RunnerJS • Proyecto Integrador

Juego tipo «dinosaurio de Google» implementado con HTML, CSS, JavaScript (Canvas) y backend Express para Leaderboard.

Autor	LUIS MANUEL CORONA
Repositorio	href="https://github.com/LuisCsr/RunnerJS.git" color="blu <link
Sitio publicado	href="https://luiscsr.github.io/RunnerJS/#inicio" color="blu

---

Objetivo y descripción del proyecto

Construir unrunner2D minimalista inspiradoen el dinosaurio de Google. El juego corre en un lienzo **<canvas>** con un bucle de juego basado en requestAnimationFrame, gestiona niveles, colisiones, puntuación y un Leaderboard persistente. El backend en Node/Express expone endpoints para consultar y registrar puntuaciones, almacenando datos en **data/scores.json**.

Estructura del repositorio

```
(repo raíz)
▪▪ /frontend
▪   ▪▪ index.html                  # Landing (enlaza prácticas y juego)
▪   ▪▪ /practicas/...              # Todas tus prácticas previas
▪   ▪▪ /proyecto
▪   ▪▪   ▪▪ game.html
▪   ▪▪   ▪▪ game.css
▪   ▪     ▪▪ game.js               # loop, niveles, colisiones, HUD
▪         ▪▪ api.js                # fetch a la API (scores)
▪         ▪▪ assets/               # sprites opcionales (o shapes)
▪▪ /backend
▪   ▪▪ server.js
▪   ▪▪ scores.routes.js
▪   ▪▪ data/scores.json
▪   ▪▪ package.json
```

Temas aplicados

• Canvas 2D:dibujo de sprites geométricos (player, obstáculos) y HUD.

• Bucle de juego: init → update(dt) → render(ctx) sincronizado con requestAnimationFrame.

• Física básica: gravedad, salto con impulso, velocidad incremental por nivel.

• Colisiones AABB (Axis-Aligned Bounding Box) y ventanas de invulnerabilidad.

• Gestión de niveles y dificultad por tiempo/score.

• Módulos JS: separación en **game.js** (lógica) y **api.js** (red).

• Persistencia: Leaderboard con API Express + **scores.json** y fallback localStorage.

• Accesibilidad: foco visible, navegación por teclado, skip link, contraste alto.

• Responsive y reduced motion respetando prefers-reduced-motion.

Diagramas de maquetado y funcionamiento

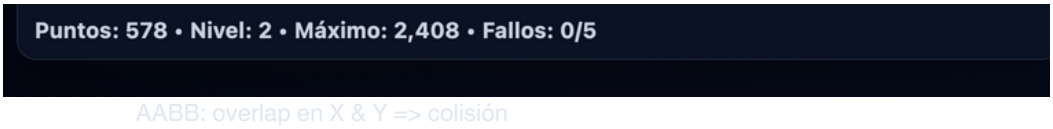
Pantalla de juego (wireframe)

RunnerJS — Juego

SCORE: 000123

Diagrama del bucle de juego

Diagrama de detección de colisiones (AABB)



Arquitectura de Leaderboard (frontend ↔ API ↔ JSON)

Código fuente comentado (extractos)

frontend/index.html (resumen)

```
<html lang="es">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>RunnerJS • Portafolio y Prácticas</title>
<!-- Estilos con variables CSS, modo oscuro/claro y accesibilidad -->
</head>
<body>
<a class="sr-only visually-hidden-focusable" href="#contenido">Saltar al contenido</a>
<header>… navegación sticky con resaltado por intersección …</header>
<main id="contenido" class="container">

    <section id="proyecto">
      <a href="./proyecto/game.html" class="button">■ Jugar RunnerJS</a>
    </section>
 </main>
<script>
    // IO para resaltar la sección activa + año dinámico
  </script>
</body>
</html>
```

proyecto/game.js (resumen)

```
// game.js — bucleprincipal,  estados y colisiones
const canvas = document.getElementById('game');
const ctx = canvas.getContext('2d');

let last = 0, score = 0, level = 1;
const player = { x:40, y:0, w:36, h:36, vy:0, onGround:false };
const obstacles = [];

function init(){
spawnObstacle();
requestAnimationFrame(loop);
}

function loop(ts){
const dt = (ts - last) / 1000; last = ts;
update(dt);
render();
requestAnimationFrame(loop);
}

function update(dt){
// gravedad + salto
player.vy += 1200 * dt;
player.y += player.vy * dt;
if(player.y >= groundY - player.h){ player.y = groundY - player.h; player.vy = 0; player.onGround = t
// mover obstáculos y reciclar
// aumentar dificultad por tiempo/score
if (checkCollision(player, obstacles)) gameOver();
}

function checkCollision(a, obs){
// AABB contra cada obstáculo
return obs.some(b => a.x < b.x + b.w && a.x + a.w > b.x && a.y < b.y + b.h && a.y + a.h > b.y);
}
```

proyecto/api.js (resumen)

```
// api.js — capa deredpara   Leaderboard
const API_URL = 'http://localhost:3000/api/scores';
```

```
export async function getScores(){
const res = await fetch(API_URL);
if(!res.ok) throw new Error('Error al leer scores');
return res.json();
}

export async function postScore(entry){
const res = await fetch(API_URL, {
    method:'POST',
    headers:{ 'Content-Type':'application/json' },
    body: JSON.stringify(entry)
 });
if(!res.ok) throw new Error('Error al guardar score');
return res.json();
}
```

backend/server.js + scores.routes.js (resumen)

```
// backend/server.js — Express + rutas
import express from 'express';
import cors from 'cors';
import scoresRouter from './scores.routes.js';

const app = express();
app.use(cors());
app.use(express.json());
app.use('/api/scores', scoresRouter);

app.listen(3000, () => console.log('API en http://localhost:3000'));
```

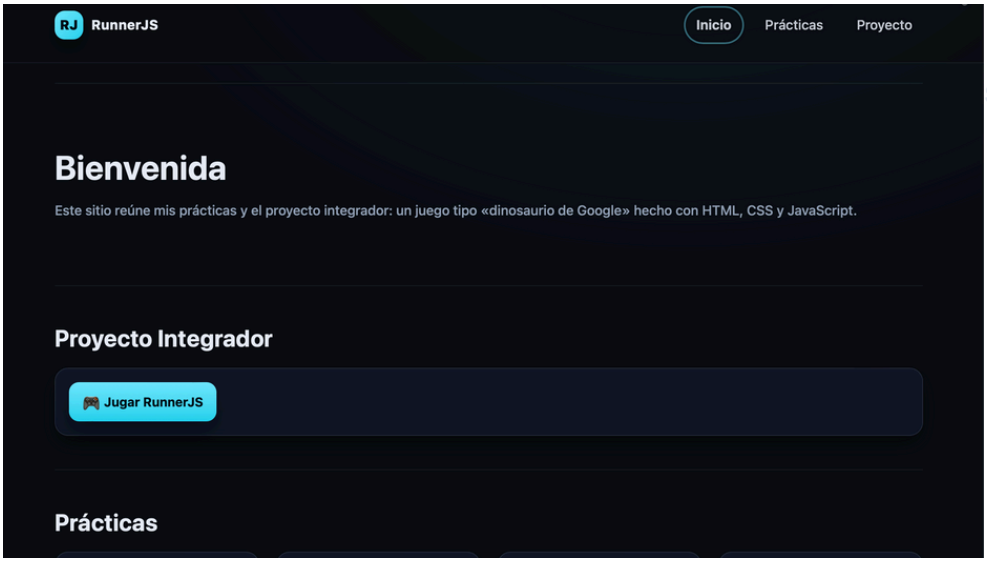
```
// backend/scores.routes.js
import { Router } from 'express';
import fs from 'fs/promises';
const router = Router();
const DB = './data/scores.json';

router.get('/', async (req, res) => {
const data = JSON.parse(await fs.readFile(DB, 'utf8'));
res.json(data.sort((a,b)=> b.score - a.score).slice(0,50));
});

router.post('/', async (req, res) => {
const { name, score, ts = Date.now() } = req.body;
if(!name || typeof score!=='number') return res.status(400).json({error:'payload inválido'});
const data = JSON.parse(await fs.readFile(DB, 'utf8'));
data.push({ name, score, ts });
await fs.writeFile(DB, JSON.stringify(data, null, 2));
res.status(201).json({ ok:true });
});

export default router;
```

Capturas de pantalla (wireframes representativos)

Seilustranlosestadosprincipalesdeljuego.EnelPDFdelrepositoriopuedesreemplazarlasporcapturas reales.

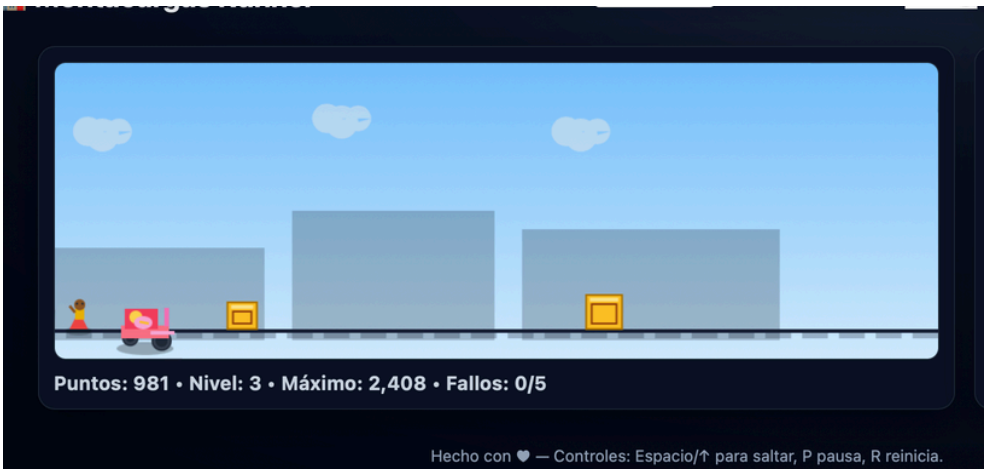
1) Inicio / Menú

Inicio — Jugar

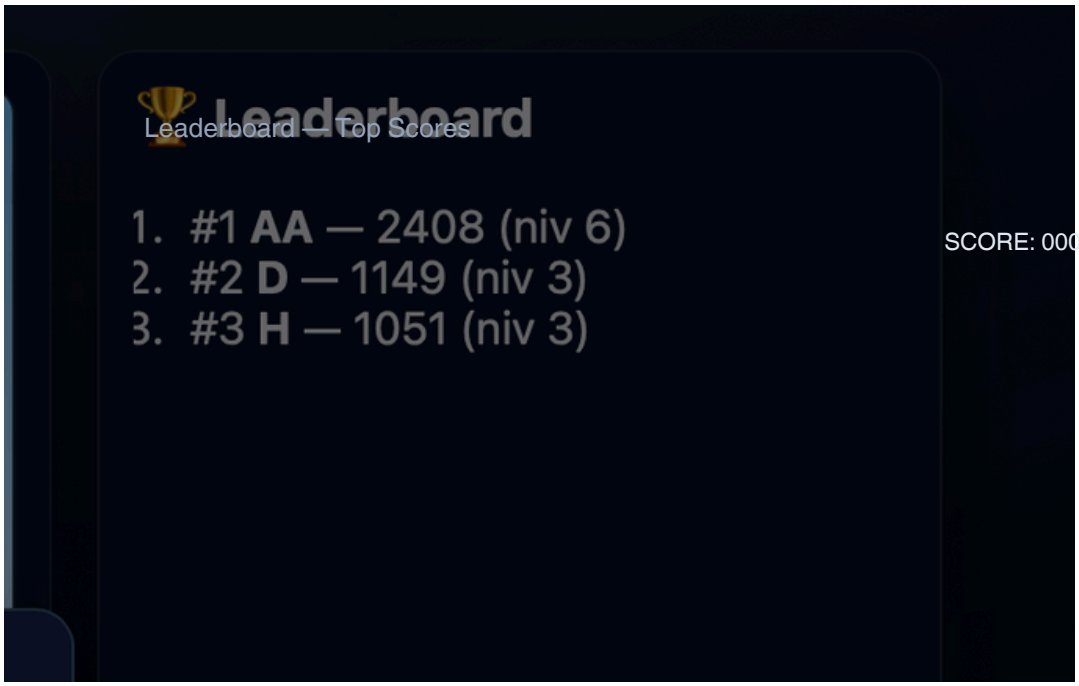


2) Gameplay (en carrera)

Gameplay — HUD & Obstáculos



3) Leaderboard



Repositorio y sitio publicado

https://github.com/LuisCsr/RunnerJS.git

https://luiscsr.github.io/RunnerJS/#inicio

Preguntas de reflexión

1. ¿Qué ventajas y limitaciones tiene usar Canvas para este juego?

Ventajas:altorendimiento para 2D simple, controltotal del render(un sololienzo),fácil animación con requestAnimationFrame, bajo peso sin dependencias. Limitaciones: accesibilidad menor que DOM/SVG, manejo manual de layout/retina, colisiones y estado más “low-level”, testing visual menos trivial.

2. ¿Cómo se diseñó la progresión de los niveles y la dificultad?

Incremento suavede velocidad de scrollyfrecuencia de obstáculos enfuncióndel tiempo/score; a cada umbral, level++ y se ajusta la variación de gaps. Se evita picos bruscos con curvas lineales y pequeñas rampas.

3. ¿Qué método de detección de colisiones se usó y por qué?

AABB (Axis-Aligned Bounding Box)porsuO(1),simpleimplementaciónysuficienteprecisión para sprites rectangulares del runner y obstáculos verticales.

4. ¿Qué mejoras harías en la accesibilidad del juego?

Añadir controles alternativos(barra espaciadora/teclas configurables),pausar con pérdida de foco, señales auditivas de colisión/level up, alternativa de alto contraste y opción ‘reduced motion’ que limita parallax y sacudidas.

5. ¿Qué validaciones aplicaste en el Leaderboard?

Enelcliente,evitarenvíos vacíos y tiposincorrectos;enel servidor,validar esquema (name obligatorio, score numérico, límites de longitud), normalizar mayúsculas/minúsculas y recortar espacios; limitar resultados y ordenar descendente.

6. ¿Cómo asegurarías la integridad de los scores guardados en el servidor?

Firmar losenvíoscon un token HMACgenerado en el servidor(no accesible al cliente),rate limiting, sanitización de entrada, almacén inmutable (append-only) y checks de integridad (hash) por registro; en producción, usar DB con roles y logs de auditoría.

7. ¿Qué diferencias notaste entre guardar datos en localStorage y en la API?

localStoragees inmediato y offline,pero inseguroy solodel navegador;laAPIcentralizadatos,

permite rankings globales y controles de integridad, aunque requiere conectividad y manejo de errores/red.

8. ¿Cómo modularizaste el código del juego para que fuera más fácil de mantener?

Separandoresponsabilidades:**game.js** (estado/bucle/input/colisiones),**api.js**(fetchy serialización), **game.css** (estilos HUD), y constantes/utilidades en módulos dedicados. Cada archivo exporta funciones claras.

9. ¿Qué pruebas hiciste para validar que las colisiones y puntuaciones funcionaran correctamente?

Pruebasmanuales con seeds deterministas (velocidadfija), escenarios forzadosde colisión (obstáculo a X,Y conocidos) y verificación del HUD; tests unitarios de checkCollision con casos borde; pruebas de API con datos válidos/ inválidos y orden correcto del Leaderboard.

10. ¿Qué mejora de alto impacto implementarías si tuvieras más tiempo?

Unsistema de misiones/desafíos diariosy power-ups (escudo/saltodoble) con persistencia, además de un modo ‘ghost’ que reproduce tu mejor carrera para incentivar la mejora.

Gracias por revisar RunnerJS. Este PDF es estático y listo para entrega; puedes reemplazar los wireframes por capturas reales del sitio publicado.