SEGUNDA ENTREGA PROYECTO: Tricosaurus

TRICOTECH

Asignatura: Programación Full Stack

Nombre del docente: Diego Mendez

ITI – CETP - Grupo 3° MG

Nombre estudiantes: Jose Madruga, Fabricio Vanrell y Facundo Benítez

Contenido

DESCRIPCIÓN DEL PROYECTO

Tricosaurus es una aplicación web que simula el juego de mesa Draftosaurus, permitiendo a los usuarios jugar de forma digital tanto en modo multijugador como en modo seguimiento individual.

OBJETIVO PRINCIPAL

Crear una experiencia de juego digital fiel al juego original, con interfaz intuitiva y responsive que funcione en dispositivos móviles y de escritorio.

ARQUITECTURA DEL SISTEMA

Arquitectura FrontEnd:

La arquitectura del frontend se organiza en tres capas principales:

1. **Presentación**

- **HTML5**: define la estructura y el contenido de las páginas.
- **CSS3**: gestiona los estilos visuales, incluyendo colores, tipografías y disposición de los elementos.

2. **Lógica**

- **JavaScript (Vanilla)**: implementa la lógica de interacción, validaciones y control de flujo en el navegador.
- **API Drag & Drop**: permite la manipulación visual de elementos arrastrables dentro de la interfaz, facilitando una experiencia dinámica para el usuario.

3. **Recursos**

- **Imágenes (PNG/JPG)**: archivos gráficos utilizados para enriquecer la interfaz.
- **Font Awesome**: biblioteca de íconos escalables que aporta elementos gráficos adicionales para mejorar la usabilidad y estética.

Arquitectura BackEnd:

Arquitectura Backend (API)

El backend está desarrollado en **PHP** y organizado en capas para mejorar la mantenibilidad y escalabilidad:

Endpoints (Rutas)

Definen los puntos de entrada de la API bajo api/v1/, por ejemplo:

- GET /api/v1/partidas → obtener todas las partidas.
- POST /api/v1/partidas → crear una nueva partida.

Controllers

Reciben las solicitudes HTTP, validan parámetros y llaman a los servicios correspondientes. Se encargan de formatear las respuestas en JSON.

Services

Contienen la lógica de negocio. Definen validaciones y transformaciones necesarias antes de interactuar con los datos.

Repositories

Gestionan la persistencia de datos y conexión a una base de dato.

Flujo de Interacción

1. El jugador accede al frontend y visualiza el tablero.
2. Al mover un dinosaurio, el frontend genera una petición HTTP hacia la API.
3. El **Controller** recibe la petición y la envía al **Service**.
4. El **Service** aplica la lógica de negocio y solicita datos al **Repository**.
5. El **Repository** recupera o modifica la información.
6. El **Service** devuelve la respuesta procesada al **Controller**.
7. El **Controller** retorna una respuesta en JSON al frontend.
8. El frontend actualiza la interfaz en tiempo real.

TECNOLOGÍAS UTILIZADAS

Frontend

- **HTML5, CSS3, JavaScript (Vanilla):** Estructura, estilos y lógica de interacción en el navegador.
- **API Drag & Drop:** Permite arrastrar y soltar elementos dentro del tablero de juego.
- **Font Awesome:** Biblioteca de iconos escalables para mejorar la interfaz.
- **Recursos gráficos (PNG/JPG):** Ilustraciones de dinosaurios, tableros y fondos.

Backend

- **PHP 7+:** Lenguaje de programación del servidor para procesar la lógica del juego.
- **Arquitectura en capas:**
    - **Endpoints:** Rutas de la API REST que exponen los servicios al frontend.
    - **Controllers:** Gestionan solicitudes HTTP y formatean respuestas.
    - **Services:** Implementan la lógica de negocio y reglas del juego.
    - **Repositories:** Encapsulan el acceso a datos y consultas SQL.
- **SQL (MySQL):** Persistencia de datos, relaciones entre tablas, consultas y scripts de inicialización.
- **JSON:** Formato de intercambio de datos entre frontend y backend.

Ventajas de la Arquitectura

- **Separación de responsabilidades:** cada capa tiene un rol específico.
- **Mantenibilidad:** cambios en lógica o persistencia no afectan al resto.
- **Escalabilidad:** fácil agregar nuevas reglas de juego o modos.
- **Compatibilidad:** accesible desde navegadores modernos y adaptable a móviles.

CONFIGURACIÓN DEL ENTERNO

Instalación del IDE

Visual Studio Code (recomendado)

1. **Descarga**: Obtener el instalador desde https://code.visualstudio.com.
2. **Instalación**: Ejecutar el instalador y seguir los pasos del asistente.

3. **Extensiones recomendadas**:

- **Live Server**: Servidor local con recarga automática.
- **HTML CSS Support**: Autocompletado de etiquetas y estilos.
- **Auto Rename Tag**: Renombrado automático de etiquetas HTML.

Instalación y Configuración de PHP

1. **Descargar PHP**

- Ir a https://www.php.net/downloads.php.
- Descargar la versión estable recomendada (mínimo PHP 8.1).

2. **Instalar PHP**

- Descomprimir en C:\php (Windows) o usar el gestor de paquetes en Linux/Mac (apt, brew).
- Agregar la ruta de PHP a las variables de entorno del sistema.

3. **Verificación de instalación**

- Comando: php -v

Debería mostrar la versión instalada.

Instalación y Configuración de MySQL Server

1. **Descarga de MySQL Server**

- Ir a https://dev.mysql.com/downloads/mysql/.
- Seleccionar la versión Community Server.

2. **Instalación**

- Seguir el asistente de instalación.
- Definir un usuario administrador (root) y contraseña segura.
- Configurar puerto de acceso (por defecto **3306**).

3. **Verificación**

- Comando: mysql -u root -p

- Si conecta, la instalación fue correcta.

Instalación y Configuración de MySQL Workbench

1. **Descarga**: Obtener desde https://dev.mysql.com/downloads/workbench/.
2. **Instalación**: Seguir las instrucciones del instalador.
3. **Configuración inicial**:

- Conectar con el servidor local (localhost).

- Puerto: **3306**.
- Usuario: root.
- Contraseña: la definida en la instalación de MySQL Server.

4. **Uso principal en el proyecto**:

- Creación y administración de bases de datos.
- Ejecución de scripts SQL.
- Visualización de diagramas EER (Entidad-Relación).

CONTROL DE VERSIONES

Creación del Repositorio Inicial

Configuración básica de Git

https://github.com/Fabri1899/TricoTech.git

```
git clone https://github.com/Fabri1899/TricoTech
cd TricoTech
git config user.name "Tu Nombre"
git config user.email "tu-email@ejemplo.com"

# Flujo de trabajo básico
git add .
git commit -m "feat: inicializar proyecto Draftosaurus"
git push origin main
```

Justificación de la Elección

¿Por qué Git?

- Sistema de control de versiones distribuido (trabajo offline + sincronización).
- Historial completo de cambios.
- Uso de ramas para desarrollo paralelo sin conflictos.
- Colaboración entre múltiples desarrolladores.

¿Por qué GitHub?

- Hosting gratuito para repositorios públicos.
- GitHub Pages para despliegue automático de sitios web.
- Amplia comunidad y ecosistema de colaboración.
- Integración con múltiples herramientas de desarrollo.

COMPATIBILIDAD DEL PROYECTO

- **Navegadores soportados**: Chrome 60+, Firefox 55+, Safari 12+, Edge 79+.
- **Dispositivos**: Móviles, tablets y escritorio.
- **Sistemas Operativos**: Windows, macOS y Linux.

CÓDIGOS

Imágenes de trozos de código fuente.

user_acces.php

```php
<?php
session_start();

header(header: 'Content-Type: application/json');
header(header: 'Access-Control-Allow-Origin: *');
header(header: "Access-Control-Allow-Methods: GET, POST, OPTIONS");
header(header: "Access-Control-Allow-Headers: Content-Type, Authorization");

require_once '../../inc/config/conectionDB.php';
require_once '../../src/repositories/UserRepository.php';
require_once '../../src/services/AuthService.php';
require_once '../../src/controllers/AuthController.php';

$action = $_GET['action'] ?? null;

$controller = new AuthController();

if ($action === 'login') {
    $controller->login();
} elseif ($action === 'register') {
    $controller->register();
} else {
    echo json_encode(value: ['success' => false, 'message' => 'Ruta no encontrada']);
}
```

script.js

```javascript
// ----------------------------
// FUNCIONES DRAG & DROP
// ----------------------------
function generarDraggables() {
    const cont = document.getElementById("contenedor-dinosaurio");
    if (!cont) return;
    cont.innerHTML = "";

    draggables.forEach(d => {
        const div = document.createElement("div");
        div.className = "arrastrable";
        div.draggable = true;
        div.id = d.id;
        div.dataset.especie = d.id;
        div.innerHTML = `<img src="${d.img}">`;
        div.addEventListener("dragstart", dragStart);
        cont.appendChild(div);
        contadorEspecies[d.id] = 0;
    });
}
```

```javascript
async function login() {
    const API_URL = 'http://localhost:8000/api/v1/user_access.php';

    const identifier = document.getElementById('login_identifier').value;
    const password = document.getElementById('login_password').value;
    const messageDiv = document.getElementById('message');

    const response = await fetch(API_URL + '?action=login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ identifier, password })
    });

    const result = await response.json();
    messageDiv.textContent = result.message;
    if (result && result.success) {
        window.location.href = '../../index.php';
    }
}
```

script.js

```php
<!DOCTYPE html>      You, 15 hours ago • si …
<html lang="es">
```

header.php

```php
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><?php echo isset($pageTitle) ? $pageTitle : 'Draftosaurus'; ?></title>
    <link rel="icon" type="image/png" href="../../assets/images/logos/logo.png">
    <link rel="stylesheet" href="../../assets/css/styles.css">

    <?php if (isset($customCSS)): ?>
        <?php if (is_array(value: $customCSS)): ?>
            <?php foreach ($customCSS as $css): ?>
                <link rel="stylesheet" href="../../assets/css/<?php echo $css; ?>">
            <?php endforeach; ?>
        <?php else: ?>
            <link rel="stylesheet" href="../../assets/css/<?php echo $customCSS; ?>">
        <?php endif; ?>
    <?php endif; ?>


    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
</head>

<body>
```

```php
public function insertarPartida($userId, $cantJugadores, $cantRondas, $datosJson, $gameId = null): bool {
    if ($gameId === null) {
        $stmt = $this->conn->prepare(query: "INSERT INTO partidas (user_id, cant_jugadores, cant_rondas, datos) VALUES (?, ?, ?, ?)");
        $stmt->bind_param(types: "iiis", var: &$userId, vars: &$cantJugadores, $cantRondas, $datosJson);
    } else {
        $stmt = $this->conn->prepare(query: "INSERT INTO partidas (user_id, game_id, cant_jugadores, cant_rondas, datos) VALUES (?, ?, ?, ?, ?)");
        $stmt->bind_param(types: "iiiis", var: &$userId, vars: &$gameId, $cantJugadores, $cantRondas, $datosJson);
    }
    return $stmt->execute();
}
```

PartidaRepository.php

```php
public function guardar(): void {
    $input = json_decode(json: file_get_contents(filename: 'php://input'), associative: true);
    $userId = $_SESSION['user_id'] ?? null;

    if (!$userId) {
        echo json_encode(value: ['success' => false, 'message' => 'Usuario no logueado']);
        return;
    }

    $result = $this->PartidaService->guardarPartida(userId: $userId, data: $input);
    echo json_encode(value: $result);
}
```

PartidaController.php

```php
public function guardarPartida($userId, $data): array
{
    $cantJugadores = $data['cantJugadores'] ?? 2;
    $cantRondas = $data['cantRondas'] ?? 4;
    $gameId = isset($data['game_id']) ? (int)$data['game_id'] : null;
    $datos = json_encode(value: [
        'puntajesPorRonda' => $data['puntajesPorRonda'] ?? [],
        'estado' => $data['estado'] ?? []
    ]);

    $success = $this->PartidaRepository->insertarPartida(userId: $userId, cantJugadores: $cantJugadores, cantRondas: $cantRondas, datosJso…$datos, $gameId);

    return $success
        ? ['success' => true, 'message' => 'Partida guardada correctamente']
        : ['success' => false, 'message' => 'Error al guardar partida'];
}
```

PartidaService.php

index.php

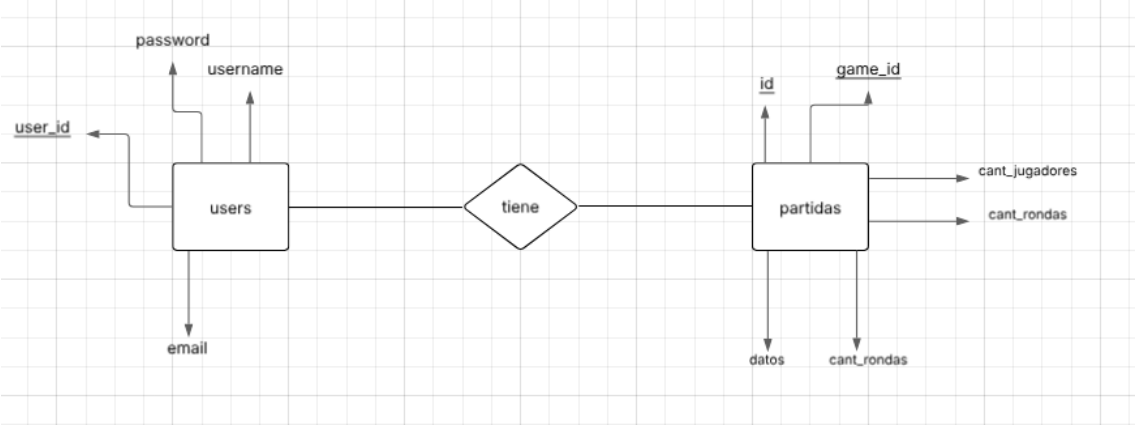
```php
<?php
$pageTitle = "Bienvenido - Draftosaurus";
include 'inc/templates/header.php';
include 'inc/templates/nav.php';
?>

<main>

    <div class="contenedor">
        <div id="contenido-bienvenida">
            <h1>Bienvenido a Draftosaurus!</h1>
            <p>Preparate para jugar y divertirte</p>
            <button onclick="mostrarModosJuego()">Comenzar Juego</button>
        </div>

        <div id="modos-juego" style="display: none;">
            <h2>Elegir modo de juego</h2>
            <div class="botones-modos">
                <button class="btn-modo" onclick="seleccionarModo('game')">Modo Juego</button>
                <button class="btn-modo" onclick="seleccionarModo('tracing')">Modo Seguimiento</button>
            </div>
            <button class="btn-volver" onclick="volverInicio()">Volver</button>
        </div>
    </div>

    <script src="assets/js/script.js"></script>


</main>

</body>
```

db_create_tricosaurus.sql

```sql
CREATE DATABASE IF NOT EXISTS tricosaurus;
USE tricosaurus;

-- Usuarios
CREATE TABLE IF NOT EXISTS users (
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(120) NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP
);

-- Partidas guardadas por usuario, con referencia a game_id
CREATE TABLE IF NOT EXISTS partidas (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    game_id BIGINT NULL,
    cant_jugadores INT NOT NULL,
    cant_rondas INT NOT NULL,
    datos JSON NOT NULL,
    fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_partidas_user FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

-- Índices para consultas frecuentes
CREATE INDEX IF NOT EXISTS ix_users_username ON users (username);
CREATE INDEX IF NOT EXISTS ix_users_email ON users (email);
CREATE INDEX IF NOT EXISTS ix_partidas_user_fecha ON partidas (user_id, fecha);
```

MODELADO DE DATOS

A) DER (Diagrama Entidad-Relación)



Notas: - `datos` encapsula estado por ronda: `puntajesPorRonda`, `estado` (tablero), `kingPorRonda`.

B) Esquema Relacional Normalizado (hasta 3FN)

Transformación DER→Tablas: - `users(user_id PK, username UK, email UK, password_hash, created_at, updated_at)` - `partidas(id PK, user_id FK→users.user_id, game_id NULL, cant_jugadores, cant_rondas, datos JSON, fecha)`

Normalización: - **1FN:** atributos atómicos; JSON usado para estado complejo. - **2FN:** dependencias completas de la PK. - **3FN:** sin dependencias transitivas; `username` y `email` no determinan otros campos fuera de PK.

Decisión sobre JSON: - Tablero altamente jerárquico, se justifica guardar en JSON para simplificar implementación.

Claves y restricciones: - PK: `users.user_id`, `partidas.id`. - UK: `users.username`, `users.email`. - FK: `partidas.user_id` → `users.user_id` (ON DELETE CASCADE). - Índices: `ix_users_username`, `ix_users_email`, `ix_partidas_user_fecha`.

C) Restricciones No Estructurales (Reglas del juego)

1. **Colocación por recinto:**
   - SAME_FOREST, DIFFERENT_MEADOW, TRIO_TREES, LONELY, KING, RIVER.
   - Implementación: JS valida antes de guardar; PHP opcional.
2. **Límites por ronda:**

- Máximo 6 piezas por ronda.

3. **Integridad de usuario y sesión:**
   - Sesión requerida para guardar/listar partidas.
4. **Seguridad:**
   - Consultas preparadas, hash de contraseñas.

3. Implementación Física de la Base de Datos

```
CREATE DATABASE IF NOT EXISTS tricosaurus;
USE tricosaurus;

-- Tabla de Usuarios
CREATE TABLE IF NOT EXISTS users (
    user_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(120) NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP
);

-- Tabla de Partidas
CREATE TABLE IF NOT EXISTS partidas (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    cant_jugadores INT NOT NULL,
    cant_rondas INT NOT NULL,
    datos JSON NOT NULL,
    fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_partidas_user FOREIGN KEY (user_id) REFERENCES users
(user_id) ON DELETE CASCADE
);

-- Índices
CREATE INDEX IF NOT EXISTS ix_users_username ON users (username);
CREATE INDEX IF NOT EXISTS ix_users_email ON users (email);
CREATE INDEX IF NOT EXISTS ix_partidas_user_fecha ON partidas (user_id
, fecha);

-- Datos de prueba
INSERT INTO users (username, email, password_hash) VALUES
('alice', 'alice@example.com', '$2y$10$$2y$12$e493ym5VyAoyt4htyW7ZDucA
e5NQC17M0R.OA5LSX1zam7GRxZWkG');

INSERT INTO partidas (user_id, cant_jugadores, cant_rondas, datos)
VALUES
(1, 3, 2, '{
  "estado": {
    "Ronda 1": {
      "KING": ["SPECIE_4_copy_1"],
```


```
      "RIVER": [null, null, null, null, null, null],
      "LONELY": [null],
      "TRIO_TREES": ["SPECIE_5_copy_5", null, null],
      "LOVE_MEADOW": [null, null, null, null, null, null],
      "SAME_FOREST": ["SPECIE_5_copy_1", "SPECIE_5_copy_2", "SPECIE_5_
copy_3", "SPECIE_5_copy_4", null, null],
      "DIFFERENT_MEADOW": [null, null, null, null, null, null]
    },
    "Ronda 2": {
      "KING": ["SPECIE_2_copy_1"],
      "RIVER": [null, null, null, null, null, null],
      "LONELY": [null],
      "TRIO_TREES": ["SPECIE_4_copy_1", "SPECIE_3_copy_1", null],
      "LOVE_MEADOW": [null, null, null, null, null, null],
      "SAME_FOREST": ["SPECIE_5_copy_1", null, null, null, null, null]
,
      "DIFFERENT_MEADOW": ["SPECIE_5_copy_2", "SPECIE_2_copy_2", null,
null, null, null]
    }
  },
  "puntajesPorRonda": {
    "Ronda 1": 12,
    "Ronda 2": 5,
    "Ronda 3": 0,
    "Ronda 4": 0
  }
}');
```