



INSTITUTO TECNOLÓGICO DE TUXTEPEC
Departamento de Sistemas y Computación
Formato para prácticas de Laboratorio

Carrera	Plan de estudios	Clave de la materia	Nombre de la materia	Semestre	Gpo.	Periodo
Ingeniería Informática	IINF-2010-220	7F5 A	Programación en ambiente cliente servidor	7	A	Ago/Dic 2025

Practica No.	Laboratorio de:	Nombre de la práctica	Duración (Hora)
1	SC-07	mini_pokeapi y ¿Quién es el Pokémon?	1

1. Enunciado

- Desarrollar una mini API REST denominada *mini_pokeapi* y una aplicación cliente llamada *¿Quién es el Pokémon?*, aplicando el modelo cliente–servidor, con el fin de consumir servicios web y realizar operaciones CRUD utilizando tecnologías del lado del servidor y del cliente.

2. Introducción

El modelo cliente–servidor es fundamental en el desarrollo de aplicaciones modernas, ya que permite separar la lógica del negocio del acceso a los datos. Las API REST facilitan esta comunicación mediante el uso del protocolo HTTP y el intercambio de información en formato JSON.

En esta práctica se integran dos desarrollos: una *mini_pokeapi*, encargada de la gestión de datos de Pokémon, y una aplicación cliente denominada *¿Quién es el Pokémon?*, la cual consume dicha API para mostrar información de manera dinámica, reforzando así el aprendizaje práctico de servicios web.

3. Objetivo (Competencia)

Implementar un sistema cliente–servidor mediante el desarrollo de una API REST y una aplicación cliente que consuma sus servicios, fortaleciendo las competencias en el manejo de operaciones CRUD, consumo de APIs y arquitectura cliente–servidor.



INSTITUTO TECNOLÓGICO DE TUXTEPEC
Departamento de Sistemas y Computación
Formato para prácticas de Laboratorio

4. Fundamento

La **arquitectura cliente–servidor** es un modelo de comunicación en el que dos componentes principales interactúan:

- **Cliente:** Solicita servicios o recursos.
- **Servidor:** Proporciona esos servicios o recursos.

Este modelo se basa en la idea de que el cliente inicia la comunicación y el servidor responde. Es ampliamente usado en aplicaciones web, sistemas distribuidos y redes.

Funciones del Cliente	Funciones del Servidor
<ul style="list-style-type: none">• Interfaz de usuario: Permite la interacción con el sistema.• Generar solicitudes: Envía peticiones al servidor (por ejemplo, consultar datos).• Procesamiento local: Puede realizar validaciones o cálculos simples.• Presentación de resultados: Muestra la información que recibe del servidor.	<ul style="list-style-type: none">• Procesar solicitudes: Recibe y atiende las peticiones del cliente.• Gestión de datos: Accede, almacena y actualiza información en bases de datos.• Control de seguridad: Autenticación, autorización y protección de datos.• Escalabilidad: Maneja múltiples clientes simultáneamente.

¿Qué es una API?

Una API (Application Programming Interface) **es un conjunto de reglas y protocolos que permite que dos aplicaciones se comuniquen entre sí.**

- Define **cómo** se deben enviar y recibir datos.
- Facilita la integración entre sistemas sin necesidad de conocer la implementación interna.

En el caso de **mini_pokeapi**, la API expone datos de Pokémon para que los clientes (apps web, móviles) puedan consultarlos.



INSTITUTO TECNOLÓGICO DE TUXTEPEC
Departamento de Sistemas y Computación
Formato para prácticas de Laboratorio

¿Qué es REST?

REST (Representational State Transfer) es un estilo de arquitectura para diseñar APIs que usan el protocolo HTTP. Características principales:

- Basado en **recursos** (por ejemplo, /pokemon, /abilities).
- Usa métodos HTTP estándar: **GET, POST, PUT, DELETE**.
- Es **ligero** y fácil de implementar.

Principios REST

1. Recursos

Todo se representa como un recurso identificado por una **URI** (por ejemplo: <https://pokeapi.co/api/v2/pokemon/1>).

2. Stateless

Cada petición del cliente al servidor debe contener toda la información necesaria. El servidor **no guarda estado** entre peticiones.

3. Uso de URI

Las URIs son únicas y describen el recurso. Ejemplo:

- GET /pokemon/25 → Obtiene información de Pikachu.

4. Operaciones HTTP

- **GET**: Consultar datos.
- **POST**: Crear un recurso.
- **PUT/PATCH**: Actualizar un recurso.
- **DELETE**: Eliminar un recurso.

Las operaciones CRUD (Create, Read, Update, Delete)

son las cuatro funciones fundamentales para gestionar datos en bases de datos y aplicaciones, permitiendo a los usuarios interactuar con la información: Crear nuevos registros, Leer (o consultar) existentes, Actualizar información y Eliminar registros, siendo la base de la manipulación de datos en cualquier sistema, desde un simple cuaderno hasta APIs RESTful.

¿Qué es JSON?

JSON (JavaScript Object Notation) es un formato ligero para el intercambio de datos. Se basa en texto y es fácil de leer y escribir para humanos, además de ser sencillo de



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

interpretar por máquinas. Se utiliza ampliamente para transmitir datos entre un servidor y una aplicación web.

Estructura básica	Ventajas en APIs
<p>Un documento JSON se compone de:</p> <ul style="list-style-type: none">• Objetos: Con llaves { }, contienen pares clave: valor.• Arreglos: Con corchetes [], contienen listas de valores.• Tipos de datos:<ul style="list-style-type: none">◦ String: "texto"◦ Número: 123◦ Booleano: true / false◦ Null: null◦ Objeto: { "clave": "valor" }◦ Arreglo: ["valor1", "valor2"]	<ul style="list-style-type: none">• Ligero y rápido: Ideal para transmitir datos por HTTP.• Compatible: Funciona con casi todos los lenguajes de programación.• Estandarizado: Fácil de integrar en servicios REST y aplicaciones web.• Legible: Tanto para humanos como para máquinas.• Flexible: Permite estructuras complejas (objetos dentro de objetos, listas, etc.).

¿Qué es Node.js?

Node.js es un **entorno de ejecución** para JavaScript que se ejecuta en el **lado del servidor**. Está construido sobre el motor **V8 de Google Chrome**, lo que le permite ejecutar código JavaScript fuera del navegador.

Características	Uso en servidores
<ul style="list-style-type: none">• Asíncrono y no bloqueante: Utiliza un modelo basado en eventos, ideal para aplicaciones que manejan muchas conexiones simultáneas.• Alto rendimiento: Gracias al motor V8 y su arquitectura orientada a eventos.• Single-thread: Opera en un solo hilo, pero maneja múltiples operaciones concurrentes mediante callbacks y promesas.	<p>Node.js se utiliza principalmente para:</p> <ul style="list-style-type: none">• Servidores web: Crear APIs REST y aplicaciones en tiempo real.• Aplicaciones en tiempo real: Chats, juegos online, streaming.• Microservicios: Arquitecturas distribuidas y escalables.



INSTITUTO TECNOLÓGICO DE TUXTEPEC
Departamento de Sistemas y Computación
Formato para prácticas de Laboratorio

- **Gran ecosistema:** A través de **npm (Node Package Manager)**, ofrece miles de librerías.
- **Multiplataforma:** Funciona en Windows, Linux y macOS.

- **Integración con bases de datos:** MongoDB, MySQL, PostgreSQL, etc.

¿Qué es Express?

Express es una librería minimalista que proporciona herramientas para gestionar **rutas**, **peticiones HTTP**, **middlewares** y **respuestas** en aplicaciones Node.js. Es muy popular para desarrollar **APIs REST** y aplicaciones web escalables.

Endpoints y Rutas

- **Rutas:** Son las direcciones dentro de una aplicación que permiten acceder a recursos o ejecutar acciones.
Ejemplo: /usuarios, /productos/123.
- **Endpoints:** Son puntos específicos de acceso a la API que responden a una petición HTTP (GET, POST, PUT, DELETE).
Cada endpoint está asociado a una **ruta** y un **método HTTP**.

5. Descripción (Procedimiento)

A) Equipo necesario

- Computadora personal
- Sistema operativo Windows

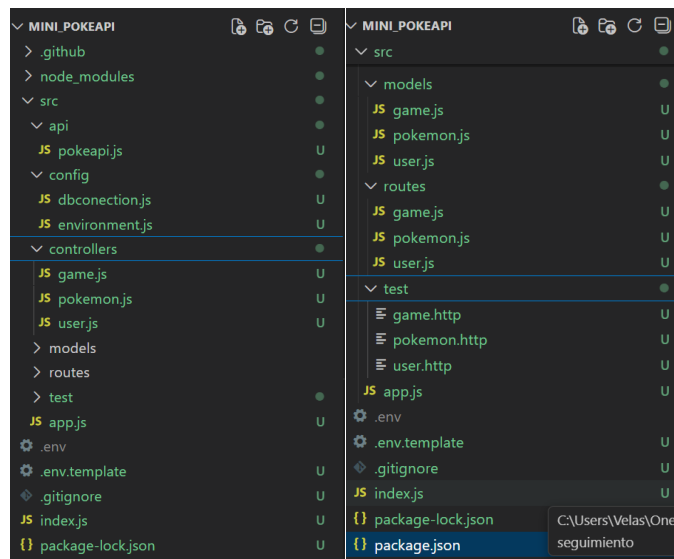
Material de apoyo

- Visual Studio Code
- Navegador web
- Node.js

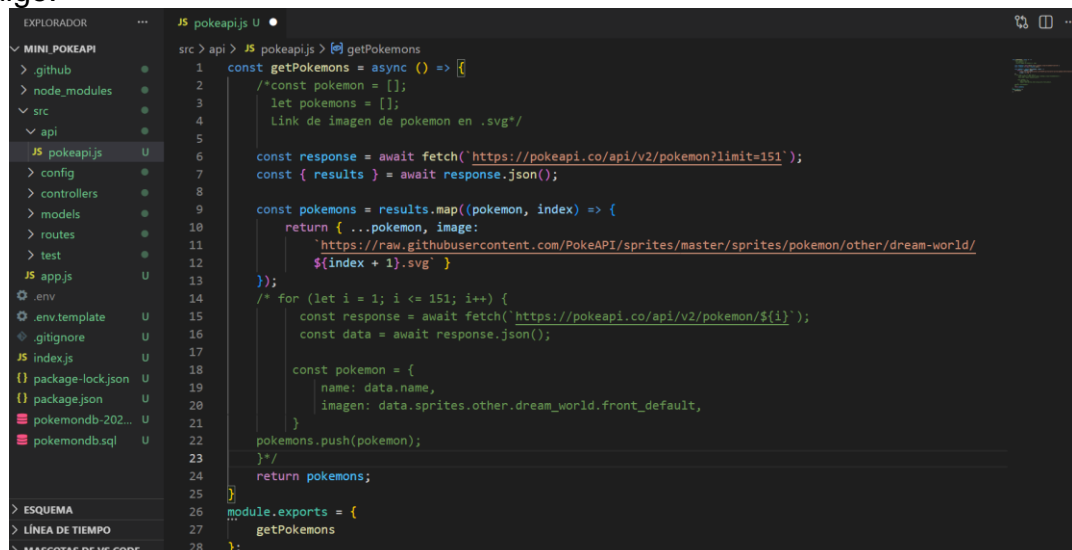
1. Desarrollo de la práctica

Parte 1: Mini PokeAPI

Se creó la estructura del proyecto **MINI_POKEAPI**. se desarrolló una API REST utilizando **Node.js** y el framework **Express**, el cual permitió la creación de rutas HTTP para el acceso a los datos. Para la conexión con la base de datos se empleó el módulo **mariadb**, facilitando la ejecución de consultas SQL. Posteriormente se creó una carpeta llamada **src** en donde se encontraran las siguientes carpetas y sus archivos con extensiones **.js** y **.http** mostrado en las imágenes, dentro de la carpeta de **MINI_POKEAPI** se debe de encontrar el siguiente archivo **.env**, **.env.template** y el **index.js**.



Empezando con la carpeta **api** y el archivo **pokeapi.js** se mostrará el siguiente código:





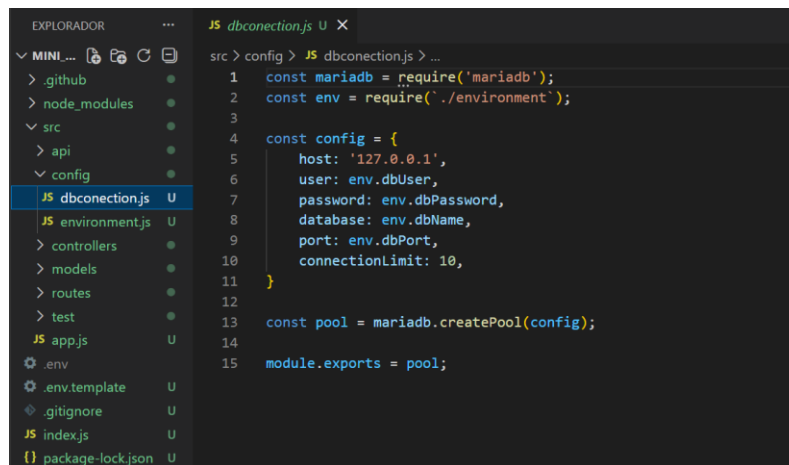
INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

El archivo **pokeapi.js** define una función asíncrona llamada **getPokemons** que consume la PokeAPI para obtener los primeros **151 Pokémon**. Mediante una petición HTTP, se recupera la lista básica de Pokémon y se procesa usando **map()** para agregar a cada uno una imagen en formato SVG proveniente del repositorio oficial de sprites de PokeAPI. Finalmente, la función retorna un arreglo con los datos del Pokémon (nombre y URL) junto con su imagen y se exporta para ser utilizada en otras partes de la aplicación, como controladores o rutas de una API REST.

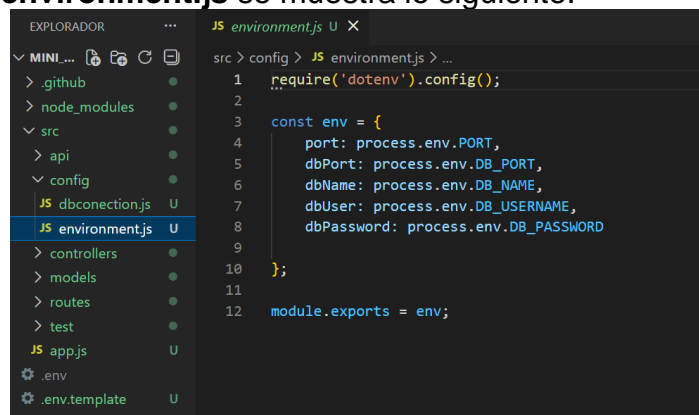
Carpeta **config** y el archivo **dbconnection.js** se muestra el siguiente código:



```
EXPLORADOR  ...  JS dbconnection.js U X
src > config > JS dbconnection.js > ...
1  const mariadb = require('mariadb');
2  const env = require('./environment');
3
4  const config = {
5    host: '127.0.0.1',
6    user: env.dbUser,
7    password: env.dbPassword,
8    database: env.dbName,
9    port: env.dbPort,
10   connectionLimit: 10,
11  }
12
13  const pool = mariadb.createPool(config);
14
15  module.exports = pool;
```

En donde el archivo **dbconnection.js** configura la conexión a una base de datos MariaDB utilizando un pool de conexiones. Los datos sensibles se cargan desde variables de entorno, lo que mejora la seguridad y facilita la configuración. Este módulo permite que toda la aplicación acceda de forma eficiente y controlada a la base de datos.

Base al archivo **environment.js** se muestra lo siguiente:



```
EXPLORADOR  ...  JS environment.js U X
src > config > JS environment.js > ...
1  require('dotenv').config();
2
3  const env = {
4    port: process.env.PORT,
5    dbPort: process.env.DB_PORT,
6    dbName: process.env.DB_NAME,
7    dbUser: process.env.DB_USERNAME,
8    dbPassword: process.env.DB_PASSWORD
9  };
10
11
12  module.exports = env;
```



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

El archivo `environment.js` carga las variables de entorno desde el archivo `.env` utilizando la librería `dotenv` y las centraliza en un objeto exportable. Esto permite configurar el puerto del servidor y los datos de conexión a la base de datos de forma segura, flexible y sin exponer información sensible en el código.

El archivo `.env` muestra lo siguiente a lo que también se relaciona con el archivo `.env.template`

The screenshot shows a code editor with two files open. The top file is `.env`, which contains the following configuration:

```
1 DB_PORT = 3306
2 DB_NAME = pokemondb
3 DB_USERNAME = root
4 DB_PASSWORD = "Belem_23"
5 PORT = 3000
```

The bottom file is `.env.template`, which shows the structure of the variables without values:

```
1 DB_PORT =
2 DB_NAME =
3 DB_USERNAME =
4 DB_PASSWORD =
5 PORT =
```

The file explorer on the left shows the project structure with files like `.github`, `node_modules`, `src`, `.env`, `.env.template`, `.gitignore`, and `index.js`.

Por su parte, el archivo `.env` contiene los valores reales de configuración, como el puerto de la base de datos, el nombre de la base de datos, el usuario, la contraseña y el puerto del servidor. Estos valores son leídos por `environment.js` en tiempo de ejecución. Asimismo, el archivo `.env.template` funciona como una **plantilla de referencia**, mostrando la estructura de las variables de entorno que deben definirse, pero sin incluir datos reales. Esto facilita la configuración del proyecto en distintos entornos y evita compartir información sensible en repositorios públicos.

Posteriormente la carpeta **controllers** en donde se encuentra los archivos **game.js**, **pokemon.js** y **user.js**



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
const win = async (req = request, res = response) => {
  const win = async (req = request, res = response) => {
    const { id } = req.params;
    if (isNaN(Number(id))) {
      res.status(400).send('Esto no es un número');
      return;
    }
    let conn;
    conn = await pool.getConnection();
    const [user] = await conn.query('SELECT * FROM users WHERE id = ?', [id]);
    if (!user) {
      res.status(500).send('No se pudo registrar la victoria');
      return;
    }
    const [game] = await conn.query('SELECT * FROM game WHERE user_id = ?', [id]);
    if (!game) {
      const module = await conn.query('INSERT INTO game (user_id, win, lose) VALUES (?, ?, ?)');
      if (module.affectedRows === 0) {
        res.status(500).send('No se pudo registrar la victoria');
      }
    } else {
      const module = await conn.query('UPDATE game SET win = win + 1 WHERE user_id = ?');
      if (module.affectedRows === 0) {
        res.status(500).send('No se pudo actualizar la victoria');
      }
    }
    conn.release();
  }
}

const lose = async (req = request, res = response) => {
  const { id } = req.params;
  if (isNaN(Number(id))) {
    res.status(400).send('Esto no es un número');
    return;
  }
  let conn;
  conn = await pool.getConnection();
  const [user] = await conn.query('SELECT * FROM users WHERE id = ?', [id]);
  if (!user) {
    res.status(500).send('No se pudo registrar la victoria');
    return;
  }
  const [game] = await conn.query('SELECT * FROM game WHERE user_id = ?', [id]);
  if (!game) {
    const module = await conn.query('INSERT INTO game (user_id, win, lose) VALUES (?, ?, ?)');
    if (module.affectedRows === 0) {
      res.status(500).send('No se pudo registrar la victoria');
    }
  } else {
    const module = await conn.query('UPDATE game SET lose = lose + 1 WHERE user_id = ?');
    if (module.affectedRows === 0) {
      res.status(500).send('No se pudo actualizar la victoria');
    }
  }
  conn.release();
}

module.exports = {
  win,
  lose
};
```

Este archivo corresponde al archivo **game.js** encargado de manejar la lógica para registrar **victorias y derrotas** de los usuarios en la base de datos. Las funciones **win** y **lose** reciben el identificador del usuario desde los parámetros de la solicitud **HTTP** y, antes de realizar cualquier operación, validan que dicho identificador sea numérico, evitando errores y peticiones inválidas. Posteriormente, el controlador establece una conexión con la base de datos mediante un **pool de conexiones**, lo que permite un acceso eficiente y controlado a MariaDB.

Una vez establecida la conexión, se verifica que el usuario exista en la base de datos. Si el usuario no se encuentra, la operación se detiene y se envía una respuesta de error, garantizando la integridad de la información. Si el usuario existe, el sistema consulta si ya cuenta con un registro de juego. En caso de que no exista dicho registro, se crea uno nuevo inicializando los valores de victorias y derrotas según corresponda. Si el registro ya existe, se actualiza incrementando el contador de victorias o derrotas. Durante todo el proceso se valida que las operaciones de inserción o actualización se realicen correctamente, verificando que se haya modificado al menos un registro. Además, el uso de bloques **try-catch-finally** permite manejar errores del servidor y asegurar que la conexión a la base de datos se cierre correctamente al finalizar la operación. Finalmente, las funciones se exportan para ser utilizadas en las rutas de la aplicación, formando parte de la lógica del **CRUD** del sistema y asegurando un manejo seguro y estructurado de las estadísticas de juego.



INSTITUTO TECNOLÓGICO DE TUXTEPEC

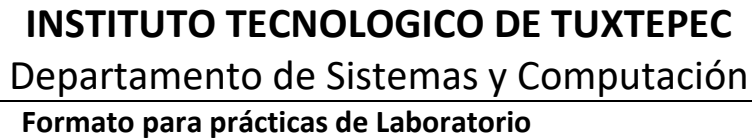
Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
EXPLORADOR
MINI...
> github
> node_modules
> src
  > api
  > config
  > controllers
  > game.js
  > pokemon.js
  > user.js
> models
> routes
> test
> app.js
env
env.template
.gitignore
index.js
package-lock.json
package.json
pokemondb-202...
pokemondb.sql
ESQUEMA
LÍNEA DE TIEMPO
MASCOTAS DE VS CODE

src > controllers > JS pokemon.js > ...
1  const { request, response } = require('express');
2  const pokeapi = require('../api/pokeapi');
3  const pool = require('../config/dbconnection');
4  const { pokemonQuery } = require('../models/pokemon');
5
6  const pokemonSeeder = async (req = request, res = response) => {
7    const pokemons = await pokeapi.getPokemons();
8
9    let conn;
10
11    try {
12      conn = await pool.getConnection();
13      await conn.query('SET FOREIGN_KEY_CHECKS = 0');
14      await conn.query('TRUNCATE TABLE pokemons');
15      await conn.query('SET FOREIGN_KEY_CHECKS = 1');
16
17      pokemons.forEach(async (pokemon) => {
18        await conn.query(pokemonQuery.add, [pokemon.name, pokemon.image]);
19      });
20      res.send("Pokémones agregados en la Base de Datos");
21    } catch (err) {
22      return res.status(500).send(err);
23    } finally {
24      if (conn) {
25        conn.end();
26      }
27    }
28  }
29
30  const randomPokemon = async (req = request, res = response) => {
31    let conn;
32
33    try {
34      conn = await pool.getConnection();
35      const pokemons = await conn.query(pokemonQuery.random);
36      if (pokemons.length === 0) {
37        return res.status(500).send("No hay pokémones en la base de datos");
38      }
39      res.send(pokemons);
40    } catch (err) {
41      return res.status(500).send(err);
42    } finally {
43      if (conn) {
44        conn.end();
45      }
46    }
47  }
48
49  const idPokemon = async (req = request, res = response) => {
50    let conn;
51
52    try {
53      const { id } = req.params
54      if (!isNaN(Number(id))) {
55        const idPokemon = async (req = request, res = response) => {
56          if (!isNaN(Number(id))) {
57            res.status(400).send("Esto no es un numero");
58            return;
59          }
60
61          conn = await pool.getConnection();
62          const pokeid = await conn.query(pokemonQuery.view, [id]);
63          if (!pokeid) {
64            res.status(404).send("Pokémon no encontrado");
65            return;
66          }
67          res.send(pokeid);
68        } catch (err) {
69          return res.status(500).send(err);
70        } finally {
71          if (conn) {
72            conn.end();
73          }
74        }
75      }
76
77      module.exports = {
78        pokemonSeeder,
79        randomPokemon,
80        idPokemon
81      };
82    }
83  }
84}
```

El archivo **pokemon.js** es un **controlador de Express** que gestiona las operaciones relacionadas con los Pokémon dentro del proyecto, conectando las rutas con la **PokeAPI** y la **base de datos MariaDB**. La función **pokemonSeeder** obtiene los Pokémon desde la PokeAPI, limpia la tabla pokemons y la vuelve a poblar con los datos actualizados. La función **randomPokemon** permite obtener un



The image displays four screenshots of a code editor showing the implementation of a REST API for a user management system. The code is written in JavaScript using Node.js, Express, and MongoDB.

Top Left Screenshot: Shows the initial setup of the API. It includes the Express app configuration, the MongoDB connection, and the initial setup of the user creation endpoint. The code defines a `createUser` function that takes a request and response object, validates the input, and creates a new user in the database.

Top Right Screenshot: Shows the implementation of the user creation endpoint. It includes the `createUser` function, which uses the `mongoose` library to create a new user and save it to the database. It also includes the `updateUser` function, which updates an existing user in the database.

Bottom Left Screenshot: Shows the implementation of the user deletion endpoint. It includes the `deleteUser` function, which uses the `mongoose` library to delete a user from the database. It also includes the `findUser` function, which finds a user in the database.

Bottom Right Screenshot: Shows the implementation of the user update endpoint. It includes the `updateUser` function, which uses the `mongoose` library to update an existing user in the database. It also includes the `findUser` function, which finds a user in the database.



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
const updateUser = async (req = request, res = response) => {
  const {id} = req.params;
  if (!Number(id)) {
    res.status(400).send("Esto no es un número");
    return;
  }
  let conn;
  try {
    conn = await pool.getConnection();
    const [user] = await conn.query(userQueries.users.view, [id]);
    if (!user) {
      res.status(404).send("Usuario no encontrado");
      return;
    }
    const {name, lastname, email, password} = req.body;
    let hashedPassword;
    if (password) {
      hashedPassword = await bcrypt.hash(password, saltRounds);
    } else {
      hashedPassword = user.password;
    }
    const updateUser = await conn.query(userQueries.users.update, [
      userToUpdate.name,
      userToUpdate.lastname,
      userToUpdate.email,
      userToUpdate.password,
      id
    ]);
    if (userToUpdate.affectedRows === 0) {
      res.status(500).send("No se pudo actualizar el usuario");
      return;
    }
    res.send(msg("Usuario actualizado", user: userToUpdate));
  } catch (err) {
    console.log(err);
    res.status(500).send("error");
  } finally {
    if (conn) conn.end();
  }
}

const loginUser = async (req = request, res = response) => {
  const {email, password} = req.body;
  if (!email || !password) {
    res.status(400).send("Faltan datos");
    return;
  }
  let conn;
  try {
    conn = await pool.getConnection();
    const [user] = await conn.query(userQueries.users.viewByEmail, [email]);
    if (!user) {
      res.status(404).send("Usuario no encontrado");
      return;
    }
    const validPassword = await bcrypt.compare(password, user.password);
    if (!validPassword) {
      res.status(401).send("Usuario o contraseña incorrectos");
      return;
    }
    delete user.password;
  } catch (err) {
    console.log(err);
    res.status(500).send("error");
  } finally {
    if (conn) conn.end();
  }
}

module.exports = {
  showUsers,
  viewUser,
  createUser,
  removeUser,
  updateUser,
  loginUser
}
```

El archivo **user.js** es un **controlador de Express** encargado de gestionar todas las operaciones relacionadas con los usuarios dentro del sistema, funcionando como parte central del **CRUD de usuarios** y del proceso de **autenticación**. En este archivo se utilizan consultas definidas en el modelo de usuario, un **pool de conexiones** a MariaDB y la librería **bcrypt** para manejar contraseñas de forma segura mediante cifrado.

La función **showUsers** obtiene y devuelve la lista completa de usuarios almacenados en la base de datos. La función **viewUser** permite consultar un usuario específico por su identificador, validando previamente que el ID sea numérico y verificando que el usuario exista. La función **createUser** se encarga de registrar nuevos usuarios, validando que los datos requeridos estén presentes, comprobando que el correo no esté previamente registrado y cifrando la contraseña antes de almacenarla en la base de datos, lo que refuerza la seguridad de la información.

Por otro lado, **removeUser** permite eliminar un usuario a partir de su ID, validando su existencia antes de realizar la eliminación. La función **updateUser** se utiliza para modificar la información de un usuario existente, permitiendo actualizar datos personales y la contraseña, la cual se vuelve a cifrar en caso de ser modificada. Finalmente, la función **loginUser** gestiona el proceso de inicio de sesión, verificando que el correo exista y comparando la contraseña ingresada con la almacenada



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

mediante bcrypt; si la autenticación es correcta, se devuelve la información del usuario sin incluir la contraseña.

En todas las funciones se emplea el uso de try-catch-finally para manejar errores y asegurar el cierre correcto de la conexión a la base de datos. En conjunto, este controlador garantiza un manejo seguro, validado y estructurado de los usuarios dentro de la aplicación.

Posteriormente la carpeta **models** contiene los siguientes archivos **game.js**, **pokemon.js** y **user.js**:

```
EXPLORADOR
└─ MINI_POKEAPI
   ├── .github
   ├── node_modules
   └── src
       ├── api
       ├── config
       ├── controllers
       └── models
           ├── JS game.js
           ├── JS pokemon.js
           ├── JS user.js
           ├── routes
           └── test

src > models > JS game.js > ...
1  const gameQuery = {
2    getGame: 'SELECT * FROM games WHERE user_id = ?',
3    addGame: 'INSERT INTO games (user_id, win, lose) VALUES (?, ?, ?)',
4    updateGame: 'UPDATE games SET win = ?, lose = ? WHERE user_id = ?'
5  }
6
7  module.exports = {
8    gameQuery
9  };
10
```

El archivo **game.js** pertenece a la carpeta **models** y define las consultas SQL necesarias para gestionar la información de los juegos en la base de datos. En él se agrupan las sentencias que permiten interactuar con la tabla **games** de forma ordenada y reutilizable.

La consulta **getGame** se utiliza para obtener el registro de un juego asociado a un usuario específico, buscando por su identificador (**user_id**). La consulta **addGame** permite crear un nuevo registro de juego para un usuario, inicializando los campos de victorias (**win**) y derrotas (**lose**). Por su parte, la consulta **updateGame** se encarga de actualizar los valores de victorias y derrotas de un usuario cuando se registra un nuevo resultado.

Finalmente, el objeto **gameQuery** se exporta para que pueda ser utilizado por los controladores, manteniendo la separación entre la lógica de negocio y el acceso a datos, lo que mejora la organización y el mantenimiento del proyecto.



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
src > models > JS pokemon.js > ...
1  const pokemonQuery = {
2    add: 'INSERT INTO pokemons (name, image) VALUES (?, ?)',
3    random: 'SELECT * FROM pokemons ORDER BY RAND() LIMIT 4',
4    view: 'SELECT * FROM pokemons WHERE id = ?'
5  }
6
7
8  module.exports = {
9    ...
10   pokemonQuery
11 };
12
```

El archivo **pokemon.js** ubicado en la carpeta **models** define las consultas SQL necesarias para interactuar con la tabla **pokemons** de la base de datos. En este archivo se agrupan las sentencias que permiten insertar y consultar información de los Pokémon de forma centralizada y reutilizable.

La consulta **add** se utiliza para insertar un nuevo Pokémon, almacenando su nombre y la URL de su imagen. La consulta **random** permite obtener cuatro Pokémon aleatorios de la base de datos, lo cual es útil para dinámicas de juego o selecciones al azar. Por su parte, la consulta **view** se encarga de buscar un Pokémon específico por su identificador, facilitando la consulta individual de registros.

Finalmente, el objeto **pokemonQuery** se exporta para que pueda ser utilizado por los controladores, manteniendo la separación entre la lógica de acceso a datos y la lógica de negocio, lo que contribuye a un código más organizado, mantenible y claro.

```
src > models > JS user.js > ...
1  const users = {
2    show: "SELECT * FROM users",
3    view: "SELECT * FROM users WHERE id = ?",
4    verifyEmail: "SELECT * FROM users WHERE email = ?",
5    create: "INSERT INTO users (name, lastname, email, password) VALUES (?, ?, ?, ?)",
6    delete: "DELETE FROM users WHERE id = ?",
7    update: "UPDATE users SET name = ?, lastname = ?, email = ?, password = ? WHERE id = ?",
8    viewByEmail: "SELECT * FROM users WHERE email = ?"
9  }
10
11
12  module.exports = {
13    ...
14   users
15 };
16
```



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

El archivo **user.js** define las consultas SQL necesarias para la gestión de usuarios en la base de datos. En este archivo se centralizan las sentencias que permiten realizar operaciones de consulta, inserción, actualización y eliminación sobre la tabla users, facilitando su reutilización desde los controladores.

La consulta show obtiene todos los usuarios registrados, mientras que **view** permite consultar un usuario específico mediante su identificador. La consulta **verifyEmail** se utiliza para verificar si un correo electrónico ya se encuentra registrado, lo cual es clave para evitar duplicados. La consulta **create** permite insertar un nuevo usuario almacenando sus datos personales y la contraseña cifrada. Por su parte, **delete** elimina un usuario por su ID y **update** actualiza la información de un usuario existente. Finalmente, **viewByEmail** se emplea durante el proceso de autenticación para buscar un usuario a partir de su correo electrónico.

El objeto **users** se exporta para ser utilizado por los controladores, manteniendo una clara separación entre el acceso a datos y la lógica de negocio, lo que mejora la organización, seguridad y mantenimiento del proyecto.

```
EXPLORADOR  ...  JS game.js U X
MINI...  .github  .
node_modules  .
src  .
  api  .
  config  .
  controllers  .
  models  .
  routes  .
    JS game.js  U
    JS pokemon.js  U
    JS user.js  U
    test  .

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

El archivo **game.js** ubicado en la carpeta routes define las rutas HTTP relacionadas con la lógica del juego dentro de la API. En este archivo se utiliza el enrutador de Express para asociar las solicitudes del cliente con las funciones correspondientes del controlador de juegos.

La ruta `/win/:id` recibe solicitudes de tipo GET y se encarga de registrar una victoria para el usuario cuyo identificador se envía como parámetro en la URL. De manera similar, la ruta `/lose/:id` registra una derrota para el usuario indicado. Ambas rutas delegan la lógica al controlador game, manteniendo separada la definición de rutas de la lógica de negocio. Finalmente, el enrutador se exporta para ser integrado en



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

la aplicación principal, permitiendo que estas rutas formen parte del flujo de la API y contribuyendo a una estructura ordenada y mantenible del proyecto.



```
EXPLORADOR
└─ MINI...
   ├── .github
   ├── node_modules
   └── src
       ├── api
       ├── config
       ├── controllers
       ├── models
       └── routes
           ├── JS game.js
           ├── JS pokemon.js
           ├── JS user.js
           └── test

JS pokemon.js U X
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;
```

El archivo `pokemon.js` ubicado en la carpeta `routes` define las rutas HTTP relacionadas con los Pokémon dentro de la API. Utiliza el enrutador de Express para enlazar las solicitudes del cliente con las funciones correspondientes del controlador de Pokémon.

La ruta `/seed` ejecuta el controlador `pokemonSeeder`, el cual obtiene los Pokémon desde la `PokeAPI` y los inserta en la base de datos. Esta ruta se utiliza principalmente para inicializar o recargar la información de Pokémon en el sistema. Por otro lado, la ruta `/random` invoca el controlador `randomPokemon`, que devuelve Pokémon aleatorios almacenados en la base de datos, permitiendo funcionalidades como juegos o selecciones al azar. El enrutador se exporta para ser integrado en la aplicación principal, manteniendo una estructura clara donde las rutas únicamente definen los endpoints y delegan la lógica al controlador correspondiente.



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
EXPLORADOR  ...  JS pokemon.js U  JS user.js U X

MINI...  > .github  > node_modules  > src
  > api  > config  > controllers  > models  > routes
    JS game.js  JS pokemon.js  JS user.js U
  > test  JS app.js U

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

El archivo user.js ubicado en la carpeta routes define las rutas HTTP relacionadas con la gestión de usuarios dentro de la API. En este archivo se utiliza el enrutador de Express para asociar cada endpoint con la función correspondiente del controlador de usuarios.

La ruta GET / permite obtener la lista completa de usuarios, mientras que GET /:id se utiliza para consultar un usuario específico mediante su identificador. La ruta POST / se encarga de crear un nuevo usuario, recibiendo los datos desde el cuerpo de la solicitud. Por su parte, DELETE /:id permite eliminar un usuario y PUT /:id actualizar la información de un usuario existente. Finalmente, la ruta POST /login gestiona el proceso de autenticación, validando las credenciales del usuario. Este archivo se exporta para integrarse en la aplicación principal, manteniendo una clara separación entre la definición de rutas y la lógica de negocio, lo que favorece una estructura organizada y mantenible del proyecto.

```
Archivo  Editar  Selección  Ver  Ir  ...  <  >  Q. m...

EXPLORADOR  ...  user.http U X

MINI...  > .github  > node_modules  > src
  > api  > config  > controllers  > models  > routes
    JS game.js  JS pokemon.js  JS user.js U
  > test  JS app.js U
  > .env  > .env.template  > .gitignore  > package-lock.json  > package.json  > pokemondb-202...  > pokemondb.sql  > ESQUEMA  > LÍNEA DE TIEMPO

src > test > user.http > GET /user/
1  GET http://localhost:3000/user/
2  ##
3  Send Request
4  POST http://localhost:3000/user/0
5  Content-Type: application/json
6  ##
7  {
8    "password": "123456"
9  }
10  ##
11  Send Request
12  DELETE http://localhost:3000/user/3
13  ##
14  Send Request
15  PUT http://localhost:3000/user/1
16  Content-Type: application/json
17  ##
18  {
19    "password": "123456"
20  }
21  ##
22  Send Request
23  POST http://localhost:3000/user/login
24  Content-Type: application/json
25  ##
26  {
27    "email": "carrerabelem23@gmail.com",
28    "password": "123456"
29  }
30  ##
31  Send Request
32  POST http://localhost:3000/user/login
33  Content-Type: application/json
34  ##
35  {
36    "email": "carrerabelem23@gmail.com",
37    "password": "123456"
38  }
39  ##
40  Send Request
41  POST http://localhost:3000/user/login
42  Content-Type: application/json
43  ##
44  {
45    "email": "carrerabelem23@gmail.com",
46    "password": "123456"
47  }
48  ##
49  Send Request
50  POST http://localhost:3000/user/login
51  Content-Type: application/json
52  ##
53  {
54    "email": "carrerabelem23@gmail.com",
55    "password": "123456"
56  }
57  ##
58  Send Request
59  POST http://localhost:3000/user/login
60  Content-Type: application/json
61  ##
62  {
63    "email": "carrerabelem23@gmail.com",
64    "password": "123456"
65  }
66  ##
67  Send Request
68  POST http://localhost:3000/user/login
69  Content-Type: application/json
70  ##
71  {
72    "email": "carrerabelem23@gmail.com",
73    "password": "123456"
74  }
75  ##
76  Send Request
77  POST http://localhost:3000/user/login
78  Content-Type: application/json
79  ##
80  {
81    "email": "carrerabelem23@gmail.com",
82    "password": "123456"
83  }
84  ##
85  Send Request
86  POST http://localhost:3000/user/login
87  Content-Type: application/json
88  ##
89  {
90    "email": "carrerabelem23@gmail.com",
91    "password": "123456"
92  }
93  ##
94  Send Request
95  POST http://localhost:3000/user/login
96  Content-Type: application/json
97  ##
98  {
99    "email": "carrerabelem23@gmail.com",
100   "password": "123456"
101  }
102  ##
103  Send Request
104  POST http://localhost:3000/user/login
105  Content-Type: application/json
106  ##
107  {
108    "email": "carrerabelem23@gmail.com",
109    "password": "123456"
110  }
111  ##
112  Send Request
113  POST http://localhost:3000/user/login
114  Content-Type: application/json
115  ##
116  {
117    "email": "carrerabelem23@gmail.com",
118    "password": "123456"
119  }
120  ##
121  Send Request
122  POST http://localhost:3000/user/login
123  Content-Type: application/json
124  ##
125  {
126    "email": "carrerabelem23@gmail.com",
127    "password": "123456"
128  }
129  ##
130  Send Request
131  POST http://localhost:3000/user/login
132  Content-Type: application/json
133  ##
134  {
135    "email": "carrerabelem23@gmail.com",
136    "password": "123456"
137  }
138  ##
139  Send Request
140  POST http://localhost:3000/user/login
141  Content-Type: application/json
142  ##
143  {
144    "email": "carrerabelem23@gmail.com",
145    "password": "123456"
146  }
147  ##
148  Send Request
149  POST http://localhost:3000/user/login
150  Content-Type: application/json
151  ##
152  {
153    "email": "carrerabelem23@gmail.com",
154    "password": "123456"
155  }
156  ##
157  Send Request
158  POST http://localhost:3000/user/login
159  Content-Type: application/json
160  ##
161  {
162    "email": "carrerabelem23@gmail.com",
163    "password": "123456"
164  }
165  ##
166  Send Request
167  POST http://localhost:3000/user/login
168  Content-Type: application/json
169  ##
170  {
171    "email": "carrerabelem23@gmail.com",
172    "password": "123456"
173  }
174  ##
175  Send Request
176  POST http://localhost:3000/user/login
177  Content-Type: application/json
178  ##
179  {
180    "email": "carrerabelem23@gmail.com",
181    "password": "123456"
182  }
183  ##
184  Send Request
185  POST http://localhost:3000/user/login
186  Content-Type: application/json
187  ##
188  {
189    "email": "carrerabelem23@gmail.com",
190    "password": "123456"
191  }
192  ##
193  Send Request
194  POST http://localhost:3000/user/login
195  Content-Type: application/json
196  ##
197  {
198    "email": "carrerabelem23@gmail.com",
199    "password": "123456"
200  }
201  ##
202  Send Request
203  POST http://localhost:3000/user/login
204  Content-Type: application/json
205  ##
206  {
207    "email": "carrerabelem23@gmail.com",
208    "password": "123456"
209  }
210  ##
211  Send Request
212  POST http://localhost:3000/user/login
213  Content-Type: application/json
214  ##
215  {
216    "email": "carrerabelem23@gmail.com",
217    "password": "123456"
218  }
219  ##
220  Send Request
221  POST http://localhost:3000/user/login
222  Content-Type: application/json
223  ##
224  {
225    "email": "carrerabelem23@gmail.com",
226    "password": "123456"
227  }
228  ##
229  Send Request
230  POST http://localhost:3000/user/login
231  Content-Type: application/json
232  ##
233  {
234    "email": "carrerabelem23@gmail.com",
235    "password": "123456"
236  }
237  ##
238  Send Request
239  POST http://localhost:3000/user/login
240  Content-Type: application/json
241  ##
242  {
243    "email": "carrerabelem23@gmail.com",
244    "password": "123456"
245  }
246  ##
247  Send Request
248  POST http://localhost:3000/user/login
249  Content-Type: application/json
250  ##
251  {
252    "email": "carrerabelem23@gmail.com",
253    "password": "123456"
254  }
255  ##
256  Send Request
257  POST http://localhost:3000/user/login
258  Content-Type: application/json
259  ##
260  {
261    "email": "carrerabelem23@gmail.com",
262    "password": "123456"
263  }
264  ##
265  Send Request
266  POST http://localhost:3000/user/login
267  Content-Type: application/json
268  ##
269  {
270    "email": "carrerabelem23@gmail.com",
271    "password": "123456"
272  }
273  ##
274  Send Request
275  POST http://localhost:3000/user/login
276  Content-Type: application/json
277  ##
278  {
279    "email": "carrerabelem23@gmail.com",
280    "password": "123456"
281  }
282  ##
283  Send Request
284  POST http://localhost:3000/user/login
285  Content-Type: application/json
286  ##
287  {
288    "email": "carrerabelem23@gmail.com",
289    "password": "123456"
290  }
291  ##
292  Send Request
293  POST http://localhost:3000/user/login
294  Content-Type: application/json
295  ##
296  {
297    "email": "carrerabelem23@gmail.com",
298    "password": "123456"
299  }
300  ##
301  Send Request
302  POST http://localhost:3000/user/login
303  Content-Type: application/json
304  ##
305  {
306    "email": "carrerabelem23@gmail.com",
307    "password": "123456"
308  }
309  ##
310  Send Request
311  POST http://localhost:3000/user/login
312  Content-Type: application/json
313  ##
314  {
315    "email": "carrerabelem23@gmail.com",
316    "password": "123456"
317  }
318  ##
319  Send Request
320  POST http://localhost:3000/user/login
321  Content-Type: application/json
322  ##
323  {
324    "email": "carrerabelem23@gmail.com",
325    "password": "123456"
326  }
327  ##
328  Send Request
329  POST http://localhost:3000/user/login
330  Content-Type: application/json
331  ##
332  {
333    "email": "carrerabelem23@gmail.com",
334    "password": "123456"
335  }
336  ##
337  Send Request
338  POST http://localhost:3000/user/login
339  Content-Type: application/json
340  ##
341  {
342    "email": "carrerabelem23@gmail.com",
343    "password": "123456"
344  }
345  ##
346  Send Request
347  POST http://localhost:3000/user/login
348  Content-Type: application/json
349  ##
350  {
351    "email": "carrerabelem23@gmail.com",
352    "password": "123456"
353  }
354  ##
355  Send Request
356  POST http://localhost:3000/user/login
357  Content-Type: application/json
358  ##
359  {
360    "email": "carrerabelem23@gmail.com",
361    "password": "123456"
362  }
363  ##
364  Send Request
365  POST http://localhost:3000/user/login
366  Content-Type: application/json
367  ##
368  {
369    "email": "carrerabelem23@gmail.com",
370    "password": "123456"
371  }
372  ##
373  Send Request
374  POST http://localhost:3000/user/login
375  Content-Type: application/json
376  ##
377  {
378    "email": "carrerabelem23@gmail.com",
379    "password": "123456"
380  }
381  ##
382  Send Request
383  POST http://localhost:3000/user/login
384  Content-Type: application/json
385  ##
386  {
387    "email": "carrerabelem23@gmail.com",
388    "password": "123456"
389  }
390  ##
391  Send Request
392  POST http://localhost:3000/user/login
393  Content-Type: application/json
394  ##
395  {
396    "email": "carrerabelem23@gmail.com",
397    "password": "123456"
398  }
399  ##
400  Send Request
401  POST http://localhost:3000/user/login
402  Content-Type: application/json
403  ##
404  {
405    "email": "carrerabelem23@gmail.com",
406    "password": "123456"
407  }
408  ##
409  Send Request
410  POST http://localhost:3000/user/login
411  Content-Type: application/json
412  ##
413  {
414    "email": "carrerabelem23@gmail.com",
415    "password": "123456"
416  }
417  ##
418  Send Request
419  POST http://localhost:3000/user/login
420  Content-Type: application/json
421  ##
422  {
423    "email": "carrerabelem23@gmail.com",
424    "password": "123456"
425  }
426  ##
427  Send Request
428  POST http://localhost:3000/user/login
429  Content-Type: application/json
430  ##
431  {
432    "email": "carrerabelem23@gmail.com",
433    "password": "123456"
434  }
435  ##
436  Send Request
437  POST http://localhost:3000/user/login
438  Content-Type: application/json
439  ##
440  {
441    "email": "carrerabelem23@gmail.com",
442    "password": "123456"
443  }
444  ##
445  Send Request
446  POST http://localhost:3000/user/login
447  Content-Type: application/json
448  ##
449  {
450    "email": "carrerabelem23@gmail.com",
451    "password": "123456"
452  }
453  ##
454  Send Request
455  POST http://localhost:3000/user/login
456  Content-Type: application/json
457  ##
458  {
459    "email": "carrerabelem23@gmail.com",
460    "password": "123456"
461  }
462  ##
463  Send Request
464  POST http://localhost:3000/user/login
465  Content-Type: application/json
466  ##
467  {
468    "email": "carrerabelem23@gmail.com",
469    "password": "123456"
470  }
471  ##
472  Send Request
473  POST http://localhost:3000/user/login
474  Content-Type: application/json
475  ##
476  {
477    "email": "carrerabelem23@gmail.com",
478    "password": "123456"
479  }
480  ##
481  Send Request
482  POST http://localhost:3000/user/login
483  Content-Type: application/json
484  ##
485  {
486    "email": "carrerabelem23@gmail.com",
487    "password": "123456"
488  }
489  ##
490  Send Request
491  POST http://localhost:3000/user/login
492  Content-Type: application/json
493  ##
494  {
495    "email": "carrerabelem23@gmail.com",
496    "password": "123456"
497  }
498  ##
499  Send Request
500  POST http://localhost:3000/user/login
501  Content-Type: application/json
502  ##
503  {
504    "email": "carrerabelem23@gmail.com",
505    "password": "123456"
506  }
507  ##
508  Send Request
509  POST http://localhost:3000/user/login
510  Content-Type: application/json
511  ##
512  {
513    "email": "carrerabelem23@gmail.com",
514    "password": "123456"
515  }
516  ##
517  Send Request
518  POST http://localhost:3000/user/login
519  Content-Type: application/json
520  ##
521  {
522    "email": "carrerabelem23@gmail.com",
523    "password": "123456"
524  }
525  ##
526  Send Request
527  POST http://localhost:3000/user/login
528  Content-Type: application/json
529  ##
530  {
531    "email": "carrerabelem23@gmail.com",
532    "password": "123456"
533  }
534  ##
535  Send Request
536  POST http://localhost:3000/user/login
537  Content-Type: application/json
538  ##
539  {
540    "email": "carrerabelem23@gmail.com",
541    "password": "123456"
542  }
543  ##
544  Send Request
545  POST http://localhost:3000/user/login
546  Content-Type: application/json
547  ##
548  {
549    "email": "carrerabelem23@gmail.com",
550    "password": "123456"
551  }
552  ##
553  Send Request
554  POST http://localhost:3000/user/login
555  Content-Type: application/json
556  ##
557  {
558    "email": "carrerabelem23@gmail.com",
559    "password": "123456"
560  }
561  ##
562  Send Request
563  POST http://localhost:3000/user/login
564  Content-Type: application/json
565  ##
566  {
567    "email": "carrerabelem23@gmail.com",
568    "password": "123456"
569  }
570  ##
571  Send Request
572  POST http://localhost:3000/user/login
573  Content-Type: application/json
574  ##
575  {
576    "email": "carrerabelem23@gmail.com",
577    "password": "123456"
578  }
579  ##
580  Send Request
581  POST http://localhost:3000/user/login
582  Content-Type: application/json
583  ##
584  {
585    "email": "carrerabelem23@gmail.com",
586    "password": "123456"
587  }
588  ##
589  Send Request
590  POST http://localhost:3000/user/login
591  Content-Type: application/json
592  ##
593  {
594    "email": "carrerabelem23@gmail.com",
595    "password": "123456"
596  }
597  ##
598  Send Request
599  POST http://localhost:3000/user/login
600  Content-Type: application/json
601  ##
602  {
603    "email": "carrerabelem23@gmail.com",
604    "password": "123456"
605  }
606  ##
607  Send Request
608  POST http://localhost:3000/user/login
609  Content-Type: application/json
610  ##
611  {
612    "email": "carrerabelem23@gmail.com",
613    "password": "123456"
614  }
615  ##
616  Send Request
617  POST http://localhost:3000/user/login
618  Content-Type: application/json
619  ##
620  {
621    "email": "carrerabelem23@gmail.com",
622    "password": "123456"
623  }
624  ##
625  Send Request
626  POST http://localhost:3000/user/login
627  Content-Type: application/json
628  ##
629  {
630    "email": "carrerabelem23@gmail.com",
631    "password": "123456"
632  }
633  ##
634  Send Request
635  POST http://localhost:3000/user/login
636  Content-Type: application/json
637  ##
638  {
639    "email": "carrerabelem23@gmail.com",
640    "password": "123456"
641  }
642  ##
643  Send Request
644  POST http://localhost:3000/user/login
645  Content-Type: application/json
646  ##
647  {
648    "email": "carrerabelem23@gmail.com",
649    "password": "123456"
650  }
651  ##
652  Send Request
653  POST http://localhost:3000/user/login
654  Content-Type: application/json
655  ##
656  {
657    "email": "carrerabelem23@gmail.com",
658    "password": "123456"
659  }
660  ##
661  Send Request
662  POST http://localhost:3000/user/login
663  Content-Type: application/json
664  ##
665  {
666    "email": "carrerabelem23@gmail.com",
667    "password": "123456"
668  }
669  ##
670  Send Request
671  POST http://localhost:3000/user/login
672  Content-Type: application/json
673  ##
674  {
675    "email": "carrerabelem23@gmail.com",
676    "password": "123456"
677  }
678  ##
679  Send Request
680  POST http://localhost:3000/user/login
681  Content-Type: application/json
682  ##
683  {
684    "email": "carrerabelem23@gmail.com",
685    "password": "123456"
686  }
687  ##
688  Send Request
689  POST http://localhost:3000/user/login
690  Content-Type: application/json
691  ##
692  {
693    "email": "carrerabelem23@gmail.com",
694    "password": "123456"
695  }
696  ##
697  Send Request
698  POST http://localhost:3000/user/login
699  Content-Type: application/json
700  ##
701  {
702    "email": "carrerabelem23@gmail.com",
703    "password": "123456"
704  }
705  ##
706  Send Request
707  POST http://localhost:3000/user/login
708  Content-Type: application/json
709  ##
710  {
711    "email": "carrerabelem23@gmail.com",
712    "password": "123456"
713  }
714  ##
715  Send Request
716  POST http://localhost:3000/user/login
717  Content-Type: application/json
718  ##
719  {
720    "email": "carrerabelem23@gmail.com",
721    "password": "123456"
722  }
723  ##
724  Send Request
725  POST http://localhost:3000/user/login
726  Content-Type: application/json
727  ##
728  {
729    "email": "carrerabelem23@gmail.com",
730    "password": "123456"
731  }
732  ##
733  Send Request
734  POST http://localhost:3000/user/login
735  Content-Type: application/json
736  ##
737  {
738    "email": "carrerabelem23@gmail.com",
739    "password": "123456"
740  }
741  ##
742  Send Request
743  POST http://localhost:3000/user/login
744  Content-Type: application/json
745  ##
746  {
747    "email": "carrerabelem23@gmail.com",
748    "password": "123456"
749  }
750  ##
751  Send Request
752  POST http://localhost:3000/user/login
753  Content-Type: application/json
754  ##
755  {
756    "email": "carrerabelem23@gmail.com",
757    "password": "123456"
758  }
759  ##
760  Send Request
761  POST http://localhost:3000/user/login
762  Content-Type: application/json
763  ##
764  {
765    "email": "carrerabelem23@gmail.com",
766    "password": "123456"
767  }
768  ##
769  Send Request
770  POST http://localhost:3000/user/login
771  Content-Type: application/json
772  ##
773  {
774    "email": "carrerabelem23@gmail.com",
775    "password": "123456"
776  }
777  ##
778  Send Request
779  POST http://localhost:3000/user/login
780  Content-Type: application/json
781  ##
782  {
783    "email": "carrerabelem23@gmail.com",
784    "password": "123456"
785  }
786  ##
787  Send Request
788  POST http://localhost:3000/user/login
789  Content-Type: application/json
790  ##
791  {
792    "email": "carrerabelem23@gmail.com",
793    "password": "123456"
794  }
795  ##
796  Send Request
797  POST http://localhost:3000/user/login
798  Content-Type: application/json
799  ##
800  {
801    "email": "carrerabelem23@gmail.com",
802    "password": "123456"
803  }
804  ##
805  Send Request
806  POST http://localhost:3000/user/login
807  Content-Type: application/json
808  ##
809  {
810    "email": "carrerabelem23@gmail.com",
811    "password": "123456"
812  }
813  ##
814  Send Request
815  POST http://localhost:3000/user/login
816  Content-Type: application/json
817  ##
818  {
819    "email": "carrerabelem23@gmail.com",
820    "password": "123456"
821  }
822  ##
823  Send Request
824  POST http://localhost:3000/user/login
825  Content-Type: application/json
826  ##
827  {
828    "email": "carrerabelem23@gmail.com",
829    "password": "123456"
830  }
831  ##
832  Send Request
833  POST http://localhost:3000/user/login
834  Content-Type: application/json
835  ##
836  {
837    "email": "carrerabelem23@gmail.com",
838    "password": "123456"
839  }
840  ##
841  Send Request
842  POST http://localhost:3000/user/login
843  Content-Type: application/json
844  ##
845  {
846    "email": "carrerabelem23@gmail.com",
847    "password": "123456"
848  }
849  ##
850  Send Request
851  POST http://localhost:3000/user/login
852  Content-Type: application/json
853  ##
854  {
855    "email": "carrerabelem23@gmail.com",
856    "password": "123456"
857  }
858  ##
859  Send Request
860  POST http://localhost:3000/user/login
861  Content-Type: application/json
862  ##
863  {
864    "email": "carrerabelem23@gmail.com",
865    "password": "123456"
866  }
867  ##
868  Send Request
869  POST http://localhost:3000/user/login
870  Content-Type: application/json
871  ##
872  {
873    "email": "carrerabelem23@gmail.com",
874    "password": "123456"
875  }
876  ##
877  Send Request
878  POST http://localhost:3000/user/login
879  Content-Type: application/json
880  ##
881  {
882    "email": "carrerabelem23@gmail.com",
883    "password": "123456"
884  }
885  ##
886  Send Request
887  POST http://localhost:3000/user/login
888  Content-Type: application/json
889  ##
890  {
891    "email": "carrerabelem23@gmail.com",
892    "password": "123456"
893  }
894  ##
895  Send Request
896  POST http://localhost:3000/user/login
897  Content-Type: application/json
898  ##
899  {
900    "email": "carrerabelem23@gmail.com",
901    "password": "123456"
902  }
903  ##
904  Send Request
905  POST http://localhost:3000/user/login
906  Content-Type: application/json
907  ##
908  {
909    "email": "carrerabelem23@gmail.com",
910    "password": "123456"
911  }
912  ##
913  Send Request
914  POST http://localhost:3000/user/login
915  Content-Type: application/json
916  ##
917  {
918    "email": "carrerabelem23@gmail.com",
919    "password": "123456"
920  }
921  ##
922  Send Request
923  POST http://localhost:3000/user/login
924  Content-Type: application/json
925  ##
926  {
927    "email": "carrerabelem23@gmail.com",
928    "password": "123456"
929  }
930  ##
931  Send Request
932  POST http://localhost:3000/user/login
933  Content-Type: application/json
934  ##
935  {
936    "email": "carrerabelem23@gmail.com",
937    "password": "123456"
938  }
939  ##
940  Send Request
941  POST http://localhost:3000/user/login
942  Content-Type: application/json
943  ##
944  {
945    "email": "carrerabelem23@gmail.com",
946    "password": "123456"
947  }
948  ##
949  Send Request
950  POST http://localhost:3000/user/login
951  Content-Type: application/json
952  ##
953  {
954    "email": "carrerabelem23@gmail.com",
955    "password": "123456"
956  }
957  ##
958  Send Request
959  POST http://localhost:3000/user/login
960  Content-Type: application/json
961  ##
962  {
963    "email": "carrerabelem23@gmail.com",
964    "password": "123456"
965  }
966  ##
967  Send Request
968  POST http://localhost:3000/user/login
969  Content-Type: application/json
970  ##
971  {
972    "email": "carrerabelem23@gmail.com",
973    "password": "123456"
974  }
975  ##
976  Send Request
977  POST http://localhost:3000/user/login
978  Content-Type: application/json
979  ##
980  {
981    "email": "carrerabelem23@gmail.com",
982    "password": "123456"
983  }
984  ##
985  Send Request
986  POST http://localhost:3000/user/login
987  Content-Type: application/json
988  ##
989  {
990    "email": "carrerabelem23@gmail.com",
991    "password": "123456"
992  }
993  ##
994  Send Request
995  POST http://localhost:3000/user/login
996  Content-Type: application/json
997  ##
998  {
999    "email": "carrerabelem23@gmail.com",
1000   "password": "123456"
1001  }
1002  ##
1003  Send Request
1004  POST http://localhost:3000/user/login
1005  Content-Type: application/json
1006  ##
1007  {
1008    "email": "carrerabelem23@gmail.com",
1009    "password": "123456"
1010  }
1011  ##
1012  Send Request
1013  POST http://localhost:3000/user/login
1014  Content-Type: application/json
1015  ##
1016  {
1017    "email": "carrerabelem23@gmail.com",
1018    "password": "123456"
1019  }
1020  ##
1021  Send Request
1022  POST http://localhost:3000/user/login
1023  Content-Type: application/json
1024  ##
1025  {
1026    "email": "carrerabelem23@gmail.com",
1027    "password": "123456"
1028  }
1029  ##
1030  Send Request
1031  POST http://localhost:3000/user/login
1032  Content-Type: application/json
1033  ##
1034  {
1035    "email": "carrerabelem23@gmail.com",
1036    "password": "123456"
1037  }
1038  ##
1039  Send Request
1040  POST http://localhost:3000/user/login
1041  Content-Type: application/json
1042  ##
1043  {
1044    "email": "carrerabelem23@gmail.com",
1045    "password": "123456"
1046  }
1047  ##
1048  Send Request
1049  POST http://localhost:3000/user/login
1050  Content-Type: application/json
1051  ##
1052  {
1053    "email": "carrerabelem23@gmail.com",
1054    "password": "123456"
1055  }
1056  ##
1057  Send Request
1058  POST http://localhost:3000/user/login
1059  Content-Type: application/json
1060  ##
1061  {
1062    "email": "carrerabelem23@gmail.com",
1063    "password": "123456"
1064  }
1065  ##
1066  Send Request
1067  POST http://localhost:3000/user/login
1068  Content-Type: application/json
1069  ##
1070  {
1071    "email": "carrerabelem23@gmail.com",
1072    "password": "123456"
1073  }
1074  ##
1075  Send Request
1076  POST http://localhost:3000/user/login
1077  Content-Type: application/json
1078  ##
1079  {
1080    "email": "carrerabelem23@gmail.com",
1081    "password": "123456"
1082  }
1083  ##
1084  Send Request
1085  POST http://localhost:3000/user/login
1086  Content-Type: application/json
1087  ##
1088  {
1089    "email": "carrerabelem23@gmail.com",
1090    "password": "123456"
1091  }
1092  ##
1093  Send Request
1094  POST http://localhost:3000/user/login
1095  Content-Type: application/json
1096  ##
1097  {
1098    "email": "carrerabelem23@gmail.com",
1099    "password": "123456"
1100  }
1101  ##
1102  Send Request
1103  POST http://localhost:3000/user/login
1104  Content-Type: application/json
1105  ##
1106  {
1107    "email": "carrerabelem23@gmail.com",
1108    "password": "123456"
1109  }
1110  ##
1111  Send Request
1112  POST http://localhost:3000/user/login
1113  Content-Type: application/json
1114  ##
1115  {
1116    "email": "carrerabelem23@gmail.com",
1117    "password": "123456"
1118  }
1119  ##
1120  Send Request
1121  POST http://localhost:3000/user/login
1122  Content-Type: application/json
1123  ##
1124  {
1125    "email": "carrerabelem23@gmail.com",
1126    "password": "123456"
1127  }
1128  ##
1129  Send Request
1130  POST http://localhost:3000/user/login
1131  Content-Type: application/json
1132  ##
1133  {
1134    "email": "carrerabelem23@gmail.com",
1135    "password": "123456"
1136  }
1137
```



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

El archivo **user.http** se utiliza para probar manualmente las rutas del módulo de usuarios de la API. Contiene solicitudes HTTP que permiten listar usuarios, crear usuarios, actualizar datos, eliminar usuarios y realizar el inicio de sesión, enviando información en formato JSON. Este archivo facilita la validación del correcto funcionamiento del CRUD de usuarios y del proceso de autenticación durante el desarrollo.

```
EXPLORADOR  ...  pokemon.http U X
src > test > pokemon.http > GET /pokemon/random
Send Request
1 GET http://localhost:3000/pokemon/seed
2
3 ###
Send Request
4 GET http://localhost:3000/pokemon/random
```

El archivo **pokemon.http** se utiliza para probar las rutas del módulo de Pokémon de la API. Incluye solicitudes HTTP para cargar los Pokémon en la base de datos mediante la ruta **/pokemon/seed** y para obtener Pokémon aleatorios a través de la ruta **/pokemon/random**. Este archivo permite verificar rápidamente el correcto funcionamiento de las rutas y controladores durante el desarrollo.

```
EXPLORADOR  ...  game.http U X
src > test > game.http > GET /game/win/1
Send Request
1 get http://localhost:3000/game/win/1
2
3 ###
Send Request
5 GET http://localhost:3000/game/lose/4
```



INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

El archivo **game.http** se utiliza para probar las rutas del módulo de juegos de la API. Contiene solicitudes HTTP que permiten registrar victorias y derrotas de usuarios mediante peticiones GET, facilitando la verificación del correcto funcionamiento de las rutas y controladores durante el desarrollo.

```
src > JS app.js > ...
1  const express = require('express');
2  const cors = require('cors');
3
4  const env = require('./config/environment');
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
17     this.app.use(express.json());
18   }
19
20   start() {
21     this.app.listen(this.port, () => {
22       console.log('Servidor esta ejecutando en el puerto ${this.port}');
23     });
24   }
25
26   routes() {
27   }
28
29   routes() {
30     this.app.get('/', (req, res) => {
31       res.send('Hola Mundo!');
32     });
33
34     this.app.use('/user', require('./routes/user'));
35     this.app.use('/pokemon', require('./routes/pokemon'));
36     this.app.use('/game', require('./routes/game'));
37   }
38 }
39 module.exports = App;
```

El archivo **app.js** es el núcleo de la aplicación. En él se configura el servidor con Express, se habilitan los middlewares como cors y el manejo de JSON, se define el puerto usando las variables de entorno y se registran las rutas principales (/user, /pokemon, /game). Además, incluye una ruta básica de prueba y el método que inicia el servidor, dejando la API lista para recibir peticiones.



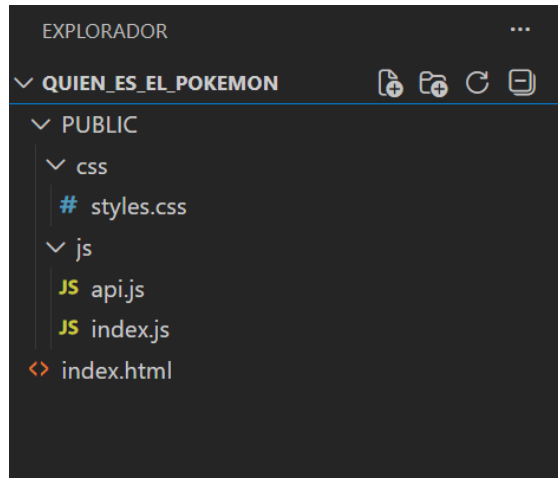
INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

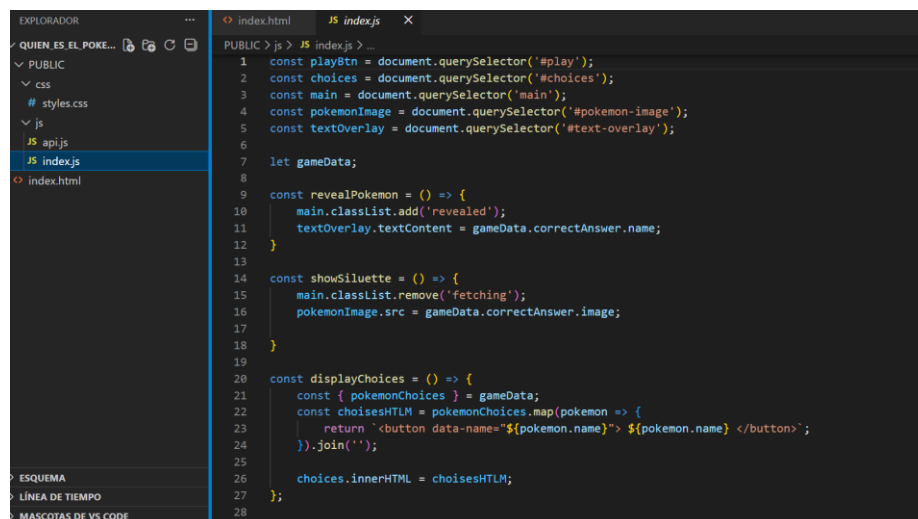
Formato para prácticas de Laboratorio

Parte 2: ¿Quién es el Pokémon?

Se desarrolló una aplicación cliente que consume los servicios de la *mini_pokeapi*. La aplicación realiza peticiones HTTP para obtener información de los Pokémon.



El proyecto **QUIÉN_ES_EL_POKEMON** es el frontend del juego. **index.html** muestra la interfaz, **styles.css** define el diseño, **api.js** se comunica con la API para obtener los Pokémon y **index.js** controla la lógica del juego y la interacción del usuario.





INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
20 const displayChoices = () => {
21   ...
26   choices.innerHTML = choicesHTML;
27 };
28
29 const resetImage = () => {
30   pokemonImage.src = '';
31   main.classList.add('fetching');
32   main.classList.remove('revealed');
33 }
34
35 const fetchData = async () => {
36   resetImage();
37   gameData = await window.getPokeData();
38   showSilhouette();
39   displayChoices();
40 }
41
42
43 playBtn.addEventListener('click', fetchData);
44
45 choices.addEventListener('click', (e) => {
46   const { name } = e.target.dataset;
47   const resultClass = (name === gameData.correctAnswer.name) ? 'correct' : 'incorrect';
48   e.target.classList.add(resultClass);
49   revealPokemon();
50 });
```

El archivo **index.js** controla la **lógica del juego “¿Quién es el Pokémon?”**. Se encarga de obtener los elementos del DOM, solicitar los datos del Pokémon a la API, mostrar la silueta y las opciones de respuesta, manejar los clics del usuario y revelar el Pokémon correcto indicando si la respuesta fue correcta o incorrecta.

```
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   const randomIndex: number
13   const correctAnswer = pokemon[randomIndex];
14
15   //const correctAnswer = pokemon.find(p => p.isCorrect);
16
17   return {
18     pokemonChoices: pokemon,
19     correctAnswer,
20   }
21   //console.log(pokemon);
22 }
23
24
25
26
```

El archivo **api.js** se encarga de la **comunicación con el backend**. Realiza una petición a la ruta **/pokemon/random** para obtener un conjunto de Pokémon, selecciona uno de forma aleatoria como **respuesta correcta** y devuelve tanto las **opciones de Pokémon** como el **Pokémon correcto**, datos que luego utiliza el archivo **index.js** para ejecutar el juego.



INSTITUTO TECNOLÓGICO DE TUXTEPEC

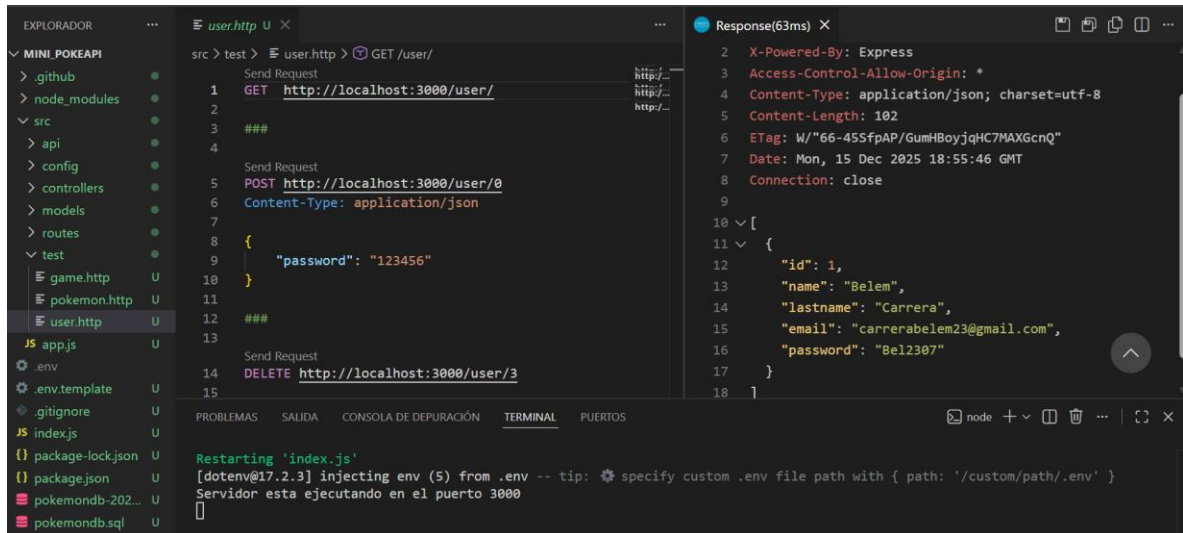
Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

```
EXPLORADOR --- index.html # styles.css
✓ QUIEN ES EL POKEMON
  PUBLIC
  ✓ PUBLIC
  # styles.css
  ✓ js
  JS apijs
  JS indexjs
  index.html
  ESQUEMA
  LINEA DE TIEMPO
  MASCOTAS DE VS CODE

PUBLIC > css > # styles.css > % main
1 body {
2   margin: 0;
3   background-color: #000;
4   color: #fff;
5 }
6
7 main {
8   position: relative;
9   width: 575px;
10  background-image: url(
11    (https://res.cloudinary.com/dzynq10l/image/upload/v1633196203/Msc/pokemon-bg_1g4uv3.jpg)
12  );
13  background-size: 110%;
14  background-repeat: no-repeat;
15 }
16
17 #pokemon-container {
18   height: 355px;
19 }
20
21 #answer {
22   display: none;
23 }
24
25 #bg-overlay {
26   color: #fff;
27   position: absolute;
28   background-color: #dd1716;
29   right: 25px;
30 }
31
32 #text-overlay {
33   position: absolute;
34   right: 20px;
35   top: 100px;
36   height: 170px;
37   top: 60px;
38   border-radius: 50%;
39 }
40
41 #controls {
42   position: relative;
43   right: 20px;
44   top: 100px;
45   background-color: #ffeb00;
46   border: solid 7px #2c70ae;
47   padding: 20px;
48   min-width: 130px;
49   font-family: 'Bangers', Arial, sans-serif;
50   font-size: 36px;
51   letter-spacing: 1.2px;
52   color: #9ea22f;
53   text-shadow: 2px 2px #d3366c;
54   border-radius: 20px;
55   text-align: center;
56 }
57
58 #pokemon-image {
59   margin: 135px 0 140px;
60   width: 75px;
61   transform: scale(2.5);
62   filter: brightness(0);
63   transition: filter .5s ease-out;
64 }
65
66 #pokeball {
67   width: 78px;
68   margin: 144px 0 144px;
69   position: absolute;
70   visibility: hidden;
71   animation: wiggle 1.35s infinite;
72 }
73
74 #play {
75   position: absolute;
76   top: -56px;
77   left: 39px;
78 }
79
80 button.correct {
81   background: #0df5d6;
82 }
83
84 button.incorrect {
85   background: #ea2e25;
86 }
87
88 #pokemon-image {
89   margin: 135px 0 140px;
90   width: 75px;
91   transform: scale(2.5);
92   filter: brightness(0);
93   transition: filter .5s ease-out;
94 }
95
96 #pokeball {
97   width: 78px;
98   margin: 144px 0 144px;
99   position: absolute;
100  visibility: hidden;
101  animation: wiggle 1.35s infinite;
102 }
103
104 #play {
105   position: absolute;
106   top: -56px;
107   left: 39px;
108 }
109
110 button.correct {
111   background: #0df5d6;
112 }
113
114 button.incorrect {
115   background: #ea2e25;
116 }
117
118 #pokemon-image {
119   margin: 135px 0 140px;
120   width: 75px;
121   transform: scale(2.5);
122   filter: brightness(0);
123   transition: filter .5s ease-out;
124 }
125
126 #pokeball {
127   width: 78px;
128   margin: 144px 0 144px;
129   position: absolute;
130   visibility: hidden;
131   animation: wiggle 1.35s infinite;
132 }
133
134 #play {
135   position: absolute;
136   top: -56px;
137   left: 39px;
138 }
139
140 button.correct {
141   background: #0df5d6;
142 }
143
144 button.incorrect {
145   background: #ea2e25;
146 }
147
148 #pokemon-image {
149   margin: 135px 0 140px;
150   width: 75px;
151   transform: scale(2.5);
152   filter: brightness(0);
153   transition: filter .5s ease-out;
154 }
155
156 #pokeball {
157   width: 78px;
158   margin: 144px 0 144px;
159   position: absolute;
160   visibility: hidden;
161   animation: wiggle 1.35s infinite;
162 }
163
164 #play {
165   position: absolute;
166   top: -56px;
167   left: 39px;
168 }
169
170 button.correct {
171   background: #0df5d6;
172 }
173
174 button.incorrect {
175   background: #ea2e25;
176 }
177
178 #pokemon-image {
179   margin: 135px 0 140px;
180   width: 75px;
181   transform: scale(2.5);
182   filter: brightness(0);
183   transition: filter .5s ease-out;
184 }
185
186 #pokeball {
187   width: 78px;
188   margin: 144px 0 144px;
189   position: absolute;
190   visibility: hidden;
191   animation: wiggle 1.35s infinite;
192 }
193
194 #play {
195   position: absolute;
196   top: -56px;
197   left: 39px;
198 }
199
200 button.correct {
201   background: #0df5d6;
202 }
203
204 button.incorrect {
205   background: #ea2e25;
206 }
207
208 #pokemon-image {
209   margin: 135px 0 140px;
210   width: 75px;
211   transform: scale(2.5);
212   filter: brightness(0);
213   transition: filter .5s ease-out;
214 }
215
216 #pokeball {
217   width: 78px;
218   margin: 144px 0 144px;
219   position: absolute;
220   visibility: hidden;
221   animation: wiggle 1.35s infinite;
222 }
223
224 #play {
225   position: absolute;
226   top: -56px;
227   left: 39px;
228 }
229
230 button.correct {
231   background: #0df5d6;
232 }
233
234 button.incorrect {
235   background: #ea2e25;
236 }
237
238 #pokemon-image {
239   margin: 135px 0 140px;
240   width: 75px;
241   transform: scale(2.5);
242   filter: brightness(0);
243   transition: filter .5s ease-out;
244 }
245
246 #pokeball {
247   width: 78px;
248   margin: 144px 0 144px;
249   position: absolute;
250   visibility: hidden;
251   animation: wiggle 1.35s infinite;
252 }
253
254 #play {
255   position: absolute;
256   top: -56px;
257   left: 39px;
258 }
259
260 button.correct {
261   background: #0df5d6;
262 }
263
264 button.incorrect {
265   background: #ea2e25;
266 }
267
268 #pokemon-image {
269   margin: 135px 0 140px;
270   width: 75px;
271   transform: scale(2.5);
272   filter: brightness(0);
273   transition: filter .5s ease-out;
274 }
275
276 #pokeball {
277   width: 78px;
278   margin: 144px 0 144px;
279   position: absolute;
280   visibility: hidden;
281   animation: wiggle 1.35s infinite;
282 }
283
284 #play {
285   position: absolute;
286   top: -56px;
287   left: 39px;
288 }
289
290 button.correct {
291   background: #0df5d6;
292 }
293
294 button.incorrect {
295   background: #ea2e25;
296 }
297
298 #pokemon-image {
299   margin: 135px 0 140px;
300   width: 75px;
301   transform: scale(2.5);
302   filter: brightness(0);
303   transition: filter .5s ease-out;
304 }
305
306 #pokeball {
307   width: 78px;
308   margin: 144px 0 144px;
309   position: absolute;
310   visibility: hidden;
311   animation: wiggle 1.35s infinite;
312 }
313
314 #play {
315   position: absolute;
316   top: -56px;
317   left: 39px;
318 }
319
320 button.correct {
321   background: #0df5d6;
322 }
323
324 button.incorrect {
325   background: #ea2e25;
326 }
327
328 #pokemon-image {
329   margin: 135px 0 140px;
330   width: 75px;
331   transform: scale(2.5);
332   filter: brightness(0);
333   transition: filter .5s ease-out;
334 }
335
336 #pokeball {
337   width: 78px;
338   margin: 144px 0 144px;
339   position: absolute;
340   visibility: hidden;
341   animation: wiggle 1.35s infinite;
342 }
343
344 #play {
345   position: absolute;
346   top: -56px;
347   left: 39px;
348 }
349
350 button.correct {
351   background: #0df5d6;
352 }
353
354 button.incorrect {
355   background: #ea2e25;
356 }
357
358 #pokemon-image {
359   margin: 135px 0 140px;
360   width: 75px;
361   transform: scale(2.5);
362   filter: brightness(0);
363   transition: filter .5s ease-out;
364 }
365
366 #pokeball {
367   width: 78px;
368   margin: 144px 0 144px;
369   position: absolute;
370   visibility: hidden;
371   animation: wiggle 1.35s infinite;
372 }
373
374 #play {
375   position: absolute;
376   top: -56px;
377   left: 39px;
378 }
379
380 button.correct {
381   background: #0df5d6;
382 }
383
384 button.incorrect {
385   background: #ea2e25;
386 }
387
388 #pokemon-image {
389   margin: 135px 0 140px;
390   width: 75px;
391   transform: scale(2.5);
392   filter: brightness(0);
393   transition: filter .5s ease-out;
394 }
395
396 #pokeball {
397   width: 78px;
398   margin: 144px 0 144px;
399   position: absolute;
400   visibility: hidden;
401   animation: wiggle 1.35s infinite;
402 }
403
404 #play {
405   position: absolute;
406   top: -56px;
407   left: 39px;
408 }
409
410 button.correct {
411   background: #0df5d6;
412 }
413
414 button.incorrect {
415   background: #ea2e25;
416 }
417
418 #pokemon-image {
419   margin: 135px 0 140px;
420   width: 75px;
421   transform: scale(2.5);
422   filter: brightness(0);
423   transition: filter .5s ease-out;
424 }
425
426 #pokeball {
427   width: 78px;
428   margin: 144px 0 144px;
429   position: absolute;
430   visibility: hidden;
431   animation: wiggle 1.35s infinite;
432 }
433
434 #play {
435   position: absolute;
436   top: -56px;
437   left: 39px;
438 }
439
440 button.correct {
441   background: #0df5d6;
442 }
443
444 button.incorrect {
445   background: #ea2e25;
446 }
447
448 #pokemon-image {
449   margin: 135px 0 140px;
450   width: 75px;
451   transform: scale(2.5);
452   filter: brightness(0);
453   transition: filter .5s ease-out;
454 }
455
456 #pokeball {
457   width: 78px;
458   margin: 144px 0 144px;
459   position: absolute;
460   visibility: hidden;
461   animation: wiggle 1.35s infinite;
462 }
463
464 #play {
465   position: absolute;
466   top: -56px;
467   left: 39px;
468 }
469
470 button.correct {
471   background: #0df5d6;
472 }
473
474 button.incorrect {
475   background: #ea2e25;
476 }
477
478 #pokemon-image {
479   margin: 135px 0 140px;
480   width: 75px;
481   transform: scale(2.5);
482   filter: brightness(0);
483   transition: filter .5s ease-out;
484 }
485
486 #pokeball {
487   width: 78px;
488   margin: 144px 0 144px;
489   position: absolute;
490   visibility: hidden;
491   animation: wiggle 1.35s infinite;
492 }
493
494 #play {
495   position: absolute;
496   top: -56px;
497   left: 39px;
498 }
499
500 button.correct {
501   background: #0df5d6;
502 }
503
504 button.incorrect {
505   background: #ea2e25;
506 }
507
508 #pokemon-image {
509   margin: 135px 0 140px;
510   width: 75px;
511   transform: scale(2.5);
512   filter: brightness(0);
513   transition: filter .5s ease-out;
514 }
515
516 #pokeball {
517   width: 78px;
518   margin: 144px 0 144px;
519   position: absolute;
520   visibility: hidden;
521   animation: wiggle 1.35s infinite;
522 }
523
524 #play {
525   position: absolute;
526   top: -56px;
527   left: 39px;
528 }
529
530 button.correct {
531   background: #0df5d6;
532 }
533
534 button.incorrect {
535   background: #ea2e25;
536 }
537
538 #pokemon-image {
539   margin: 135px 0 140px;
540   width: 75px;
541   transform: scale(2.5);
542   filter: brightness(0);
543   transition: filter .5s ease-out;
544 }
545
546 #pokeball {
547   width: 78px;
548   margin: 144px 0 144px;
549   position: absolute;
550   visibility: hidden;
551   animation: wiggle 1.35s infinite;
552 }
553
554 #play {
555   position: absolute;
556   top: -56px;
557   left: 39px;
558 }
559
560 button.correct {
561   background: #0df5d6;
562 }
563
564 button.incorrect {
565   background: #ea2e25;
566 }
567
568 #pokemon-image {
569   margin: 135px 0 140px;
570   width: 75px;
571   transform: scale(2.5);
572   filter: brightness(0);
573   transition: filter .5s ease-out;
574 }
575
576 #pokeball {
577   width: 78px;
578   margin: 144px 0 144px;
579   position: absolute;
580   visibility: hidden;
581   animation: wiggle 1.35s infinite;
582 }
583
584 #play {
585   position: absolute;
586   top: -56px;
587   left: 39px;
588 }
589
590 button.correct {
591   background: #0df5d6;
592 }
593
594 button.incorrect {
595   background: #ea2e25;
596 }
597
598 #pokemon-image {
599   margin: 135px 0 140px;
600   width: 75px;
601   transform: scale(2.5);
602   filter: brightness(0);
603   transition: filter .5s ease-out;
604 }
605
606 #pokeball {
607   width: 78px;
608   margin: 144px 0 144px;
609   position: absolute;
610   visibility: hidden;
611   animation: wiggle 1.35s infinite;
612 }
613
614 #play {
615   position: absolute;
616   top: -56px;
617   left: 39px;
618 }
619
620 button.correct {
621   background: #0df5d6;
622 }
623
624 button.incorrect {
625   background: #ea2e25;
626 }
627
628 #pokemon-image {
629   margin: 135px 0 140px;
630   width: 75px;
631   transform: scale(2.5);
632   filter: brightness(0);
633   transition: filter .5s ease-out;
634 }
635
636 #pokeball {
637   width: 78px;
638   margin: 144px 0 144px;
639   position: absolute;
640   visibility: hidden;
641   animation: wiggle 1.35s infinite;
642 }
643
644 #play {
645   position: absolute;
646   top: -56px;
647   left: 39px;
648 }
649
650 button.correct {
651   background: #0df5d6;
652 }
653
654 button.incorrect {
655   background: #ea2e25;
656 }
657
658 #pokemon-image {
659   margin: 135px 0 140px;
660   width: 75px;
661   transform: scale(2.5);
662   filter: brightness(0);
663   transition: filter .5s ease-out;
664 }
665
666 #pokeball {
667   width: 78px;
668   margin: 144px 0 144px;
669   position: absolute;
670   visibility: hidden;
671   animation: wiggle 1.35s infinite;
672 }
673
674 #play {
675   position: absolute;
676   top: -56px;
677   left: 39px;
678 }
679
680 button.correct {
681   background: #0df5d6;
682 }
683
684 button.incorrect {
685   background: #ea2e25;
686 }
687
688 #pokemon-image {
689   margin: 135px 0 140px;
690   width: 75px;
691   transform: scale(2.5);
692   filter: brightness(0);
693   transition: filter .5s ease-out;
694 }
695
696 #pokeball {
697   width: 78px;
698   margin: 144px 0 144px;
699   position: absolute;
700   visibility: hidden;
701   animation: wiggle 1.35s infinite;
702 }
703
704 #play {
705   position: absolute;
706   top: -56px;
707   left: 39px;
708 }
709
710 button.correct {
711   background: #0df5d6;
712 }
713
714 button.incorrect {
715   background: #ea2e25;
716 }
717
718 #pokemon-image {
719   margin: 135px 0 140px;
720   width: 75px;
721   transform: scale(2.5);
722   filter: brightness(0);
723   transition: filter .5s ease-out;
724 }
725
726 #pokeball {
727   width: 78px;
728   margin: 144px 0 144px;
729   position: absolute;
730   visibility: hidden;
731   animation: wiggle 1.35s infinite;
732 }
733
734 #play {
735   position: absolute;
736   top: -56px;
737   left: 39px;
738 }
739
740 button.correct {
741   background: #0df5d6;
742 }
743
744 button.incorrect {
745   background: #ea2e25;
746 }
747
748 #pokemon-image {
749   margin: 135px 0 140px;
750   width: 75px;
751   transform: scale(2.5);
752   filter: brightness(0);
753   transition: filter .5s ease-out;
754 }
755
756 #pokeball {
757   width: 78px;
758   margin: 144px 0 144px;
759   position: absolute;
760   visibility: hidden;
761   animation: wiggle 1.35s infinite;
762 }
763
764 #play {
765   position: absolute;
766   top: -56px;
767   left: 39px;
768 }
769
770 button.correct {
771   background: #0df5d6;
772 }
773
774 button.incorrect {
775   background: #ea2e25;
776 }
777
778 #pokemon-image {
779   margin: 135px 0 140px;
780   width: 75px;
781   transform: scale(2.5);
782   filter: brightness(0);
783   transition: filter .5s ease-out;
784 }
785
786 #pokeball {
787   width: 78px;
788   margin: 144px 0 144px;
789   position: absolute;
790   visibility: hidden;
791   animation: wiggle 1.35s infinite;
792 }
793
794 #play {
795   position: absolute;
796   top: -56px;
797   left: 39px;
798 }
799
800 button.correct {
801   background: #0df5d6;
802 }
803
804 button.incorrect {
805   background: #ea2e25;
806 }
807
808 #pokemon-image {
809   margin: 135px 0 140px;
810   width: 75px;
811   transform: scale(2.5);
812   filter: brightness(0);
813   transition: filter .5s ease-out;
814 }
815
816 #pokeball {
817   width: 78px;
818   margin: 144px 0 144px;
819   position: absolute;
820   visibility: hidden;
821   animation: wiggle 1.35s infinite;
822 }
823
824 #play {
825   position: absolute;
826   top: -56px;
827   left: 39px;
828 }
829
830 button.correct {
831   background: #0df5d6;
832 }
833
834 button.incorrect {
835   background: #ea2e25;
836 }
837
838 #pokemon-image {
839   margin: 135px 0 140px;
840   width: 75px;
841   transform: scale(2.5);
842   filter: brightness(0);
843   transition: filter .5s ease-out;
844 }
845
846 #pokeball {
847   width: 78px;
848   margin: 144px 0 144px;
849   position: absolute;
850   visibility: hidden;
851   animation: wiggle 1.35s infinite;
852 }
853
854 #play {
855   position: absolute;
856   top: -56px;
857   left: 39px;
858 }
859
860 button.correct {
861   background: #0df5d6;
862 }
863
864 button.incorrect {
865   background: #ea2e25;
866 }
867
868 #pokemon-image {
869   margin: 135px 0 140px;
870   width: 75px;
871   transform: scale(2.5);
872   filter: brightness(0);
873   transition: filter .5s ease-out;
874 }
875
876 #pokeball {
877   width: 78px;
878   margin: 144px 0 144px;
879   position: absolute;
880   visibility: hidden;
881   animation: wiggle 1.35s infinite;
882 }
883
884 #play {
885   position: absolute;
886   top: -56px;
887   left: 39px;
888 }
889
890 button.correct {
891   background: #0df5d6;
892 }
893
894 button.incorrect {
895   background: #ea2e25;
896 }
897
898 #pokemon-image {
899   margin: 135px 0 140px;
900   width: 75px;
901   transform: scale(2.5);
902   filter: brightness(0);
903   transition: filter .5s ease-out;
904 }
905
906 #pokeball {
907   width: 78px;
908   margin: 144px 0 144px;
909   position: absolute;
910   visibility: hidden;
911   animation: wiggle 1.35s infinite;
912 }
913
914 #play {
915   position: absolute;
916   top: -56px;
917   left: 39px;
918 }
919
920 button.correct {
921   background: #0df5d6;
922 }
923
924 button.incorrect {
925   background: #ea2e25;
926 }
927
928 #pokemon-image {
929   margin: 135px 0 140px;
930   width: 75px;
931   transform: scale(2.5);
932   filter: brightness(0);
933   transition: filter .5s ease-out;
934 }
935
936 #pokeball {
937   width: 78px;
938   margin: 144px 0 144px;
939   position: absolute;
940   visibility: hidden;
941   animation: wiggle 1.35s infinite;
942 }
943
944 #play {
945   position: absolute;
946   top: -56px;
947   left: 39px;
948 }
949
950 button.correct {
951   background: #0df5d6;
952 }
953
954 button.incorrect {
955   background: #ea2e25;
956 }
957
958 #pokemon-image {
959   margin: 135px 0 140px;
960   width: 75px;
961   transform: scale(2.5);
962   filter: brightness(0);
963   transition: filter .5s ease-out;
964 }
965
966 #pokeball {
967   width: 78px;
968   margin: 144px 0 144px;
969   position: absolute;
970   visibility: hidden;
971   animation: wiggle 1.35s infinite;
972 }
973
974 #play {
975   position: absolute;
976   top: -56px;
977   left: 39px;
978 }
979
980 button.correct {
981   background: #0df5d6;
982 }
983
984 button.incorrect {
985   background: #ea2e25;
986 }
987
988 #pokemon-image {
989   margin: 135px 0 140px;
990   width: 75px;
991   transform: scale(2.5);
992   filter: brightness(0);
993   transition: filter .5s ease-out;
994 }
995
996 #pokeball {
997   width: 78px;
998   margin: 144px 0 144px;
999   position: absolute;
1000  visibility: hidden;
1001  animation: wiggle 1.35s infinite;
1002 }
1003
1004 #play {
1005   position: absolute;
1006   top: -56px;
1007   left: 39px;
1008 }
1009
1010 button.correct {
1011   background: #0df5d6;
1012 }
1013
1014 button.incorrect {
1015   background: #ea2e25;
1016 }
1017
1018 #pokemon-image {
1019   margin: 135px 0 140px;
1020   width: 75px;
1021   transform: scale(2.5);
1022   filter: brightness(0);
1023   transition: filter .5s ease-out;
1024 }
1025
1026 #pokeball {
1027   width: 78px;
1028   margin: 144px 0 144px;
1029   position: absolute;
1030   visibility: hidden;
1031   animation: wiggle 1.35s infinite;
1032 }
1033
1034 #play {
1035   position: absolute;
1036   top: -56px;
1037   left: 39px;
1038 }
1039
1040 button.correct {
1041   background: #0df5d6;
1042 }
1043
1044 button.incorrect {
1045   background: #ea2e25;
1046 }
1047
1048 #pokemon-image {
1049   margin: 135px 0 140px;
1050   width: 75px;
1051   transform: scale(2.5);
1052   filter: brightness(0);
1053   transition: filter .5s ease-out;
1054 }
1055
1056 #pokeball {
1057   width: 78px;
1058   margin: 144px 0 144px;
1059   position: absolute;
1060   visibility: hidden;
1061   animation: wiggle 1.35s infinite;
1062 }
1063
1064 #play {
1065   position: absolute;
1066   top: -56px;
1067   left: 39px;
1068 }
1069
1070 button.correct {
1071   background: #0df5d6;
1072 }
1073
1074 button.incorrect {
1075   background: #ea2e25;
1076 }
1077
1078 #pokemon-image {
1079   margin: 135px 0 140px;
1080   width: 75px;
1081   transform: scale(2.5);
1082   filter: brightness(0);
1083   transition: filter .5s ease-out;
1084 }
1085
1086 #pokeball {
1087   width: 78px;
1088   margin: 144px 0 144px;
1089   position: absolute;
1090   visibility: hidden;
1091   animation: wiggle 1.35s infinite;
1092 }
1093
1094 #play {
1095   position: absolute;
1096   top: -56px;
1097   left: 39px;
1098 }
1099
1100 button.correct {
1101   background: #0df5d6;
1102 }
1103
1104 button.incorrect {
1105   background: #ea2e25;
1106 }
1107
1108 #pokemon-image {
1109   margin: 135px 0 140px;
1110   width: 75px;
1111   transform: scale(2.5);
1112   filter: brightness(0);
1113   transition: filter .5s ease-out;
1114 }
1115
1116 #pokeball {
1117   width: 78px;
1118   margin: 144px 0 144px;
1119   position: absolute;
1120   visibility: hidden;
1121   animation: wiggle 1.35s infinite;
1122 }
1123
1124 #play {
1125   position: absolute;
1126   top: -56px;
1127   left: 39px;
1128 }
1129
1130 button.correct {
1131   background: #0df5d6;
1132 }
1133
1134 button.incorrect {
1135   background: #ea2e25;
1136 }
1137
1138 #pokemon-image {
1139   margin: 135px 0 140px;
1140   width: 75px;
1141   transform: scale(2.5);
1142   filter: brightness(0);
1143   transition: filter .5s ease-out;
1144 }
1145
1146 #pokeball {
1147   width: 78px;
1148   margin: 144px 0 144px;
1149   position: absolute;
1150   visibility: hidden;
1151   animation: wiggle 1.35s infinite;
1152 }
1153
1154 #play {
1155   position: absolute;
1156   top: -56px;
1157   left: 39px;
1158 }
1159
1160 button.correct {
1161   background: #0df5d6;
1162 }
1163
1164 button.incorrect {
1165   background: #ea2e25;
1166 }
1167
1168 #pokemon-image {
1169   margin: 135px 0 140px;
1170   width: 75px;
1171   transform: scale(2.5);
1172   filter: brightness(0);
1173   transition: filter .5s ease-out;
1174 }
1175
1176 #pokeball {
1177   width: 78px;
1178   margin: 144px 0 144px;
1179   position: absolute;
1180   visibility: hidden;
1181   animation: wiggle 1.35s infinite;
1182 }
1183
1184 #play {
1185   position: absolute;
1186   top: -56px;
1187   left: 39px;
1188 }
1189
1190 button.correct {
1191   background: #0df5d6;
1192 }
1193
1194 button.incorrect {
1195   background: #ea2e25;
1196 }
1197
1198 #pokemon-image {
1199   margin: 135px 0 140px;
1200   width: 75px;
1201   transform: scale(2.5);
1202   filter: brightness(0);
1203   transition: filter .5s ease-out;
1204 }
1205
1206 #pokeball {
1207   width: 78px;
1208   margin: 144px 0 144px;
1209   position: absolute;
1210   visibility: hidden;
1211   animation: wiggle 1.35s infinite;
1212 }
1213
1214 #play {
1215   position: absolute;
1216   top: -56px;
1217   left: 39px;
1218 }
1219
1220 button.correct {
1221   background: #0df5d6;
1222 }
1223
1224 button.incorrect {
1225   background: #ea2e25;
1226 }
1227
1228 #pokemon-image {
1229   margin: 135px 0 140px;
1230   width: 75px;
1231   transform: scale(2.5);
1232   filter: brightness(0);
1233   transition: filter .5s ease-out;
1234 }
1235
1236 #pokeball {
1237   width: 78px;
1238   margin: 144px 0 144px;
1239   position: absolute;
1240   visibility: hidden;
1241   animation: wiggle 1.35s infinite;
1242 }
1243
1244 #play {
1245   position: absolute;
1246   top: -56px;
1247   left: 39px;
1248 }
1249
1250 button.correct {
1251   background: #0df5d6;
1252 }
1253
1254 button.incorrect {
1255   background: #ea2e25;
1256 }
1257
1258 #pokemon-image {
1259   margin: 135px 0 140px;
1260   width: 75px;
1261   transform: scale(2.5);
1262   filter: brightness(0);
1263   transition: filter .5s ease-out;
1264 }
1265
1266 #pokeball {
1267   width: 78px;
1268   margin: 144px 0 144px;
1269   position: absolute;
1270   visibility: hidden;
1271   animation: wiggle 1.35s
```

6. Resultados y conclusiones



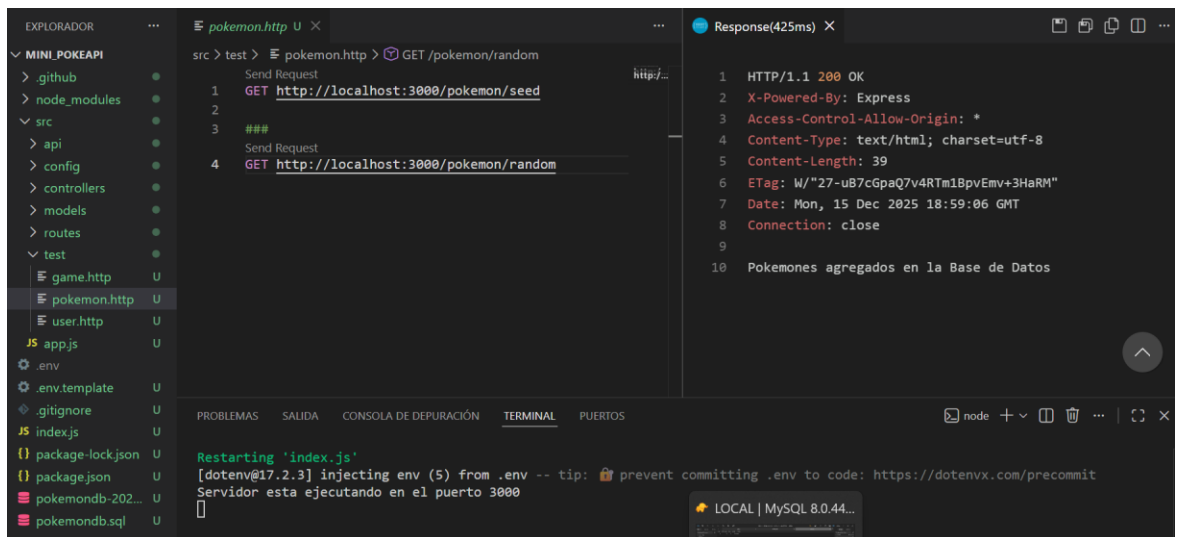
```
EXPLORADOR  ...  user.http  X
MINI_POKEAPI
  > .github
  > node_modules
  > src
    > api
    > config
    > controllers
    > models
    > routes
    > test
      game.http  U
      pokemon.http  U
      user.http  U
  JS app.js  U
  .env
  .env.template  U
  .gitignore  U
  JS index.js  U
  package-lock.json  U
  package.json  U
  pokemondb-202...  U
  pokemondb.sql  U

src > test > user.http > GET /user/
1 Send Request
2 GET http://localhost:3000/user/
3 ###
4
5 Send Request
6 POST http://localhost:3000/user/0
7 Content-Type: application/json
8 {
9   "password": "123456"
10 }
11 ###
12
13 Send Request
14 DELETE http://localhost:3000/user/3
15

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
Restarting 'index.js'
[dotenv@17.2.3] injecting env (5) from .env -- tip: specify custom .env file path with { path: '/custom/path/.env' }
Servidor esta ejecutando en el puerto 3000

Response(63ms)  X
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 102
6 ETag: W/"66-45SfpAP/GumHBoyjqHC7MAXGcnQ"
7 Date: Mon, 15 Dec 2025 18:55:46 GMT
8 Connection: close
9
10 [
11   {
12     "id": 1,
13     "name": "Belem",
14     "lastname": "Carrera",
15     "email": "carrerabelem23@gmail.com",
16     "password": "Bel2307"
17   }
18 ]
```

La prueba del endpoint `/user` usando un archivo `user.http`. Se realiza una petición **GET** a `http://localhost:3000/user/`, y el servidor responde correctamente con un **JSON** que contiene la lista de usuarios registrados. La respuesta confirma que el **servidor Express está activo en el puerto 3000**, que CORS está habilitado y que la API devuelve los datos del usuario (id, nombre, apellido, correo y contraseña) desde la base de datos, validando así el funcionamiento correcto del módulo de usuarios y su conexión con el backend.



```
EXPLORADOR  ...  pokemon.http  X
MINI_POKEAPI
  > .github
  > node_modules
  > src
    > api
    > config
    > controllers
    > models
    > routes
    > test
      game.http  U
      pokemon.http  U
      user.http  U
  JS app.js  U
  .env
  .env.template  U
  .gitignore  U
  JS index.js  U
  package-lock.json  U
  package.json  U
  pokemondb-202...  U
  pokemondb.sql  U

src > test > pokemon.http > GET /pokemon/random
1 Send Request
2 GET http://localhost:3000/pokemon/seed
3 ###
4 GET http://localhost:3000/pokemon/random

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
Restarting 'index.js'
[dotenv@17.2.3] injecting env (5) from .env -- tip: prevent committing .env to code: https://dotenvx.com/precommit
Servidor esta ejecutando en el puerto 3000

Response(425ms)  X
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 39
6 ETag: W/"27-uB7cGpaQ7v4RTm18pvEmv+3HaRM"
7 Date: Mon, 15 Dec 2025 18:59:06 GMT
8 Connection: close
9
10 Pokemones agregados en la Base de Datos
```




INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio

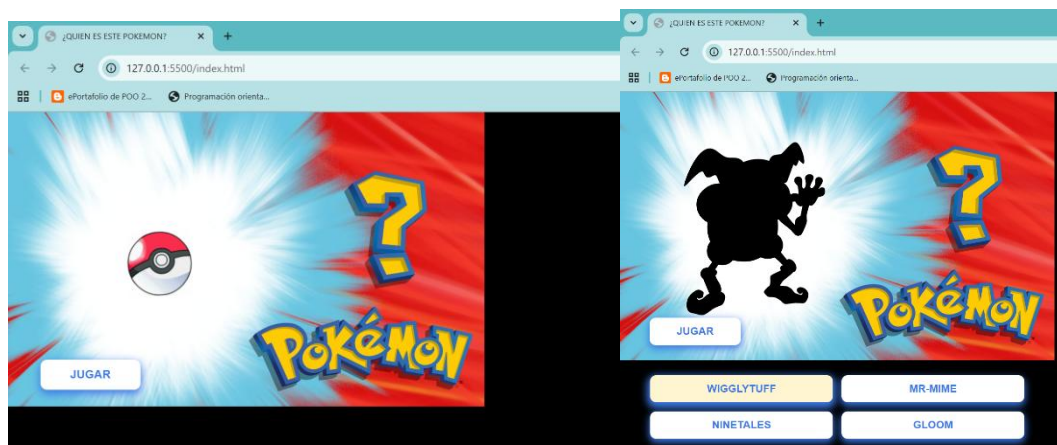
la **prueba del endpoint /pokemon** desde el archivo **pokemon.http**. Primero se ejecuta la petición **GET /pokemon/seed**, la cual responde con **200 OK** y el mensaje *"Pokémones agregados en la Base de Datos"*, confirmando que los Pokémon fueron cargados correctamente en la base de datos. Posteriormente, la ruta **GET /pokemon/random** queda disponible para obtener Pokémon de forma aleatoria, validando que el módulo Pokémon y su conexión con la base de datos funcionan correctamente.

```
EXPLORADOR  game.http  X  Response(24ms)  X
src > test > game.http > GET /game/win/1
Send Request
1 get http://localhost:3000/game/win/1
2
3 ###
4
5 Send Request
6 GET http://localhost:3000/game/lose/4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
Restarting 'index.js'
[dotenv@17.2.3] injecting env (5) from .env -- tip: prevent committing .env to code: https://dotenvx.com/precommit
Servidor está ejecutando en el puerto 3000

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 86
6 ETag: W/"56-ISMIG8qEf8nPh2Pd8xwJHPY2JE"
7 Date: Mon, 15 Dec 2025 19:02:20 GMT
8 Connection: close
9
10 {
11   "msg": "Victoria registrada",
12   "game": {
13     "id": 1,
14     "user_id": 1,
15     "win": 1,
16     "lose": 0,
17     "date": null
18   }
19 }
```

El **endpoint /game** usando el archivo **game.http**. Al ejecutar la petición **GET /game/win/1**, el servidor responde con **200 OK** y confirma que la **victoria fue registrada correctamente** para el usuario con ID 1, incrementando el contador de **wins** en la base de datos. Esto valida que el módulo de juego funciona correctamente, actualiza las estadísticas del usuario y mantiene la conexión adecuada entre Express y la base de datos.

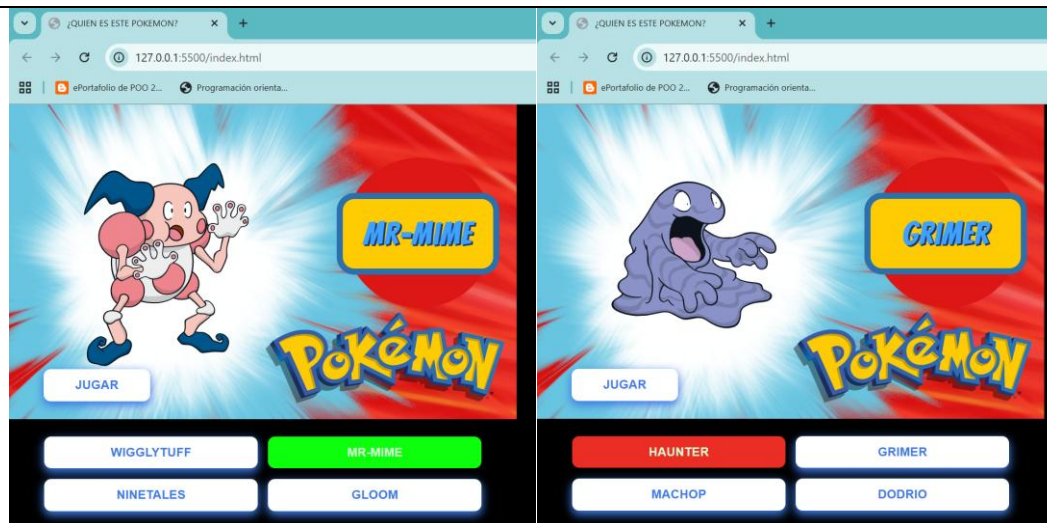




INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

Formato para prácticas de Laboratorio



El funcionamiento final del juego “¿Quién es este Pokémon?”. Al iniciar, el usuario presiona **JUGAR** y se muestra la **silueta de un Pokémon** junto con varias **opciones de respuesta**. Al seleccionar una opción, el juego **revela la imagen real del Pokémon**, muestra su **nombre** y resalta la respuesta en **verde si es correcta** o en **rojo si es incorrecta**. Luego, el jugador puede volver a jugar para intentar con otro Pokémon. En conjunto, se evidencia una interacción completa entre el **frontend (HTML y CSS)** y el **backend**, ofreciendo una experiencia visual clara y dinámica.

Conclusión:

En la presente práctica se logró desarrollar e integrar exitosamente un sistema basado en la arquitectura cliente–servidor, compuesto por una API REST (mini_pokeapi) y una aplicación cliente denominada ¿Quién es el Pokémon?, cumpliendo con los objetivos planteados. A través del uso de Node.js y Express, se implementaron correctamente los endpoints necesarios para realizar operaciones CRUD, permitiendo la gestión de usuarios, Pokémon y estadísticas de juego de manera estructurada y segura.

La correcta conexión con la base de datos MariaDB, mediante el uso de pools de conexión y variables de entorno, permitió garantizar un acceso eficiente a la información y una adecuada separación entre la lógica de negocio y el acceso a datos. Asimismo, la implementación de mecanismos de validación y cifrado de contraseñas con bcrypt reforzó la seguridad del sistema.

Por otro lado, el desarrollo de la aplicación cliente permitió comprobar de forma práctica el consumo de servicios web REST, logrando una interacción dinámica entre el frontend y el backend. El juego “¿Quién es el Pokémon?” evidenció el flujo completo de datos, desde la solicitud HTTP hasta la presentación visual de los resultados, demostrando el correcto



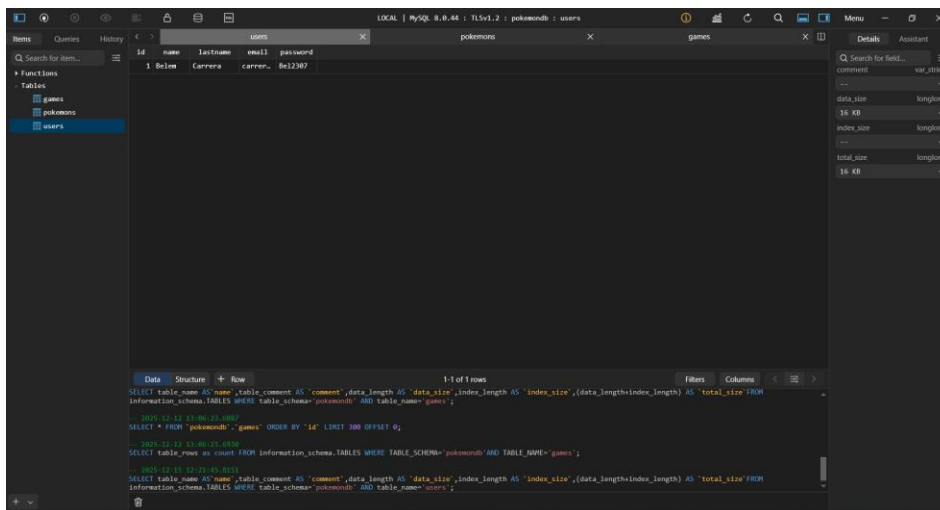
INSTITUTO TECNOLÓGICO DE TUXTEPEC

Departamento de Sistemas y Computación

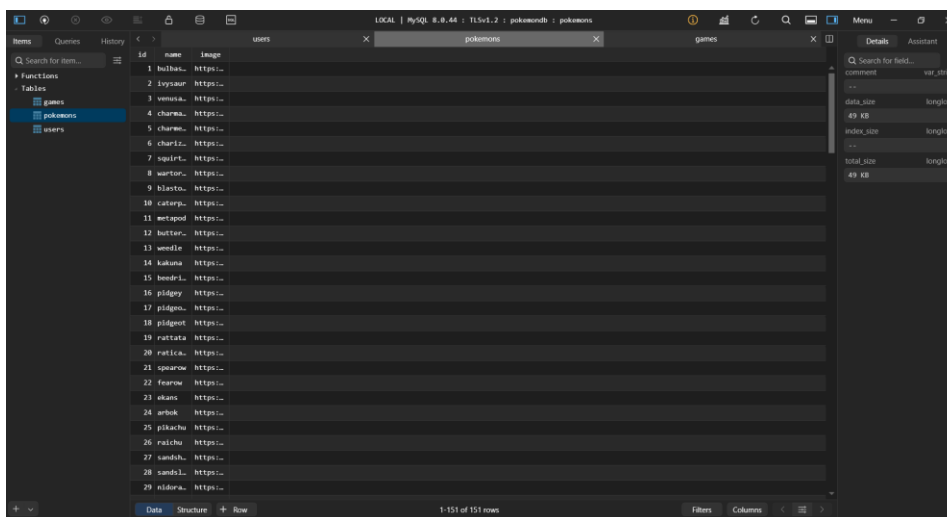
Formato para prácticas de Laboratorio

funcionamiento de la comunicación entre ambas capas, esta práctica fortaleció las competencias en el desarrollo de aplicaciones cliente–servidor, el consumo de APIs REST y la implementación de operaciones CRUD, brindando una experiencia integral que combina teoría y práctica, y sentando bases sólidas para el desarrollo de aplicaciones web y móviles más complejas en entornos reales

7. Anexos



Anexo 1: resultados de los datos de la base de datos en la tabla user



Anexo 2: resultados de los datos de la base de datos en la tabla pokemons



INSTITUTO TECNOLÓGICO DE TUXTEPEC
Departamento de Sistemas y Computación
Formato para prácticas de Laboratorio

id	user_id	win	lose	date
1	1	1	0	NULL

Anexo 3: resultados de los datos de la base de datos en la tabla games


Fecha de realización:

Formuló:

Julio Aguilar Carmona

Catedrático(a)

Realizó


María Belén Barrera Velasco
22350634

Nombre, No. Control y firma del
participante