinámico que se envía al navegador del usuario. Esto significa que el código PHP se procesa en el servidor antes de que se envíe el resultado al cliente, generalmente en forma de HTML.
2. Código Abierto: PHP es de código abierto y gratuito. Esto ha permitido que sea ampliamente utilizado y modificado por la comunidad de desarrolladores.
3. Integración con Bases de Datos: PHP tiene una fuerte integración con varias bases de datos, como MySQL, PostgreSQL, SQLite y otras. Esto lo convierte en una opción popular para aplicaciones web que requieren almacenamiento y recuperación de datos.
4. Simplicidad y Flexibilidad: PHP es conocido por su sintaxis relativamente sencilla, lo que lo hace accesible para principiantes. A la vez, ofrece una gran flexibilidad y potencia para desarrolladores experimentados.
5. Amplio Soporte de Hosting: Debido a su popularidad, la mayoría de los proveedores de servicios de alojamiento web soportan PHP, lo que facilita la implementación de aplicaciones PHP en una variedad de entornos.
6. Frameworks y Herramientas: Existen numerosos frameworks PHP, como Laravel, Symfony, y CodeIgniter, que ayudan a acelerar el desarrollo y a seguir las mejores prácticas de programación. También hay muchas herramientas y bibliotecas que amplían las capacidades de PHP.

7. Manejo de Formularios y Sesiones: PHP facilita el manejo de formularios web y la gestión de sesiones de usuario, lo que es esencial para el desarrollo de aplicaciones web interactivas.

**Apache:** Apache es un servidor web de código abierto que se encarga de recibir las solicitudes de los navegadores y enviar las respuestas adecuadas, ya sea en forma de archivos HTML, imágenes, o cualquier otro tipo de contenido. Es conocido por su estabilidad, flexibilidad y robustez.

Cómo Funciona Apache para Páginas PHP

Apache, en sí mismo, no entiende PHP directamente. Para servir páginas PHP, Apache necesita trabajar con un módulo que interprete el código PHP. Aquí está un resumen de cómo funciona el proceso:

1. Recepción de Solicitud:
    - Un navegador web (cliente) envía una solicitud HTTP al servidor Apache para acceder a una página web.
2. Manejo de Solicitud:
    - Apache recibe la solicitud y examina el tipo de archivo solicitado. Si el archivo tiene una extensión que Apache ha sido configurado para manejar con PHP (por ejemplo, `.php`), Apache pasa la solicitud a un intérprete de PHP.
3. Interacción con PHP:
    - Apache utiliza un módulo como mod_php o una interfaz CGI/FastCGI para pasar la solicitud al intérprete de PHP.
    - El módulo `mod_php` es un módulo de Apache que se integra directamente con el servidor, permitiendo que Apache ejecute el código PHP directamente.
    - Alternativamente, PHP puede ejecutarse a través de CGI o FastCGI, lo que implica que Apache pasa la solicitud a un proceso PHP externo que maneja la ejecución del código.
4. Ejecución del Código PHP:
    - El intérprete de PHP procesa el código PHP en el archivo solicitado, generando contenido dinámico. Este contenido es generalmente en formato HTML, pero puede incluir otros tipos de datos, como JSON o XML.

5. Respuesta al Navegador:
    - Una vez que el intérprete de PHP ha procesado el código, Apache recibe el contenido generado (HTML u otro) y lo envía de vuelta al navegador del cliente.
6. Visualización del Contenido:
    - El navegador del cliente recibe el contenido HTML y lo renderiza para que el usuario lo vea.

**XAMPP:**es un paquete de software que proporciona una solución todo-en-uno para el desarrollo web local. Es una forma fácil de instalar y configurar un entorno de servidor web en tu propia máquina, que incluye varias herramientas esenciales para el desarrollo de aplicaciones web.

Componentes de XAMPP

XAMPP incluye:

1. Apache: El servidor web que maneja las solicitudes HTTP y sirve archivos a los navegadores.
2. MySQL/MariaDB: Un sistema de gestión de bases de datos relacional que almacena y gestiona datos.
3. PHP: Un lenguaje de programación del lado del servidor que se utiliza para generar contenido web dinámico.
4. Perl: Un lenguaje de programación adicional incluido en algunas versiones de XAMPP.

**MYSQL**: Es un software de base de datos que permite almacenar, organizar, y gestionar datos en tablas relacionadas entre sí dentro de una base de datos. Es ampliamente utilizado en aplicaciones web y software debido a su robustez, rendimiento y facilidad de uso.

**Características Clave**

1. **Relacional**: MySQL organiza los datos en tablas que pueden tener relaciones entre sí mediante claves primarias y foráneas. Esto facilita la estructuración y el acceso eficiente a los datos.

2. **SQL**: Utiliza el lenguaje SQL para realizar operaciones como consultas, inserciones, actualizaciones y eliminaciones de datos. SQL es un estándar ampliamente utilizado para interactuar con bases de datos relacionales.
3. **Código Abierto**: MySQL es un software de código abierto, lo que significa que el código fuente está disponible para que cualquiera pueda estudiarlo, modificarlo y distribuirlo.
4. **Rendimiento y Escalabilidad**: MySQL está diseñado para ser rápido y eficiente en la gestión de grandes volúmenes de datos. Ofrece características como índices, optimización de consultas y replicación para mejorar el rendimiento y la escalabilidad.
5. **Compatibilidad y Portabilidad**: MySQL es compatible con una amplia gama de sistemas operativos, incluidos Windows, Linux y macOS. También es compatible con varios lenguajes de programación, como PHP, Python, Java y más.
6. **Seguridad**: MySQL incluye características de seguridad como autenticación de usuarios, control de acceso y cifrado de datos para proteger la integridad y la privacidad de los datos.
7. **Soporte para Transacciones**: MySQL admite transacciones, lo que garantiza que las operaciones en la base de datos sean completas y coherentes. Esto es crucial para mantener la integridad de los datos en sistemas críticos.
8. **Gestión de Datos**: Permite la creación, modificación y eliminación de bases de datos, tablas y registros. También proporciona herramientas para realizar copias de seguridad y recuperación de datos.