S.E.P. TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO de Tuxtepec

PROGRAMACION CLIENTE-SERVIDOR

“REPORTE TIPO MEMORIA PokéAPI”

PRESENTA:

RIVERA MARTÍNEZ JESÚS

N. DE CONTROL:

22350405

DOCENTE:

JULIO AGUILAR CARMONA

CARRERA:

INGENIERIA INFORMÁTICA

SEMESTRE Y GRUPO:

7º “A”

DICIEMBRE/2025

Contenido

INTRODUCCIÓN

El presente reporte documenta el desarrollo completo de un juego web interactivo denominado “¿Quién es ese Pokémon?”, el cual combina un backend implementado con Node.js junto a Express, una base de datos MySQL, y un frontend construido con JavaScript, HTML y CSS. El objetivo de la práctica es integrar conceptos fundamentales del desarrollo backend y frontend, manejo de rutas HTTP, consumo de API mediante fetch, manipulación del DOM, almacenamiento de imágenes y selección aleatoria de elementos.

El sistema se compone de dos módulos principales:

- **Backend:** Encargado de exponer rutas para obtener Pokémon, generar selecciones aleatorias y proveer datos estructurados al frontend.

- **Frontend:** Interfaz gráfica del juego que permite al usuario descubrir cuál es el Pokémon visible en silueta y seleccionar la opción correcta entre varias alternativas.

OBJETIVO GENERAL

Desarrollar un juego web funcional que permita reforzar conocimientos sobre consumo de API, estructura de proyectos Node.js, manipulación del DOM y uso de bases de datos para generar dinámicas interactivas.

OBJETIVOS ESPECIFICOS

- Implementar un backend que provea datos de Pokémon mediante diversas rutas HTTP.
- Consumir el backend desde el frontend utilizando fetch().
- Crear una UI interactiva que permita seleccionar una respuesta entre varias opciones.
- Aplicar conceptos de aleatoriedad, manipulación del DOM y actualización dinámica de elementos.
- Integrar imágenes, estilos y eventos para construir una experiencia de usuario fluida.

JUSTIFICACIÓN

Esta práctica es útil porque combina el ciclo completo de desarrollo web: desde la lógica del servidor hasta la presentación visual del cliente. Además, permite reforzar conceptos relacionados con:

- Node.js y Express.
- Buenas prácticas de separación entre backend y frontend.
- Comunicación HTTP.
- Manejo de datos JSON.
- Selección aleatoria y validación de respuestas.

FUNDAMENTO TEORICO

Arquitectura Cliente-Servidor

La arquitectura cliente-servidor es un modelo en el que las responsabilidades de una aplicación se dividen entre dos entidades principales: el cliente y el servidor. El cliente es quien solicita recursos o servicios, mientras que el servidor procesa dichas solicitudes y devuelve una respuesta. Esta separación permite una mejor organización del sistema, así como mayor escalabilidad y facilidad de mantenimiento (Tanenbaum & Van Steen, 2017).

En aplicaciones web modernas, esta arquitectura es fundamental, ya que permite que el frontend y el backend evolucionen de manera independiente, comunicándose a través de protocolos estándar como HTTP (Arsys, 2023).

JavaScript como Lenguaje de Programación

JavaScript es un lenguaje de programación interpretado, orientado a eventos y multiparadigma, ampliamente utilizado en el desarrollo web. Inicialmente fue diseñado para ejecutarse en el navegador, permitiendo crear interfaces dinámicas e interactivas, pero con el paso del tiempo su uso se ha extendido al lado del servidor (Flanagan, 2020).

Gracias a su flexibilidad y popularidad, JavaScript se ha convertido en un lenguaje clave tanto para el desarrollo frontend como backend, lo que permite utilizar un mismo lenguaje en todo el sistema (MDN Web Docs, 2024).

Node.js

Node.js es un entorno de ejecución que permite ejecutar JavaScript del lado del servidor. Está basado en el motor V8 de Google Chrome y se caracteriza por su modelo de ejecución asíncrono y orientado a eventos, lo que lo hace eficiente para

aplicaciones que manejan múltiples solicitudes simultáneas (Node.js Foundation, 2023).

Node.js es ampliamente utilizado para la creación de APIs y servicios web debido a su alto rendimiento, escalabilidad y facilidad de integración con otras tecnologías (Tilkov & Vinoski, 2010).

Express.js

Express.js es un framework para Node.js que facilita la creación de aplicaciones web y APIs REST. Proporciona un conjunto de herramientas para el manejo de rutas, solicitudes HTTP, middlewares y respuestas del servidor, simplificando el desarrollo del backend (Express.js, 2023).

El uso de Express permite estructurar el código de forma modular, separando rutas, controladores y configuraciones, lo que mejora la legibilidad y el mantenimiento del proyecto (MDN Web Docs, 2024).

APIs REST

Una API REST (Representational State Transfer) es un estilo arquitectónico que define un conjunto de restricciones para el diseño de servicios web. Las APIs REST utilizan los métodos HTTP estándar como GET, POST, PUT y DELETE para operar sobre recursos identificados mediante URLs (Fielding, 2000).

Este tipo de API es ampliamente adoptado debido a su simplicidad, independencia de plataforma y facilidad de consumo desde aplicaciones frontend, como las desarrolladas con JavaScript (AWS, 2023).

Comunicación HTTP y Formato JSON

HTTP es el protocolo base para la comunicación entre clientes y servidores en la web. Permite la transferencia de información mediante solicitudes y respuestas,

acompañadas de códigos de estado que indican el resultado de la operación (Kurose & Ross, 2021).

Por su parte, JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos que se utiliza comúnmente en APIs REST. Su estructura basada en pares clave-valor facilita la lectura y manipulación de la información, especialmente en aplicaciones desarrolladas con JavaScript (ECMA International, 2017).

Bases de Datos Relacionales (MySQL)

MySQL es un sistema de gestión de bases de datos relacional que utiliza el lenguaje SQL para la definición y manipulación de datos. Las bases de datos relacionales organizan la información en tablas relacionadas entre sí, garantizando la integridad y consistencia de los datos (Silberschatz, Korth & Sudarshan, 2020).

MySQL es ampliamente utilizado en aplicaciones web debido a su rendimiento, confiabilidad y compatibilidad con múltiples lenguajes de programación, incluyendo Node.js (Oracle, 2023).

Consumo de APIs desde el Frontend

El consumo de APIs en aplicaciones web permite obtener y enviar datos entre el frontend y el backend de manera dinámica. JavaScript ofrece mecanismos como la Fetch API para realizar peticiones HTTP asíncronas y procesar las respuestas en formato JSON (MDN Web Docs, 2024).

Este enfoque mejora la experiencia del usuario, ya que permite actualizar el contenido de la aplicación sin necesidad de recargar la página completa (Flanagan, 2020).

Aplicaciones Web Interactivas

Las aplicaciones web interactivas combinan frontend, backend y bases de datos para ofrecer experiencias dinámicas al usuario. La integración de tecnologías como JavaScript, Node.js, Express y bases de datos relacionales permite desarrollar sistemas completos que responden en tiempo real a las acciones del usuario (Red Hat, 2022).

Este tipo de aplicaciones es ideal para proyectos educativos, ya que permite aplicar conceptos de programación, bases de datos y arquitectura de software en un solo desarrollo.

DESARROLLO DE LA PRACTICA

TECNOLOGIAS UTILIZADAS

Para implementar el juego “¿Quién es ese Pokémon?”, se utilizaron las siguientes tecnologías:

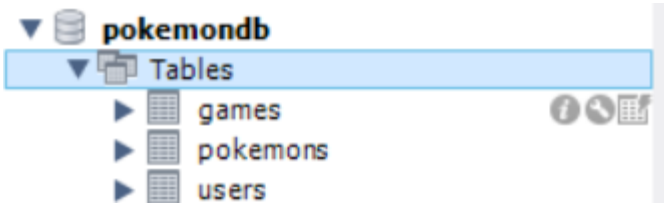
Backend

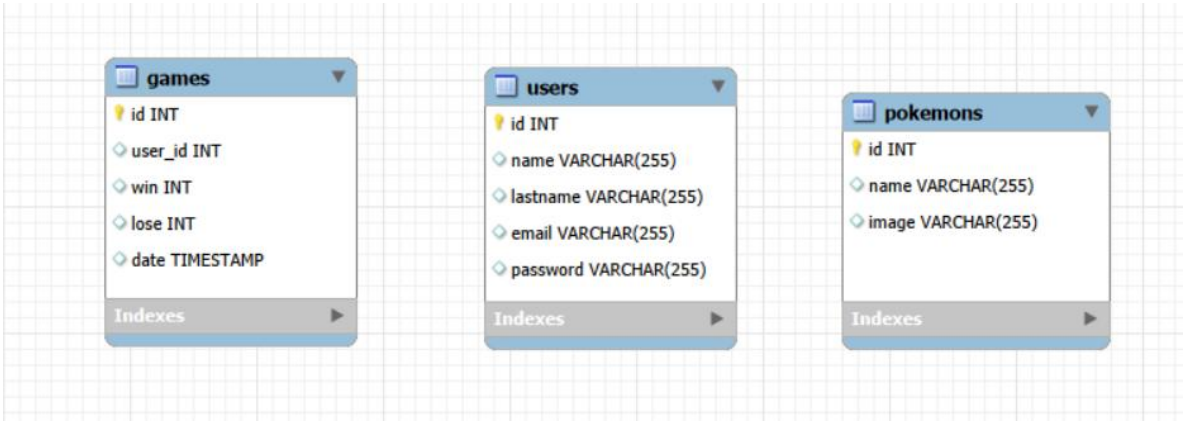
- **Node.js**: Entorno de ejecución para JavaScript del lado del servidor. Permite manejar rutas, peticiones HTTP y lógica del backend.
- **Express.js**: Framework minimalista para crear APIs REST de forma sencilla.
- **MySQL**: Sistema gestor de bases de datos relacional utilizado para almacenar información de Pokémon.
- **dotenv**: Carga de variables de entorno para parámetros de conexión.
- **Cors**: Permitir llamadas desde el frontend alojado en un servidor distinto.

Frontend

- **HTML5**: Estructura del documento y contenedores visuales del juego.
- **CSS3**: Estilos visuales del juego, animaciones y diseño responsivo.
- **JavaScript (Vanilla JS)**: Lógica del juego en el navegador, consumo del backend y manipulación del DOM.
- **Fetch API**: Peticiones HTTP desde el cliente.

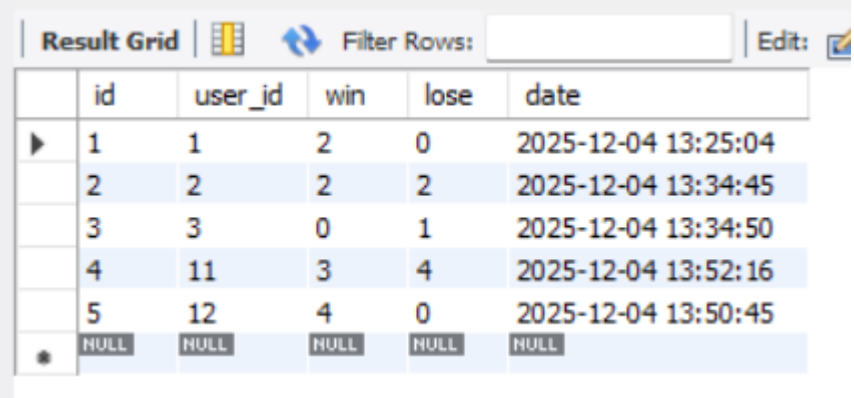
DISEÑO DE LA BASE DE DATOS





CONTENIDO DE LA BASE DE DATOS

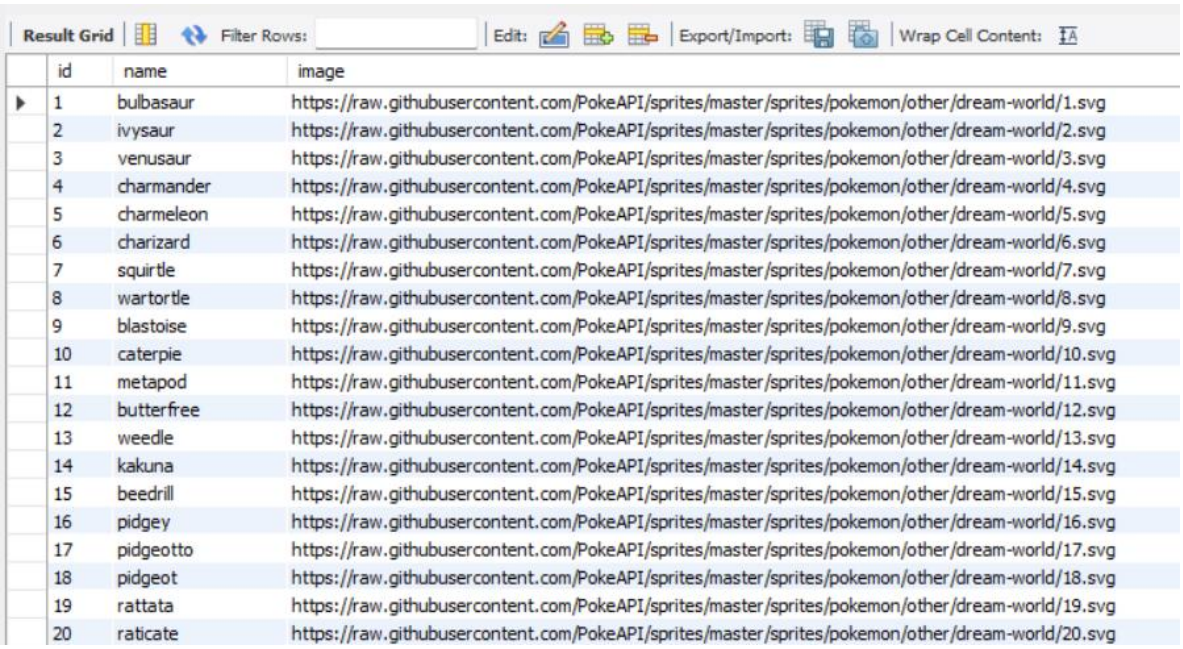
GAMES



Result Grid | Filter Rows: | Edit:

id	user_id	win	lose	date
1	1	2	0	2025-12-04 13:25:04
2	2	2	2	2025-12-04 13:34:45
3	3	0	1	2025-12-04 13:34:50
4	11	3	4	2025-12-04 13:52:16
5	12	4	0	2025-12-04 13:50:45
NULL	NULL	NULL	NULL	NULL

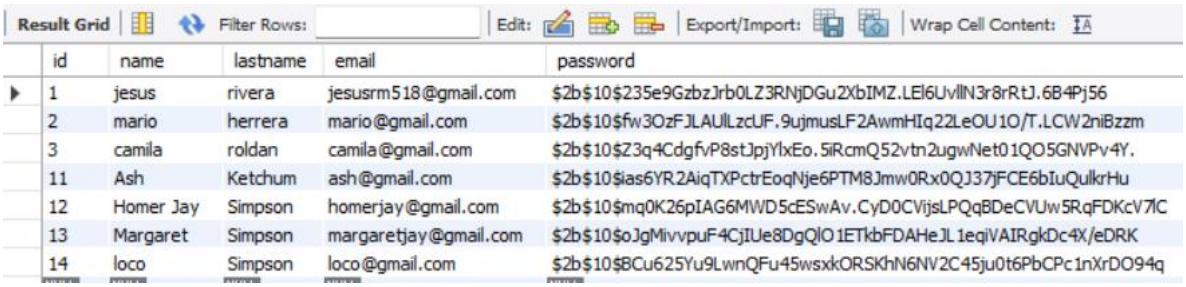
POKEMONS



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	image
1	bulbasaur	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/1.svg
2	ivysaur	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/2.svg
3	venusaur	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/3.svg
4	charmander	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/4.svg
5	charmeleon	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/5.svg
6	charizard	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/6.svg
7	squirtle	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/7.svg
8	wartortle	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/8.svg
9	blastoise	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/9.svg
10	caterpie	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/10.svg
11	metapod	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/11.svg
12	butterfree	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/12.svg
13	weedle	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/13.svg
14	kakuna	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/14.svg
15	beedrill	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/15.svg
16	pidgey	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/16.svg
17	pidgeotto	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/17.svg
18	pidgeot	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/18.svg
19	rattata	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/19.svg
20	raticate	https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/dream-world/20.svg

USERS

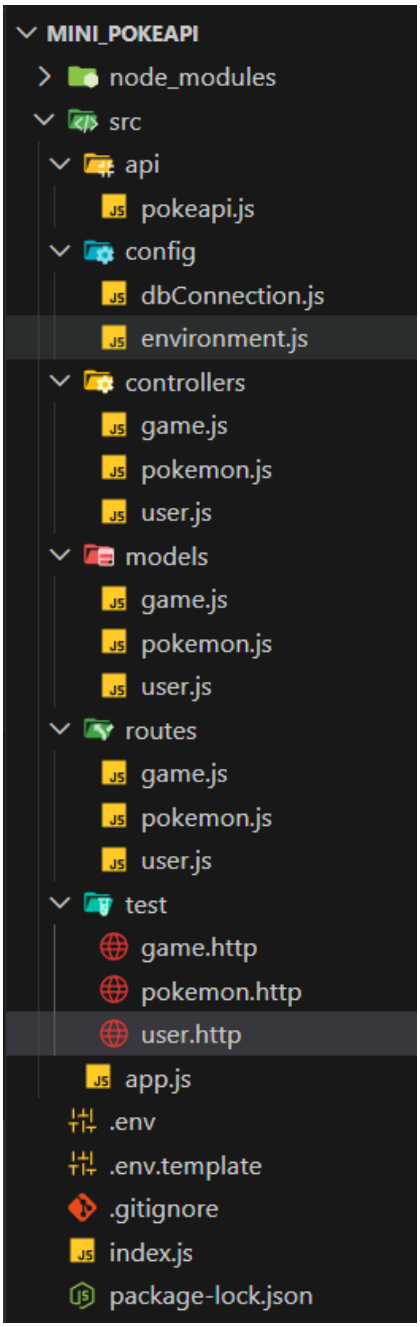


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	lastname	email	password
1	jesus	rivera	jesusrm518@gmail.com	$2b$10$235e9GzbzJrb0LZ3RNjDGu2XbIMZ.LEl6UvllN3r8rRtJ.6B4Pj56
2	mario	herrera	mario@gmail.com	$2b$10$fw3OzFJLAUlLzcUF.9ujmusLF2AwmHIq22LeOU1O/T.LCW2niBzzm
3	camila	roldan	camila@gmail.com	$2b$10$Z3q4CdgfvP8stJpjYlxEo.5iRcmQ52vtn2ugwNet01QO5GNVPv4Y.
11	Ash	Ketchum	ash@gmail.com	$2b$10$ias6YR2AiqTXPctrEoqNje6PTM8Jmw0Rx0QJ37jFCE6bIuQulkrHu
12	Homer Jay	Simpson	homerjay@gmail.com	$2b$10$mq0K26pIAG6MWD5cESwAv.CyD0CVijsLPQqBDeCVUw5RqFDKcV7lC
13	Margaret	Simpson	margaretjay@gmail.com	$2b$10$oJgMivvpuF4CjIUe8DgQlO1ETkbFDAHeJL1eqiVAIRgkDc4X/eDRK
14	loco	Simpson	loco@gmail.com	$2b$10$BCu625Yu9LwnQFu45wsxkORSKhN6NV2C45ju0t6PbCPc1nXrDO94q

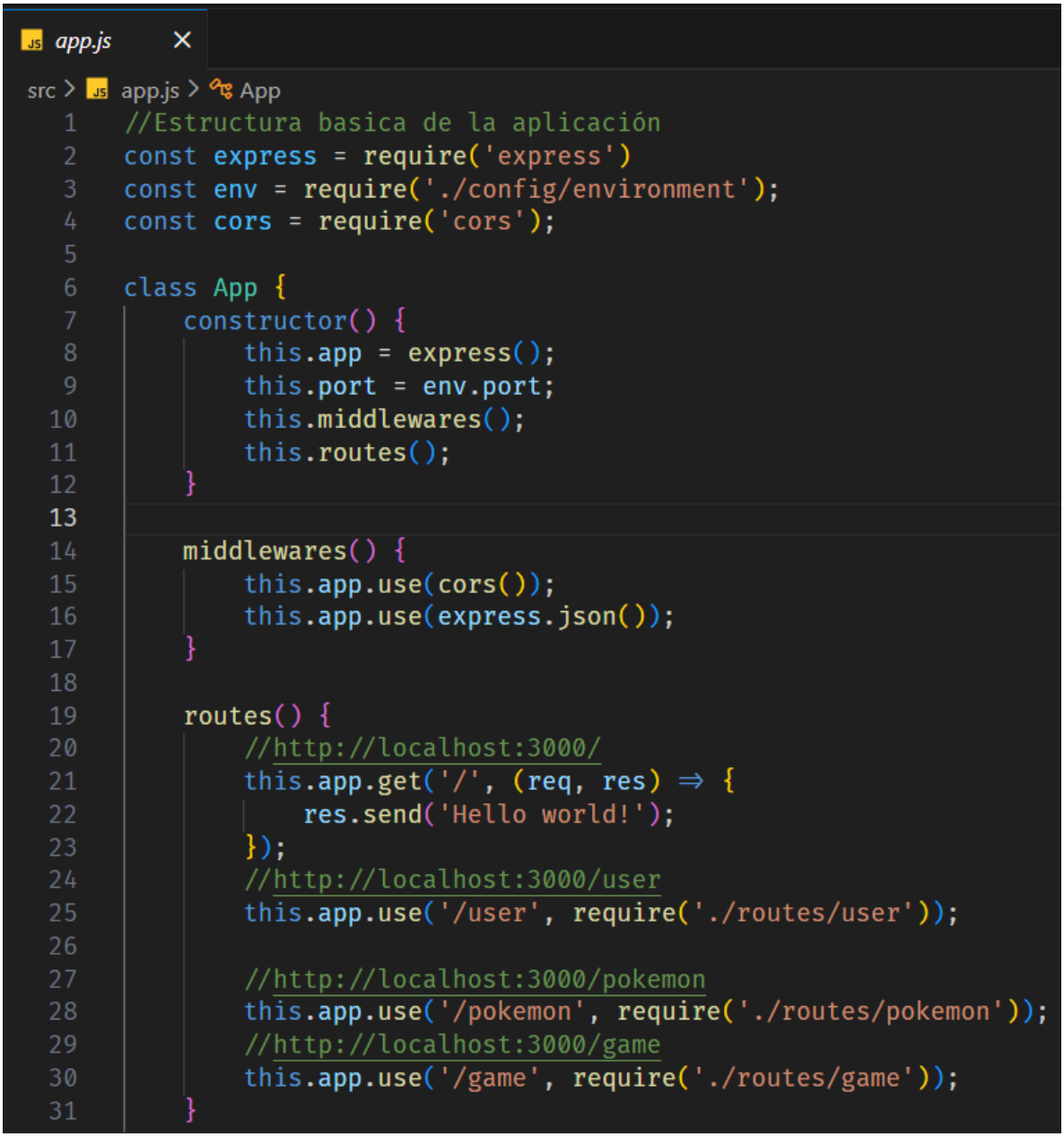
ESTRUCTURA DEL PROYECTO

BACKEND (NODE.JS + EXPRESS)



1. src/app.js

Es la clase principal de la aplicación. Configura Express, registra middlewares y monta las rutas.



```
app.js
src > app.js > App
1   //Estructura basica de la aplicación
2   const express = require('express')
3   const env = require('./config/environment');
4   const cors = require('cors');
5
6   class App {
7       constructor() {
8           this.app = express();
9           this.port = env.port;
10          this.middlewares();
11          this.routes();
12      }
13
14      middlewares() {
15          this.app.use(cors());
16          this.app.use(express.json());
17      }
18
19      routes() {
20          //http://localhost:3000/
21          this.app.get('/', (req, res) => {
22              res.send('Hello world!');
23          });
24          //http://localhost:3000/user
25          this.app.use('/user', require('./routes/user'));
26
27          //http://localhost:3000/pokemon
28          this.app.use('/pokemon', require('./routes/pokemon'));
29          //http://localhost:3000/game
30          this.app.use('/game', require('./routes/game'));
31      }
```

- Se creó una clase App que encapsula la configuración del servidor.
- En el constructor se instancia Express, se define el puerto desde variables de entorno y se inicializan middlewares y rutas.
- En middlewares() se habilitó CORS y el parsing de JSON.

- En routes() se registraron los módulos de rutas:
    - /user
    - /pokemon
    - /game
- También se agregó una ruta raíz (/) que responde “Hello world”.
- start() inicia el servidor escuchando en el puerto correspondiente.

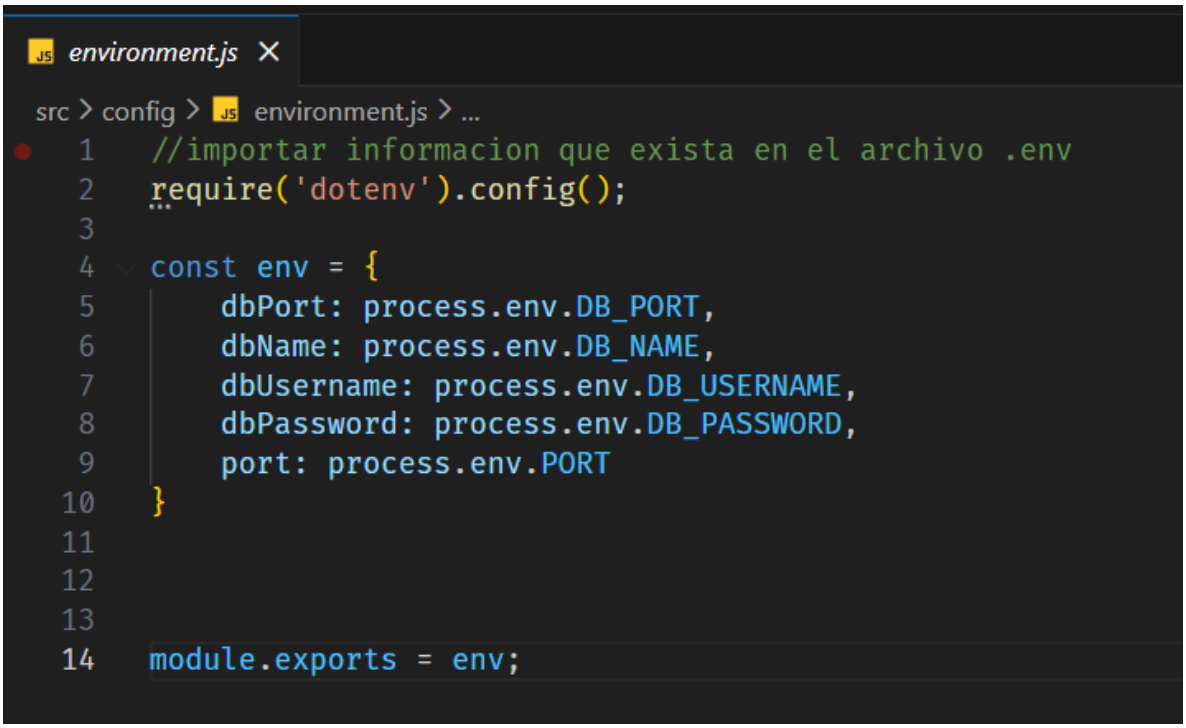
2. src/index.js

Archivo ejecutable principal.

- Se importó la clase App.
- Se creó una instancia y se invocó app.start() para poner el servidor en marcha.

3. src/config/environment.js

Gestionar variables del archivo .env.



- Se usó dotenv.config() para cargar las variables de entorno.
- Se expuso un objeto env con:
    - Puerto del servidor
    - Datos de conexión a base de datos (nombre, usuario, password, puerto)

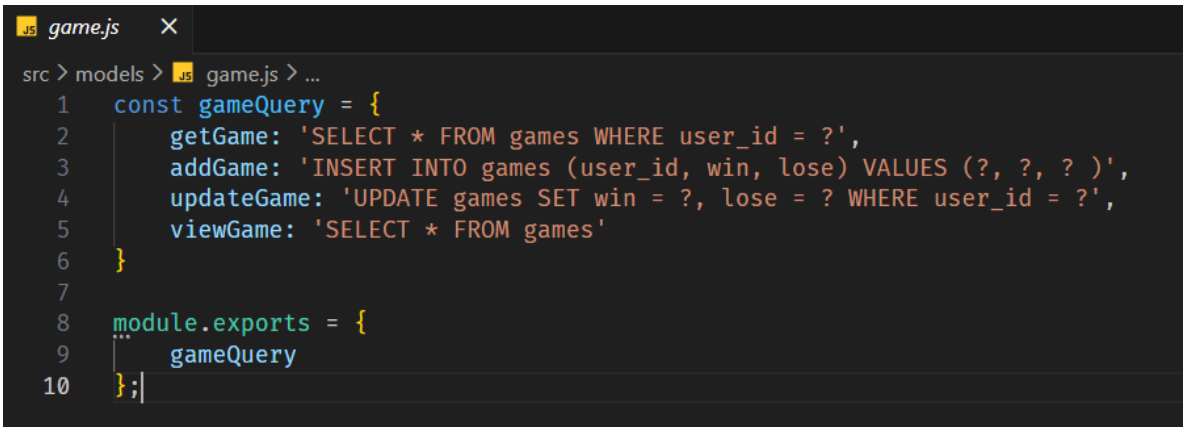
4. src/database/dbConnection.js

Crear un pool de conexiones a MariaDB/MySQL.

- Se importó mariadb.
- Se leyó la configuración desde environment.js.
- Se creó un pool con createPool().
- Se exportó el pool para que los controladores puedan ejecutar consultas SQL.

5. src/models/game.js

Contiene las consultas SQL relacionadas al juego.



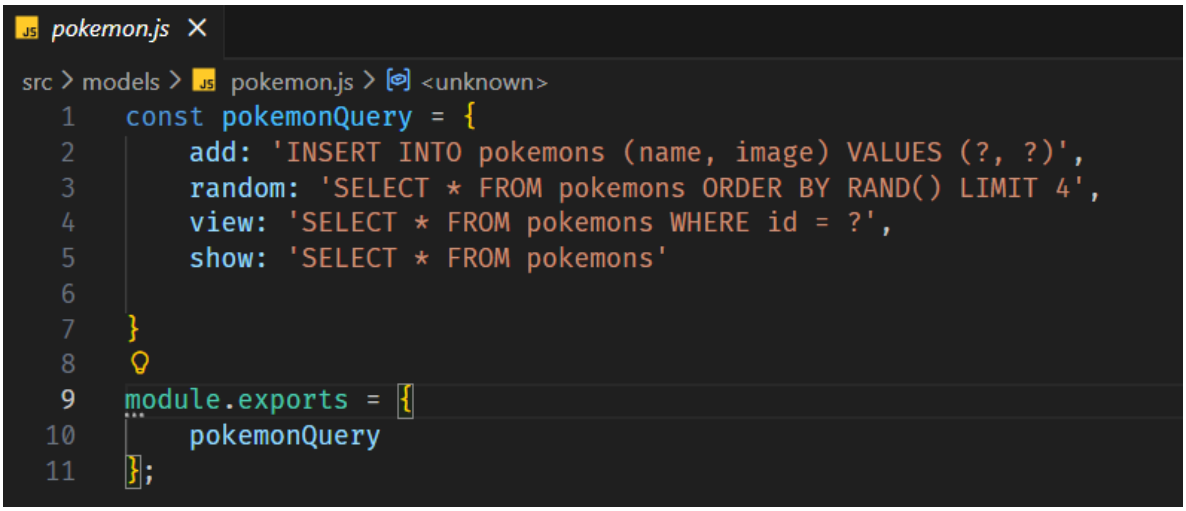
Se definieron queries para gestionar estadísticas del juego:

- getGame – obtener datos del usuario.
- addGame – insertar registro.
- updateGame – actualizar win/lose.
- viewGame – consultar todos los registros.

- Se exportó un objeto gameQuery.

6. src/models/pokemon.js

Consultas SQL para la tabla de Pokémon.



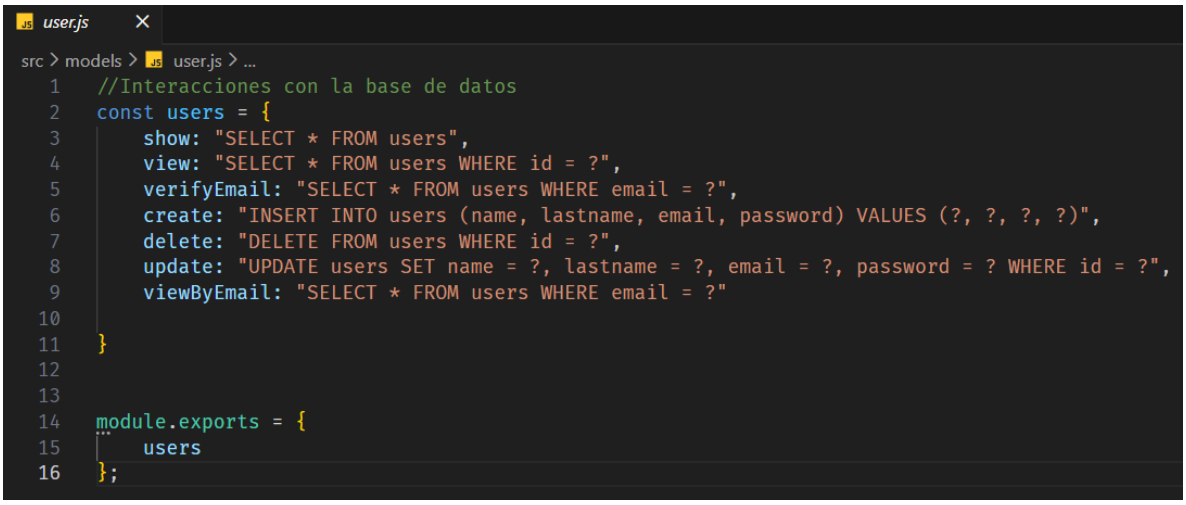
Se definieron queries:

- add – insertar Pokémon.
- random – obtener 4 Pokémon aleatorios.
- view – consultar Pokémon por id.
- show – listar todos.

- Se exportó pokemonQuery.

7. src/models/user.js

Consultas SQL relacionadas con usuarios.



Se definieron operaciones:

- Ver todos los usuarios.
- Ver por id.
- Verificar email.
- Crear usuario.
- Eliminar usuario.
- Actualizar usuario.
- Obtener usuario por email.

- Se exportó users.

8. Rutas del juego – src/routes/game.js

Definir endpoints relacionados a estadísticas del juego.

- Se creó un router con Express.
- Se definieron:
    - GET /win/:id
    - GET /lose/:id
    - GET /view
- Cada ruta llama a un método de su controlador correspondiente.

9. Rutas de Pokémon – src/routes/pokemon.js

Manejar peticiones sobre Pokémon.

- Se definieron endpoints:
    - GET /seed – llenar la BD desde pokeapi.
    - GET /random – obtener 4 Pokémon aleatorios.
    - GET /:id – consultar Pokémon por id.
    - GET / – mostrar todos.
- Se enlazaron a métodos del controlador pokemon.

10. Rutas de usuarios – src/routes/user.js

Controlar acciones sobre los usuarios.

user.js

src > routes > user.js > ...

```
1   //Rutas de los Endpoints
2   const {Router} = require('express');
3
4   const router = Router();
5   //http:localhost:3000/user/
6   router.get('/', require('../controllers/user').showUsers);
7   //http:localhost:3000/user/1
8   router.get('/:id', require('../controllers/user').viewUser);
9   router.post('/', require('../controllers/user').createUser);
10  router.delete('/:id', require('../controllers/user').removeUser);
11  router.put('/:id', require('../controllers/user').updateUser);
12  router.post('/login', require('../controllers/user').loginUser);
13
14  module.exports = router;
```

- Se definieron:
    - GET /
    - GET /:id
    - POST /
    - DELETE /:id
    - PUT /:id
    - POST /login
- Todas las rutas delegan la lógica a controllers/user.

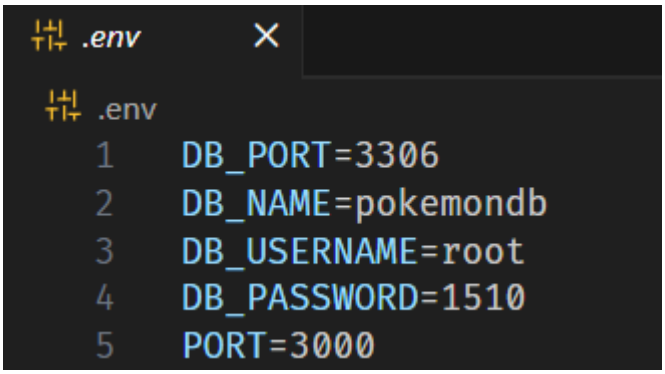
11. pokeapi.js

Obtener los 151 Pokémon iniciales desde PokeAPI.

- Se creó la función getPokemons() que:
    - Llama a la API pública con fetch(...).
    - Obtiene el array results.
    - Mapea cada Pokémon para agregar campo image con URL de DreamWorld.
- Se exportó la función.

12. .env

Configurar datos sensibles.



- Se almacenaron:
    - Puerto de MySQL
    - Usuario y contraseña
    - Nombre de la BD
    - Puerto del backend

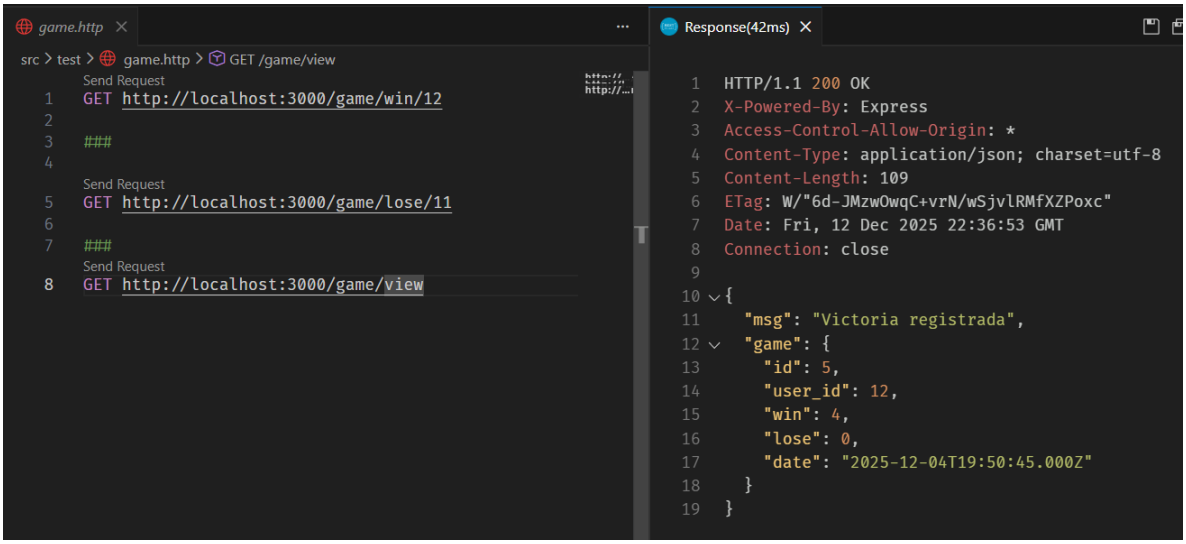
13. test

La carpeta **test** se utilizó para **probar manualmente los endpoints del backend** durante el desarrollo de la práctica, sin necesidad de un frontend o herramientas externas como Postman.

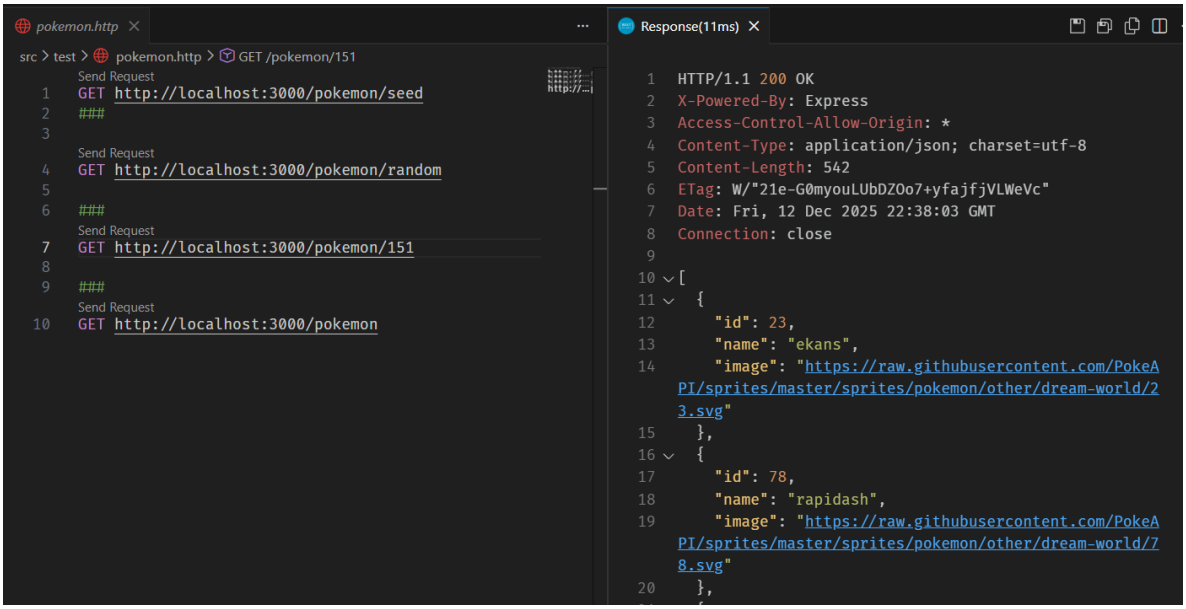
Los archivos .http permiten:

- Ejecutar peticiones HTTP directamente desde Visual Studio Code.
- Verificar el correcto funcionamiento de cada endpoint.
- Comprobar respuestas, códigos de estado y flujo de datos.
- Facilitar el proceso de depuración durante el desarrollo.

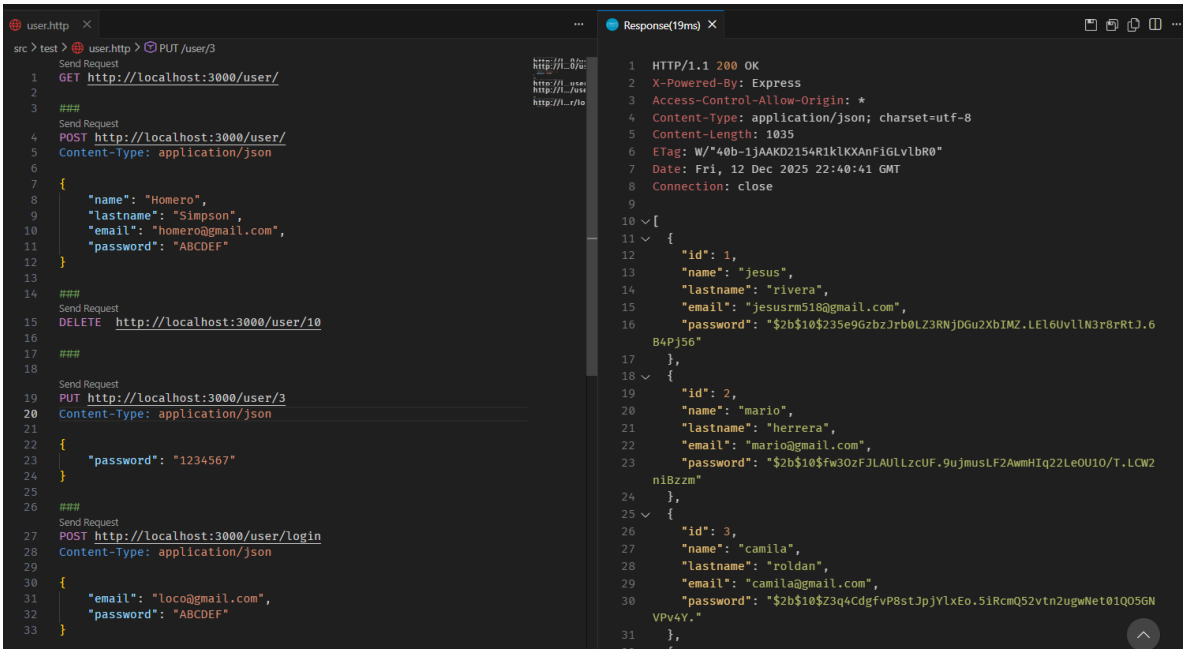
Game.http



Pokemon.http



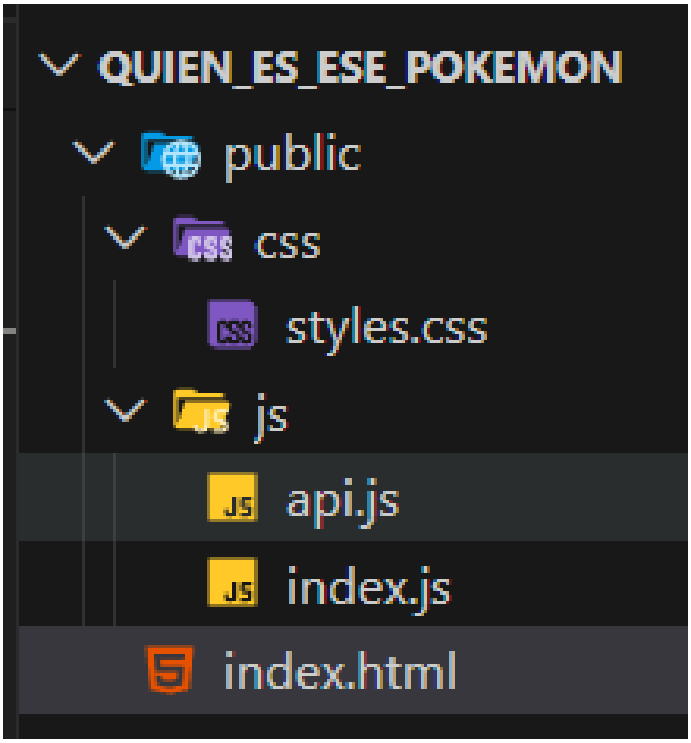
User.http



```
user.http
src > test > user.http > PUT /user/3
Send Request
GET http://localhost:3000/user/

###
Send Request
POST http://localhost:3000/user/
Content-Type: application/json

{
    "name": "Homero",
    "lastname": "Simpson",
    "email": "homero@gmail.com",
    "password": "ABCDEF"
}

###
Send Request
DELETE  http://localhost:3000/user/10

###

Send Request
PUT http://localhost:3000/user/3
Content-Type: application/json

{
    "password": "1234567"
}

###
Send Request
POST http://localhost:3000/user/login
Content-Type: application/json

{
    "email": "loco@gmail.com",
    "password": "ABCDEF"
}
```

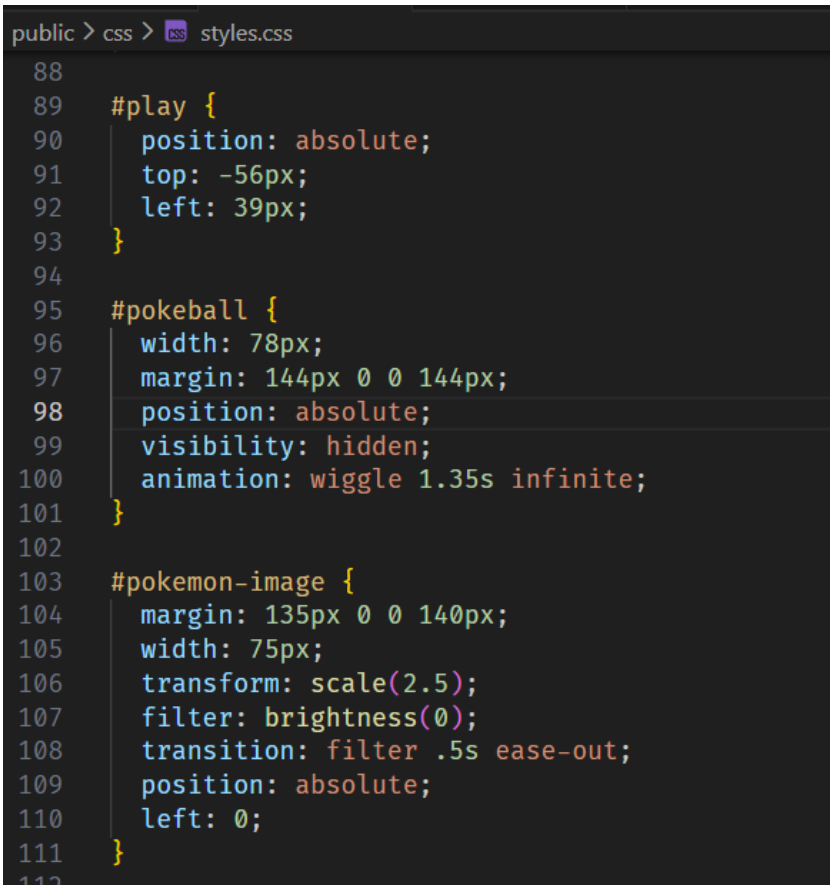
```
Response(19ms)
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 1035
ETag: W/"40b-1jAAKD2154R1klKXAnFiGLvlbR0"
Date: Fri, 12 Dec 2025 22:40:41 GMT
Connection: close

[
  {
    "id": 1,
    "name": "jesus",
    "lastname": "rivera",
    "email": "jesusrm518@gmail.com",
    "password": "$2b$10$235e9GzbzJrb0LZ3RNjDGu2XbIMZ.LEl6UvllN3r8rRtJ.6B4Pj56"
  },
  {
    "id": 2,
    "name": "mario",
    "lastname": "herrera",
    "email": "mario@gmail.com",
    "password": "$2b$10$fw3OzFJLAUlLzcUF.9ujmusLF2AwmHIq22LeOU1O/T.LCW2niBzzm"
  },
  {
    "id": 3,
    "name": "camila",
    "lastname": "roldan",
    "email": "camila@gmail.com",
    "password": "$2b$10$Z3q4CdgfvP8stJpjYlxEo.5iRcmQ52vtn2ugwNet01QO5GNVPv4Y."
  },
```

FRONTEND


QUIEN_ES_ESE_POKEMON
public
css
styles.css
js
api.js
index.js
index.html

1. public/css/styles.css

Diseñar la interfaz gráfica del juego.



- Fondo oscuro general.
- Imagen de fondo del juego dentro de <main>.
- Estilo y animaciones del pokeball.
- Efecto de silueta aplicando filter: brightness(0).
- Posicionamiento absoluto de imágenes y contenedores.
- Botones personalizados con hover, estados correct/incorrect.
- Animación @keyframes wiggle aplicada al pokeball.
- Diseño del texto emergente al revelar el Pokémon.

2. public/js/api.js

Encapsular la petición HTTP al backend.

```
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];

    return {
        pokemonChoices: pokemon,
        correctAnswer,
    }
};
```

- Se creó la función getPokemon(), que:
    - Llama a GET http://localhost:3000/pokemon/random.
    - Devuelve los 4 Pokémon enviados por el backend.
- Se definió window.getPokeData() que:
    - Obtiene los Pokémon.
    - Selecciona uno aleatorio como respuesta correcta.
    - Retorna:
        - pokemonChoices
        - correctAnswer

3. public/js/index.js

Lógica completa del juego en el navegador.

```
public > js > index.js > revealPokemon
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 showSilouette = () => {
  main.classList.remove('fetching');
  pokemonImage.src = gameData.correctAnswer.image;
  console.log(gameData.correctAnswer);
};

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

const resetImage = () => {
  pokemonImage.src = 'https://res.cloudinary.com/dzynqn10l/image/upload/v1633196610/Msc/pokeball_zeoqii.webp';
  main.classList.remove('revealed');
}

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

- Se capturaron elementos del DOM (playBtn, choices, pokemonImage, etc.).
- Se creó la variable global gameData.
- **Funciones implementadas:**
    - revealPokemon() – Muestra la imagen completa y el nombre correcto.
    - showSilouette() – Coloca la imagen real pero oscurecida.
    - displayChoices() – Genera los botones de opciones.
    - resetImage() – Regresa a la pokebola antes de cargar datos.
    - fetchData() – Llama a getPokeData(), muestra silueta y opciones.
- **Eventos:**
    - playBtn.addEventListener('click', fetchData);
        - Inicia nueva ronda.
    - choices.addEventListener('click', ...)
        - Determina si el usuario acertó o falló.
        - Colorea el botón con la clase correcta/incorrect.
        - Revela el Pokémon.

4. index.html

Estructura principal del juego.

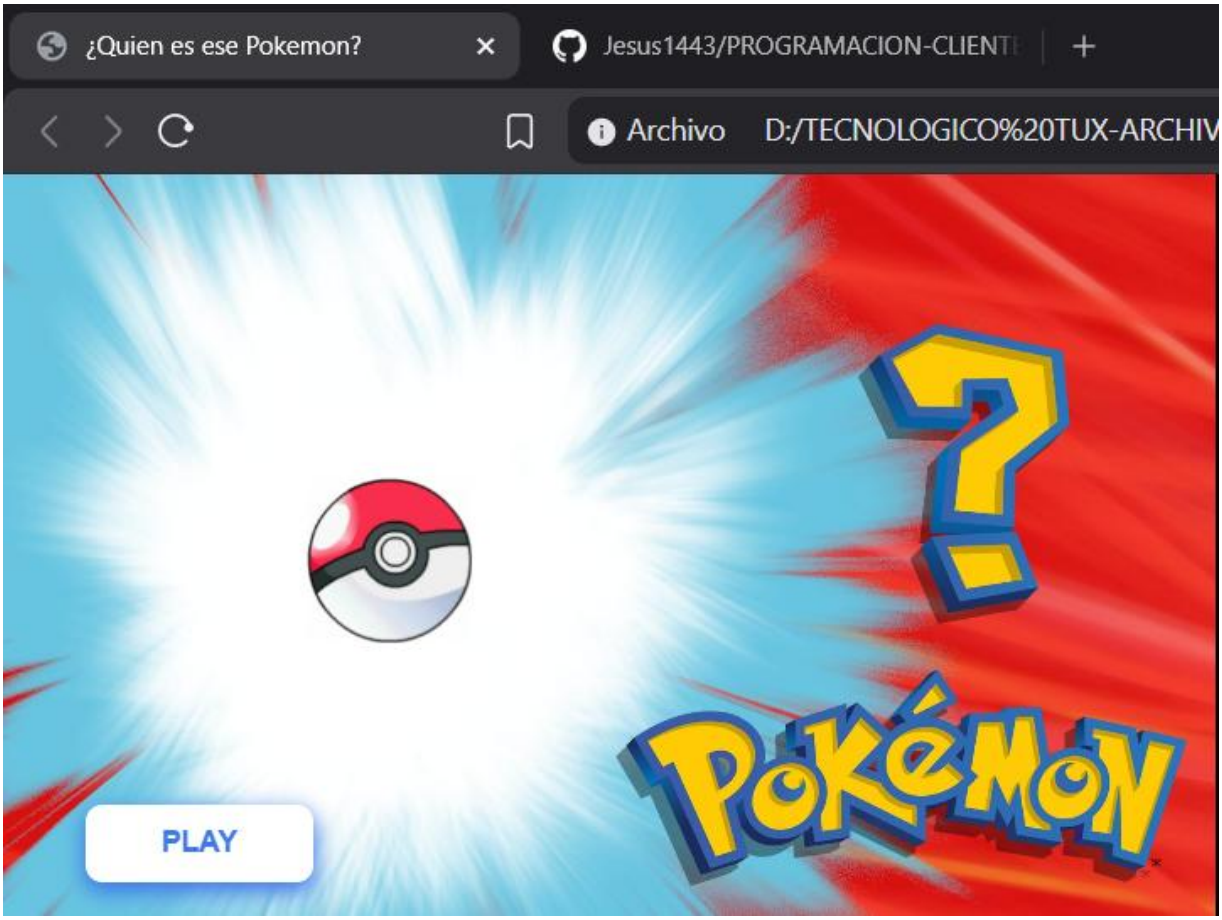
```
index.html > html > body > script
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="./public/css/styles.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Bangers&display=swap">
    <title>¿Quien es ese 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">Play</button>
            <div id="choices"></div>
        </section>
    </main>

    <script src="./public/js/api.js"></script>
    <script src="./public/js/index.js"></script>
</body>
</html>
```

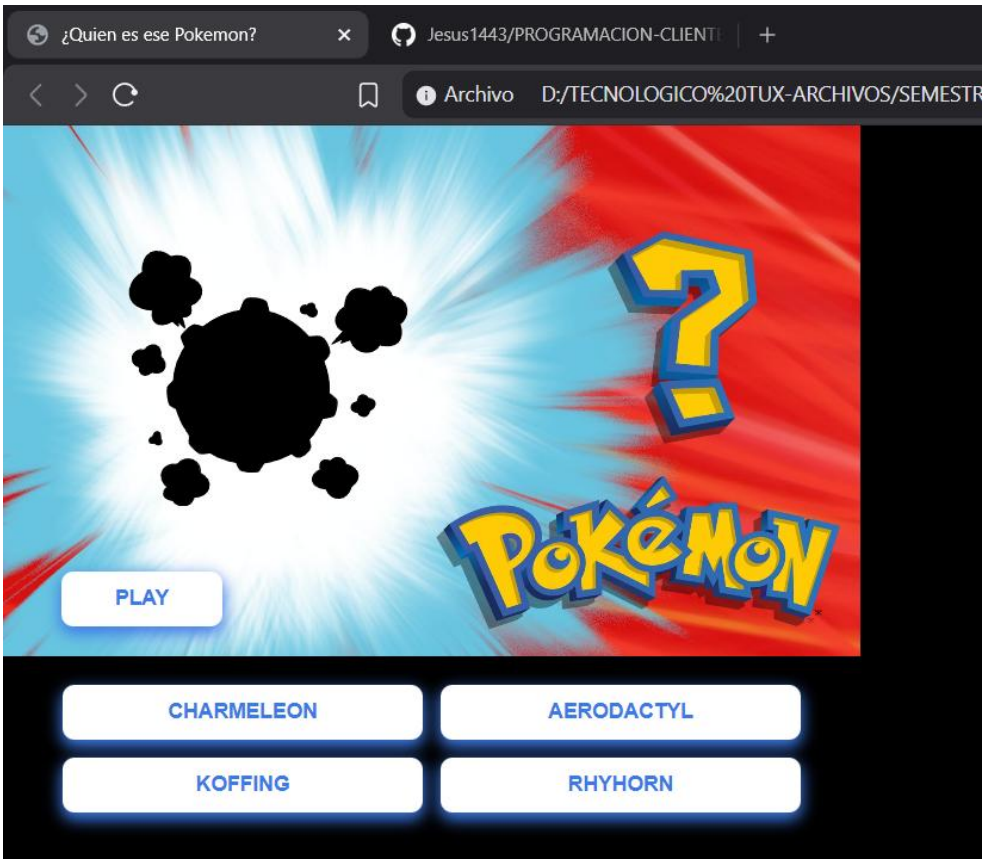
Se creó un main con estado inicial .fetching.

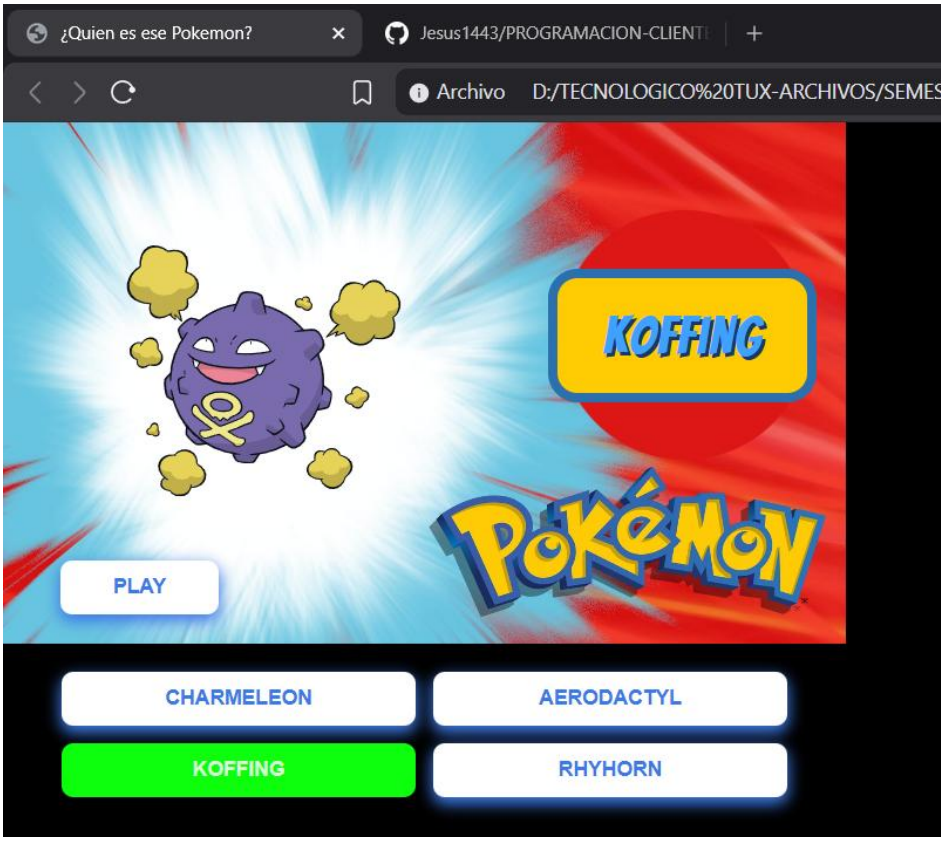
- Se incluyeron:
    - Imagen del pokeball animado.
    - Imagen del Pokémon que inicia como silueta.
    - Contenedores:
        - #pokemon-container
        - #answer
        - #controls
- Se agregó el botón **Play**.
- Se agregó el contenedor dinámico #choices.
- Se enlazaron los scripts:
    - api.js
    - index.js
- Se aplicaron las fuentes de Google.

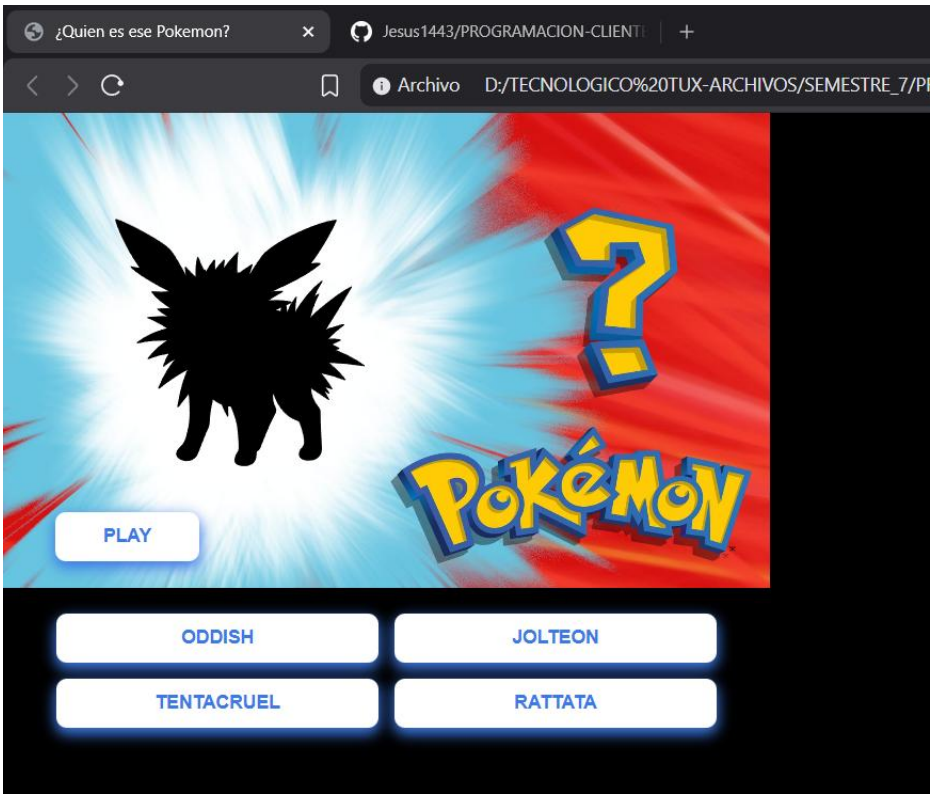
RESULTADOS

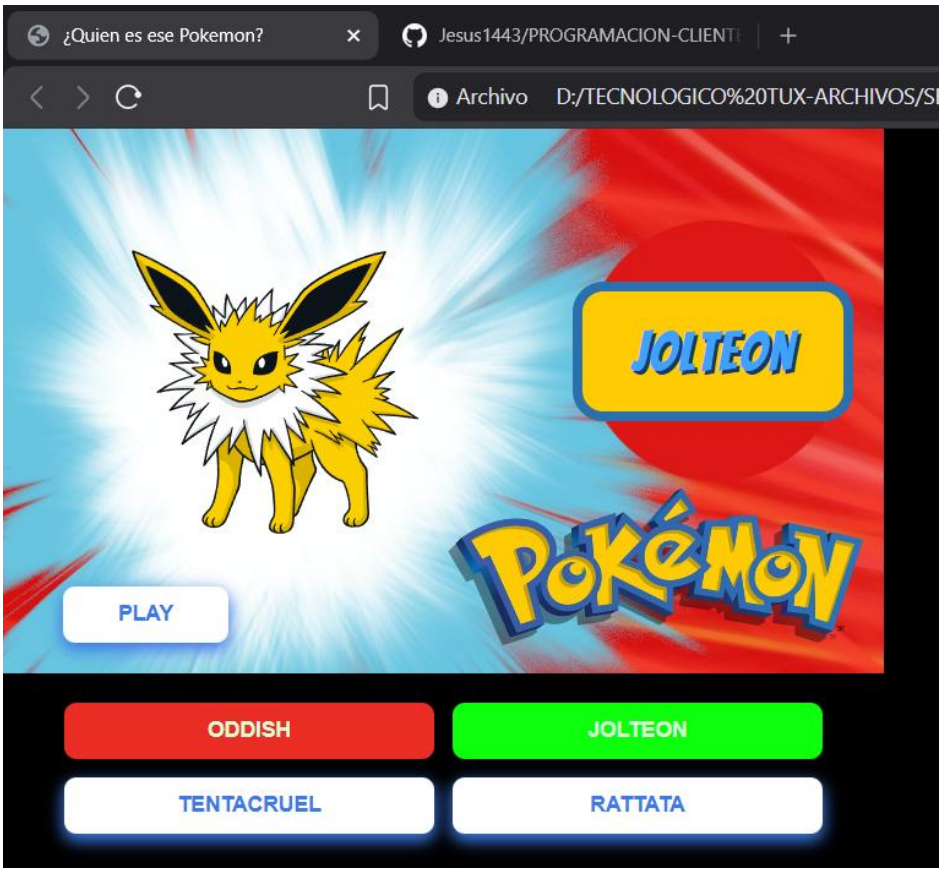
Al final se logró que el juego ¿Quién es ese Pokémon? Funcionará, primero muestra una pokebola que se mueve pero que al darle play muestra la silueta del Pokémon y debajo aparecen 4 opciones de las cuales se tiene que seleccionar la correcta, si es correcta el botón cambia de color a verde y muestra el Pokémon con su nombre pero si es incorrecta el botón se vuelve color rojo.




¿Quien es ese Pokemon?
Jesus1443/PROGRAMACION-CLIENT
Archivo
D:/TECNOLOGICO%20TUX-ARCHIVOS/SEMESTR
?
PLAY
CHARMELEON
AERODACTYL
KOFFING
RHYHORN


¿Quien es ese Pokemon?
Jesus1443/PROGRAMACION-CLIENT
Archivo
D:/TECNOLOGICO%20TUX-ARCHIVOS/SEMES
KOFFING
PLAY
CHARMELEON
AERODACTYL
KOFFING
RHYHORN


¿Quien es ese Pokemon?
Jesus1443/PROGRAMACION-CLIENT
Archivo
D:/TECNOLOGICO%20TUX-ARCHIVOS/SEMESTRE_7/P
?
Pokémon
PLAY
ODDISH
JOLTEON
TENTACRUEL
RATTATA


¿Quien es ese Pokemon?
Jesus1443/PROGRAMACION-CLIENT
Archivo
D:/TECNOLOGICO%20TUX-ARCHIVOS/SI
JOLTEON
Pokémon
PLAY
ODDISH
JOLTEON
TENTACRUEL
RATTATA

CONCLUSIÓN

Al finalizar la PokéAPI empecé a entender mejor el manejo de las APIs, rutas, endpoints, conexión a la base de datos y diseño del frontend. Conocer nuevas tecnologías como Node, express y JavaScript ampliaron mi conocimiento sobre herramientas para el desarrollo web, hubo algunas complicaciones en el desarrollo de la practica como rutas mal definidas, fallos en la conexión de la base de datos, importaciones incorrectas, errores sintácticos y al ejecutar acciones para los botones, pero fueron resueltos gracias al apoyo del docente. Quizá no se pudo ver todo lo propuesto al inicio del semestre, pero al menos se pudo llevar a cabo lo más importante (la PokéAPI), en lo personal el conocimiento adquirido me será de utilidad para futuros proyectos durante la carrera y ámbito laboral.

REFERENCIAS

Amazon Web Services. (2023). *What is a RESTful API?* https://aws.amazon.com/what-is/restful-api/

Arsys. (2023). *Arquitectura cliente-servidor: qué es y cómo funciona*. https://www.arsys.es/blog/arquitectura-cliente-servidor

ECMA International. (2017). *The JSON data interchange syntax (ECMA-404)*. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

Express.js. (2023). *Express – Node.js web application framework*. https://expressjs.com/

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California).

Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.

Kurose, J. F., & Ross, K. W. (2021). *Computer networking: A top-down approach* (8th ed.). Pearson.

MDN Web Docs. (2024). *JavaScript guide*. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

Node.js Foundation. (2023). *About Node.js*. https://nodejs.org/en/about

Oracle. (2023). *MySQL documentation*. https://dev.mysql.com/doc/

Red Hat. (2022). *What is an API?* https://www.redhat.com/en/topics/api

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts* (7th ed.). McGraw-Hill.

Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed systems: Principles and paradigms* (2nd ed.). Pearson.