PROYECTO TERCERA ENTREGA PROGRAMACIÓN

TRICOTECH

ROL	C.I	APELLIDO	NOMBRE	E-MAIL
Coordinador	5.484.815-1	Vanrell	Fabricio	fabriciovanrell2@gmail.com
Subcoordinador	5.214.522-2	Madruga	José	josemadruga241999@gmail.com
Integrante 1	5.297.422-3	Benítez	Facundo	facuhbt98@gmail.com

Asignatura: Prog. FullStack

Nombre Docente: Diego Mendez

Nombre Estudiantes: José Madruga, Fabricio Vanrell y Facundo Benítez

Grupo: 3° MG

Institución educativa: ITI

Fecha de Entrega: 03/11/25

Contenido

Visión General

Tricosaurus es una aplicación web para jugar **Draftosaurus**, un juego de estrategia donde los jugadores colocan dinosaurios en diferentes recintos de un zoológico para obtener puntos. El proyecto está implementado en PHP (backend) y JavaScript (frontend), con una base de datos MySQL.

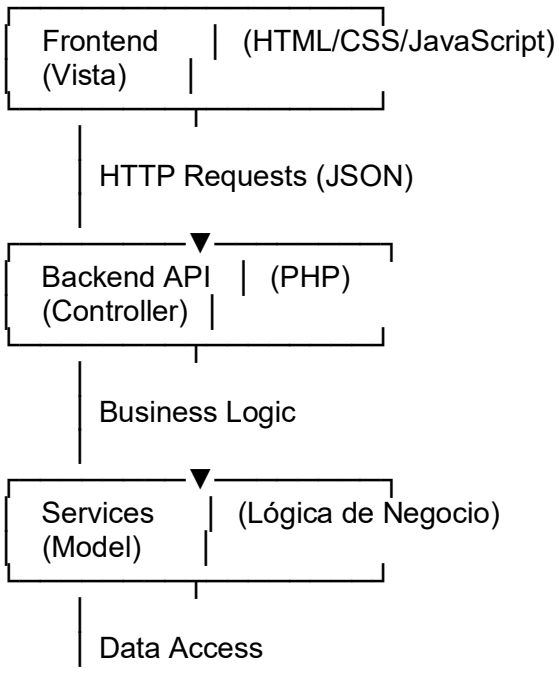
Características Principales

- **Sistema de autenticación** completo (registro, login, gestión de perfil)
- **Dos modos de juego**:
  - **Modo Seguimiento**: Para un solo jugador que registra manualmente sus partidas
  - **Modo Juego**: Para múltiples jugadores con turnos, bolsas de dinosaurios y tableros individuales
- **Guardado y restauración** de partidas
- **Sistema de puntuación** automático basado en reglas del juego
- **Interfaz drag & drop** para colocar dinosaurios
- **Visualización de partidas guardadas** con navegación por rondas y jugadores

Arquitectura del Proyecto

Patrón de Arquitectura

El proyecto sigue una arquitectura **MVC (Model-View-Controller)** con separación de responsabilidades:



```
        │
        ▼
┌──────────────┐
│  Repository  │ (Acceso a Datos)
│  (Database)  │
└──────────────┘
```

Tecnologías Utilizadas

- **Backend**: PHP 7.4+, MySQL, JSON
- **Frontend**: HTML5, CSS3, JavaScript (Vanilla)
- **Servidor**: Apache/XAMPP
- **Base de Datos**: MySQL 8.0+

Estructura de Directorios

```
Tricosaurus/
│
├── backend/                    # Código del servidor
│   ├── api/                    # Punto de entrada de la API
│   │   ├── index.php           # Router principal
│   │   └── v1/                 # Versiones futuras
│   │
│   ├── inc/                    # Includes y configuración
│   │   └── config/
│   │       ├── conectionDB.php # Singleton de conexión DB
│   │       └── config          # Archivo JSON de configuración
│   │
│   └── src/                    # Código fuente
│       ├── controllers/        # Controladores (Auth, Partida)
│       ├── services/           # Lógica de negocio
│       └── repositories/       # Acceso a datos
│
├── frontend/                   # Cliente web
│   ├── assets/
│   │   ├── css/                # Estilos
│   │   ├── js/                 # JavaScript
│   │   └── images/             # Imágenes (dinosaurios, fondos, logos)
│   │
│   ├── pages/                  # Páginas HTML
│   │   ├── modes/              # Modos de juego
│   │   └── user/               # Páginas de usuario
│   │
│   └── index.html              # Página principal
│
├── dataBaseScripts/            # Scripts SQL
│   ├── create_database_full.sql
│   ├── add_game_mode_tables.sql
│   └── insert_sample_data.sql
```

└── DOCUMENTACION_COMPLETA.md # Esta documentación

Base de Datos

Esquema de Base de Datos

La base de datos utiliza un diseño **normalizado** con tablas separadas para cada entidad.

Tablas Principales

1. users - Usuarios del Sistema

```
CREATE TABLE 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
);
```

Campos: - user_id: ID único del usuario (PK) - username: Nombre de usuario único - email: Email único (opcional) - password_hash: Hash bcrypt de la contraseña - created_at: Fecha de creación - updated_at: Fecha de última actualización

2. partidas - Partidas Principales

```
CREATE TABLE partidas (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    tipo ENUM('seguimiento', 'juego') NOT NULL DEFAULT 'seguimiento',
    user_id BIGINT NOT NULL,
    game_id BIGINT NULL,
    cant_jugadores INT NOT NULL,
    cant_rondas INT NOT NULL,
    datos JSON NULL,
    fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Campos: - id: ID único de la partida (PK) - tipo: Tipo de partida (‘seguimiento’ o ‘juego’) - user_id: Usuario propietario (FK) - game_id: ID de juego asociado (opcional) - cant_jugadores: Número de jugadores - cant_rondas: Número de rondas - datos: JSON con datos adicionales (legacy) - fecha: Fecha de creación

3. partida_rondas - Rondas de Partidas (Modo Seguimiento)

```
CREATE TABLE partida_rondas (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
```

```
    puntaje INT NOT NULL DEFAULT 0,
    es_rey BOOLEAN NOT NULL DEFAULT FALSE,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE,
    UNIQUE KEY (partida_id, numero_ronda)
);
```

Campos: - id: ID único (PK) - partida_id: ID de la partida (FK) - numero_ronda: Número de ronda (1, 2, 3, …) - puntaje: Puntos obtenidos en la ronda - es_rey: Si tiene el rey en esta ronda

4. posiciones_dinosaurios - Posiciones (Modo Seguimiento)

```
CREATE TABLE posiciones_dinosaurios (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
    recinto VARCHAR(50) NOT NULL,
    posicion_slot INT NOT NULL,
    dinosaurio_id VARCHAR(100) NOT NULL,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE
);
```

Campos: - id: ID único (PK) - partida_id: ID de la partida (FK) - numero_ronda: Número de ronda - recinto: ID del recinto (SAME_FOREST, DIFFERENT_MEADOW, KING, etc.) - posicion_slot: Índice del slot (0, 1, 2, …) - dinosaurio_id: ID del dinosaurio (ej: “tRex_copy_1234567890”)

Tablas del Modo Juego

5. bolsas_ronda - Bolsa Grande por Ronda

```
CREATE TABLE bolsas_ronda (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
    dinosaurios JSON NOT NULL,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE,
    UNIQUE KEY (partida_id, numero_ronda)
);
```

Campos: - dinosaurios: JSON array con todos los dinosaurios de la ronda

6. bolsas_jugador - Bolsas Individuales por Jugador

```
CREATE TABLE bolsas_jugador (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
    numero_jugador INT NOT NULL,
    dinosaurios JSON NOT NULL,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE,
    UNIQUE KEY (partida_id, numero_ronda, numero_jugador)
);
```

Campos: - numero_jugador: Número del jugador (1, 2, 3, …) - dinosaurios: JSON array con dinosaurios del jugador

7. posiciones_jugador - Posiciones por Jugador

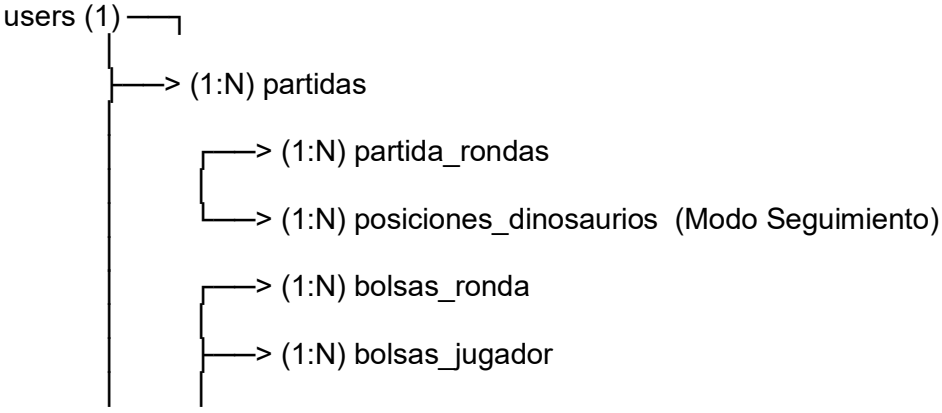
```
CREATE TABLE posiciones_jugador (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
    numero_jugador INT NOT NULL,
    recinto VARCHAR(50) NOT NULL,
    posicion_slot INT NOT NULL,
    dinosaurio_id VARCHAR(100) NOT NULL,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE,
    UNIQUE KEY (partida_id, numero_ronda, numero_jugador, recinto, posicion_slot)
);
```

Campos: - numero_jugador: Número del jugador - Similar a posiciones_dinosaurios pero por jugador

8. puntajes_jugador - Puntajes por Jugador por Ronda

```
CREATE TABLE puntajes_jugador (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    partida_id BIGINT NOT NULL,
    numero_ronda INT NOT NULL,
    numero_jugador INT NOT NULL,
    puntaje INT NOT NULL DEFAULT 0,
    es_rey BOOLEAN NOT NULL DEFAULT FALSE,
    fecha_creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (partida_id) REFERENCES partidas(id) ON DELETE CASCADE,
    UNIQUE KEY (partida_id, numero_ronda, numero_jugador)
);
```

Relaciones entre Tablas



```
│       ├──> (1:N) posiciones_jugador
│       │
│       └──> (1:N) puntajes_jugador       (Modo Juego)
```

Índices Optimizados

- users: username, email
- partidas: user_id, fecha
- partida_rondas: partida_id
- posiciones_dinosaurios: partida_id, numero_ronda
- bolsas_ronda: partida_id, numero_ronda
- bolsas_jugador: partida_id, numero_ronda, numero_jugador
- posiciones_jugador: partida_id, numero_ronda, numero_jugador
- puntajes_jugador: partida_id, numero_ronda, numero_jugador

Backend (PHP)

Arquitectura del Backend

El backend utiliza el patrón **Singleton** y **Repository Pattern** para gestionar la conexión a la base de datos y las operaciones de datos.

Estructura de Clases

```
conectionDB (Singleton)
    └──> getConnection()

PartidaRepository (Singleton)
    ├──> insertarPartida()
    ├──> insertarPartidaJuego()
    ├──> actualizarPartida()
    ├──> actualizarPartidaJuego()
    ├──> obtenerPartidaPorId()
    └──> obtenerPartidasPorUsuario()

UserRepository (Singleton)
    ├──> createUser()
    ├──> findByUsername()
    ├──> findByEmail()
    └──> findById()

PartidaService (Singleton)
    ├──> guardarPartida()
    ├──> listarPartidas()
    ├──> obtenerPartida()
    └──> actualizarPartida()

AuthService (Singleton)
    ├──> register()
```

```
    ├──> login()
    ├──> updateUsername()
    ├──> updateEmail()
    └──> updatePassword()

PartidaController
    ├──> guardar()
    ├──> listar()
    ├──> obtener()
    └──> actualizar()

AuthController
    ├──> register()
    ├──> login()
    ├──> checkSession()
    ├──> logout()
    ├──> updateUsername()
    ├──> updateEmail()
    └──> updatePassword()
```

Archivos Principales

1. backend/api/index.php - Router Principal

Responsabilidad: Enrutar las peticiones HTTP a los controladores correspondientes.

Características: - Manejo de CORS - Autoloading de clases - Manejo de errores centralizado - Sistema de rutas basado en parámetros GET

Rutas Definidas:

```
'partidas' => [
    'GET' => ['listar', 'obtener'],
    'POST' => 'guardar',
    'PUT' => ['actualizar']
],
'auth' => [
    'GET' => ['session' => 'checkSession'],
    'POST' => [
        'login' => 'login',
        'register' => 'register',
        'logout' => 'logout',
        'update-username' => 'updateUsername',
        'update-email' => 'updateEmail',
        'update-password' => 'updatePassword'
    ]
]
```

Ejemplo de URL:

GET /backend/api/index.php?path=partidas/listar
GET /backend/api/index.php?path=partidas/obtener&id=123
POST /backend/api/index.php?path=partidas
PUT /backend/api/index.php?path=partidas/actualizar&id=123
POST /backend/api/index.php?path=auth/login

2. backend/inc/config/conectionDB.php - Singleton de Conexión

Patrón: Singleton

Responsabilidad: Gestionar una única conexión a la base de datos MySQL.

Métodos: - getInstance(): Obtiene la instancia única - getConnection(): Retorna el objeto mysqli - close(): Cierra la conexión

Configuración: Lee desde config/config (archivo JSON)

3. backend/src/controllers/PartidaController.php

Responsabilidad: Controlar las peticiones relacionadas con partidas.

Métodos:

- **guardar()**: Guarda una nueva partida
  - Valida sesión de usuario
  - Llama a PartidaService->guardarPartida()
- **listar()**: Lista todas las partidas del usuario
  - Retorna array de partidas
- **obtener()**: Obtiene una partida por ID
  - Valida que la partida pertenezca al usuario
- **actualizar()**: Actualiza una partida existente
  - Valida sesión y propiedad

4. backend/src/controllers/AuthController.php

Responsabilidad: Controlar autenticación y gestión de usuarios.

Métodos:

- **register()**: Registra un nuevo usuario
  - Valida campos requeridos
  - Llama a AuthService->register()
- **login()**: Inicia sesión
  - Normaliza identifier (username o email)
  - Establece $_SESSION['user_id']
- **checkSession()**: Verifica sesión activa
  - Retorna datos del usuario si está autenticado
- **logout()**: Cierra sesión
  - Destruye la sesión
- **updateUsername()**: Actualiza nombre de usuario

- **updateEmail()**: Actualiza email
- **updatePassword()**: Actualiza contraseña

Validaciones: - Campos requeridos - Formato de email - Longitud mínima de username y password - Duplicados (username/email)

5. backend/src/services/PartidaService.php

Responsabilidad: Lógica de negocio para partidas.

Métodos:

- **guardarPartida($userId, $data)**:
  - Determina tipo de partida (seguimiento/juego)
  - Procesa datos según el tipo
  - Llama al repository correspondiente
- **procesarDatosRondas($data)**: Procesa datos del modo seguimiento
  - Convierte formato frontend a estructura de BD
- **procesarDatosRondasJuego($data)**: Procesa datos del modo juego
  - Incluye bolsas y datos por jugador
- **procesarPosicionesRonda($estadoRonda)**: Procesa posiciones de dinosaurios
  - Filtra valores nulos/vacíos
- **listarPartidas($userId)**: Lista partidas del usuario
- **obtenerPartida($partidaId, $userId)**: Obtiene una partida
- **actualizarPartida(...)**: Actualiza una partida

Patrón: Singleton

6. backend/src/repositories/PartidaRepository.php

Responsabilidad: Acceso directo a la base de datos.

Métodos Principales:

- **insertarPartida($userId, $cantJugadores, $cantRondas, $rondasData, $gameId)**:
  - Inserta partida en partidas
  - Inserta rondas en partida_rondas
  - Inserta posiciones en posiciones_dinosaurios
  - Usa transacciones
- **insertarPartidaJuego(...)**:
  - Inserta partida tipo ‘juego’
  - Inserta bolsas en bolsas_ronda y bolsas_jugador
  - Inserta posiciones en posiciones_jugador
  - Inserta puntajes en puntajes_jugador
- **actualizarPartidaJuego(...)**:
  - Actualiza datos de partida existente
  - Usa ON DUPLICATE KEY UPDATE

- **obtenerPartidaPorId($partidaId, $userId)**:
  - Reconstruye estructura JSON desde tablas normalizadas
  - Retorna formato compatible con frontend
- **cargarDatosPartidaJuego($partidaId)**:
  - Carga bolsas, puntajes y posiciones
  - Organiza por rondas y jugadores

Características: - Uso de transacciones para operaciones complejas - Prepared statements para prevenir SQL injection - Manejo de errores con excepciones

7. backend/src/services/AuthService.php

Responsabilidad: Lógica de autenticación y validación.

Métodos:

- **register($username, $email, $password)**:
  - Valida formato y longitud
  - Verifica duplicados
  - Hashea contraseña con password_hash()
  - Crea usuario en BD
- **login($identifier, $password)**:
  - Busca por username o email
  - Verifica contraseña con password_verify()
  - Retorna datos del usuario
- **updateUsername($userId, $newUsername, $password)**:
  - Verifica contraseña actual
  - Valida nuevo username
  - Actualiza en BD
- **updateEmail($userId, $newEmail, $password)**: Similar
- **updatePassword($userId, $newPassword, $currentPassword)**: Similar

Validaciones: - Username: mínimo 5 caracteres - Password: mínimo 8 caracteres - Email: formato válido - Duplicados: username y email únicos

8. backend/src/repositories/UserRepository.php

Responsabilidad: Acceso a datos de usuarios.

Métodos: - createUser($username, $email, $passwordHash): Crea usuario - findByUsername($username): Busca por username - findByEmail($email): Busca por email - findById($userId): Busca por ID - updateUsername($userId, $newUsername): Actualiza username - updateEmail($userId, $newEmail): Actualiza email - updatePassword($userId, $passwordHash): Actualiza contraseña

Frontend (HTML/CSS/JavaScript)

Estructura del Frontend

El frontend está organizado en páginas HTML independientes con scripts JavaScript modulares.

Páginas HTML

1. frontend/index.html - Página Principal

Funcionalidad: - Página de bienvenida - Selector de modo de juego - Navegación principal

Scripts: - auth.js: Manejo de autenticación - script.js: Funciones generales - game.js: Navegación entre modos

Funciones JavaScript: - mostrarModosJuego(): Muestra selector de modos - volverInicio(): Regresa a vista inicial - seleccionarModo(modo): Redirige a modo seleccionado

2. frontend/pages/modes/game.html - Modo Juego

Funcionalidad: - Interfaz de juego multi-jugador - Tablero activo del jugador actual - Sistema de turnos - Bolsas de dinosaurios por jugador - Guardado y restauración de partidas - Vista de revisión de partidas completadas

Elementos Principales: - Selector de cantidad de jugadores - Formularios de login por jugador - Botones: Crear Partida, Reanudar, Restaurar - Panel de información de partida - Bolsa de dinosaurios del jugador activo - Tablero del jugador activo - Selectores de ronda y jugador (modo revisión) - Contenedor de tableros múltiples (modo revisión)

Script: scriptGame.js

3. frontend/pages/modes/modoSeguimiento.html - Modo Seguimiento

Funcionalidad: - Interfaz para un solo jugador - Registro manual de partidas - Navegación por rondas - Guardado y restauración

Elementos Principales: - Selector de rondas - Tablero único - Bolsa de dinosaurios - Panel de puntajes - Lista de partidas guardadas

Script: scriptSeguimiento.js

4. frontend/pages/user/user_access.html - Login/Registro

Funcionalidad: - Formulario de login - Formulario de registro - Cambio entre formularios

Scripts: - auth.js - login.js

5. frontend/pages/user/user_panel.html - Panel de Usuario

Funcionalidad: - Información del usuario - Lista de partidas guardadas - Edición de partidas guardadas - Cerrar sesión

Script: auth.js (inline)

6. frontend/pages/howPlay.html - Reglas del Juego

Funcionalidad: - Explicación de reglas - Tipos de recintos - Sistema de puntuación

Scripts JavaScript

1. frontend/assets/js/auth.js - Clase Auth

Responsabilidad: Gestión de autenticación en el cliente.

Clase: Auth (estática)

Métodos: - init(): Inicializa listeners de navegación - isAuthenticated(): Verifica si hay token en localStorage - redirectToUserPage(): Redirige según estado de autenticación - checkAccess(): Verifica acceso a páginas protegidas - login(token, userData): Guarda token y datos de usuario - logout(): Limpia localStorage y redirige

LocalStorage: - token: Token de sesión - user: Datos del usuario (JSON)

2. frontend/assets/js/login.js - Funciones de Login

Responsabilidad: Manejo de formularios de login/registro.

Funciones: - mostrarFormulario(tipo): Cambia entre login/registro - togglePassword(inputId): Muestra/oculta contraseña - login(): Envía petición de login - register(): Envía petición de registro - showMessage(text, type): Muestra mensajes

Event Listeners: - Toggles de mostrar contraseña - Submit de formularios

3. frontend/assets/js/script.js - Funciones Generales

Responsabilidad: Funciones compartidas entre páginas.

Funciones: - togglePassword(inputId): Cambia tipo de input password - mostrarFormulario(tipo): Cambia formularios - checkSession(): Verifica sesión con servidor - logout(): Cierra sesión - initUserAccess(): Inicializa página de acceso - mostrarUsuarioLogueado(user): Muestra UI de usuario logueado - actualizarUsername(): Actualiza username - actualizarEmail(): Actualiza email - actualizarPassword(): Actualiza contraseña - register(): Registra usuario - mostrarModosJuego(): Muestra selector de modos - volverInicio(): Regresa a inicio - seleccionarModo(modo): Redirige a modo

4. frontend/assets/js/game.js - Navegación de Modos

Responsabilidad: Navegación entre modos de juego.

Funciones: - mostrarModosJuego(): Muestra selector - volverInicio(): Oculta selector - seleccionarModo(modo): Redirige a modo (‘game’ o ‘tracing’)

5. frontend/assets/js/scriptGame.js - Lógica del Modo Juego

Responsabilidad: Toda la lógica del modo juego multi-jugador.

Variables Globales:

```
let cantRondas = 4;
let rondaActual = 1;
let cantJugadores = 2;
let jugadorActual = 1;
let historialMovimientos = [];
let puntajesPorJugadorPorRonda = {}; // { "Jugador1_Ronda1": 10 }
let estadoPorJugadorPorRonda = {}; // { "Jugador1_Ronda1": {...} }
let kingPorJugadorPorRonda = {}; // { "Jugador1_Ronda1": true }
let bolsaGrandeRonda = []; // Bolsa principal
let bolsasJugadoresRonda = {}; // { "1": [...], "2": [...] }
let jugadoresLogueados = {}; // { "1": { id, username, token } }
let partidaIniciada = false;
let partidaIdActual = null;
let ordenTurnos = [];
let indiceTurnoActual = 0;
let dinosauriosColocadosRonda = {}; // { "1": 0, "2": 0 }
```

Funciones Principales:

Inicialización: - inicializarControles(): Configura selectores - actualizarJugadores(): Actualiza UI según cantidad de jugadores - mostrarFormulariosLogin(): Muestra formularios de login por jugador

Gestión de Jugadores: - loginJugador(numeroJugador): Login de un jugador específico - cerrarSesionJugador(numeroJugador): Logout de un jugador - verificarTodosLogueados(): Verifica que todos estén logueados - guardarJugadoresEnLocalStorage(): Persiste jugadores - cargarJugadoresDesdeLocalStorage(): Restaura jugadores

Gestión de Partidas: - crearPartida(): Inicia nueva partida - guardarPartida(): Guarda partida en BD - mostrarPartidas(): Lista partidas guardadas - cargarPartida(partidaId): Carga partida desde BD - continuarPartida(partida, datos): Reanuda partida - reanudarPartida(): Reanuda partida actual

Gestión de Rondas: - crearBolsaRonda(): Crea bolsa con 10 de cada especie - mezclarArray(array): Algoritmo Fisher-Yates - distribuirBolsas(): Distribuye dinosaurios a jugadores - guardarRonda(): Calcula puntajes y guarda estado - guardarRondaEnBD(): Guarda ronda en BD automáticamente - terminarRonda(): Finaliza ronda y pasa a siguiente - restaurarRonda(): Restaura estado de una ronda

Sistema de Turnos: - elegirOrdenTurnos(): Aleatoriza orden de turnos - comenzarTurno(): Inicia turno del jugador actual - siguienteTurno(): Pasa al siguiente jugador - actualizarInfoPartida(): Actualiza UI de información

Drag & Drop: - dragStart(e): Inicia arrastre - dropRecintoJugador(e, recintoId, jugadorNum): Maneja drop - handleRecintoClickJugador(e, recintoId, jugadorNum): Click en recinto - handleDinoClick(e): Click en dinosaurio de bolsa

Cálculo de Puntajes: - calcularPuntajeJugadorRonda(jug, ronda): Calcula puntaje total - actualizarPuntajes(): Actualiza UI de puntajes

Renderizado: - renderizarTableroActivo(): Renderiza tablero del jugador activo - actualizarTableroJugadorActivo(): Actualiza tablero - actualizarBolsaJugadorActivo():

Actualiza bolsa visible - actualizarBolsaGrande(): Actualiza bolsa grande (UI) - actualizarBolsasJugadores(): Actualiza bolsas de jugadores (UI)

Vista de Revisión: - mostrarVistaRevisionPartida(partida, datos): Muestra modo revisión - mostrarTablerosRondaJuego(partida, datos, numeroRonda, jugadorSeleccionado): Renderiza tableros de ronda - renderizarTableroJugadorCargadoJuego(tablero, posiciones): Renderiza tablero individual - aplicarLayoutTableroJuego(tablero): Aplica CSS layout

Utilidades: - deshacerCambio(): Deshace último movimiento - mostrarMensaje(texto, tipo): Muestra mensajes - reiniciarTablerosRonda(): Limpia tableros para nueva ronda

6. frontend/assets/js/scriptSeguimiento.js - Lógica del Modo Seguimiento

Responsabilidad: Lógica del modo seguimiento (un jugador).

Variables Globales:

```
let cantRondas = 4;
let rondaActual = 1;
let historialMovimientos = [];
let puntajesPorRonda = {}; // { "Ronda 1": 10 }
let estadoPorRonda = {}; // { "Ronda 1": {...} }
let kingPorRonda = {}; // { "Ronda 1": true }
let contadorEspecies = {};
let bolsaRonda = []; // 60 dinosaurios (10 de cada especie)
let selectedDino = null;
```

Funciones Principales:

Gestión de Rondas: - actualizarRondas(): Actualiza selector de rondas - restaurarRonda(): Restaura estado de ronda - guardarRonda(): Guarda estado actual

Partidas: - guardarPartida(): Guarda partida completa - mostrarPartidas(): Lista partidas guardadas - cargarPartida(partida): Carga partida desde BD - cargarPartidaDesdeURL(): Carga desde parámetro URL

Vista de Partidas Cargadas: - mostrarVistaPartidaCargada(partida): Muestra vista de múltiples tableros - mostrarTablerosRonda(partida, numeroRonda): Renderiza tableros de ronda - renderizarTableroJugadorCargado(tablero, posiciones): Renderiza tablero individual - aplicarLayoutTablero(tablero): Aplica CSS layout

Drag & Drop: - dragStart(e): Inicia arrastre - dropRecinto(e, recintoId): Maneja drop - handleRecintoClick(e, recintoId): Click en recinto

Cálculo de Puntajes: - calcularPuntaje(): Calcula puntaje total - inicializarPuntajes(): Inicializa UI de puntajes

Renderizado: - renderizarTablero(): Renderiza tablero completo - crearBolsaRonda(): Crea bolsa de 60 dinosaurios

Estilos CSS

1. frontend/assets/css/styles.css - Estilos Generales

Responsabilidad: Estilos compartidos de la aplicación.

Componentes: - Navegación (barra superior) - Botones y formularios - Contenedores y layouts - Responsive design

2. frontend/assets/css/game.css - Estilos del Modo Juego

Responsabilidad: Estilos específicos del modo juego.

Componentes: - Layout de juego (sidebar + tablero) - Tableros de jugadores - Bolsas de dinosaurios - Paneles de información - Vista de revisión (múltiples tableros)

3. frontend/assets/css/seguimiento.css - Estilos del Modo Seguimiento

Responsabilidad: Estilos del modo seguimiento.

Componentes: - Tablero único - Selector de rondas - Vista de partidas cargadas - Contenedor de tableros múltiples

Recintos del Juego

Los recintos son áreas del tablero donde se colocan dinosaurios:

1. **SAME_FOREST** (Bosque de la Semejanza)
   - 4 slots
   - Regla: Todos deben ser de la misma especie
   - Puntos: Valor del último slot ocupado
2. **DIFFERENT_MEADOW** (Pradera de la Diferencia)
   - 4 slots
   - Regla: Todos deben ser de especies diferentes
   - Puntos: Valor del último slot ocupado
3. **KING** (Rey de la Selva)
   - 1 slot
   - Regla: +7 puntos si hay exactamente 1 dinosaurio y tienes el rey
   - Puntos: 7 (si cumple condiciones)
4. **TRIO_TREES** (Trío Frondoso)
   - 3 slots
   - Regla: +7 puntos si hay exactamente 3 dinosaurios
   - Puntos: 7 (si hay 3)
5. **LONELY** (Isla Solitaria)
   - 1 slot
   - Regla: +7 puntos si hay exactamente 1 dinosaurio y no se repite en otro recinto
   - Puntos: 7 (si cumple condiciones)
6. **LOVE_MEADOW** (Pradera del Amor)

   - 6 slots
   - Regla: +5 puntos por cada par de la misma especie
   - Puntos: 5 × número de pares
7. **RIVER** (Río)
   - 10 slots
   - Regla: +1 punto por cada dinosaurio
   - Puntos: Número de dinosaurios

Bonus: Cada recinto con un T-Rex otorga +1 punto adicional.

Especies de Dinosaurios

1. **tRex**: T-Rex (imagen: tRex.png)
2. **SPECIE_1**: Especie 1 (imagen: specie1.png)
3. **SPECIE_2**: Especie 2 (imagen: specie2.png)
4. **SPECIE_3**: Especie 3 (imagen: specie3.png)
5. **SPECIE_4**: Especie 4 (imagen: specie4.png)
6. **SPECIE_5**: Especie 5 (imagen: specie5.png)

API Endpoints

Base URL

http://localhost:8012/Tricosaurus/backend/api/index.php

Autenticación

POST /backend/api/index.php?path=auth/register

Registra un nuevo usuario.

Request Body:

```
{
  "username": "usuario123",
  "email": "usuario@example.com",
  "password": "password123"
}
```

Response (201 Created):

```
{
  "success": true,
  "message": "Usuario creado exitosamente.",
  "user": {
    "id": 1,
    "username": "usuario123",
    "email": "usuario@example.com"
  }
}
```

Response (400 Bad Request):

```
{
  "success": false,
  "code": "short_username",
  "message": "El nombre de usuario debe tener al menos 5 caracteres."
}
```

POST /backend/api/index.php?path=auth/login

Inicia sesión.

Request Body:

```
{
  "identifier": "usuario123",  // o email
  "password": "password123"
}
```

Response (200 OK):

```
{
  "success": true,
  "message": "Inicio de sesión exitoso",
  "token": "session_token_here",
  "user": {
    "id": 1,
    "username": "usuario123",
    "email": "usuario@example.com"
  }
}
```

Response (401 Unauthorized):

```
{
  "success": false,
  "code": "unauthorized",
  "message": "Credenciales inválidas"
}
```

GET /backend/api/index.php?path=auth/session

Verifica sesión activa.

Response (200 OK):

```
{
  "success": true,
  "authenticated": true,
  "user": {
    "id": 1,
    "username": "usuario123",
    "email": "usuario@example.com"
```

```
  }
}
```

POST /backend/api/index.php?path=auth/logout

Cierra sesión.

Response (200 OK):

```
{
  "success": true,
  "message": "Sesión cerrada correctamente"
}
```

POST /backend/api/index.php?path=auth/update-username

Actualiza nombre de usuario.

Request Body:

```
{
  "username": "nuevo_usuario",
  "password": "password_actual"
}
```

POST /backend/api/index.php?path=auth/update-email

Actualiza email.

Request Body:

```
{
  "email": "nuevo@email.com",
  "password": "password_actual"
}
```

POST /backend/api/index.php?path=auth/update-password

Actualiza contraseña.

Request Body:

```
{
  "newPassword": "nueva_password",
  "password": "password_actual"
}
```

Partidas

GET /backend/api/index.php?path=partidas/listar

Lista todas las partidas del usuario autenticado.

Response (200 OK):


```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "tipo": "juego",
      "cant_jugadores": 2,
      "cant_rondas": 4,
      "fecha": "2024-01-15 10:30:00"
    },
    {
      "id": 2,
      "tipo": "seguimiento",
      "cant_jugadores": 1,
      "cant_rondas": 4,
      "fecha": "2024-01-14 15:20:00"
    }
  ]
}
```


GET /backend/api/index.php?path=partidas/obtener&id=123

Obtiene una partida específica por ID.

Response (200 OK):


```
{
  "success": true,
  "data": {
    "id": 123,
    "tipo": "juego",
    "cant_jugadores": 2,
    "cant_rondas": 4,
    "datos": {
      "rondas": {
        "1": {
          "bolsaGrande": ["tRex", "SPECIE_1", ...],
          "bolsasJugadores": {
            "1": ["tRex", "SPECIE_2"],
            "2": ["SPECIE_1", "SPECIE_3"]
          },
          "jugadores": {
            "1": {
              "puntaje": 15,
              "esRey": true,
              "posiciones": {
                "SAME_FOREST": {
                  "0": "tRex_copy_1234567890",
                  "1": "tRex_copy_1234567891"
                },
```


```
            "KING": {
              "0": "tRex_copy_1234567892"
            }
          }
        },
        "2": {
          "puntaje": 12,
          "esRey": false,
          "posiciones": {
            "DIFFERENT_MEADOW": {
              "0": "SPECIE_1_copy_1234567890",
              "1": "SPECIE_2_copy_1234567891"
            }
          }
        }
      }
    }
  },
  "fecha": "2024-01-15 10:30:00"
 }
}
```


POST /backend/api/index.php?path=partidas

Guarda una nueva partida.

Request Body (Modo Juego):


```
{
 "tipo": "juego",
 "cantJugadores": 2,
 "cantRondas": 4,
 "rondas": {
  "1": {
   "bolsaGrande": ["tRex", "SPECIE_1", ...],
   "bolsasJugadores": {
    "1": ["tRex", "SPECIE_2"],
    "2": ["SPECIE_1", "SPECIE_3"]
   },
   "jugadores": {
    "1": {
     "puntaje": 15,
     "esRey": true,
     "posiciones": {
      "SAME_FOREST": {
       "0": "tRex_copy_1234567890"
      }
     }
    }
```

```
    }
   }
  }
}
```

Response (200 OK):

```
{
  "success": true,
  "message": "Partida guardada correctamente",
  "partida_id": 123
}
```

PUT /backend/api/index.php?path=partidas/actualizar&id=123

Actualiza una partida existente.

Request Body: Mismo formato que POST

Response: Mismo formato que POST

Flujos de Datos

Flujo de Guardado de Partida (Modo Juego)

```
Frontend (scriptGame.js)
    │
    ├─> guardarPartida()
    │    └─> Construye objeto con datos de rondas
    │       {
    │         tipo: "juego",
    │         cantJugadores: 2,
    │         cantRondas: 4,
    │         rondas: {
    │           1: {
    │             bolsaGrande: [...],
    │             bolsasJugadores: {...},
    │             jugadores: {
    │               1: { puntaje, esRey, posiciones: {...} }
    │             }
    │           }
    │         }
    │       }
    │
    └─> POST /backend/api/index.php?path=partidas
         │
         └─> PartidaController->guardar()
              │
              └─> PartidaService->guardarPartida()
```

```
│
├─> procesarDatosRondasJuego()
│    └─> Organiza datos por ronda
│
└─> PartidaRepository->insertarPartidaJuego()
     │
     ├─> INSERT INTO partidas (tipo='juego')
     ├─> INSERT INTO bolsas_ronda (dinosaurios JSON)
     ├─> INSERT INTO bolsas_jugador (dinosaurios JSON)
     ├─> INSERT INTO puntajes_jugador
     └─> INSERT INTO posiciones_jugador
       (dinosaurio_id como VARCHAR, no INT)
```

Flujo de Carga de Partida

```
Frontend (scriptGame.js)
  │
  └─> GET /backend/api/index.php?path=partidas/obtener&id=123
       │
       └─> PartidaController->obtener()
            │
            └─> PartidaService->obtenerPartida()
                 │
                 └─> PartidaRepository->obtenerPartidaPorId()
                      │
                      ├─> SELECT FROM partidas
                      ├─> SELECT FROM bolsas_ronda
                      ├─> SELECT FROM bolsas_jugador
                      ├─> SELECT FROM puntajes_jugador
                      ├─> SELECT FROM posiciones_jugador
                      │
                      └─> Reconstruye estructura JSON
                        {
                          datos: {
                            rondas: {
                              1: {
                                bolsaGrande: [...],
                                bolsasJugadores: {...},
                                jugadores: {
                                  1: { puntaje, esRey, posiciones: {...} }
                                }
                              }
                            }
                          }
                        }
                      │
                      └─> Frontend recibe datos
                           │
                           └─> cargarPartida()
                              ├─> Restaura variables globales
```

```
        ├─> mostrarVistaRevisionPartida()
        └─> Renderiza tableros de todas las rondas
```

Flujo de Autenticación

```
Frontend (login.js)
  │
  └─> login()
      │
      └─> POST /backend/api/index.php?path=auth/login
          │   { identifier, password }
          │
          └─> AuthController->login()
              │
              └─> AuthService->login()
                  │
                  ├─> UserRepository->findByUsername()
                  │   o findByEmail()
                  │
                  └─> password_verify()
                      │
                      ├─> Si válido:
                      │   ├─> $_SESSION['user_id'] = user_id
                      │   └─> Retorna { success: true, token, user }
                      │
                      └─> Si inválido:
                          └─> Retorna { success: false, message }
          │
          └─> Frontend recibe respuesta
              │
              ├─> Si success:
              │   ├─> localStorage.setItem('token', token)
              │   ├─> localStorage.setItem('user', JSON.stringify(user))
              │   └─> Redirige a index.html
              │
              └─> Si error:
                  └─> Muestra mensaje de error
```

Flujo de Cálculo de Puntajes

```
Frontend (scriptGame.js)
  │
  └─> guardarRonda()
      │
      └─> calcularPuntajeJugadorRonda(jug, ronda)
          │
          ├─> Recorre recintos: SAME_FOREST, DIFFERENT_MEADOW, ...
          │
          ├─> SAME_FOREST:
          │   └─> Verifica que todos sean misma especie
          │       └─> Puntos = valor del último slot ocupado
          │
          │
```

```
├─> DIFFERENT_MEADOW:
│   └─> Verifica que todos sean especies diferentes
│       └─> Puntos = valor del último slot ocupado
│
├─> KING:
│   └─> Si hay exactamente 1 dinosaurio
│       └─> Puntos = 7 (y esRey = true)
│
├─> TRIO_TREES:
│   └─> Si hay exactamente 3 dinosaurios
│       └─> Puntos = 7
│
├─> LONELY:
│   └─> Si hay exactamente 1 y no se repite en otros recintos
│       └─> Puntos = 7
│
├─> LOVE_MEADOW:
│   └─> Cuenta pares de misma especie
│       └─> Puntos = 5 × número de pares
│
├─> RIVER:
│   └─> Puntos = número de dinosaurios
│
└─> Bonus T-Rex:
    └─> +1 por cada recinto con T-Rex
    │
    └─> Total = suma de todos los puntos
        │
        └─> Guarda en puntajesPorJugadorPorRonda
            │
            └─> Actualiza UI
```

Configuración e Instalación

Requisitos del Sistema

- **Servidor Web**: Apache (XAMPP recomendado)
- **PHP**: 7.4 o superior
- **MySQL**: 8.0 o superior
- **Navegador**: Chrome, Firefox, Edge (últimas versiones)

Instalación

1. Configurar Base de Datos

```
# 1. Crear base de datos
mysql -u root -p < dataBaseScripts/create_database_full.sql
```

```
# 2. (Opcional) Insertar datos de ejemplo
mysql -u root -p tricosaurus < dataBaseScripts/insert_sample_data.sql
```

2. Configurar Conexión a Base de Datos

Editar backend/inc/config/config:

```
{
  "conectionDB": {
    "server": "localhost",
    "user": "root",
    "password": "",
    "database": "tricosaurus",
    "port": 3306
  }
}
```

3. Configurar URL Base

En todos los archivos JavaScript, actualizar las URLs:

```
// Cambiar de:
http://localhost:8012/Tricosaurus/

// A tu configuración:
http://tu-dominio.com/Tricosaurus/
```

Archivos a actualizar: - frontend/assets/js/auth.js - frontend/assets/js/login.js - frontend/assets/js/scriptGame.js - frontend/assets/js/scriptSeguimiento.js - frontend/assets/js/script.js - Todos los archivos HTML que referencian imágenes o scripts

4. Configurar Servidor

XAMPP: 1. Colocar proyecto en C:\xampp\htdocs\Tricosaurus 2. Iniciar Apache y MySQL 3. Acceder a http://localhost:8012/Tricosaurus/frontend/index.html

Apache Manual: - Configurar VirtualHost apuntando a frontend/ - Habilitar mod_rewrite si es necesario

Variables de Configuración

Backend

- **Database**: backend/inc/config/config (JSON)
- **CORS**: Configurado en backend/api/index.php
- **Session**: Usa sesiones PHP nativas

Frontend

- **API URL**: Definida en cada script JavaScript
- **LocalStorage**: Usa token y user
- **Imágenes**: Rutas absolutas en JavaScript

Guía de Desarrollo

Estructura de Código

Backend

1. **Controllers**: Manejan peticiones HTTP
2. **Services**: Contienen lógica de negocio
3. **Repositories**: Acceso a base de datos
4. **Config**: Configuración y conexiones

Frontend

1. **HTML**: Estructura de páginas
2. **CSS**: Estilos por módulo
3. **JavaScript**: Lógica por funcionalidad

Convenciones de Código

PHP

- **Nombres de clases**: PascalCase (PartidaController)
- **Nombres de métodos**: camelCase (guardarPartida)
- **Variables**: camelCase ($cantJugadores)
- **Constantes**: UPPER_SNAKE_CASE

JavaScript

- **Variables**: camelCase (cantJugadores)
- **Funciones**: camelCase (guardarPartida)
- **Constantes**: UPPER_SNAKE_CASE (API_URL)
- **Clases**: PascalCase (Auth)

Debugging

Backend

- Errores PHP: ini_set('display_errors', 1) en index.php
- Logs: Revisar error_log de Apache
- Base de datos: Usar herramientas como phpMyAdmin

Frontend

- Console: console.log(), console.error()
- Network: Revisar peticiones en DevTools
- LocalStorage: Ver en Application/Storage

Testing

Manual

1. **Autenticación**:
   - Registrar usuario
   - Iniciar sesión
   - Verificar sesión

   - Cerrar sesión
2. **Modo Seguimiento**:
   - Crear partida
   - Colocar dinosaurios
   - Guardar partida
   - Restaurar partida
3. **Modo Juego**:
   - Crear partida con 2+ jugadores
   - Login de jugadores
   - Colocar dinosaurios
   - Avanzar turnos
   - Guardar y restaurar

Mejoras Futuras

1. **Seguridad**:
   - Implementar CSRF tokens
   - Validar inputs más estrictamente
   - Sanitizar outputs
2. **Performance**:
   - Cachear consultas frecuentes
   - Optimizar imágenes
   - Minificar JavaScript/CSS
3. **Funcionalidades**:
   - Sistema de amigos
   - Estadísticas de partidas
   - Modo online en tiempo real
   - Notificaciones
4. **Testing**:
   - Unit tests (PHPUnit)
   - Integration tests
   - E2E tests (Selenium)

Notas Adicionales

IDs de Dinosaurios

Los dinosaurios tienen IDs únicos generados con timestamp:

```
clone.id = especie + "_copy_" + Date.now();
// Ejemplo: "tRex_copy_1234567890"
```

Al guardar, se almacena el ID completo en dinosaurio_id (VARCHAR).

Formato de Posiciones

Las posiciones se guardan como objetos con índices:


```
{
  "SAME_FOREST": {
    "0": "tRex_copy_1234567890",
    "1": "tRex_copy_1234567891"
  },
  "KING": {
    "0": "tRex_copy_1234567892"
  }
}
```


Esto permite mantener las posiciones exactas sin arrays con null.

Sistema de Sesiones

- Backend usa $_SESSION['user_id']
- Frontend usa localStorage con token
- La sesión PHP se mantiene mediante cookies

Manejo de Errores

- Backend: Retorna JSON con success: false y message
- Frontend: Muestra mensajes al usuario con mostrarMensaje()
- Errores críticos: Se registran en logs del servidor

Repositorio en GitHub

El código fuente completo del proyecto TricoTech (Tricosaurus) está disponible en GitHub. Allí se encuentran las versiones actualizadas, ramas de desarrollo y documentación técnica adicional.

Repositorio oficial: https://github.com/Fabri1899/TricoTech

Para contribuir o revisar cambios:
- Clonar el repositorio usando: git clone https://github.com/Fabri1899/TricoTech
- Consultar el archivo README.md para instrucciones de instalación y ejecución.
- Enviar pull requests o issues en caso de errores o mejoras propuestas.