

DOCUMENTACION TECNICA LUDOTECA

Desarrollador Jr : Veraza Garcia
Amy Valentina

Fecha de elaboración: 18 de Febrero de 2026

Se desarrolló una aplicación web de tipo full-stack enfocada principalmente en el backend, cuyo propósito es administrar el inventario de una ludoteca.

El sistema permite registrar juegos, consultarlos, realizar búsquedas por texto y aplicar filtros básicos, incorporando además controles mínimos de seguridad.

La solución fue construida utilizando el stack solicitado en el reto técnico:

Node.js con TypeScript

- Express como framework HTTP

PostgreSQL como sistema de base de datos relacional

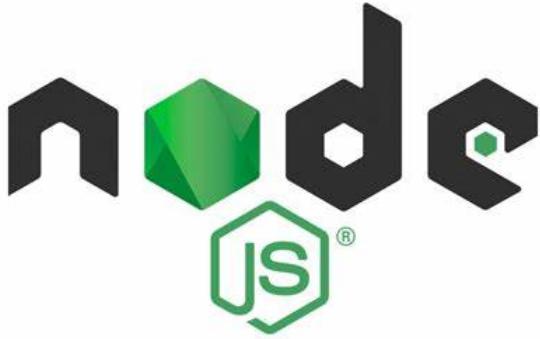
- Zod para validación de datos de entrada

The screenshot shows the Ludoteca application's inventory management interface. On the left, there's a sidebar with navigation links: 'Inventario', 'Búsqueda', 'Filtros', 'Import XML', and 'Eliminar'. The main area has two main sections: 'Buscar y filtrar' (Search and filter) and 'Registrar juego' (Register game). In the search section, there are fields for 'Nombre o tag' (e.g., UNO, familia, rápido), 'Categoría' (e.g., cartas, mesa), 'Edad mínima (min_age >=)', and 'Jugadores (players <=)'. Below these are dropdowns for 'Solo con stock' (Only with stock) and 'stock' (stock level), with 'Stock' set to 0. There are also 'Aplicar' (Apply) and 'Limpiar' (Clear) buttons. To the right, the 'Registrar juego' section shows a form with fields for 'Nombre *' (UNO), 'Categoría' (cartas), 'Edad mínima' (jugadores), 'Stock' (0), and 'Tags (coma)' (familia, rápido). A 'Crear' (Create) button is present. At the bottom, there's an 'Importar XML' (Import XML) section with a file input field ('Elegir archivo') and a 'Subir XML' (Upload XML) button. A table at the bottom lists games with columns: ID, Nombre, Categoría, Edad, Jugadores, Stock, Tags, Creado, and Acciones. It contains two entries: 'MEMORAMA' (ID 2) and 'UNO' (ID 1).

The screenshot shows the Ludoteca application's login page. The header says 'Ludoteca' and provides instructions: 'Acceso al inventario. Login con JWT + API en Node/TS + PostgreSQL.' Below are links for 'Inventario', 'Búsqueda', 'Filtros', 'Import XML', and 'Eliminar'. The main area has fields for 'Email' (e.g., amy@ludoteca.com) and 'Password'. There are 'Entrar' (Enter) and 'Registrar' (Register) buttons. At the bottom, there's a link to '/api/health' and a note: 'Tip: si aún no tienes usuario, usa "Registrar" (se crea en la DB)'.

Introducción

El presente proyecto desarrolla una aplicación backend mínima orientada a la administración de una ludoteca digital, permitiendo el registro, consulta y gestión de juegos mediante un sistema estructurado de inventario. La solución integra funcionalidades de búsqueda por texto y filtros básicos que facilitan la localización de elementos dentro del catálogo, proporcionando una experiencia de uso clara y funcional.



This project implements a minimal backend application for managing a digital board game library (“ludoteca”), providing core functionality for inventory registration, querying, text-based search, and basic filtering. The system is developed using Node.js, TypeScript, and PostgreSQL, and follows a layered architecture that separates routing, controllers, services, and database access to ensure clarity, maintainability, and scalability.

Adicionalmente, se incorporan controles esenciales de seguridad, tales como validación de entradas, consultas parametrizadas a la base de datos, almacenamiento seguro de contraseñas mediante hashing y autenticación basada en tokens JWT. Estas medidas permiten proteger la información del sistema y restringir el acceso al inventario únicamente a usuarios autenticados.

Funcionalidades implementadas

- La aplicación permite:
- Registrar juegos dentro del inventario.
- Consultar la lista completa de juegos almacenados.
- Realizar búsquedas por coincidencia de texto en nombre y etiquetas.
- Aplicar filtros por:
 - a. categoría
 - b. edad mínima
 - c. número de jugadores
 - d. disponibilidad de stock
- Verificar el estado del servicio mediante un endpoint de salud (/api/health).

The screenshot shows the Ludoteca application interface. At the top, there is a header with the logo, the title "Ludoteca", and links for "Inventario", "Búsqueda", "Filtros", "Node + TS + PostgreSQL". On the right side of the header are buttons for "/api/health" (OK), "Refrescar" (Refresh), and "Salir" (Logout).

Buscar y filtrar

- Buscar (nombre o tags): ej. UNO, familia, rapido
- Categoría: ej. cartas, mesa
- Edad mínima (min_age ≥):
- Jugadores (players =):
- Solo con stock:

 - (cualquiera)

Registrar juego

- Nombre *: UNO
- Categoría: cartas
- Edad mínima:
- Jugadores:
- Stock: 0
- Tags (coma): familia, rápido

Importar XML

Archivo XML *

Elegir archivo: games.xml

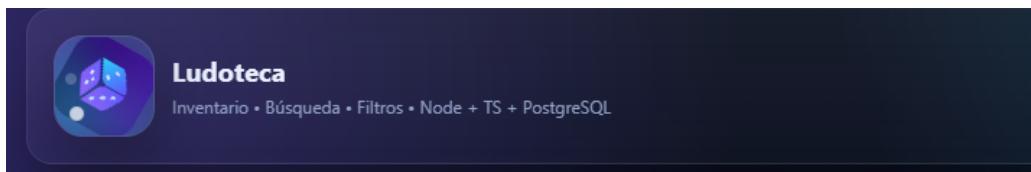
Se enviará a /api/games/import-xml

Subir XML Importado

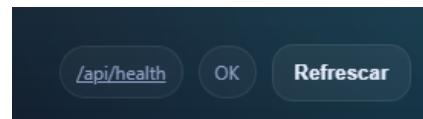
Lista de Juegos

ID	Nombre	Categoría	Edad	Jugadores	Stock	Tags	Creado	Acciones
4	twister	tacto	6	5	20	familia	18/2/2026, 12:52:48	Eliminar
2	MEMORAMA	cartas	7	3	1	familia	16/2/2026, 21:47:24	Eliminar

Código

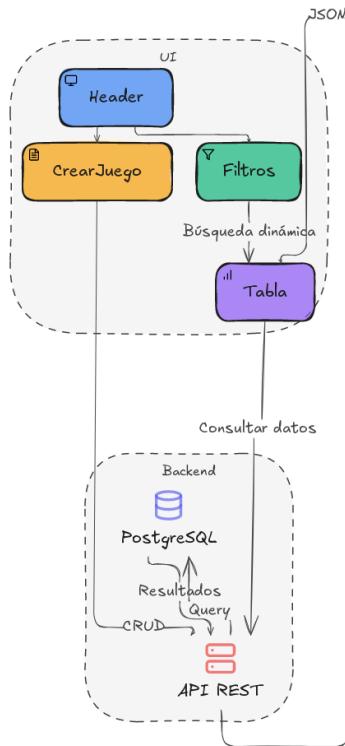


- Descripción: Inventario · Búsqueda · Filtros · Node + TS + PostgreSQL
- Endpoint de salud: /api/health → OK
- Botones:
 - Refrescar: vuelve a consultar los juegos al backend.

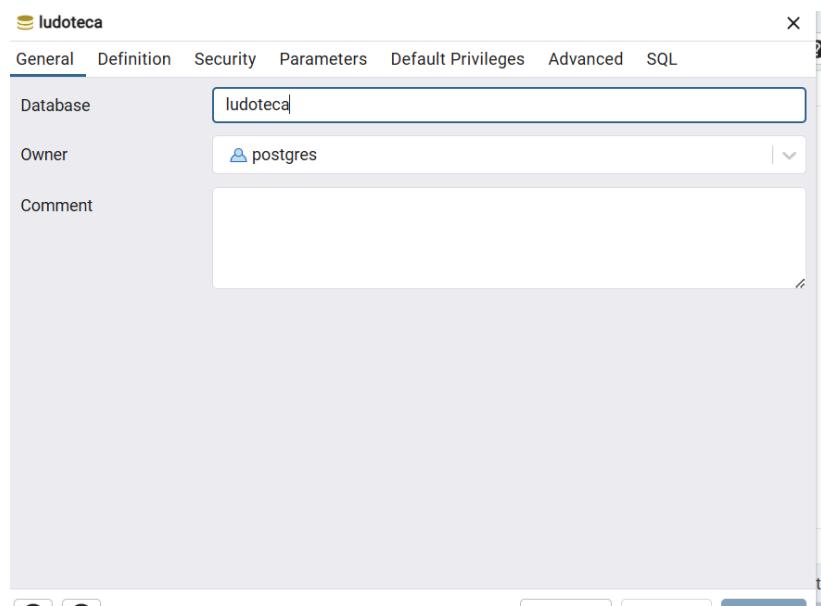


Salir: cierra sesión o limpia estado (según implementación).

```
{"ok":true,"dbTime":{"now":"2026-02-18T04:57:46.343Z"},"gamesTableExists":true,"env":{"DB_HOST":"localhost","DB_PORT":5432,"DB_USER":"postgres","DB_NAME":"ludoteca"}}
```



Base de datos



.env

```

PORT=3000
DB_HOST=localhost

DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=
DB_NAME=ludoteca

```

Categoría

Filtra por category.

WHERE category = \$1

Edad mínima (min age ≥)

Muestra juegos cuya edad mínima sea **mayor o igual**.

WHERE min_age >= \$1

Jugadores (players =)

Filtra por número exacto de jugadores.

WHERE players = \$1

The screenshot shows a database interface with a SQL editor and a results table.

SQL Editor:

```
SELECT * FROM games
WHERE category = 'cartas';
```

Results Table:

	id [PK] integer	name text	category text	min_age integer	players integer	stock integer	tags text[]	created_at timestamp without time zone
1	1	UNO	cartas	6	4	10	{familia,rapido}	2026-02-16 21:31:35.447818
2	2	MEMORAMA	cartas	7	3	1	{familia}	2026-02-16 21:47:24.762902

Diagrama UML

USERS		
int	id	PK
string	email	
string	password_hash	
timestamp	created_at	

creates

GAMES		
int	id	PK
string	name	
string	category	
int	min_age	
int	players	
int	stock	
string	tags	
timestamp	created_at	
int	user_id	FK

Carga variables .env con dotenv.config()

Crea Express con const app = express()

Habilita JSON en requests: app.use(express.json())

Sirve archivos estáticos desde public/ (tu HTML/CSS/JS)

Expone rutas:

/ manda login.html

/app manda index.html

/api/health prueba DB y si existe tabla games

/api/auth login/register (token)

/api/games todo lo de juegos

Si no coincide nada 404 Not found}

SCHEMA



```
ludoteca-backend > src > validators > game.schema.ts > ...
1 import { z } from "zod";
2
3 export const createGameSchema = z.object({
4   name: z.string().min(1).max(120),
5   category: z.string().min(1).max(60).optional(),
6   min_age: z.number().int().min(0).max(99).optional(),
7   players: z.number().int().min(1).max(50).optional(),
8   stock: z.number().int().min(0).max(100000).optional(),
9   tags: z.array(z.string().min(1).max(30)).max(20).optional(),
10 });
11
```

Campo	Regla
name	texto obligatorio, 1-120 caracteres
category	texto opcional
min_age	número entero ≥0
players	entero ≥1
stock	entero ≥0
tags	lista de textos

utils/db.ts y pool

backend usa PostgreSQL con pg (pool de conexiones).

pool.query(...) ejecuta SQL

Reutiliza conexiones

1. Que DB responde SELECT NOW()
2. Que existe la tabla games

```
roteca-backend > src > utils > db.ts > ...
1 ↴ import { Pool } from "pg";
2 import dotenv from "dotenv";
3
4 dotenv.config();
5
6 ↴ export const pool = new Pool({
7   host: process.env.DB_HOST,
8   port: Number(process.env.DB_PORT),
9   user: process.env.DB_USER,
10  password: process.env.DB_PASSWORD,
11  database: process.env.DB_NAME,
12 });
13
14 ↴ export const query = async (text: string, params?: any[]) => {
15   return pool.query(text, params);
16 };
17
```

Routes

Definen la URL y el método:

- GET /api/games
- POST /api/games
- DELETE /api/games/:id
- POST /api/games/import-xml

Controllers

Son “el puente” entre HTTP y tu lógica.

- Leen req.params, req.body, req.query, req.file
- Llaman a un service
- Responden con res.json(...)

Services

Aquí está la lógica real y el SQL.

- createGame(...) inserta
- getGames(...) filtra y consulta
- deleteGameById(id) borra
- (más adelante) importXml(...) parsea y mete a DB

Middlewares

Se ejecutan antes del controller:

- validate(createGameSchema) valida body
- upload.single("file") procesa archivo XML con multer
- (posible) auth middleware que revisa JWT

- POST / → crear juego
- GET / → listar juegos
- POST /import-xml → subir XML
- DELETE /:id → eliminar juego

Si te falta el router.delete("/:id"... entonces **tu botón jamás podrá borrar** (te dará 404).

create GameController

- recibe JSON del formulario (name, category, min_age, players, stock, tags)
- llama createGame(req.body)
- responde 201 con el juego creado

```

export const createGameController = async (req: Request, res: Response) => {
  try {
    const game = await createGame(req.body);
    return res.status(201).json(game);
  } catch (err) {
    console.error("createGameController ERROR:", err);
    return res.status(500).json({ message: "Error creating game" });
  }
};

```

getGamesController

- recibe filtros en query: search, category, min_age, players, in_stock
- llama getGames(req.query)
- responde lista

```

export const getGamesController = async (req: Request, res: Response) => {
  try {
    const games = await getGames(req.query);
    return res.json(games);
  } catch (err) {
    console.error("getGamesController ERROR:", err);
    return res.status(500).json({ message: "Error fetching games" });
  }
};

```

importGamesXmlController

- requiere archivo en req.file
- lo lee con req.file.buffer.toString("utf-8")
- por ahora solo confirma “XML recibido”
- (luego lo parseas y haces inserts)

```

export async function importGamesXmlController(req: Request, res: Response) {
  try {
    if (!req.file) return res.status(400).json({ message: "Archivo XML requerido" });
    const xml = req.file.buffer.toString("utf-8");
    return res.json({ message: "XML recibido", size: xml.length });
  } catch {
    return res.status(500).json({ message: "Error importando XML" });
  }
}

```

deleteGameController

- lee req.params.id
- lo convierte a número
- llama deleteGameById(id)
- responde:
 - 200 si borró

- 404 si no existía

```

export const deleteGameController = async (req: Request, res: Response) => {
  try {
    const id = Number(req.params.id);
    if (!Number.isFinite(id)) return res.status(400).json({ message: "ID inválido" });

    const ok = await deleteGameById(id);
    if (!ok) return res.status(404).json({ message: "Juego no encontrado" });

    return res.json({ message: "Juego eliminado" });
  } catch (err) {
    console.error("deleteGameController ERROR:", err);
    return res.status(500).json({ message: "Error deleting game" });
  }
};

```

Crear:

Hace INSERT INTO games (...) VALUES (...) RETURNING *

Listar:

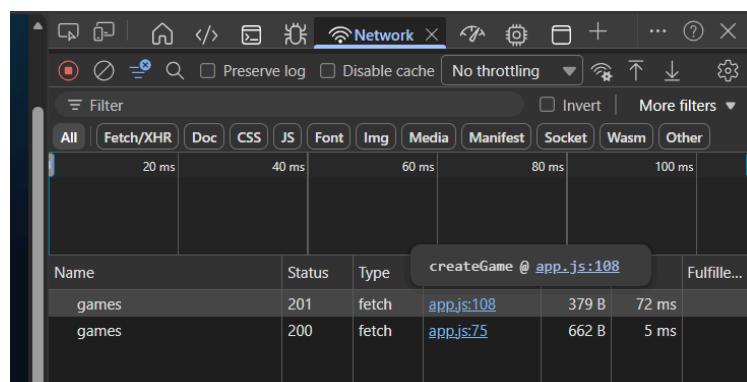
Hace SELECT ... FROM games WHERE ... con filtros

Eliminar:

Debe hacer exactamente:

DELETE FROM games WHERE id = \$1 RETURNING id

1. F12 → Network
2. Click Eliminar
3. Abre la request DELETE
4. Mira:
 - Status: 200/401/404/500
 - Response JSON



POSTMAN

```

1
2   "ok": true,
3   "dbTime": {
4     "now": "2026-02-18T05:07:43.231Z"
5   },
6   "gamesTableExists": true,
7   "env": {
8     "DB_HOST": "localhost",
9     "DB_PORT": "5432",
10    "DB_USER": "postgres",
11    "DB_NAME": "ludoteca"
12  }
13

```

Express está corriendo

PostgreSQL responde

La tabla principal existe

Las variables .env son correctas

Es como el “pulso vital” de tu sistema.

Key	Value
X-Powered-By	Express
Content-Type	application/json; charset=utf-8
Content-Length	168
ETag	W/"a8-lbVKaPkFFMxp9UMy9fNdfILDCzk"
Date	Wed, 18 Feb 2026 05:09:33 GMT
Connection	keep-alive
Keep-Alive	timeout=5

El endpoint /api/health básicamente te está diciendo que tu backend está sano. El "ok": true confirma que el servidor está corriendo sin errores. El dbTime demuestra que sí hay conexión real con PostgreSQL y que la base de datos está respondiendo en tiempo real. El campo gamesTableExists: true significa que la tabla principal (games) sí existe, así que tu base está bien estructurada para hacer crear, listar o borrar juegos.

SEGURIDAD

Primero implementamos un sistema de autenticación basado en JSON Web Tokens (JWT). Creamos dos endpoints principales: /api/auth/register y /api/auth/login. En el registro, la contraseña no se guarda en texto plano; usamos bcrypt para generar un hash seguro antes de almacenarlo en PostgreSQL. Esto garantiza que, aunque alguien acceda a la base de datos, no pueda ver las contraseñas reales.

En el login, el backend valida el email y compara la contraseña enviada con el hash guardado usando bcrypt.compare(). Si las credenciales son correctas, el servidor genera un token JWT firmado con una clave secreta (JWT_SECRET) y lo devuelve al cliente. Ese token contiene información como el sub (id del usuario) y el email, y tiene fecha de expiración.

```
PS C:\WINDOWS\system32> Invoke-RestMethod -Method Post `>>   -Uri "http://localhost:3000/api/auth/register" `>>   -ContentType "application/json" `>>   -Body '>{"email":"test@test.com", "password":"123456"}'  
  
id email      created_at  
-- ----  
3 test@test.com 2026-02-18T19:11:23.596Z  
  
PS C:\WINDOWS\system32> |
```

Token

```
PS C:\WINDOWS\system32> Invoke-RestMethod -Method Post `>>   -Uri "http://localhost:3000/api/auth/login" `>>   -ContentType "application/json" `>>   -Body '>{"email":"test@test.com", "password":"123456"}'  
  
token  
----  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwidWlhawWiOiJ0ZXN0QHRLc3QuY29tIiwiaWF0IjoxNzcxNDQxOTMyLCJleHAiOjE...  
  
PS C:\WINDOWS\system32> |
```

```
PS C:\WINDOWS\system32> $login = Invoke-RestMethod -Method Post `>>   -Uri "http://localhost:3000/api/auth/login" `>>   -ContentType "application/json" `>>   -Body '>{"email":"test@test.com", "password":"123456"}'  
PS C:\WINDOWS\system32> Invoke-RestMethod -Method Delete `>>   -Uri "http://localhost:3000/api/games/1" `>>   -Headers @{ Authorization = "Bearer $($login.token)" }  
  
message  
----  
Juego eliminado  
  
PS C:\WINDOWS\system32> |
```

Código

```
ca-backend > src > routes > TS game.routes.ts > ...
} from "../controllers/game.controller";
import { validate } from "../middlewares/validate";
import { createGameSchema } from "../validators/game.schema";
import { upload } from "../middlewares/upload";
import { requireAuth } from "../middlewares/requireAuth";
```

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";

const JWT_SECRET = process.env.JWT_SECRET || "dev_secret";

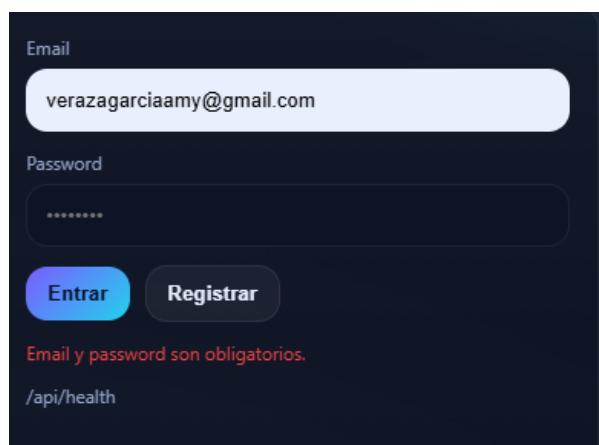
export function requireAuth(req: Request, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return res.status(401).json({ message: "Token requerido" });
  }

  const token = authHeader.split(" ")[1];

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    (req as any).user = decoded;
    next();
  } catch {
    return res.status(401).json({ message: "Token inválido" });
  }
}
```

Autenticación basada en JWT. Implementé registro y login con encriptación de contraseñas usando bcrypt, protegí rutas sensibles mediante middleware personalizado y aseguré la comunicación con PostgreSQL usando consultas parametrizadas. Además, añadí un endpoint de health check para monitoreo del sistema.



Estructura

ludoteca-backend/

src/

 controllers/ Manejo de peticiones HTTP

 routes/ Definición de endpoints de la API

 services/ Lógica de negocio y acceso a base de datos

 middlewares/ Autenticación mediante JWT

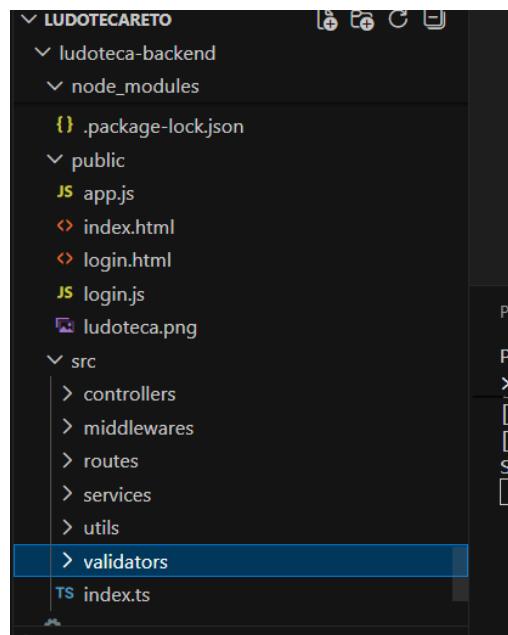
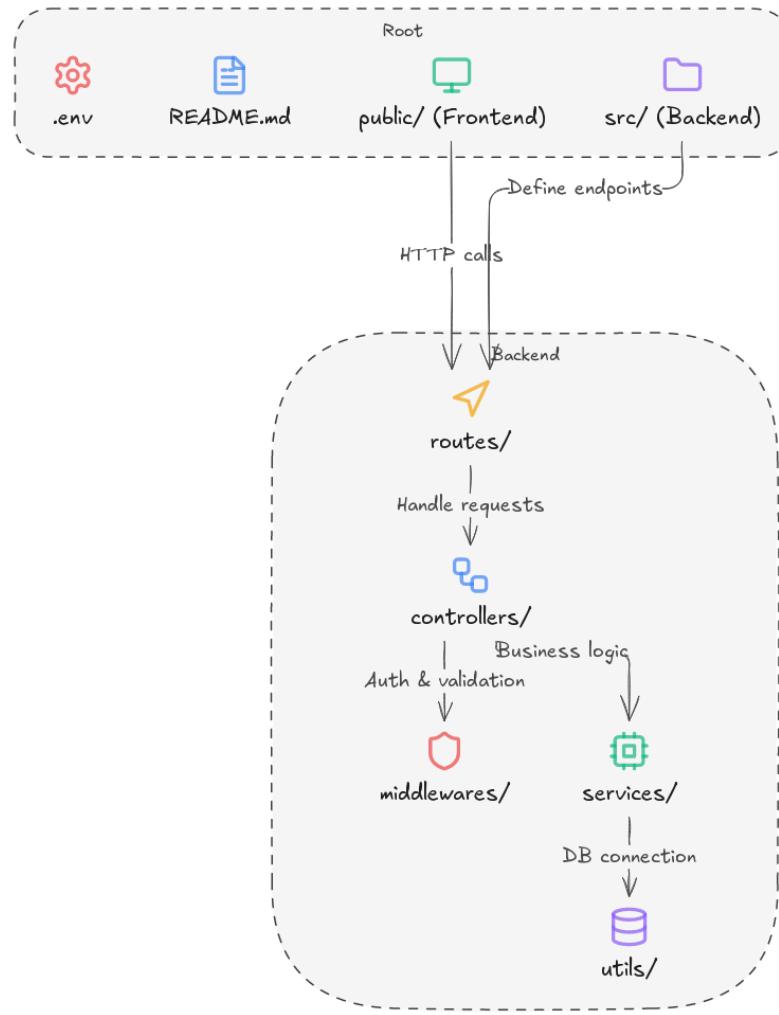
 utils/ Conexión a PostgreSQL

 public/ Frontend estático (login e inventario)

 .env Variables de entorno

README.md

Ruta / Carpeta	Descripción
README.md	Documentación general del proyecto: instalación, uso, endpoints y pruebas.
.env	Variables de entorno locales (credenciales de BD, JWT, puerto, etc.). No se sube al repositorio.
public/	Frontend estático servido por Express. Contiene login, inventario, JS y logo.
src/	Código fuente principal del backend en Node.js + TypeScript.
src/routes/	Definición de los endpoints de la API y su conexión con los controladores.
src/controllers/	Manejo de peticiones HTTP (req, res), validaciones y respuestas en JSON.
src/services/	Lógica de negocio y acceso a la base de datos PostgreSQL.
src/middlewares/	Seguridad de la aplicación, principalmente autenticación mediante JWT.
src/utils/	Configuración de infraestructura, como la conexión a PostgreSQL y utilidades comunes.



Frontend

Buscar y filtrar

Buscar (nombre o tags)

ej. UNO, familia, rapido

Categoría

ej. cartas, mesa

Edad mínima (min_age \geq)

Jugadores (players =)

Solo con stock

(cualquiera)



Aplicar

Limpiar

3 juego(s)

Esa sección es el módulo de filtrado del inventario, es decir, la parte del frontend que permite consultar los juegos con distintos criterios antes de mostrarlos en la tabla.

Ahí el usuario puede buscar por nombre o tags, lo que internamente se envía como un parámetro search al backend para hacer coincidencias parciales. También puede filtrar por categoría, por edad mínima (usando una condición tipo $\text{min_age} \geq \text{valor}$) y por número exacto de jugadores. El selector "Solo con stock" permite decidir si se quieren ver únicamente juegos disponibles, lo que en el backend normalmente se traduce en una condición como $\text{stock} > 0$.

Cuando se presiona Aplicar, el frontend construye una query string con los campos llenos y hace un GET /api/games?parametros, y el backend responde solo con los registros que cumplen esos filtros. El botón Limpiar simplemente borra los campos y vuelve a cargar todo el inventario sin restricciones. El texto "3 juego(s)" muestra dinámicamente cuántos resultados devolvió la consulta.

Registrar juego

Nombre *

UNO

Categoría

cartas

Edad mínima

Jugadores

Stock

0

Tags (coma)

familia, rapido

Crear

Ese bloque corresponde al módulo de registro de juegos dentro de la aplicación. Su función es permitir que el usuario agregue un nuevo juego al inventario enviando la información al backend mediante una petición POST a /api/games.

El campo Nombre es obligatorio y representa el identificador principal del juego. Los demás campos categoría, edad mínima, número de jugadores, stock y tags— son opcionales y sirven para clasificar y describir mejor el juego dentro del sistema. Los tags se escriben separados por comas y en el frontend se transforman en un arreglo antes de enviarse al servidor, lo que permite búsquedas más flexibles posteriormente.

Importar XML

Archivo XML *

Elegir archivo No se eligió ningún archivo

Se enviará a /api/games/import-xml

Subir XML

Importar XML

Archivo XML *

Elegir archivo games.xml

Se enviará a /api/games/import-xml

Subir XML

Selecciona XML

ID	Nombre	Categoría	Edad	Jugadores	Stock	Tags	Creado	Acciones
4	twister	tacto	6	5	20	familia	18/2/2026, 12:52:48	<button>Eliminar</button>
2	MEMORAMA	cartas	7	3	1	familia	16/2/2026, 21:47:24	<button>Eliminar</button>

ID	Nombre	Categoría	Edad	Jugadores	Stock	Tags	Creado	Acciones
3	twister	tacto	4	5	10	familia	18/2/2026, 12:39:57	<button>Eliminar</button>
2	MEMORAMA	cartas	7	3	1	familia	16/2/2026, 21:47:24	<button>Eliminar</button>
1	UNO	cartas	6	4	10	familia,rapido	16/2/2026, 21:31:35	<button>Eliminar</button>

Ese bloque corresponde al módulo de importación masiva de juegos mediante archivo XML. Su objetivo es permitir que el usuario cargue varios registros al sistema de una sola vez, en lugar de agregarlos manualmente uno por uno.

El usuario selecciona un archivo con formato .xml desde su equipo y, al presionar “Subir XML”, el frontend envía el archivo al endpoint POST /api/games/import-xml usando una petición multipart/form-data. En el backend, el controlador recibe el archivo, lo convierte a texto y posteriormente puede parsearlo para extraer cada juego definido en el XML y guardarlo en PostgreSQL. Este proceso facilita la carga inicial de inventario o la migración de datos desde otros sistemas.

En tu src/index.ts hicimos esto:

- express.static(public, { index:false })
Sirve archivos estáticos pero NO “elige” index.html automáticamente cuando entras a /.
- GET / manda login.html
- GET /app manda index.html (inventario)

<http://localhost:3000/> Login

<http://localhost:3000/app> Inventario

En login.html:

1. Cuando pegas Entrar o Registrar, hace:
 - POST /api/auth/login o POST /api/auth/register
 - manda { email, password }
2. Si el backend responde { token: "..." }:
 - se guarda en el navegador con:
 - localStorage.setItem("token", data.token);
3. Luego te manda al inventario:
4. window.location.href = "/app";



En public/app.js hay una función clave:

```
function requireTokenOrRedirect() {  
  
  const token = getToken();  
  
  if (!token) window.location.href = "/";  
  
}
```

Cada vez que el inventario carga juegos (loadGames()), primero revisa si hay token.

- Si NO hay token, te regresa al login.
- Si sí hay token, sigue normal.

Para que el backend sepa que estás logueado, cada request va con header:

```
function authHeaders(extra = {}) {  
  
  const token = getToken();
```

```
        return token ? { ...extra, Authorization: `Bearer ${token}` } : extra;  
    }  
  

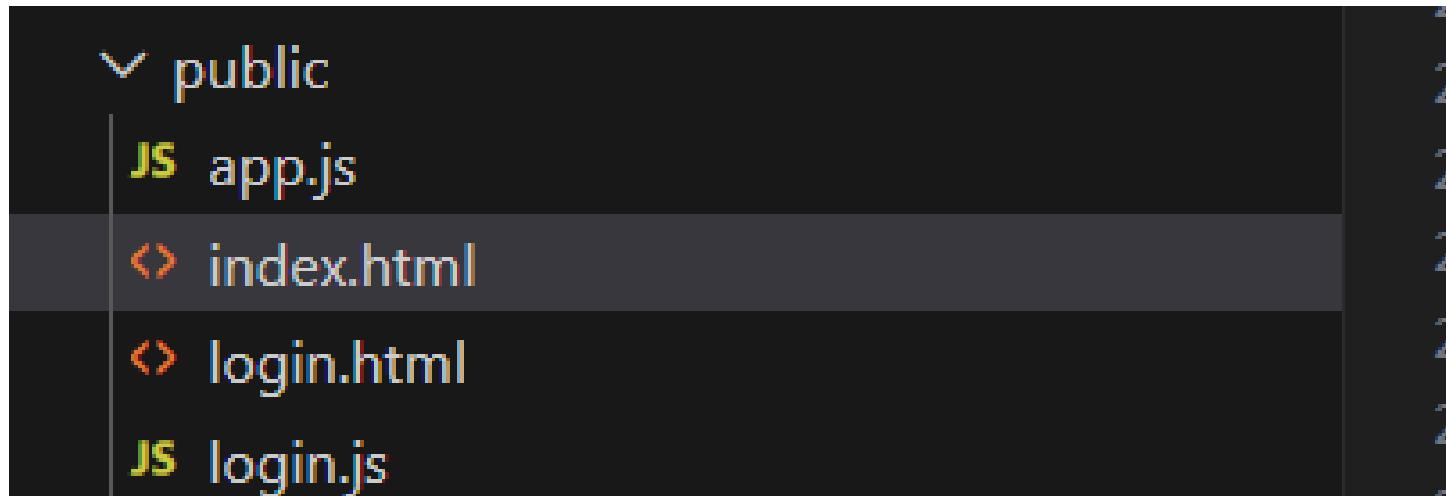

## Salir


```

```
<button id="logoutBtn" class="btn">Salir</button>
```

```
function logout() {  
  
    localStorage.removeItem("token");  
  
    window.location.href = "/";  
  
}  
  
$("#logoutBtn").addEventListener("click", logout);
```

- borra el token
- te manda al login
- y ya no puedes ver inventario porque el token ya no existe
- Eliminar: en cada fila aparece botón “Eliminar” que llama:
DELETE /api/games/:id con token.
- XML: se sube un .xml con FormData a:
POST /api/games/import/xml con token.



El sistema cuenta con una carpeta denominada public, la cual contiene los archivos estáticos que conforman la interfaz web del proyecto. Dentro de esta carpeta se encuentra index.html, que corresponde a la vista principal de la aplicación y permite la gestión del inventario de juegos, la aplicación de filtros de búsqueda, el registro de nuevos elementos y la importación de archivos XML; dicha vista está protegida y solo puede ser accedida después de un proceso de autenticación exitoso. Asimismo, se incluye login.html, que representa la interfaz de inicio de sesión donde el usuario introduce sus credenciales para acceder al sistema.

La lógica asociada a este proceso de autenticación se implementa en login.js, archivo encargado de enviar las credenciales al backend, manejar posibles errores y redirigir al usuario a la vista principal cuando el acceso es válido. Por otra parte, app.js concentra la lógica funcional principal del frontend,

incluyendo la consulta de juegos registrados, la aplicación de filtros, la creación y eliminación de registros, la importación de información desde archivos XML y la gestión del cierre de sesión. Esta organización permite separar claramente el proceso de autenticación de la operación general del sistema, lo que favorece la mantenibilidad, claridad estructural y escalabilidad del proyecto.



Amy Veraza
valeamy3@gmail.com
+5634369888