S.E.P. TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO de Tuxtepec

PROGRAMACION EN AMBIENTE CLIENTE-SERVIDOR

REPORTE DE PRACTICA

NOMBRE DEL TRABAJO:
“Pokemon”

ESTUDIANTE
SAMUEL GRACIANO AZAMAR
22350635

CARRERA: **INGENIERÍA INFORMATICA**

NOMBRE DEL DOCENTE:
DR. JULIO AGUILAR CARMONA

PERIODO DE REALIZACIÓN:
AGOSTO-DICIEMBRE/2025

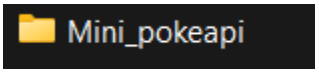
SEMESTRE Y GRUPO: 7°A

INTRODUCCION

En la presente práctica se desarrolló una aplicación web interactiva haciendo uso de herramientas y tecnologías actuales orientadas al desarrollo de software, tales como Visual Studio Code como entorno de programación y MySQL Workbench para la gestión de la base de datos. El objetivo principal de esta práctica fue aplicar los conocimientos adquiridos sobre el consumo de APIs, la manipulación del DOM, la programación asíncrona en JavaScript, así como el diseño y administración de una base de datos relacional.

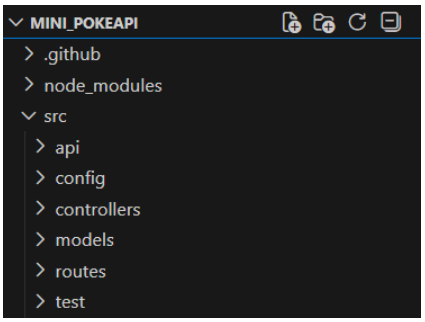
El proyecto implementado consiste en un juego interactivo el cual permite al usuario interactuar con una interfaz gráfica desarrollada en HTML, CSS y JavaScript, donde se obtiene información dinámica de Pokémon mediante el consumo de la PokeAPI y de un servidor local. A través de esta interacción, el sistema muestra una silueta de un Pokémon y presenta varias opciones para que el usuario seleccione la respuesta correcta, reforzando así el uso de lógica de programación y eventos.

1. Primero creamos la carpeta llamada Mini_pokeapi.

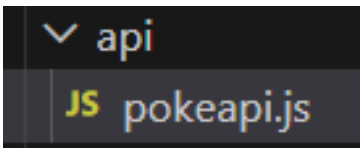


2. Después vamos a instalar los modulos necesarios.

Creamos una carpeta src, ahí mismo creamos las carpetas api, config, controllers, models, routes, test.



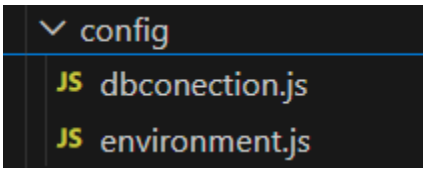
3. En cada carpeta tiene su js.
   API:



Primero, realiza una única consulta para traer los nombres y enlaces básicos de los Pokémon; luego, usa el método. map para recorrer esa lista y agregarle a cada uno una URL directa hacia su imagen alojada en GitHub. El código incluye un bloque comentado que muestra una forma alternativa (pero más lenta) de obtener los datos haciendo 151 peticiones individuales. Finalmente, exporta la función para que pueda ser usada en otras partes de tu aplicación.

```
const getPokemons = async () => {
    //const pokemon = [];
    //let pokemons = [];
    // Link de imagen de pokemon en .svg


    const response = await fetch(`https://pokeapi.co/api/v2/pokemon?limit=151`);
    const { results } = await response.json();

    const pokemons = results.map((pokemon, index) => {
        return { ...pokemon, image: `https://raw.githubusercontent.com/PokeAPI/sprites/master/
    });

    // for (let i = 1; i <= 151; i++) {
    //     // const response = await fetch(`https://pokeapi.co/api/v2/pokemon/${i}`);
    //     // const data = await response.json();

    //     const pokemon = {
    //         name: data.name,
    //         imagen: data.sprites.other.dream_world.front_default,
    //     }
    //pokemons.push(pokemon);
    //}


    return pokemons;

}

module.exports = {
    getPokemons
};
```

4. CONFIG:



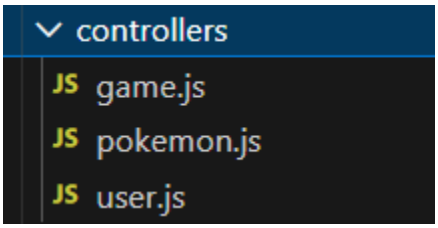
Carga la librería dotenv para leer un archivo secreto. env y organiza los valores (puerto, usuario, contraseña) en un objeto llamado env.

```
const mariadb = require('mariadb');
const env = require(`./environment`);

const config = {
    host: '127.0.0.1',
    user: env.dbUser,
    password: env.dbPassword,
    database: env.dbName,
    port: env.dbPort,
    connectionLimit: 10,
}

const pool = mariadb.createPool(config);

module.exports = pool;
```

Importa esa configuración para crear un pool de conexiones, el cual permite gestionar múltiples consultas de forma eficiente al servidor local, exportando finalmente este "pool" para que el resto de la aplicación pueda realizar operaciones en la base de datos.

```
// Se importara la información que exista en el archivo .env

require('dotenv').config();

const env = {
    port: process.env.PORT,
    dbPort: process.env.DB_PORT,
    dbName: process.env.DB_NAME,
    dbUser: process.env.DB_USERNAME,
    dbPassword: process.env.DB_PASSWORD

};

module.exports = env;
```

5. CONTROLLERS:



El código carga variables de entorno seguras (como contraseñas y puertos) para configurar un pool de conexiones a MariaDB. Finalmente, se presenta una función llamada pokemonSeeder que obtiene la lista de Pokémon desde una API externa, limpia la tabla existente en la base de datos ignorando temporalmente las restricciones de llaves foráneas y recorre cada personaje para insertarlo individualmente con su nombre e imagen.

Game.js

```
const { request, response } = require('express');
const pool = require('../config/dbconection');
const { gameQuery } = require('../models/game');
const { users } = require('../models/user');

function isNaN(number: number): boolean

Returns a Boolean value that indicates whether a value is the reserved value NaN (not a number

@param number — A numeric value.

const wi
    cons

    if (isNaN(Number(id))) {
        res.status(400).send("Esto no es un numero");
        return;
    }



    let conn;
    try {
        conn = await pool.getConnection();
        const [user] = await conn.query(users.view, [id]);

        if (!user) {
            res.status(500).send("No se pudo registrar la victoria");
            return;
        }
        const [game] = await conn.query(gameQuery.getGame, [id]);

        if (!game) {
            const newGame = await conn.query(gameQuery.addGame, [id, 1, 0]);
            if (newGame.affectedRows === 0) {
                res.status(500).send("No se pudo registrar la victoria");
                return;
            }
            res.send({ msg: "registro del juego creado" });
          return;
        }

        const gameUpdated = await conn.query(gameQuery.updateGame, [game.win + 1, game.lose, id]);
        if (gameUpdated.affectedRows === 0) {
            res.status(500).send("No se pudo actualizar la victoria");
            return;
        }
        res.send({ msg: "Victoria registrada", game });
    } catch (err) {
        res.status(500).send("error");
    } finally {
        if (conn) conn.end();
    }
}

const lose = async (req = request, res = response) => {
    const { id } = req.params;

    if (isNaN(Number(id))) {
        res.status(400).send("Esto no es un numero");
        return;
    }



    let conn;
    try {
        conn = await pool.getConnection();
        const [user] = await conn.query(users.view, [id]);
```

```
        if (!user) {
            res.status(404).send("Usuario no encontrado")
            return;
        }
        const [game] = await conn.query(gameQuery.getGame, [id]);

        if (!game) {
            const newGame = await conn.query(gameQuery.addGame, [id, 0, 1]);
            if (newGame.affectedRows === 0) {
                res.status(500).send("No se pudo registrar");
                return;
            }
            res.send({ msg: "registro del juego creado" });
            return;
        }

        const gameUpdated = await conn.query(gameQuery.updateGame, [game.win, game.lose + 1, id]);
        if (gameUpdated.affectedRows === 0) {
            res.status(500).send("No se pudo actualizar el juego");
            return;
        }
        res.send({ msg: "Juego registrada", game });

    } catch (err) {
        res.status(500).send("error");
    } finally {
        if (conn) conn.end();
    }
}
```

Pokemon.js

```
const { request, response } = require('express');
const pokeapi = require('../api/pokeapi');
const pool = require('../config/dbconection');
const { pokemonQuery } = require('../models/pokemon');

const pokemonSeeder = async (req = request, res = response) => {
    const pokemons = await pokeapi.getPokemons();

    let conn;

    try {
        conn = await pool.getConnection();
        await conn.query('SET FOREIGN_KEY_CHECKS = 0');
        await conn.query('TRUNCATE TABLE pokemons');
        await conn.query('SET FOREIGN_KEY_CHECKS = 1');

        pokemons.forEach(async (pokemon) => {
            await conn.query(pokemonQuery.add, [pokemon.name, pokemon.image]);
        });
        res.send("Pokemones agregados en la Base de Datos");
    } catch (err) {
        return res.status(500).send(err);
    } finally {
        if (conn) {
            conn.end();
        }
    }

}

const randomPokemon = async (req = request, res = response) => {
    let conn;
    try {
```

```
        conn = await pool.getConnection();
        const pokemons = await conn.query(pokemonQuery.random);
        if (pokemons.length === 0) {
            return res.status(500).send("No hay pokemones en la base de datos");
        }
        res.send(pokemons);

    } catch (err) {
        return res.status(500).send(err);
    } finally {
        if (conn) {
            conn.end();
        }
    }
}

const idPokemon = async (req = request, res = response) => {
    let conn;
    try {
        const { id } = req.params
        if (isNaN(Number(id))) {
            res.status(400).send("Esto no es un numero");
            return;
        }

        conn = await pool.getConnection();
        const pokeid = await conn.query(pokemonQuery.view, [id]);
        if (!pokeid) {
            res.status(404).send("Pokemon no encontrado");
            return;
        }
```

```
            res.send(pokeid);
        } catch (err) {
            return res.status(500).send(err);
        } finally {
            if (conn) {
                conn.end();
            }
        }
    }

    module.exports = {
        pokemonSeeder,
        randomPokemon,
        idPokemon
    };
```

User.js

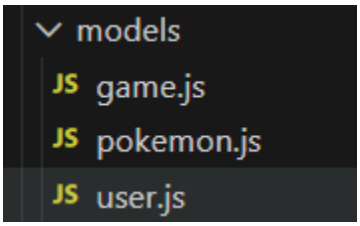
```
// Acciones que hacen los endpoints

const {request , response } = require('express');
const userQueries = require('../models/user');
const pool = require('../config/dbconection');
const bcrypt = require('bcrypt');

const saltRounds = 10;

const showUsers =  async (req= request, res =response) => {
    let conn;
    try {
        conn = await pool.getConnection();
        const users = await conn.query(userQueries.users.show);
        res.send(users);
    } catch (err) {
        res.status(500).send("error");
    } finally {
        if (conn) conn.end();
    }
}
```

```
const viewUser = async (req= request, res =response) => {
    let conn;
    try {
        const {id} = req.params
        if (isNaN(Number(id))) {
            res.status(400).send("Esto no es un numero");
            return;
        }
        conn = await pool.getConnection();
        const user = await conn.query(userQueries.users.view, [id]);
        if (!user) {
            res.status(404).send("Usuario no encontrado");
            return;
        }
        //console.log(user);
        res.send(user);
    } catch (err) {
        res.status(500).send("error");
    } finally {
        if (conn) conn.end(viewUser);
    }

}

const createUser = async (req = request, res = response) => {
    const {name, lastname, email, password} = req.body;
    if (!name || !lastname || !email || !password){
        res.status(400).send("Faltan datos");
        return;
    }
    //if(require('../models/user').users.find((u)=> u.email === email)){
```

```
    ///    res.status(400).send(`Este usuario con correo ${email} ya existe`);
    //    return;
    //}
    if (password.length < 6){
        res.status(400).send("La contraseña debe tener al menos 6 caracteres");
        return;
    }
    //require('../models/user').users.push({
    //    id: require('../models/user').users.length + 1,
    //    name,
    //    lastname,
    //    email,
    //    password
    //});
    //res.send({msg: "Usuario creado ", user : {name, lastname, email, password}});
    //res.send(req.body)
    let conn;
    try {
        conn = await pool.getConnection();
        const userExist = await conn.query( userQueries.users.verifyEmail, [email]);
        if (userExist.length > 0) {
            res.status(400).send(`Este usuario con correo ${email} ya existe`);
            return;
        }

        const hashedPassword = await bcrypt.hash(password, saltRounds);

        const userCreated = await conn.query(userQueries.users.create, [name, lastname, email, hashe
```

```
        if (userCreated.affectedRows === 0) {
            res.status(500).send("No se pudo crear el usuario");
            return;
        }
        res.send(`Usuario creado con id ${userCreated.insertId}`);
    } catch (err) {
        res.status(500).send("error");
    } finally {
        if (conn) conn.end();
    }
}

const removeUser = async (req = request, res = response) => {
    const {id} = req.params;

    if (isNaN(Number(id))) {
        res.status(400).send("Esto no es un numero");
        return
    }
```

6. MODELS:



Se observa la lógica de un "seeder" que descarga información de una API externa, limpia la tabla de la base de datos desactivando temporalmente las restricciones de llaves foráneas e inserta los nuevos Pokémon con su nombre e imagen.

Game.js

```
const gameQuery = {
    getGame: 'SELECT * FROM games WHERE user_id = ?',
    addGame: 'INSERT INTO games (user_id, win, lose) VALUES (?, ?, ?)',
    updateGame: 'UPDATE games SET win = ?, lose = ? WHERE user_id = ?'
}

module.exports = {
    gameQuery
};

```

Pokemos.js

```
const pokemonQuery = {
    add: 'INSERT INTO pokemons (name, image) VALUES (?, ?)',
    random: 'SELECT * FROM pokemons ORDER BY RAND() LIMIT 4',
    view: 'SELECT * FROM pokemons WHERE id = ?'
}


module.exports = {
    pokemonQuery
};
```

User.js

```
// Se escriben las interacciónes con la BD

const users = {
    show: "SELECT * FROM users",
    view: "SELECT * FROM users WHERE id = ?",
    verifyEmail: "SELECT * FROM users WHERE email = ?",
    create : "INSERT INTO users (name, lastname, email, password) VALUES (?, ?, ?, ?)",
    delete: "DELETE FROM users WHERE id = ?",
    update: "UPDATE users SET name = ?, lastname = ?, email = ?, password = ? WHERE id = ?",
    viewByEmail: "SELECT * FROM users WHERE email = ?"
}


module.exports = {
    users
};
```

7. ROUTES:

Se observa un controlador de "sembrado" (seeder) que consume una API externa, limpia la tabla de Pokémon ignorando restricciones de integridad temporalmente e inserta los nuevos registros masivamente

Game.js

src > routes > JS game.js > ...

```
1   const { Router } = require('express');
2   const router = Router();
3   // Controladores
4
5   router.get('/win/:id', require('../controllers/game').win);
6   router.get('/lose/:id', require('../controllers/game').lose);
7   // router.get('/:id', require('../controllers/game').view);
8
9
10  module.exports = router;
```

Pokemon.js

src > routes > JS pokemon.js > ...

```
1   const { Router } = require('express');
2
3   const router = Router();
4
5   // Controladores
6
7   router.get('/seed', require('../controllers/pokemon').pokemonSeeder);
8   router.get('/random', require('../controllers/pokemon').randomPokemon);
9
10
11  module.exports = router;
```

User.js

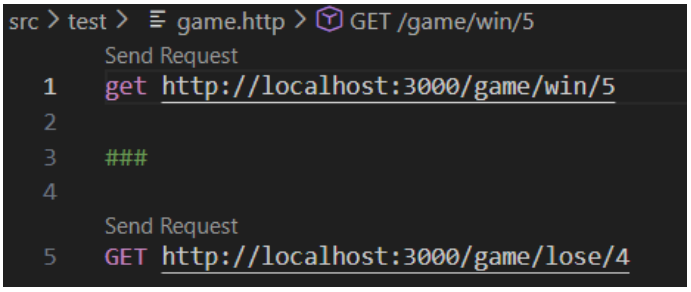
src > routes > JS user.js > ...

```
1   //Rutas de los Endpoints
2   const { Router } = require('express');
3   const router = Router();
4   router.get('/', require('../controllers/user').showUsers);
5   router.get('/:id',require('../controllers/user').viewUser);
6   router.post('/',require('../controllers/user').createUser);
7   router.delete('/:id',require('../controllers/user').removeUser);
8   router.put('/:id',require('../controllers/user').updateUser);
9   router.post('/login',require('../controllers/user').loginUser);
10  module.exports = router;
```

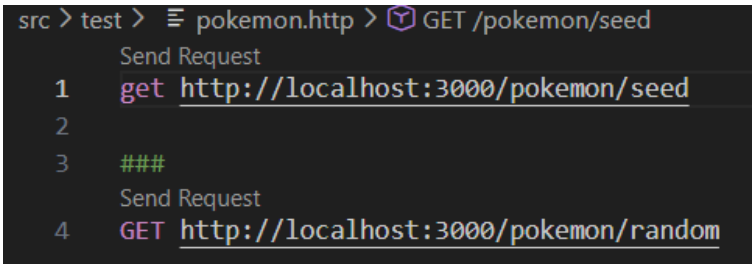
8. TEST:

La capa de acceso se define mediante rutas de Express que exponen puntos finales como /win/:id y /lose/:id, los cuales son validados a través de archivos de pruebas. http que simulan peticiones de inicio de sesión, actualización de contraseñas y eventos de juego en el entorno local.

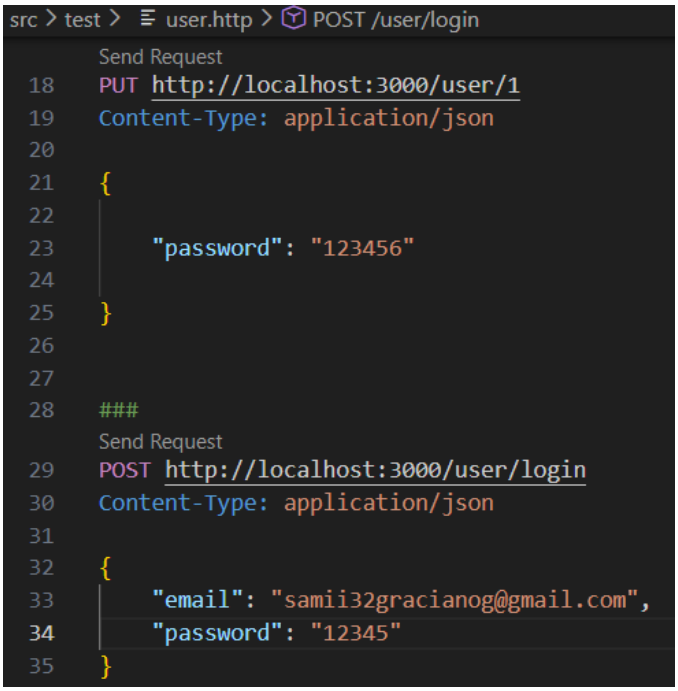
Game.http



Pokemon.http



User.http



9. PACKAGE-LOCK. JASON

La lógica de datos se divide en dos partes: un controlador de "sembrado" (pokemonSeeder) que descarga información de la PokeAPI para llenar la base de datos local tras limpiar las tablas existentes, y un modelo de consultas SQL (gameQuery) diseñado para registrar victorias, derrotas y consultar el estado de los juegos por usuario.

```
{} package-lock.json > ...
{
  "name": "mini_pokeapi",
  "version": "1.0.0",
  "lockfileVersion": 3,
  "requires": true,
  "packages": {
    "": {
      "name": "mini_pokeapi",
      "version": "1.0.0",
      "license": "ISC",
      "dependencies": {
        "bcrypt": "^6.0.0",
        "cors": "^2.8.5",
        "dotenv": "^17.2.3",
        "express": "^5.1.0",
        "mariadb": "^3.4.5"
      }
    },
    "node_modules/@types/geojson": {
      "version": "7946.0.16",
      "resolved": "https://registry.npmjs.org/@types/geojson/-/geojson-7946.0.16.tgz",
      "integrity": "sha512-6C8nqWur3j98U6+lXDfTUWIfgvZU+EumvpHKcYjujKH7woYyLj2sUmff0tRhrqM7BohUw7Pz3
      "license": "MIT"
    },
    "node_modules/@types/node": {
      "version": "24.10.1",
      "resolved": "https://registry.npmjs.org/@types/node/-/node-24.10.1.tgz",
      "integrity": "sha512-GNWcUTRBgIRJD5zj+Tq0fKOJ5XZajIiBroOF0yvj2bSU1WvNdYS/dn9UxwsujGW4JX06dnHyj
      "license": "MIT",
      "dependencies": {
        "undici-types": "~7.16.0"
      }
```

```
    },
    "node_modules/accepts": {
      "version": "2.0.0",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-2.0.0.tgz",
      "integrity": "sha512-5cvg6CtKwfgdmVqY1WIiXKc3Q1bkRqGLi+2W/6ao+6Y7gu/RCwRuAhGEzh5B4KlszSuTLgZYu
      "license": "MIT",
      "dependencies": {
        "mime-types": "^3.0.0",
        "negotiator": "^1.0.0"
      },
      "engines": {
        "node": ">= 0.6"
      }
    },
    "node_modules/bcrypt": {
      "version": "6.0.0",
      "resolved": "https://registry.npmjs.org/bcrypt/-/bcrypt-6.0.0.tgz",
      "integrity": "sha512-cU8v/EGSrnH+HnxV2z0J7/blxH8gq7Xh2JFT6Aroax7UohdmiJJlxApMxtKfuI7z68NvvVcmR
      "hasInstallScript": true,
      "license": "MIT",
      "dependencies": {
        "node-addon-api": "^8.3.0",
        "node-gyp-build": "^4.8.4"
      },
      "engines": {
        "node": ">= 18"
      }
    },
    "node_modules/body-parser": {
```

10. PACKAGE.js

Esta serie de imágenes detalla la arquitectura de "mini_pokeapi", un sistema backend en Node.js para un juego de Pokémon. El flujo comienza con la carga de credenciales seguras mediante dotenv para establecer un pool de conexiones a MariaDB.

11. POKEMONDB.sql

Un esquema SQL con tablas para usuarios, Pokémon y estadísticas de partidas, además de un controlador de "sembrado" (pokemonSeeder) que descarga información de la PokeAPI externa para poblar la base de datos local. El sistema define rutas de Express para registrar eventos como victorias o derrotas (/win/:id, /lose/:id) y utiliza modelos SQL específicos para actualizar el progreso del jugador.

pokemondb-20251128.sql

```sql
-- ------------------------------------------------------------
-- TablePlus 6.7.0(634)
--
-- https://tableplus.com/
--
-- Database: pokemondb
-- Generation Time: 2025-11-28 13:16:05.7000
-- ------------------------------------------------------------


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;


DROP TABLE IF EXISTS `games`;
CREATE TABLE `games` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) DEFAULT NULL,
  `win` int(11) DEFAULT NULL,
  `lose` int(11) DEFAULT NULL,
  `date` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci;

DROP TABLE IF EXISTS `pokemons`;
CREATE TABLE `pokemons` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `image` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci;

DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `lastname` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_uca1400_ai_ci;



/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

12. POKEMON DB

La base de datos cuenta con un esquema SQL que define tablas para usuarios, Pokémon y estadísticas de juego, las cuales se llenan mediante un controlador de "sembrado" o seeder que descarga datos directamente desde la PokeAPI.

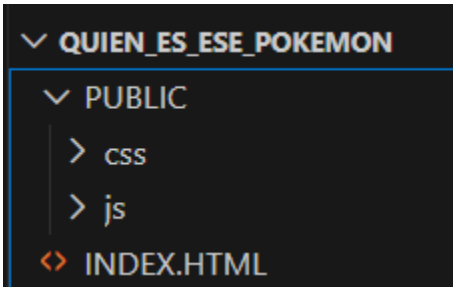
pokemondb.sql

```sql
CREATE DATABASE  IF NOT EXISTS `pokemondb` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT EN
USE `pokemondb`;
-- MySQL dump 10.13  Distrib 8.0.44, for Win64 (x86_64)
--
-- Host: 127.0.0.1    Database: pokemondb
-- ------------------------------------------------------
-- Server version 8.0.44

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `games`
--

DROP TABLE IF EXISTS `games`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `games` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_id` int DEFAULT NULL,
  `win` int DEFAULT NULL,
  `lose` int DEFAULT NULL,
  `date` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `games`
--

LOCK TABLES `games` WRITE;
/*!40000 ALTER TABLE `games` DISABLE KEYS */;
/*!40000 ALTER TABLE `games` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `pokemons`
--

DROP TABLE IF EXISTS `pokemons`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `pokemons` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `image` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `pokemons`
--
```

pokemondb.sql

```sql
LOCK TABLES `pokemons` WRITE;
/*!40000 ALTER TABLE `pokemons` DISABLE KEYS */;
/*!40000 ALTER TABLE `pokemons` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `lastname` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `email` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `password` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `users`
--

LOCK TABLES `users` WRITE;
/*!40000 ALTER TABLE `users` DISABLE KEYS */;
INSERT INTO `users` VALUES (0,'samuel','graciano','samii32gracianog@gmail.com','12345'),(1,'sami','azamar','azamarsamuel@gmail.com'
/*!40000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```

13. Ahora en la carpeta QUIEN ES ESE POKEMON, creamos el modulo css y js.



14. CSS

Styles.css

Esta secuencia de imágenes presenta la arquitectura completa de una aplicación web llamada "mini_pokeapi" centrada en un juego de Pokémon. El sistema comienza con una base de datos MariaDB definida por esquemas SQL que crean tablas para usuarios, Pokémon y estadísticas de partidas.

PUBLIC > css > # styles.css > body

```css
body {
  margin: 0;
  background-color: #000;
  color: #fff;
}

main {
  position: relative;
  width: 575px;
  background-image: url('https://res.cloudinary.com/dzynqn10l/image/upload/v1633196203/Msc/pokemon-bg_ig4uv3.jpg');
  background-size: 110%;
  background-repeat: no-repeat;
}

#pokemon-container {
  height: 355px;
}

#answer {
  display: none;
}

#bg-overlay {
  color: #fff;
  position: absolute;
  background-color: #dd1716;
  right: 25px;
  width: 170px;
  height: 170px;
  top: 60px;
  border-radius: 50%;
}

#text-overlay {
  position: absolute;
  right: 20px;
  top: 100px;
  background-color: #ffcb02;
  border: solid 7px #2c70ae;
  padding: 20px;
  min-width: 130px;
  font-family: 'Bangers', Arial, sans-serif;
  font-size: 36px;
  letter-spacing: 1.2px;
  color: #3ea2fe;
  text-shadow: 2px 2px #1d366c;
  border-radius: 20px;
  text-align: center;
}

#controls {
  position: relative;
  padding: 20px 40px;
}

button {
  background: #fff;
  border: none;
  color: #3e7ae7;
  font-weight: 600;
  font-size: 14px;
  padding: 10px 36px;
  border-radius: 8px;
  text-transform: uppercase;
  box-shadow: 0px 4px 10px 1px;
  cursor: pointer;
  transition: all .2s ease-out;
}

button:hover {
  box-shadow: 0px 4px 10px 4px;
```

15. JS

Api.js

PUBLIC > js > JS api.js > ...

```js
const getPokemon = async () => {
    const res = await fetch('http://localhost:3000/pokemon/random');
    const data = await res.json();
    return data;
}


window.getPokeData = async () => {
    const pokemon = await getPokemon();

    const randomIndex = Math.floor(Math.random() * pokemon.length);


    const correctAnswer = pokemon[randomIndex];

    //const correctAnswer = pokemon.find(p => p.isCorrect);


    return {
        pokemonChoices: pokemon,
        correctAnswer,
    }
    //console.log(pokemon);
}


```

Index.js

PUBLIC > js > JS index.js > ...

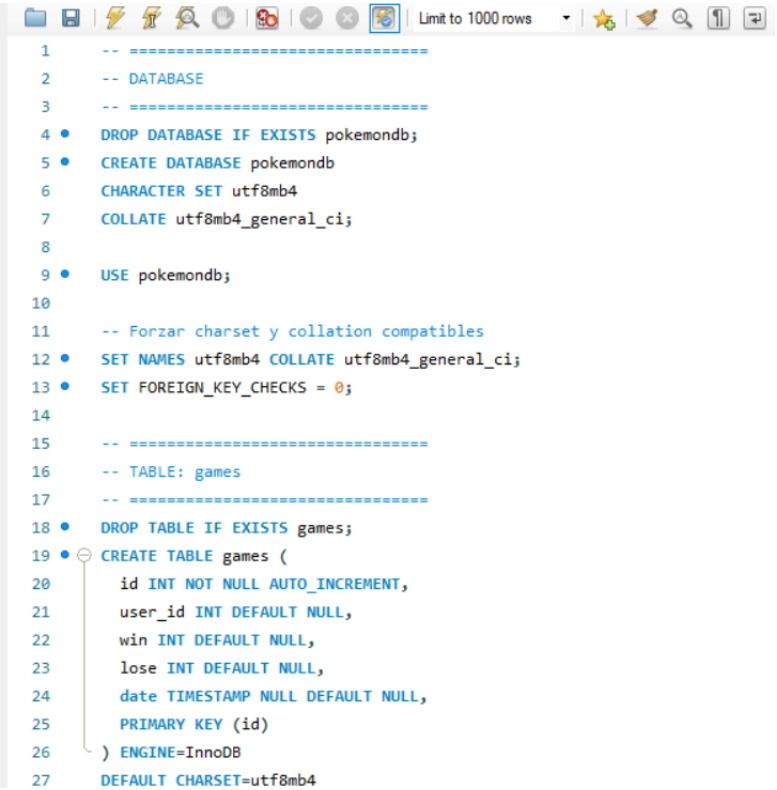
```js
const playBtn = document.querySelector('#play');
const choices = document.querySelector('#choices');
const main = document.querySelector('main');
const pokemonImage = document.querySelector('#pokemon-image');
const textOverlay = document.querySelector('#text-overlay');

let gameData;

const revealPokemon = () => {
    main.classList.add('revealed');
    textOverlay.textContent = gameData.correctAnswer.name;
}

const showSiluette = () => {
    main.classList.remove('fetching');
    pokemonImage.src = gameData.correctAnswer.image;

}

const displayChoices = () => {
    const { pokemonChoices } = gameData;
    const choisesHTLM = pokemonChoices.map(pokemon => {
        return `<button data-name="${pokemon.name}"> ${pokemon.name} </button>`;
    }).join('');

    choices.innerHTML = choisesHTLM;
};

const resetImage = () => {
    pokemonImage.src = '';
    main.classList.add('fetching');
    main.classList.remove('revealed');
}

const fetchData = async () => {
    resetImage();
    gameData = await window.getPokeData();
```

```
    showSiluette();
    displayChoices();
    //console.log(gameData)
}

playBtn.addEventListener('click', fetchData);

choices.addEventListener('click', (e) => {
    const { name } = e.target.dataset;
    const resultClass = (name === gameData.correctAnswer.name) ? 'correct' : `incorrect`;
    e.target.classList.add(resultClass);
    revealPokemon();
});
```

16. INDEX.HTML

```
INDEX.HTML > html
<!DOCTYPE html>
<html leng="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./PUBLIC/css/styles.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Bangers&display=swap">


    <title>¿QUIEN ES ESTE POKEMON?</title>
</head>

<body>
    <main class="fetching">
        <div id="pokemon-container">
            <img id="pokeball"
                src="https://res.cloudinary.com/dzynqn10l/image/upload/v1633196610/Msc/pokeball_zeoqii.webp" />
            <img id="pokemon-image" src="" />
        </div>

        <div id="answer">
            <div id="bg-overlay"> </div>
            <div id="text-overlay"> </div>
        </div>
        <section id="controls">
            <button id="play">JUGAR</button>
            <div id="choices"></div>
        </section>
    </main>

    <script src="./public/js/api.js">  </script>
    <script src="./PUBLIC/js/index.js"> </script>

</body>

</html>
```

17. En la parte de MySQL Workbench, vamos a crear la base de datos, primero creamos una nueva, en mi caso le puse como nombre pokemondb.



```
-- ================================
-- DATABASE
-- ================================
DROP DATABASE IF EXISTS pokemondb;
CREATE DATABASE pokemondb
CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;

USE pokemondb;

-- Forzar charset y collation compatibles
SET NAMES utf8mb4 COLLATE utf8mb4_general_ci;
SET FOREIGN_KEY_CHECKS = 0;

-- ================================
-- TABLE: games
-- ================================
DROP TABLE IF EXISTS games;
CREATE TABLE games (
  id INT NOT NULL AUTO_INCREMENT,
  user_id INT DEFAULT NULL,
  win INT DEFAULT NULL,
  lose INT DEFAULT NULL,
  date TIMESTAMP NULL DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

-- ================================
-- TABLE: pokemons
-- ================================
DROP TABLE IF EXISTS pokemons;
CREATE TABLE pokemons (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) DEFAULT NULL,
  image VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

-- ================================
-- TABLE: users
-- ================================
DROP TABLE IF EXISTS users;
CREATE TABLE users (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) DEFAULT NULL,
  lastname VARCHAR(255) DEFAULT NULL,
  email VARCHAR(255) DEFAULT NULL,
  password VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB
```

```
55      DEFAULT CHARSET=utf8mb4
56      COLLATE=utf8mb4_general_ci;
57
58      -- ================================
59      -- DATA: users
60      -- ================================
61  •   INSERT INTO users (name, lastname, email, password) VALUES
62      ('samuel','graciano','samii32gracianog@gmail.com','54321'),
63      ('sami','azamar','azamarsami@gmail.com','54812');
64
65  •   SET FOREIGN_KEY_CHECKS = 1;
```

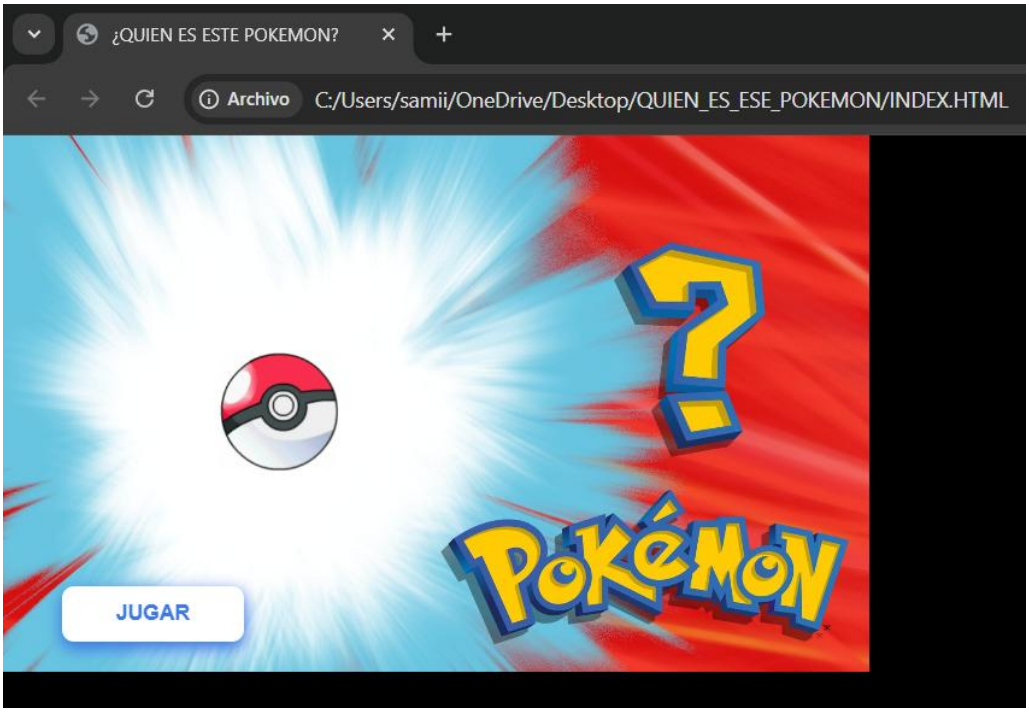
18. Después vamos a ejecutar el código y nos debe dar lo siguiente.

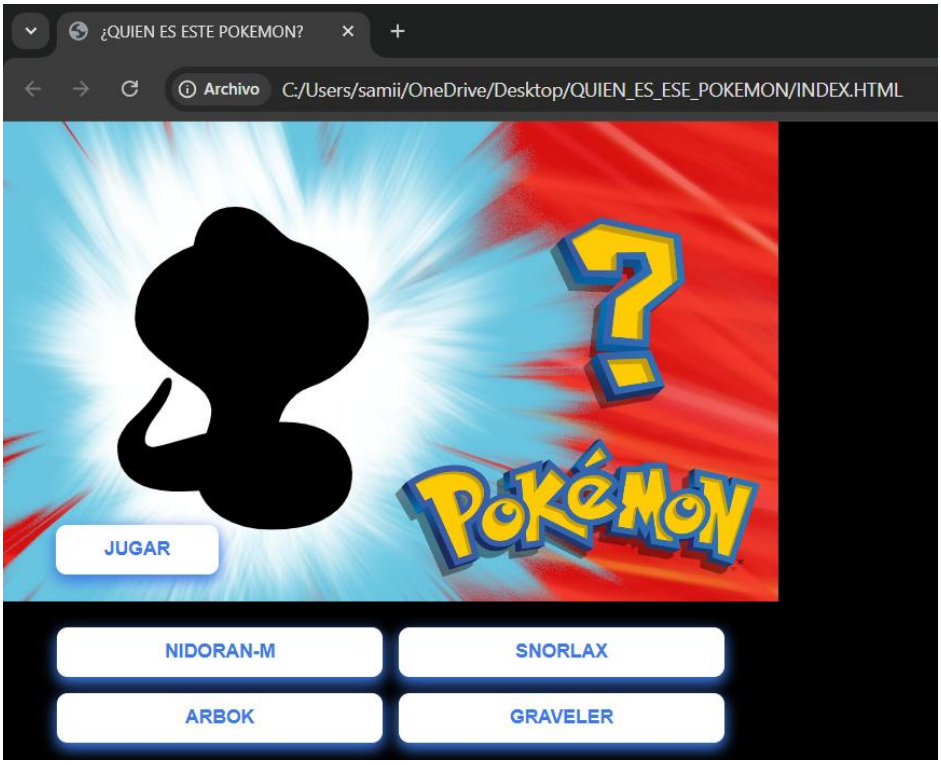
Output

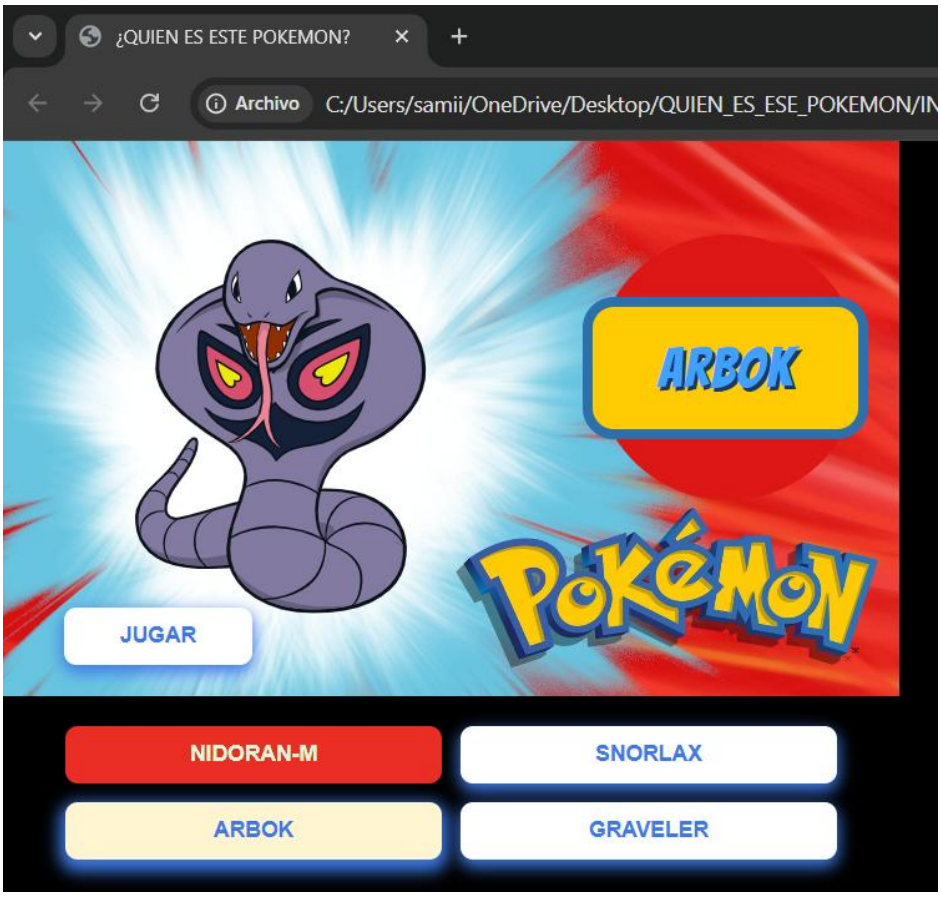
Action Output

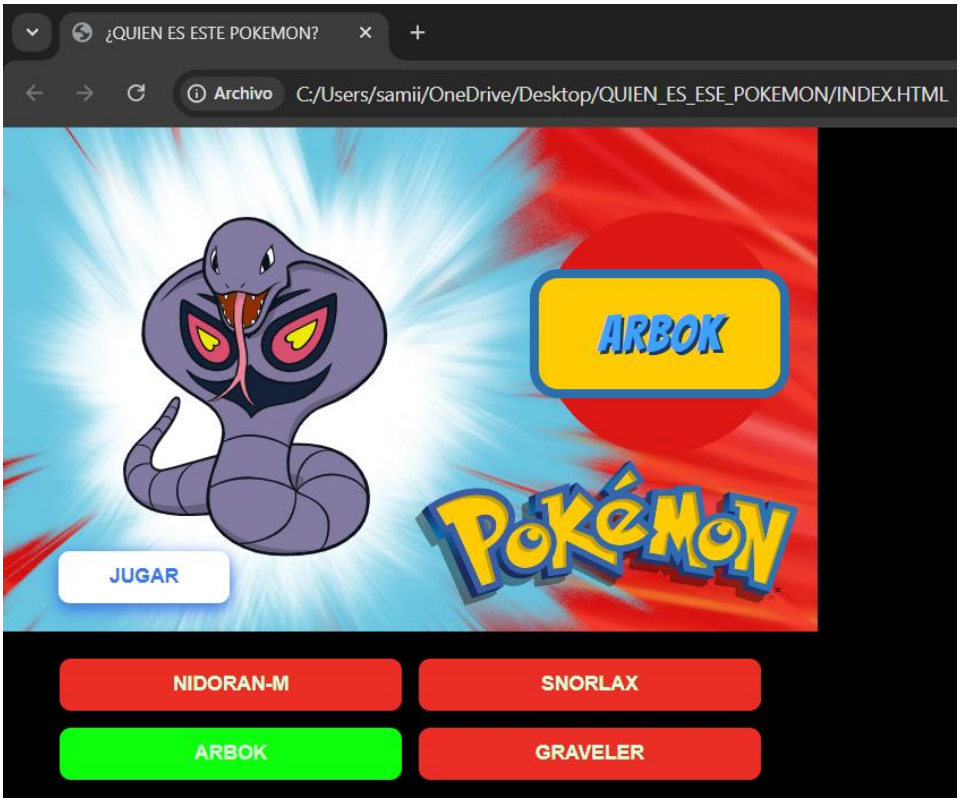
#	Time	Action	Message	Duration / Fetch
1	13:20:42	DROP DATABASE IF EXISTS pokemondb	0 row(s) affected, 1 warning(s): 1008 Can't drop database 'pokemondb'; database doesn't exist	0.000 sec
2	13:20:42	CREATE DATABASE pokemondb CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci	1 row(s) affected	0.000 sec
3	13:20:42	USE pokemondb	0 row(s) affected	0.000 sec
4	13:20:42	SET NAMES utf8mb4 COLLATE utf8mb4_general_ci	0 row(s) affected	0.000 sec
5	13:20:42	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
6	13:20:42	DROP TABLE IF EXISTS games	0 row(s) affected, 1 warning(s): 1051 Unknown table 'pokemondb.games'	0.016 sec
7	13:20:42	CREATE TABLE games (id INT NOT NULL AUTO_INCREMENT, user_id INT DEFAULT NULL, win INT DEFA...	0 row(s) affected	0.015 sec
8	13:20:42	DROP TABLE IF EXISTS pokemons	0 row(s) affected, 1 warning(s): 1051 Unknown table 'pokemondb.pokemons'	0.000 sec
9	13:20:42	CREATE TABLE pokemons (id INT NOT NULL AUTO_INCREMENT, name VARCHAR(255) DEFAULT NULL, i...	0 row(s) affected	0.016 sec
10	13:20:42	DROP TABLE IF EXISTS users	0 row(s) affected, 1 warning(s): 1051 Unknown table 'pokemondb.users'	0.015 sec
11	13:20:42	CREATE TABLE users (id INT NOT NULL AUTO_INCREMENT, name VARCHAR(255) DEFAULT NULL, lastna...	0 row(s) affected	0.016 sec
12	13:20:42	INSERT INTO users (name, lastname, email, password) VALUES ('samuel','graciano','samii32gracianog@gmail.com','...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.016 sec
13	13:20:42	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.000 sec

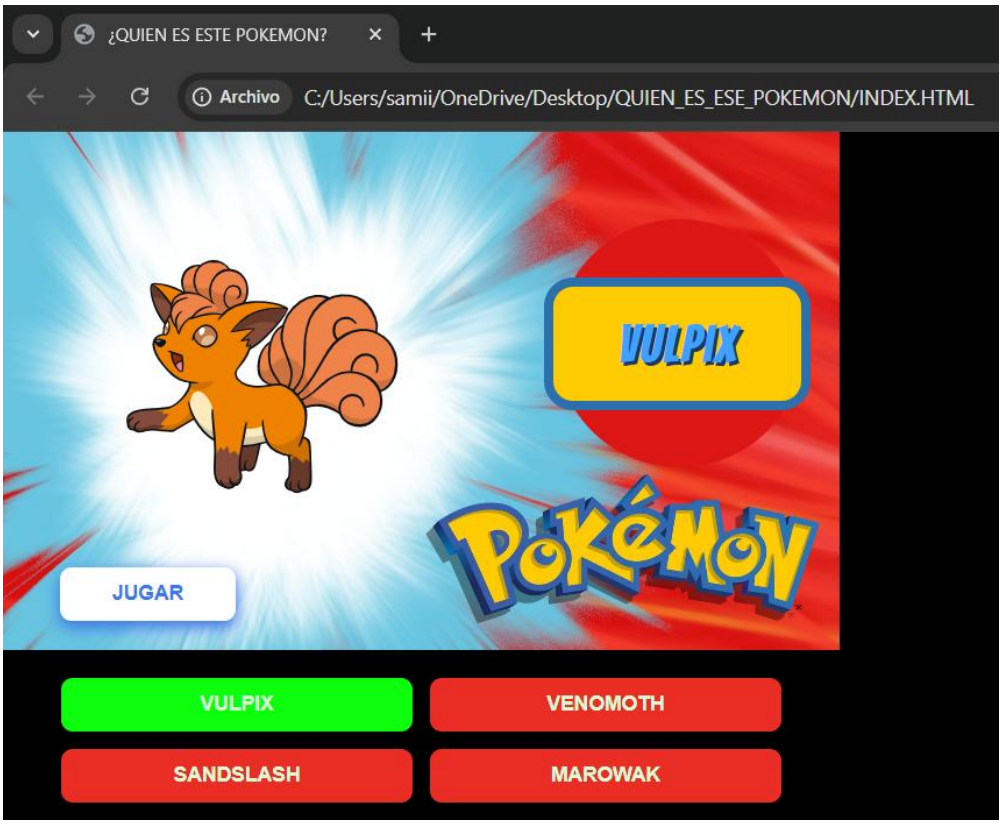
19. RESULTADO FINAL DEL VIDEOJUEGO y lo ejecutamos en Visual Studio Code.

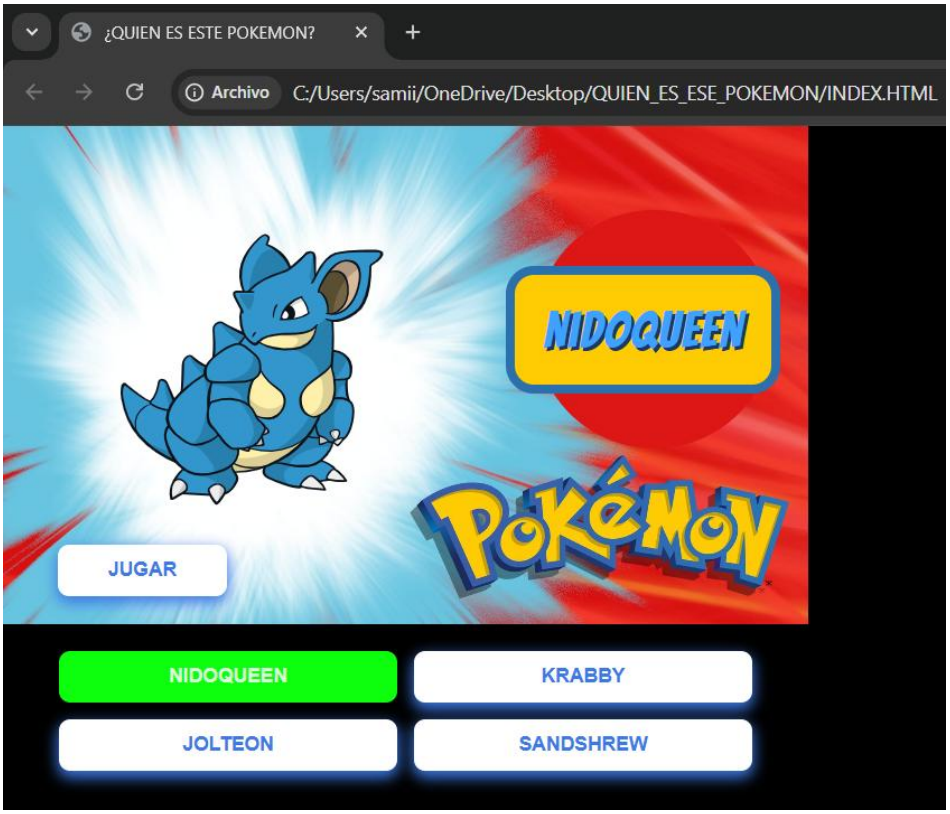


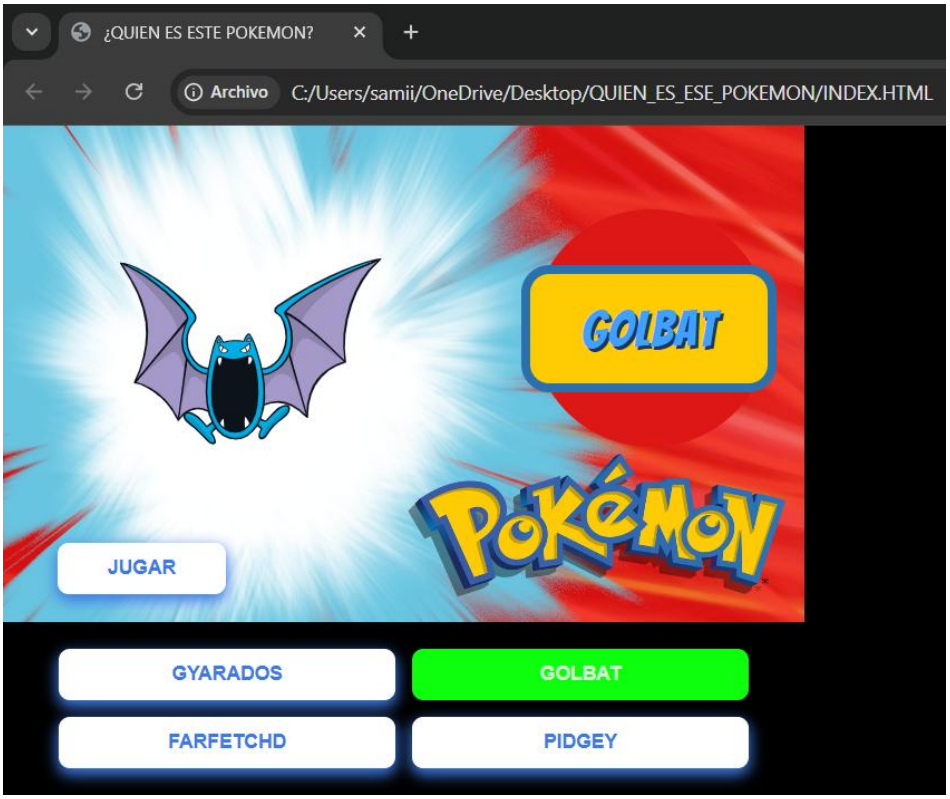

¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
?
Pokémon
JUGAR
NIDORAN-M
SNORLAX
ARBOK
GRAVELER

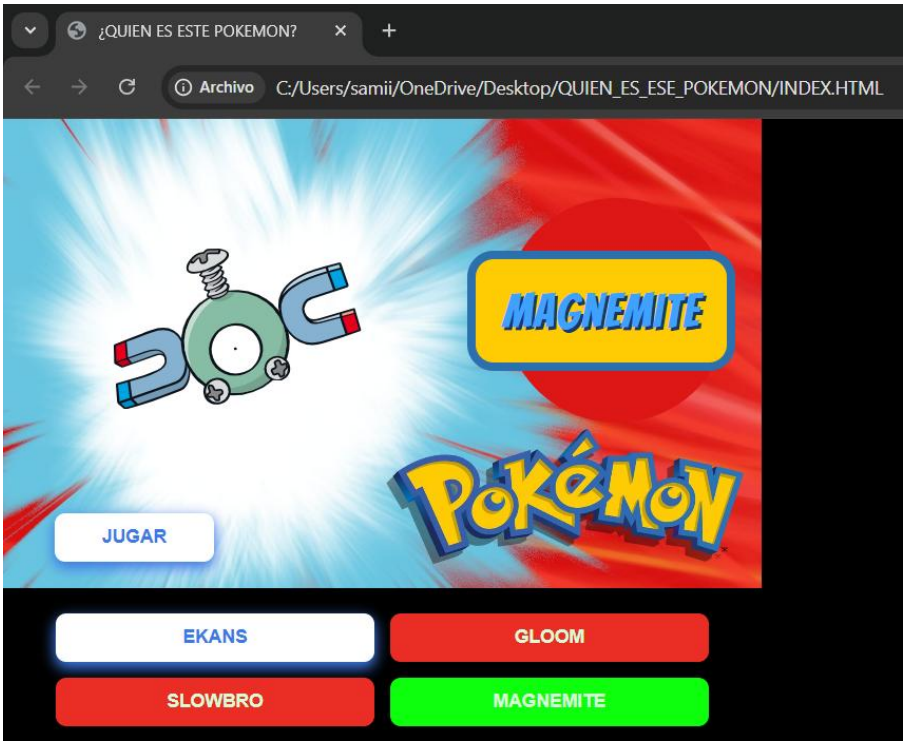

¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/IN
ARBOK
Pokémon
JUGAR
NIDORAN-M
SNORLAX
ARBOK
GRAVELER

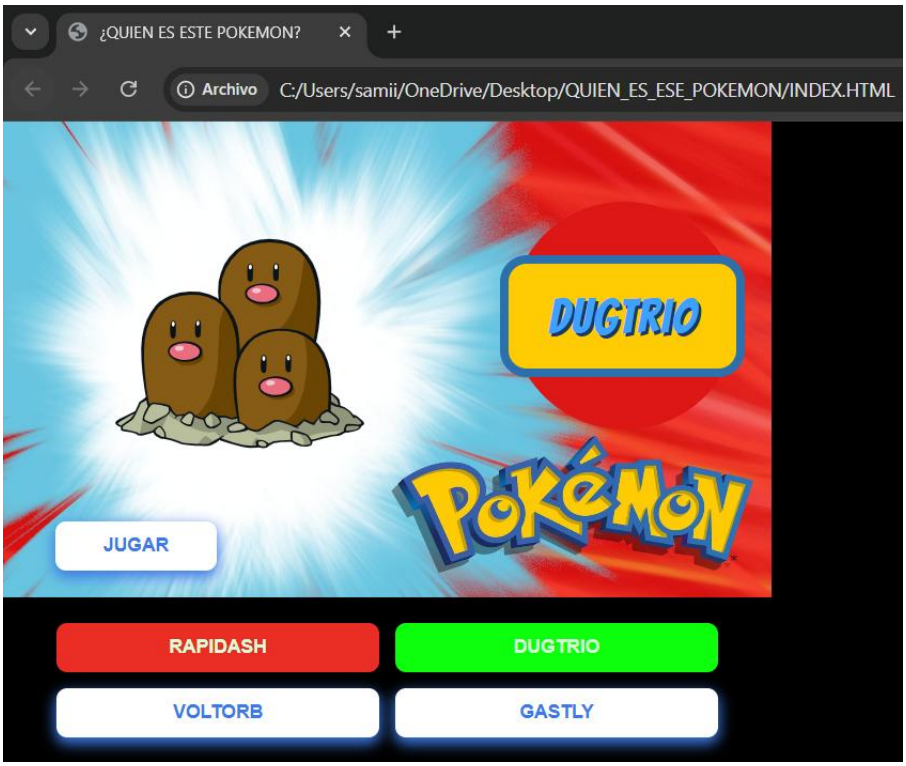

¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
ARBOK
JUGAR
NIDORAN-M
SNORLAX
ARBOK
GRAVELER


¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
VULPIX
JUGAR
VULPIX
VENOMOTH
SANDSLASH
MAROWAK


¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
NIDOQUEEN
JUGAR
NIDOQUEEN
KRABBY
JOLTEON
SANDSHREW


¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
GOLBAT
JUGAR
GYARADOS
GOLBAT
FARFETCHD
PIDGEY


¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
MAGNEMITE
JUGAR
EKANS
GLOOM
SLOWBRO
MAGNEMITE


¿QUIEN ES ESTE POKEMON?
Archivo
C:/Users/samii/OneDrive/Desktop/QUIEN_ES_ESE_POKEMON/INDEX.HTML
DUGTRIO
JUGAR
RAPIDASH
DUGTRIO
VOLTORB
GASTLY

CONCLUSIÓN

En conclusión, la práctica realizada permitió consolidar y aplicar de manera efectiva los conocimientos teóricos y prácticos relacionados con el desarrollo de aplicaciones web y la gestión de bases de datos. A través del uso de Visual Studio Code y MySQL Workbench, se logró integrar correctamente el frontend con el backend, demostrando la importancia de la comunicación entre la interfaz de usuario y la base de datos para el funcionamiento adecuado de una aplicación interactiva.

El desarrollo del juego facilitó la comprensión del consumo de APIs externas, el manejo de funciones asíncronas mediante fetch *y* async/await, así como la manipulación del DOM para actualizar dinámicamente los elementos de la interfaz. Asimismo, el uso de eventos permitió mejorar la interacción con el usuario y reforzó la lógica de programación aplicada en tiempo real.