S.E.P. TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO de Tuxtepec

REPORTE DE PAGINA DE POKEMON

ASIGNATURA:

CLIENTE SERVIDOR

PRESENTA(N):

RANGEL VASQUEZ ANGEL ARTURO
22350404

CARRERA:
INGENIERÍA INFORMÁTICA

DOCENTE:

DR.JULIO CARMONA AGUILAR

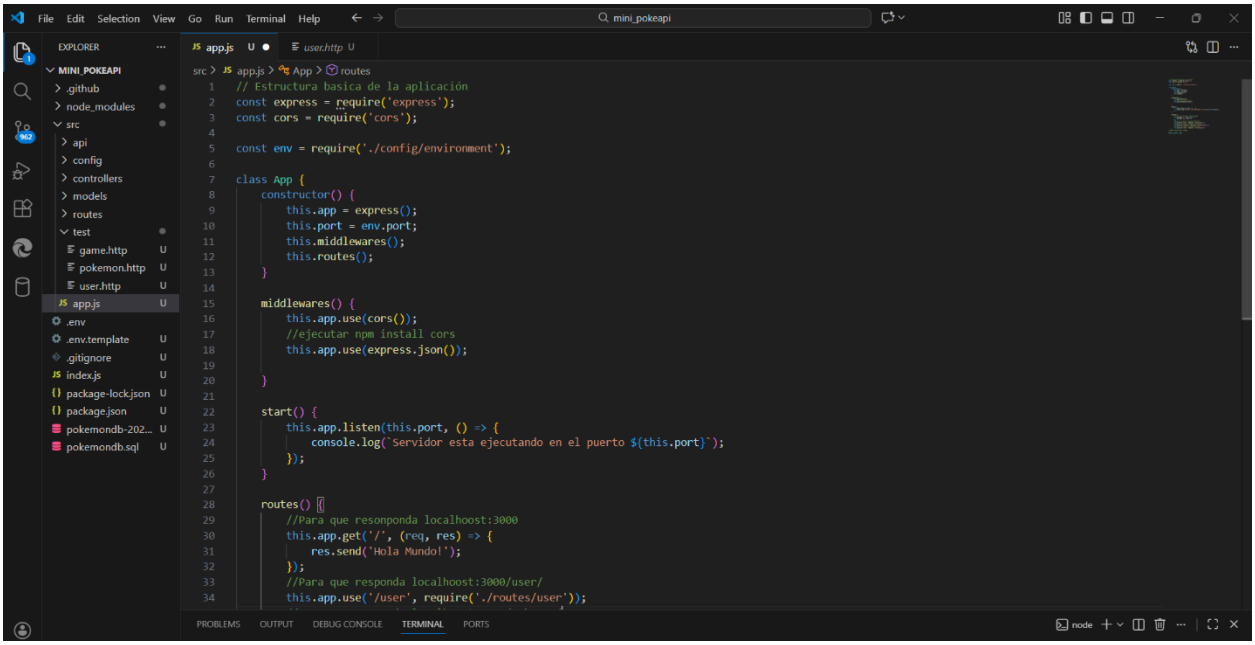
14 DE diciembre

INTRODUCCION

El presente reporte detalla el diseño, arquitectura y desarrollo de una aplicación robusta programada en el entorno de desarrollo Visual Studio Code. El proyecto surge de la necesidad de crear un sistema backend capaz de gestionar la lógica central de una experiencia inspirada en el universo de Pokémon GO, centrada en la interacción dinámica entre entrenadores y criaturas digitales.

La arquitectura del sistema se ha estructurado bajo un patrón de diseño por capas, utilizando Node.js como motor de ejecución y una base de datos relacional definida mediante SQL. Esta estructura permite una separación clara de responsabilidades, facilitando la escalabilidad del juego y garantizando un manejo eficiente de los datos relacionados con el inventario de pokémon, las estadísticas de los usuarios y el historial de encuentros o "games".

A lo largo de este documento, se analizarán los componentes clave del código fuente, desde la configuración de las conexiones a la base de datos hasta la implementación de las rutas que permiten la comunicación entre el servidor y el cliente.



Para el desarrollo de este servidor, he implementado una arquitectura basada en Programación Orientada a Objetos, encapsulando toda la lógica de la infraestructura en una clase denominada App. Mi objetivo con este enfoque es garantizar el encapsulamiento y facilitar la escalabilidad del sistema, permitiendo que el punto de entrada de la aplicación sea limpio y modular."

1. Gestión de Dependencias y Variables de Entorno

"En la parte superior, realizo la importación de los módulos críticos: express para la gestión del servidor y cors para las políticas de seguridad. Cabe destacar que estoy importando un objeto env desde mi configuración de entorno; esto lo hago para seguir las mejores prácticas de 12-Factor App, abstrayendo configuraciones sensibles como el puerto para que no estén 'hardcodeadas' en el código fuente."

2. El Constructor (Fase de Inicialización)

"Dentro del constructor, inicializo mi instancia de Express y asigno el puerto de escucha. Inmediatamente después, disparo los métodos middlewares() y routes(). He diseñado esta secuencia de ejecución para asegurar que, al momento de instanciar la clase, el servidor ya posea todas sus capas de procesamiento y definiciones de rutas correctamente configuradas."

3. Implementación de Middlewares (Capa de Pre-procesamiento)

"En mi método middlewares(), he configurado dos funciones de paso esenciales:

CORS: Lo utilizo para habilitar el intercambio de recursos de origen cruzado, lo cual es fundamental para que mi API sea consumible por clientes desde diferentes dominios.

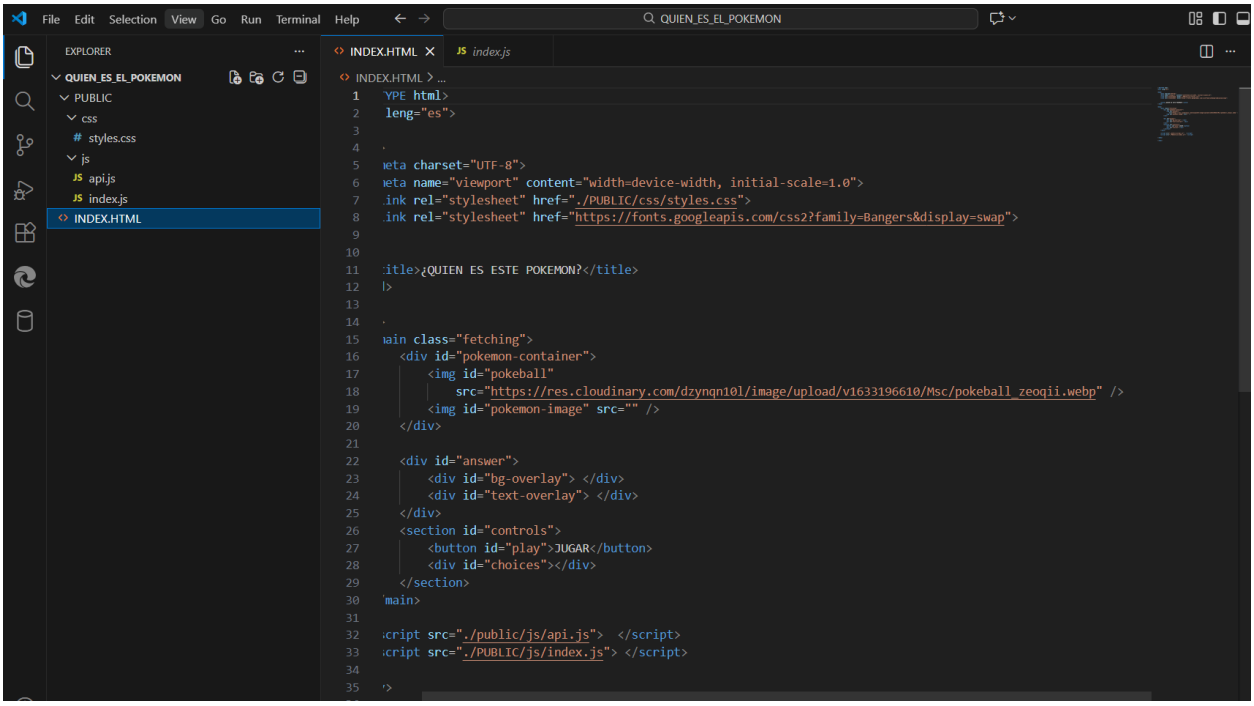
express.json(): Este es un middleware de parsing. Lo implemento para que el servidor sea capaz de interpretar y deserializar cuerpos de peticiones en formato JSON, transformándolos automáticamente en objetos de JavaScript."

4. Modularización del Enrutamiento

"En el método routes(), he aplicado un patrón de enrutamiento delegado. Además de definir una ruta raíz para pruebas de conectividad, utilizo app.use para vincular prefijos de ruta específicos (/user, /pokemon, /game) con sus respectivos módulos de rutas externos. Esta técnica me permite mantener un bajo acoplamiento, ya que cada dominio de la aplicación gestiona su propia lógica interna en archivos independientes."

5. Ciclo de Vida y Exportación

"Finalmente, el método start() es el encargado de realizar el binding del servidor al puerto TCP. Utilizo un callback para imprimir una confirmación en consola una vez que el servidor esté en estado ready. Al finalizar el archivo, exporto la clase App mediante module.exports, permitiendo que el arranque del sistema sea tan sencillo como instanciar la clase y ejecutar un único método."



1. Arquitectura de Archivos

En el panel del explorador a la izquierda, se identifica una estructura organizada bajo una carpeta principal denominada QUIEN_ES_EL_POKEMON. El proyecto sigue una jerarquía de carpeta pública (PUBLIC) que separa las responsabilidades de la siguiente manera:

Hojas de estilo: Los archivos CSS se encuentran en una subcarpeta dedicada.

Lógica de programación: El directorio js contiene dos scripts: api.js (presumiblemente para el manejo de datos externos) e index.js.

Raíz: El archivo INDEX.HTML actúa como el punto de entrada de la aplicación.

2. Estructura del Documento HTML

El archivo abierto en el editor revela un marcado semántico diseñado para la interactividad:

Metadatos y Tipografía: El documento está configurado para el idioma español y utiliza la fuente externa "Bangers" de Google Fonts, alineada con la estética visual de la franquicia.

Interfaz de Usuario (UI):

Contenedor Principal: La etiqueta <main> posee la clase fetching, lo que sugiere un estado de carga inicial.

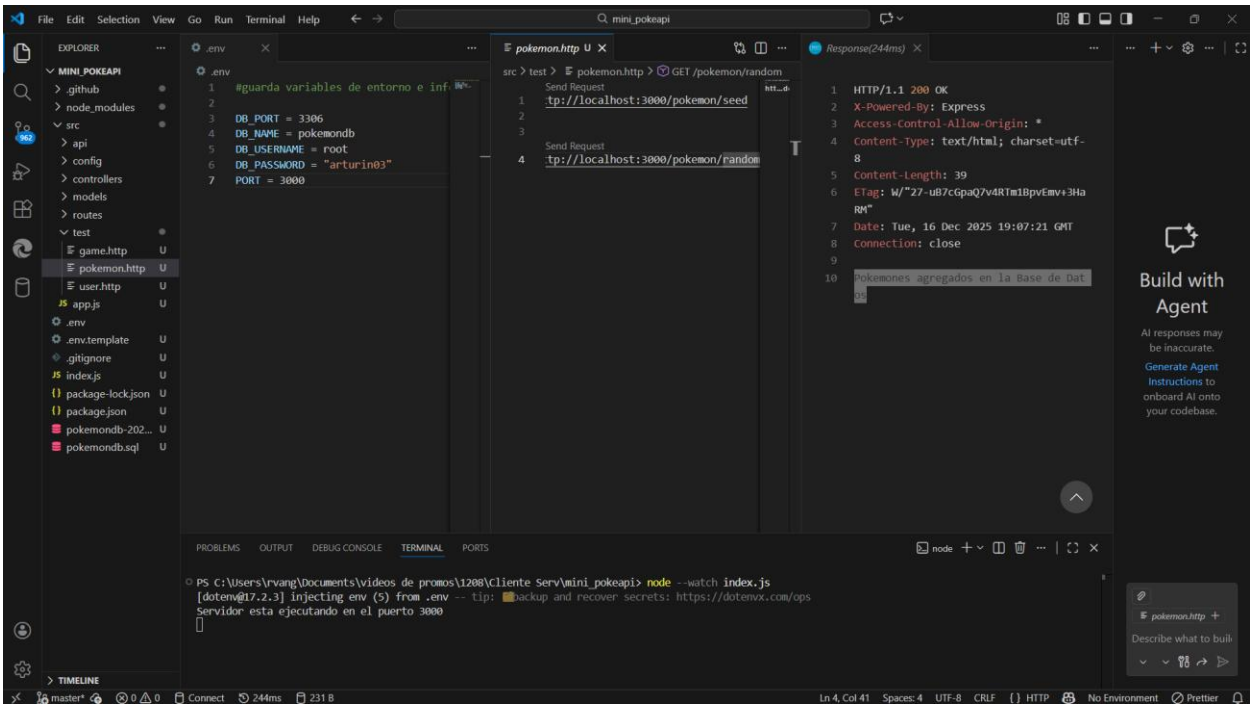
Visualización de Imágenes: El pokemon-container incluye una imagen de una Pokébola alojada en Cloudinary y un elemento <img> vacío destinado a mostrar dinámicamente al Pokémon.

Sección de Respuesta: El ID answer contiene contenedores para superposiciones de texto y fondo, diseñados para dar retroalimentación al jugador.

Interactividad: La sección de controles dispone de un botón con el texto "JUGAR" y un contenedor de opciones (choices) para la selección múltiple.

3. Integración de Recursos

El código muestra una correcta vinculación de recursos externos. Se observa el uso de rutas relativas para enlazar la hoja de estilos en el <head> y la carga secuencial de los scripts al final del <body>, asegurando que api.js cargue antes que la lógica principal en index.js.



1. Panel de Archivos (Izquierda)

Estructura del Proyecto: Se ve una estructura clásica de API: carpetas para controllers, models, routes y una carpeta test donde guardas archivos .http.

Archivo .env: Tienes abierta la configuración de las variables de entorno. Aquí defines cómo se conecta tu aplicación a la base de datos MySQL (puerto 3306, usuario root, etc.).

2. Editor Central (Pruebas de API)

Tienes abierto el archivo pokemon.http. Esta es una extensión de VS Code (probablemente "REST Client") que te permite hacer peticiones directamente desde el editor sin usar Postman.

Se ve que ejecutaste una petición GET a la ruta http://localhost:3000/pokemon/seed.

3. Respuesta de la API (Derecha)

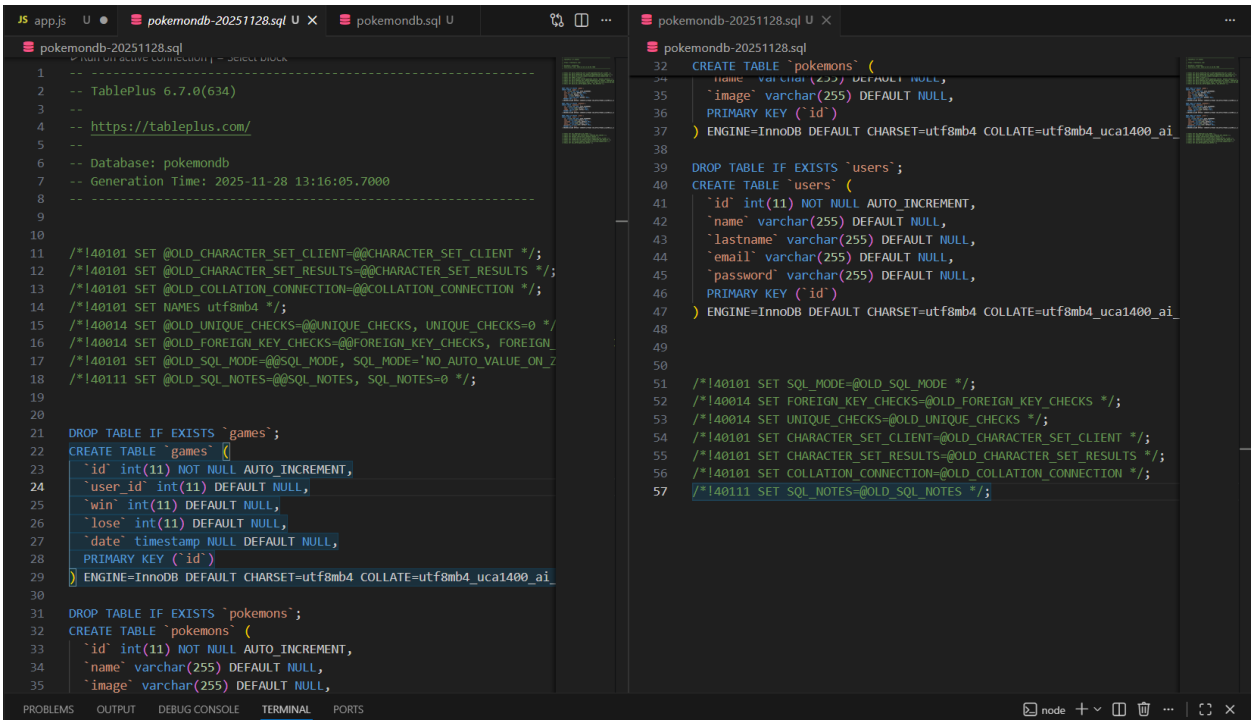
Resultado: La petición fue exitosa (HTTP/1.1 200 OK).

Cuerpo de la respuesta: En la línea 10 se lee: "Pokemones agregados en la Base de Datos". Esto indica que la ruta seed (semilla) funcionó correctamente para llenar tu base de datos con datos iniciales.

4. Terminal (Abajo)

Estado del Servidor: El servidor está corriendo en modo observación (node --watch index.js).

Confirmación: El mensaje "Servidor está ejecutando en el puerto 3000" confirma que el backend está listo y escuchando peticiones.



Parte 1: Metadatos y Configuración (Líneas 1-19)

Líneas 1-4: Son comentarios informativos. Indican la herramienta usada (TablePlus), la versión y su sitio web.

Líneas 6-8: Nombre de la base de datos (pokemondb) y el momento exacto en que se exportó este archivo.

Líneas 11-13: Configuran los CHARACTER_SET (juegos de caracteres). Esto asegura que si tienes nombres con tildes o la letra "ñ", se vean correctamente.

Línea 14: SET NAMES utf8mb4: Establece que la comunicación entre el cliente y el servidor usará este formato (soporta hasta emojis).

Líneas 15-18: Desactiva temporalmente las restricciones de seguridad (FOREIGN_KEY_CHECKS, UNIQUE_CHECKS). Se hace para que, al importar los datos, MySQL no se detenga si una fila hace referencia a otra que aún no se ha creado.

Parte 2: Tabla de Juegos (Líneas 21-28)

Línea 21: DROP TABLE IF EXISTS games: Borra la tabla si ya existe para evitar errores de "tabla duplicada".

Línea 22: CREATE TABLE games (: Inicia la definición de la tabla de partidas.

Línea 23: id int(11) NOT NULL AUTO_INCREMENT: Crea una columna para el ID. No puede ser nula y el sistema le asigna el siguiente número disponible automáticamente.

Línea 24: user_id int(11) DEFAULT NULL: Columna para saber qué usuario jugó.

Línea 25-26: win y lose: Columnas enteras para registrar victorias y derrotas.

Línea 27: date timestamp NULL DEFAULT NULL: Registra el momento de la partida (año, mes, día, hora).

Línea 28: PRIMARY KEY (id): Define que el id es la llave maestra de la tabla.

Línea 29: Configura el motor de la tabla como InnoDB (seguro y moderno) y el idioma del texto.

Parte 3: Tabla de Pokémon (Líneas 30-37)

Línea 30-31: Borra y crea la tabla pokemons.

Línea 33: id int(11): El identificador único del Pokémon.

Línea 34: name varchar(255): El nombre del Pokémon (hasta 255 letras).

Línea 35: image varchar(255): Aquí se guarda el nombre del archivo de imagen o su link.

Línea 36: Define el id como llave primaria.

Parte 4: Tabla de Usuarios (Líneas 39-47)

Línea 39-40: Borra y crea la tabla users.

Línea 41: id: ID único del usuario, autoincrementable.

Línea 42-43: name y lastname: Nombre y apellido del jugador.

Línea 44: email: El correo electrónico del usuario.

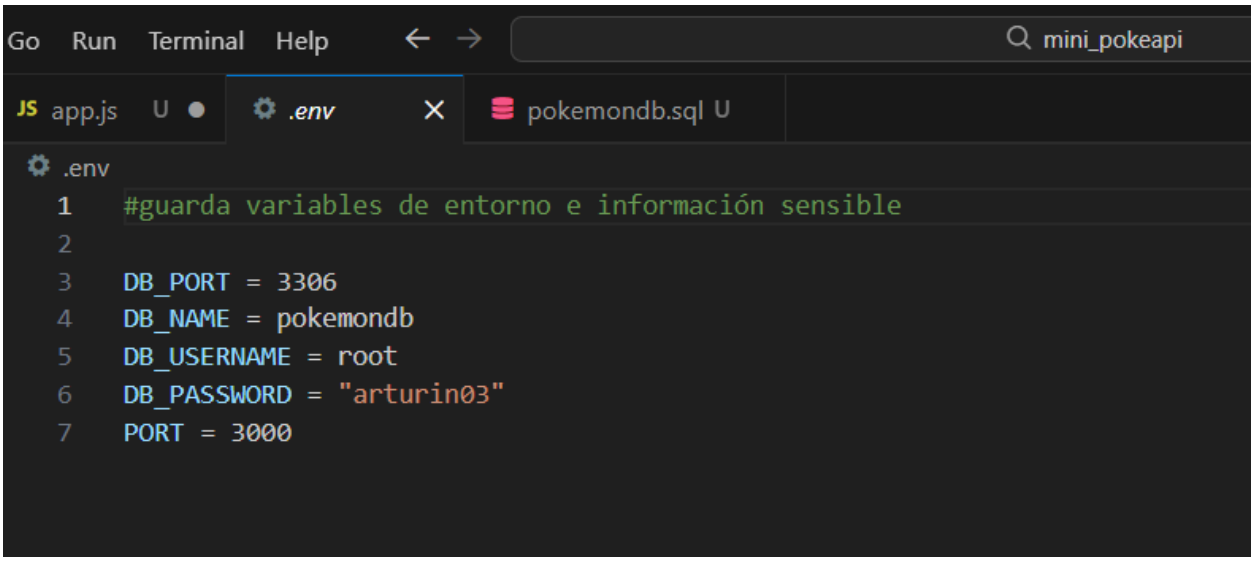
Línea 45: password: La contraseña (generalmente guardada como un código cifrado).

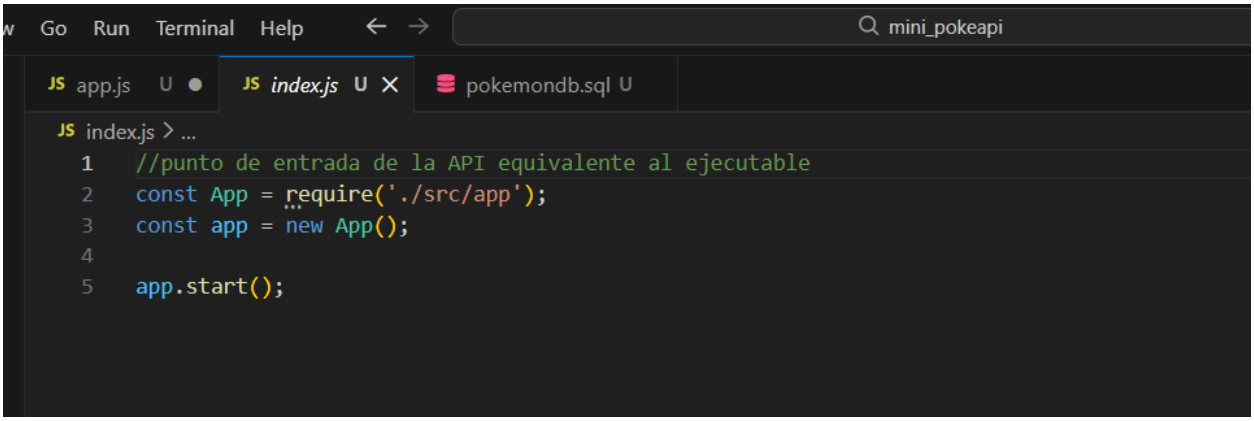
Línea 46: Define el id como llave primaria.

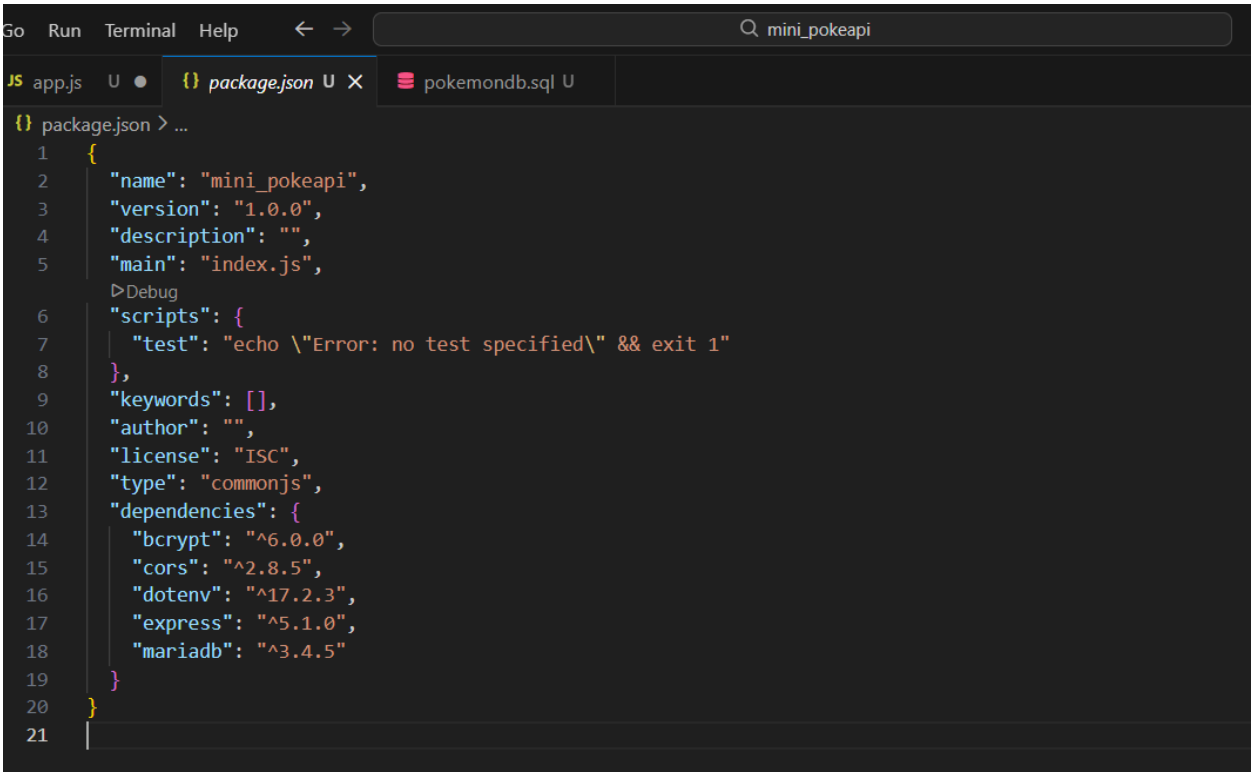
Parte 5: Finalización (Líneas 51-57)

Este bloque es el "cierre del telón". Devuelve todas las configuraciones a la normalidad.

Línea 51-57: Reactiva los chequeos que se pausaron al inicio (SQL_MODE, FOREIGN_KEY_CHECKS, UNIQUE_CHECKS). Esto asegura que, a partir de este momento, la base de datos sea estricta y proteja la integridad de tus datos.


Go Run Terminal Help
mini_pokeapi
app.js U
.env
pokemondb.sql U
.env
#guarda variables de entorno e información sensible
DB_PORT = 3306
DB_NAME = pokemondb
DB_USERNAME = root
DB_PASSWORD = "arturin03"
PORT = 3000


Go Run Terminal Help
mini_pokeapi
app.js U
index.js U
pokemondb.sql U
index.js > ...
//punto de entrada de la API equivalente al ejecutable
const App = require('./src/app');
const app = new App();
app.start();



Línea 1 ({): Abre el objeto principal que contiene toda la configuración del proyecto.

Línea 2 ("name": "mini_pokeapi"): Es el nombre técnico de tu proyecto. No debe tener espacios.

Línea 3 ("version": "1.0.0"): La versión actual de tu software. Se suele empezar en 1.0.0 al lanzarlo.

Línea 4 ("description": ""): Un espacio para escribir de qué trata tu API (actualmente está vacío).

Línea 5 ("main": "index.js"): Define el punto de entrada de tu aplicación. Es el primer archivo que Node.js buscará para ejecutar el programa.

Línea 6 ("scripts": {): Abre una sección para definir comandos personalizados que puedes ejecutar en la terminal (usando npm run ...).

Línea 7 ("test": "..."): Un comando predeterminado para pruebas. Actualmente solo imprime un error porque no has configurado tests todavía.

Línea 8 (},): Cierra la sección de scripts.

Línea 9 ("keywords": [],): Una lista de palabras clave para que otros encuentren tu proyecto si lo subes a un repositorio público.

Línea 10 ("author": ""): Tu nombre o el nombre del creador.

Línea 11 ("license": "ISC"): Define el tipo de licencia legal de tu código (ISC es una licencia libre común).

Línea 12 ("type": "commonjs"): Indica que el proyecto usa el sistema de módulos antiguo de Node (module.exports y require).

Línea 13 ("dependencies": {): Abre la sección de librerías externas que tu código necesita para funcionar.

Línea 14 ("bcrypt": "^6.0.0"): Librería para proteger contraseñas.

Línea 15 ("cors": "^2.8.5"): Permite que tu API responda a peticiones de diferentes orígenes (necesario si vas a conectar un Frontend).

Línea 16 ("dotenv": "^17.2.3"): Sirve para cargar variables secretas desde un archivo .env (como las claves de la base de datos).

Línea 17 ("express": "^5.1.0"): El motor que crea el servidor web y maneja las rutas de tu API.

Línea 18 ("mariadb": "^3.4.5"): El conector que permite que tu código hable con la base de datos MariaDB.

Línea 19 (}): Cierra la sección de dependencias.

Línea 20 (}): Cierra el archivo completo.

INDEX.HTML | api.js

PUBLIC > js > api.js > ...

```
1   const getPokemon = async () => {
2       const res = await fetch('http://localhost:3000/pokemon/random');
3       const data = await res.json();
4       return data;
5   }
6
7
8   window.getPokeData = async () => {
9       const pokemon = await getPokemon();
10
11      const randomIndex = Math.floor(Math.random() * pokemon.length);
12
13
14      const correctAnswer = pokemon[randomIndex];
15
16      //const correctAnswer = pokemon.find(p => p.isCorrect);
17
18
19      return {
20          pokemonChoices: pokemon,
21          correctAnswer,
22      }
23      //console.log(pokemon);
24  }
25
26
```

getPokemon (Líneas 1-5)

Esta función se encarga de ir a buscar la información al servidor.

Línea 1: const getPokemon = async () => { Define una función asíncrona (async). Esto permite usar await dentro de ella para esperar a que las respuestas de internet lleguen sin bloquear el resto del programa.

Línea 2: const res = await fetch('http://localhost:3000/pokemon/random'); Usa la función fetch para hacer una petición HTTP a tu servidor local. Se queda "esperando" (await) hasta que el servidor responda.

Línea 3: const data = await res.json(); La respuesta cruda del servidor se convierte a un formato de objeto JavaScript (JSON). También es un proceso asíncrono.

Línea 4: return data; Devuelve la información del Pokémon (o la lista de ellos) para que pueda ser usada en otra parte.

Función getPokeData (Líneas 8-24)

Esta función organiza los datos para el juego y se asigna al objeto window para que sea accesible globalmente en el navegador.

Línea 8: window.getPokeData = async () => { Declara otra función asíncrona y la guarda en window.getPokeData. Esto sirve para poder llamarla desde otros archivos o desde la consola del navegador.

Línea 9: const pokemon = await getPokemon(); Llama a la función anterior y guarda el resultado (que parece ser un arreglo o lista de opciones) en la variable pokemon.

Línea 11: const randomIndex = Math.floor(Math.random() * pokemon.length); Genera un índice aleatorio basado en el tamaño de la lista recibida.

Math.random() da un número entre 0 y 1.

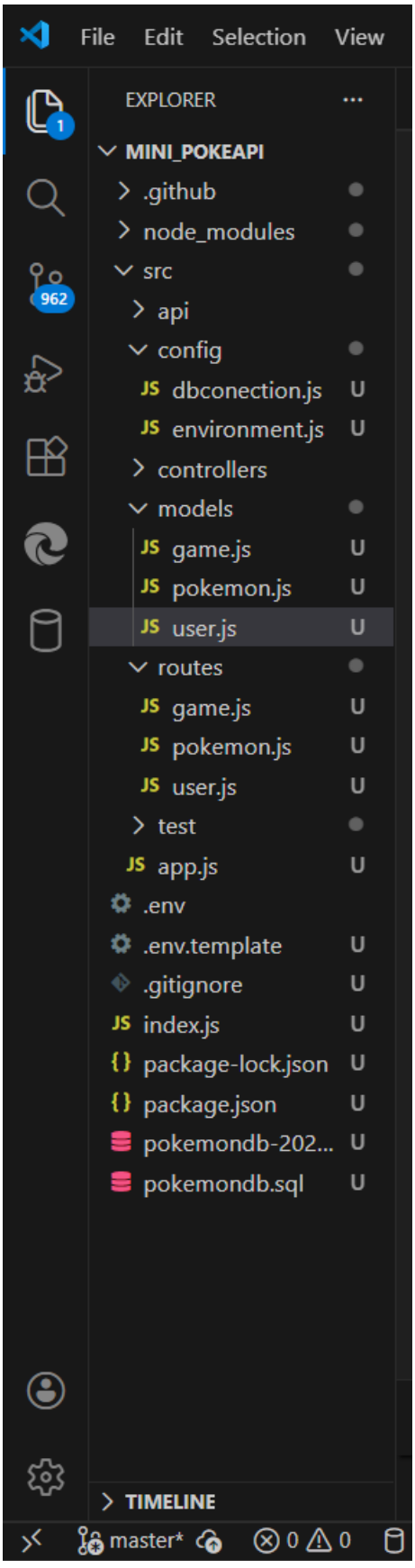
Se multiplica por el largo de la lista y Math.floor lo redondea hacia abajo para obtener un número entero válido (0, 1, 2, etc.).

Línea 14: const correctAnswer = pokemon[randomIndex]; Usa el índice aleatorio para elegir cuál de los Pokémon de la lista será el "ganador" o la respuesta que el usuario debe adivinar.

Línea 16: //const correctAnswer = pokemon.find(p => p.isCorrect); Es una línea comentada (no se ejecuta). Es una forma alternativa de buscar la respuesta correcta si el servidor ya enviara un marcador de quién es el correcto.

Líneas 19-22: return { pokemonChoices: pokemon, correctAnswer, } La función termina devolviendo un objeto con dos cosas: la lista completa de opciones para mostrar en los botones (pokemonChoices) y el Pokémon que es la respuesta correcta (correctAnswer).

Línea 23: //console.log(pokemon); Otra línea comentada que probablemente se usó para pruebas (debugging) para ver qué llegaba del servidor.


File
Edit
Selection
View
EXPLORER
MINI_POKEAPI
.github
node_modules
src
api
config
dbconection.js U
environment.js U
controllers
models
game.js U
pokemon.js U
user.js U
routes
game.js U
pokemon.js U
user.js U
test
app.js U
.env
.env.template U
.gitignore U
index.js U
package-lock.json U
package.json U
pokemondb-202... U
pokemondb.sql U
TIMELINE
master*
0 0

Carpetas Principales

.github: Contiene configuraciones específicas para GitHub, como flujos de trabajo de integración continua (CI/CD).

node_modules: Carpeta donde se instalan todas las librerías y dependencias externas del proyecto (gestionadas por npm).

src (Source): Es la carpeta raíz del código fuente. Dentro se divide en:

api: Probablemente contiene funciones para consumir servicios de terceros (como la PokeAPI oficial).

config: Archivos de configuración técnica. Incluye dbconection.js (conexión a base de datos) y environment.js (ajustes de variables de entorno).

controllers: Contiene la lógica que procesa las peticiones y envía las respuestas.

models: Define la estructura de los datos (tablas o colecciones) para game, pokemon y user.

routes: Define las rutas o URLs del servidor para interactuar con los modelos mencionados.

test: Espacio dedicado a scripts de pruebas para asegurar que el código funciona correctamente.

Archivos de la Raíz

app.js: Archivo donde se inicializa la aplicación (Express, middlewares, etc.).

.env: Archivo oculto que guarda credenciales sensibles (claves de API, puertos, contraseñas de BD).

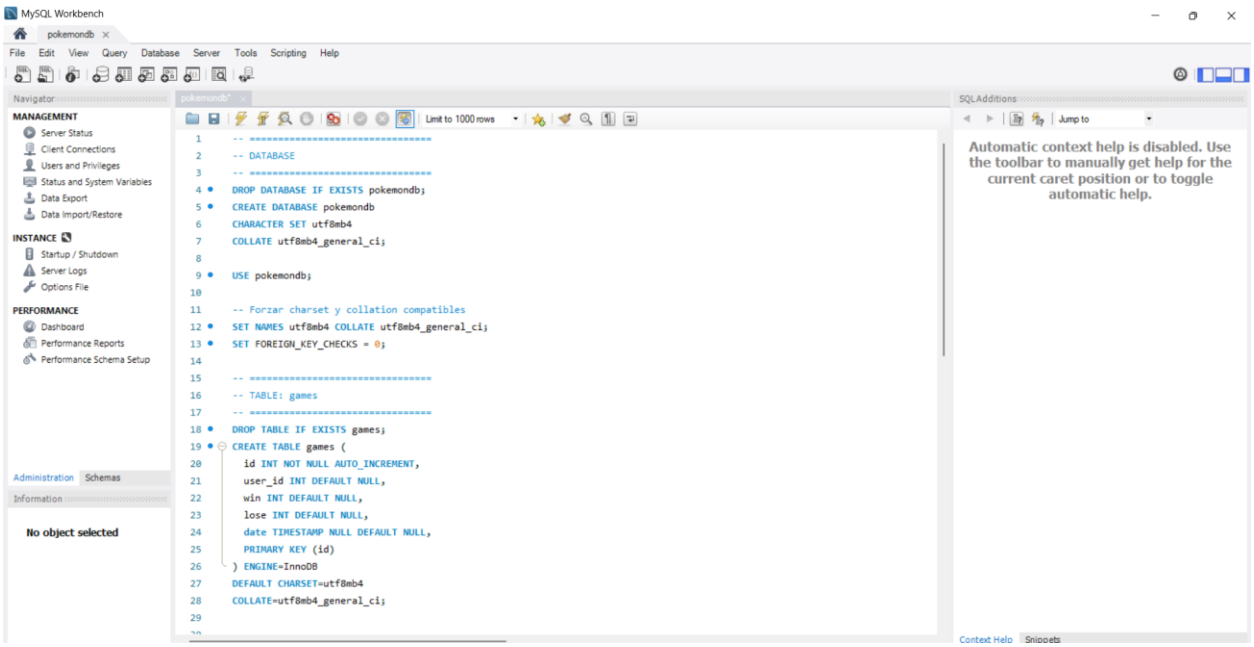
.env.template: Una guía para que otros desarrolladores sepan qué variables deben configurar en su propio .env.

.gitignore: Indica a Git qué archivos o carpetas (como node_modules) no deben subirse al repositorio.

index.js: El punto de entrada principal que pone en marcha el servidor.

package.json: El "DNI" del proyecto; contiene el nombre, versión y la lista de dependencias necesarias.

pokemondb.sql: Script de base de datos SQL para crear las tablas y datos iniciales del proyecto.



Bloque 1: Inicialización de la Base de Datos

Línea 1: -- ========== Es un comentario decorativo. Las dos rayas -- le dicen a MySQL que ignore el resto de la línea. Se usa para organizar visualmente el código.

Línea 2: -- DATABASE Otro comentario que indica que lo que sigue es la configuración de la base de datos.

Línea 3: -- ========== Cierre del separador visual.

Línea 4: DROP DATABASE IF EXISTS pokemondb; Orden: Borrar la base de datos si existe. Evita que el script falle si intentas crear algo que ya está ahí.

Línea 5: CREATE DATABASE pokemondb Crea el contenedor de datos llamado pokemondb.

Línea 6: CHARACTER SET utf8mb4 Configura la base de datos para que acepte casi cualquier símbolo del mundo (incluyendo emojis de Pokémon o caracteres japoneses).

Línea 7: COLLATE utf8mb4_general_ci; Establece que las búsquedas no distingan entre mayúsculas y minúsculas (Case Insensitive).

Línea 9: USE pokemondb; Selecciona la base de datos para trabajar. Sin esto, MySQL no sabría en qué base de datos crear las tablas.

Bloque 2: Configuración del Entorno

Línea 11: -- Forzar charset y collation compatibles Un comentario explicativo en español para el programador.

Línea 12: SET NAMES utf8mb4 COLLATE utf8mb4_general_ci; Asegura que la conexión entre tu computadora y el servidor use la misma codificación de texto.

Línea 13: SET FOREIGN_KEY_CHECKS = 0; Desactiva la revisión de "llaves foráneas". Se hace para poder borrar o modificar tablas sin que el sistema se queje de que hay datos relacionados en otras tablas.

Bloque 3: Creación de la Tabla de Juegos

Línea 15 al 17: Son separadores visuales (comentarios) para indicar el inicio de la tabla de juegos.

Línea 18: DROP TABLE IF EXISTS games; Borra la tabla "games" si ya existe, para sobreescribirla con la nueva estructura.

Línea 19: CREATE TABLE games (Abre el comando para definir las columnas de la tabla.

Línea 20: id INT NOT NULL AUTO_INCREMENT,

id: Nombre de la columna.

INT: Solo acepta números enteros.

NOT NULL: No puede estar vacío.

AUTO_INCREMENT: MySQL le suma 1 automáticamente a cada registro nuevo.

Línea 21: user_id INT DEFAULT NULL, Columna para el ID del jugador. Por defecto empieza vacía (NULL).

Línea 22: win INT DEFAULT NULL, Columna para registrar si ganó (o cuántas veces ganó).

Línea 23: lose INT DEFAULT NULL, Columna para registrar derrotas.

Línea 24: date TIMESTAMP NULL DEFAULT NULL, Almacena la fecha y hora exacta. TIMESTAMP es el formato estándar para tiempo.

Línea 25: PRIMARY KEY (id) Define que la columna id es la identificación oficial y única de cada fila.

Bloque 4: Propiedades Finales de la Tabla

Línea 26:) ENGINE=InnoDB Cierra la definición de columnas. InnoDB es el "motor" que gestiona los datos; es el más moderno y seguro porque permite transacciones.

Línea 27: DEFAULT CHARSET=utf8mb4 Reafirma que esta tabla específica usará el juego de caracteres de emojis y símbolos.

Línea 28: COLLATE=utf8mb4_general_ci; Cierra la instrucción con el punto y coma ;, confirmando las reglas de ordenamiento de texto.

```
29
30      -- ================================
31      -- TABLE: pokemons
32      -- ================================
33      DROP TABLE IF EXISTS pokemons;
34      CREATE TABLE pokemons (
35        id INT NOT NULL AUTO_INCREMENT,
36        name VARCHAR(255) DEFAULT NULL,
37        image VARCHAR(255) DEFAULT NULL,
38        PRIMARY KEY (id)
39      ) ENGINE=InnoDB
40      DEFAULT CHARSET=utf8mb4
41      COLLATE=utf8mb4_general_ci;
42
43      -- ================================
44      -- TABLE: users
45      -- ================================
46      DROP TABLE IF EXISTS users;
47      CREATE TABLE users (
48        id INT NOT NULL AUTO_INCREMENT,
49        name VARCHAR(255) DEFAULT NULL,
50        lastname VARCHAR(255) DEFAULT NULL,
51        email VARCHAR(255) DEFAULT NULL,
52        password VARCHAR(255) DEFAULT NULL,
53        PRIMARY KEY (id)
54      ) ENGINE=InnoDB
55      DEFAULT CHARSET=utf8mb4
56      COLLATE=utf8mb4_general_ci;
57
58      -- ================================
59      -- DATA: users
```

Bloque de la Tabla pokemons (Líneas 30-41)

Líneas 30-32: Comentarios decorativos que indican el inicio de la sección para la tabla "pokemons".

Línea 33: DROP TABLE IF EXISTS pokemons; Borra la tabla si ya existe para evitar errores al intentar crearla de nuevo.

Línea 34: CREATE TABLE pokemons (Inicia la definición de la tabla donde se guardarán los datos de los Pokémon.

Línea 35: id INT NOT NULL AUTO_INCREMENT, Crea un identificador único numérico que se suma automáticamente.

Línea 36: name VARCHAR(255) DEFAULT NULL, Columna para el nombre del Pokémon. VARCHAR(255) significa que acepta texto de hasta 255 caracteres.

Línea 37: image VARCHAR(255) DEFAULT NULL, Columna para guardar la ruta o URL de la imagen del Pokémon.

Línea 38: PRIMARY KEY (id) Define al id como la llave primaria.

Líneas 39-41: Cierra la tabla definiendo el motor de almacenamiento (InnoDB) y la codificación de caracteres (utf8mb4).

Bloque de la Tabla users (Líneas 43-56)

Líneas 43-45: Comentarios decorativos para la sección de usuarios.

Línea 46: DROP TABLE IF EXISTS users; Elimina la tabla de usuarios si ya existía previamente.

Línea 47: CREATE TABLE users (Comienza la creación de la tabla para los perfiles de usuario.

Línea 48: id INT NOT NULL AUTO_INCREMENT, Identificador único para cada usuario.

Línea 49: name VARCHAR(255) DEFAULT NULL, Almacena el nombre de pila del usuario.

Línea 50: lastname VARCHAR(255) DEFAULT NULL, Almacena el apellido del usuario (esta línea aparece sombreada en tu imagen porque es donde tienes el cursor).

Línea 51: email VARCHAR(255) DEFAULT NULL, Columna para el correo electrónico.

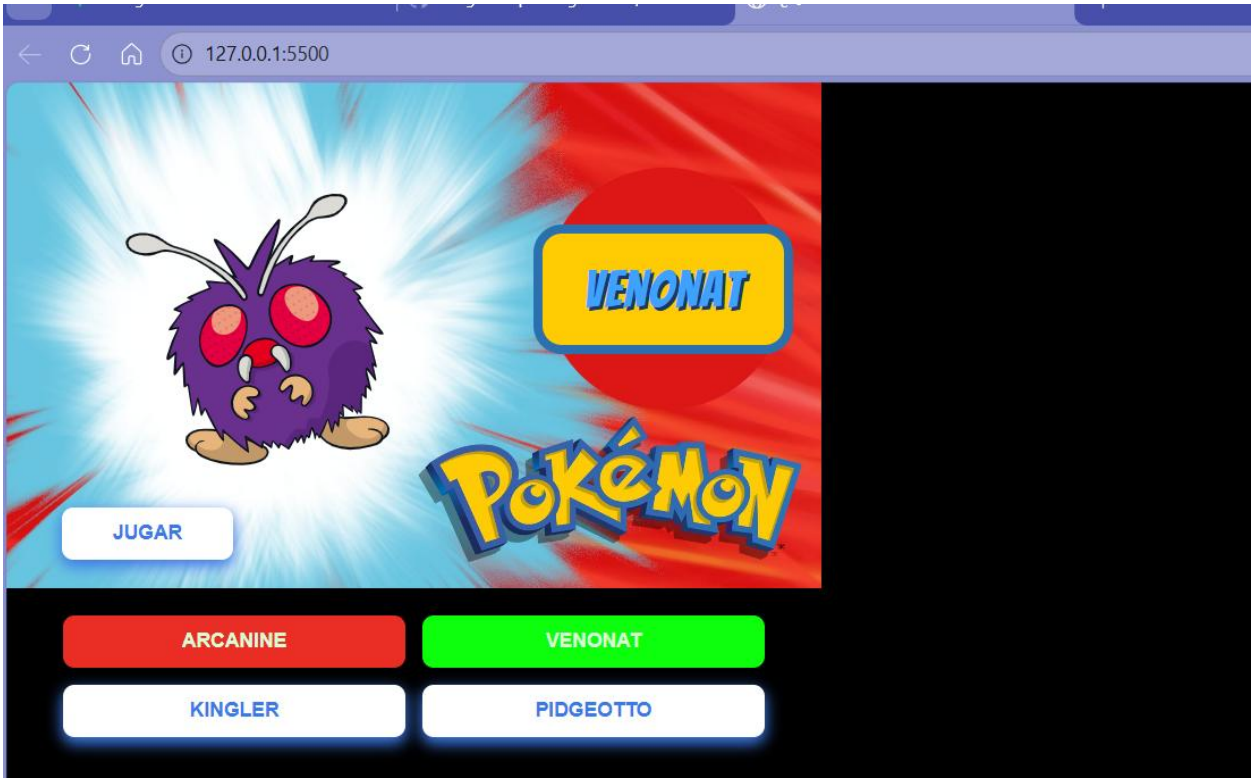
Línea 52: password VARCHAR(255) DEFAULT NULL, Columna para la contraseña (usualmente se guarda de forma encriptada).

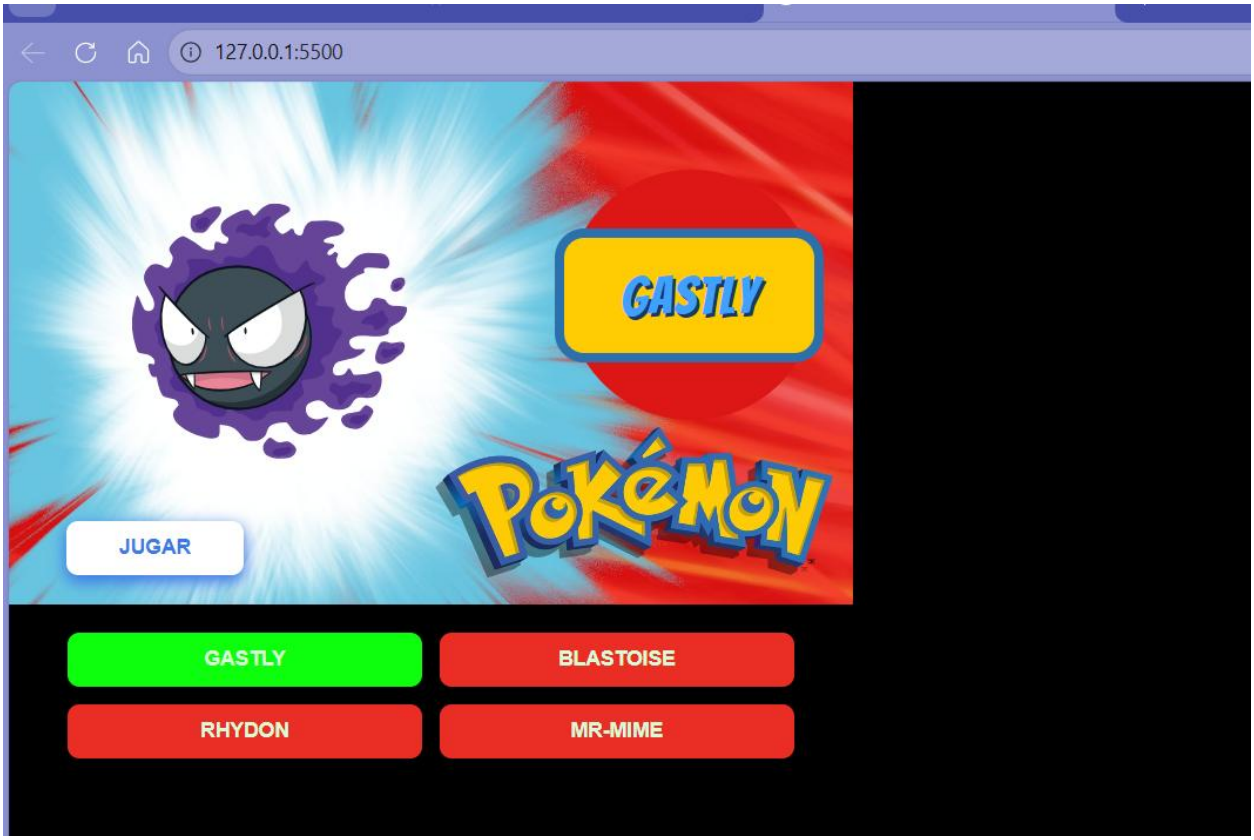
Línea 53: PRIMARY KEY (id) Establece el id como la llave principal.

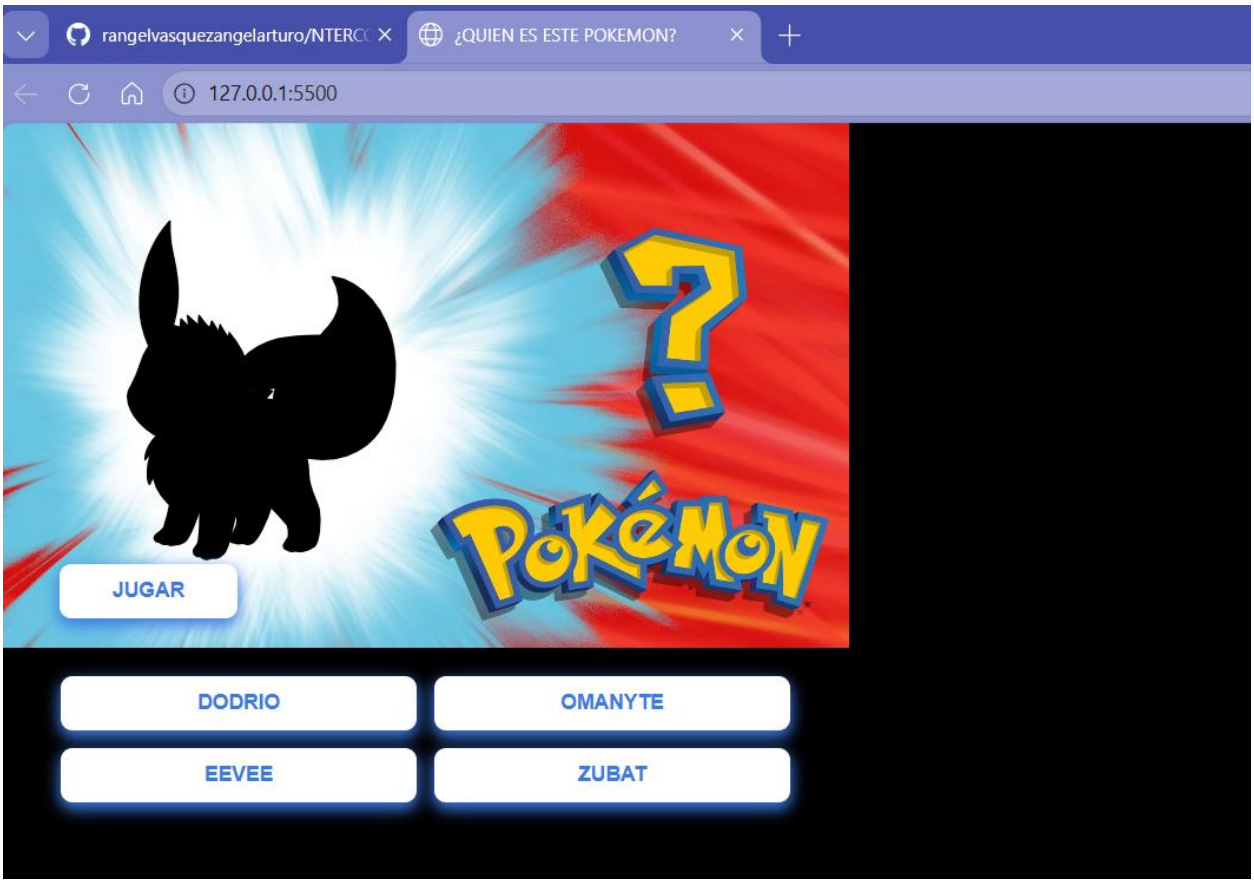
Líneas 54-56: Al igual que las anteriores, define el motor InnoDB y la compatibilidad con caracteres especiales.

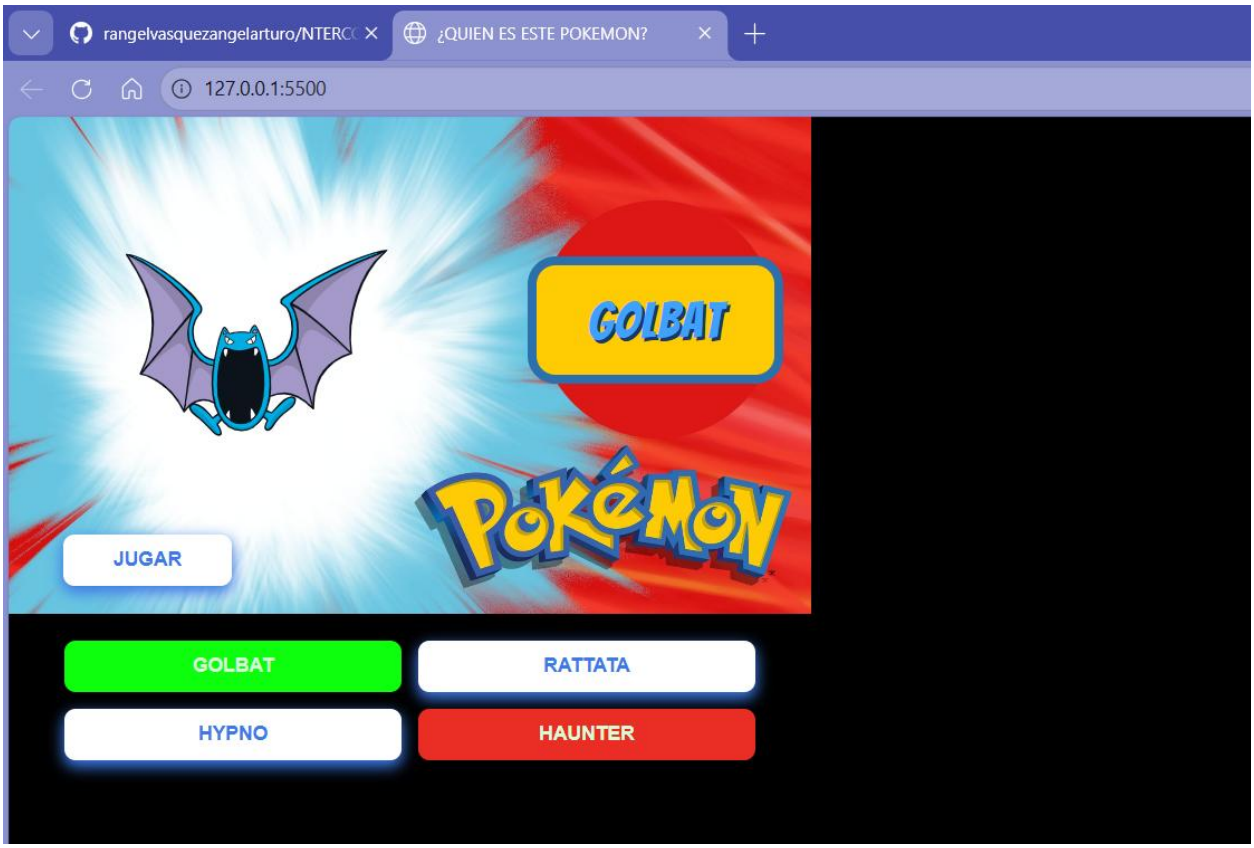
Final del Script (Líneas 58-59)

Línea 58-59: Inicia una nueva sección de comentarios llamada -- DATA: users. Esto indica que las líneas siguientes (que no se ven en la foto) probablemente serán comandos INSERT para empezar a llenar la tabla con usuarios de prueba.


127.0.0.1:5500
VENONAT
Pokémon
JUGAR
ARCANINE
VENONAT
KINGLER
PIDGEOTTO


127.0.0.1:5500
GASTLY
JUGAR
GASTLY
BLASTOISE
RHYDON
MR-MIME


rangelvasquezangelarturo/NTERC
¿QUIEN ES ESTE POKEMON?
127.0.0.1:5500
JUGAR
DODRIO
OMANYTE
EEVEE
ZUBAT


rangelvasquezangelarturo/NTERC
¿QUIEN ES ESTE POKEMON?
127.0.0.1:5500
GOLBAT
JUGAR
GOLBAT
RATTATA
HYPNO
HAUNTER