

Universidad Rosario Castellanos

Prueba técnica Back-End

PokeAPI REST

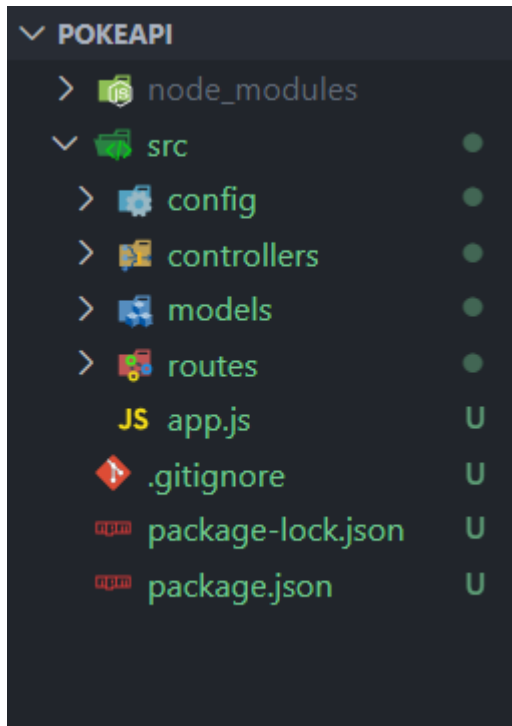
Miguel Contreras Ortiz

Lenguaje: JavaScript

Framework: Node.js

Dependencias: Express, mysql2, nodemon, sequelize

El proyecto cuenta con una estructura MVC la cual ayuda bastante en la escalabilidad de la API, a continuación, muestro la estructura del proyecto.



Dentro de la carpeta config se utilizó sequelize para interactuar y configurar la conexión necesaria con la base de datos.

```
JS db.js U x
src > config > JS db.js > [0] default
1 import { Sequelize } from "sequelize";
2
3 const db = new Sequelize('pokedex_4', 'PostulanteURC4', 'Puertoancla.', {
4   host: '172.20.1.11',
5   port: 3306,
6   dialect: 'mysql',
7   pool: {
8     max: 5,
9     min: 0,
10    acquire: 30000,
11    idle: 10000
12  },
13   logging: false
14 });
15
16 export default db;
```

En la carpeta config se encuentra pokemonController.js que es donde se almacena la lógica en la creación de los controladores, la cual obtiene información de las tablas que se encuentra dentro de models.

Estos controladores utilizan los modelos Pokemons y Moves para consultar toda la información respectivamente.

Cada controlador cuenta con la sentencia try catch para el manejo de posibles excepciones.

Para devolver los parámetros id y name de cada pokemon, devolviendo una lista con formato JSON al cliente.

```
// Obtener la lista de todos los pokemons
export const getAllPokemon = async (req, res) => {
  try {
    const pokemons = await Pokemons.findAll({
      attributes: ['id', 'name'],
    });
    res.json(pokemons);
  } catch (error) {
    res.status(500).json({ error: 'Error, lista no encontrada' });
  }
};
```

## Peticiones realizadas con Postman

The screenshot shows a Postman interface for a GET request to `http://localhost:8080/api/pokemons`. The response is a JSON array of 6 Pokémon objects, displayed in the 'Body' tab. The status bar indicates a 200 OK response with a 68 ms latency and 4.31 KB of data.

```
{
  "name": "Sceptile",
  "type_1": "Grass",
  "type_2": "None",
  "description": "Sceptile has seeds growing on its back. They are said to be bursting with nutrients that revitalize trees. This Pokémon raises the trees in a forest with loving care.",
  "weight": "12.2",
  "height": "1.7"
}
```

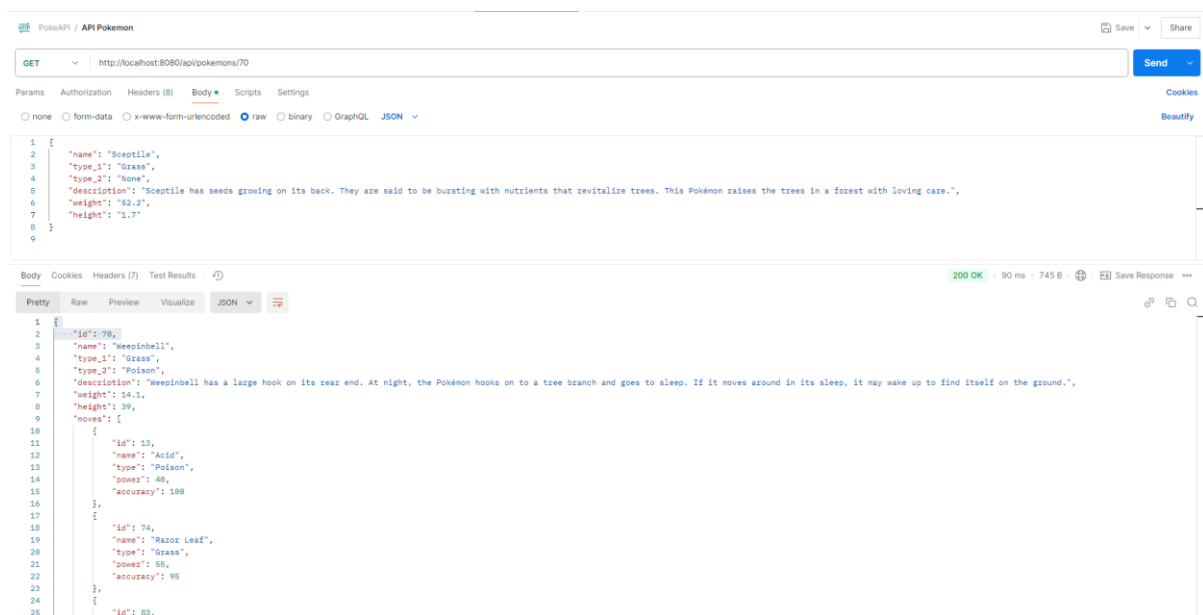
```
[
  {
    "id": 1,
    "name": "Bulbasaur"
  },
  {
    "id": 2,
    "name": "Sceptile"
  },
  {
    "id": 3,
    "name": "Venusaur"
  },
  {
    "id": 4,
    "name": "Charmander"
  },
  {
    "id": 5,
    "name": "Charmeleon"
  },
  {
    "id": 6,
    "name": "Charizard"
  }
]
```

Con respecto al controlador para obtener el pokemon por su id, se utiliza findByPk de Sequelize ya que obtiene una sola entrada de una clave proporcionada, además incluye una consulta hacia la tabla moves con la respectiva información del pokemon.

```
// Obtener Pokemon por Id
export const getPokemonById = async (req, res) => {
  try {
    const pokemon = await Pokemons.findByPk(req.params.id, {
      include: {
        model: Moves,
        through: { attributes: [] }
      }
    });

    if (!pokemon) {
      return res.status(404).json({ error: 'Pokémon no encontrado' });
    }

    res.json(pokemon);
  } catch (error) {
    res.status(500).json({ error: 'Error, Pokemon no encontrado ...' });
  }
};
```



The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:8080/api/pokemons/70`
- Response:** 200 OK, 90 ms, 745 B
- JSON Body:**

```
{
  "id": 78,
  "name": "Weepinbell",
  "type_1": "Grass",
  "type_2": "Poison",
  "description": "Weepinbell has seeds growing on its back. They are said to be bursting with nutrients that revitalize trees. This Pokémon raises the trees in a forest with loving care.",
  "weight": "52.2",
  "height": "1.7",
  "moves": [
    {
      "id": 13,
      "name": "Acid",
      "type": "Poison",
      "power": 40,
      "accuracy": 100
    },
    {
      "id": 74,
      "name": "Razor Leaf",
      "type": "Grass",
      "power": 55,
      "accuracy": 95
    }
  ]
}
```

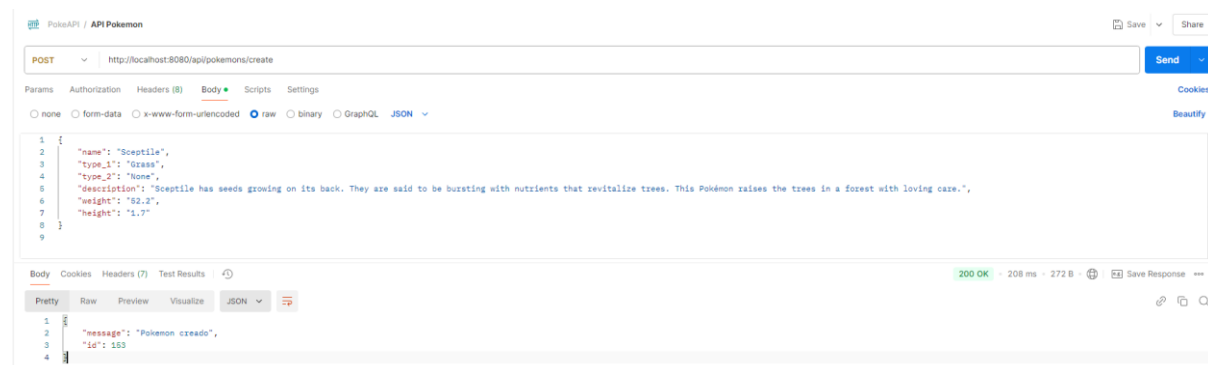
Con el controlador para crear nuevo pokemon, se extrae los datos enviados en la solicitud mediante un const, para despues realizar una validación, verifica que los datos, name, type\_1, weight y height esten presentes, de faltar un dato, responde con un estado HTTP 400 mostrando mensaje de error.

Genera una creación usando el metodo .create() para insertar el nuevo registro dentro de la tabla.

```
// Crear un nuevo pokemon
export const createPokemon = async (req, res) => {
  try {
    const { name, type_1, type_2, description, weight, height } = req.body;

    if(!name || !type_1 || !weight || !height){
      return res.status(400).json({ error: 'Faltan datos requeridos' });
    }
    const newPokemon = await Pokemons.create({
      name,
      type_1,
      type_2,
      description,
      weight,
      height,
    });

    res.status(200).json({ message: 'Pokemon creado', id:newPokemon.id});
  } catch (error) {
    res.status(500).json({error: 'Error al crear un nuevo Pokemon'});
  }
};
```



Para modificar un registro, seguí la estructura del anterior controlador, verifica la existencia del pokemon y si no lo encuentra, responde con un error.

En la parte donde cuenta con varios if sirve para comprobar que cada campo este presente en req.body y si lo está, actualiza la propiedad del objeto, para avanzar con el guardado de los cambios realizados en la base de datos usando .save()

```
// Modificar registros de un pokemon
export const updatePokemon = async (req, res) => {
  try {
    const pokemonId = req.params.id;
    const { name, type_1, type_2, description, weight, height } = req.body;

    const pokemon = await Pokemons.findByPk(pokemonId);

    if(!pokemon){
      return res.status(400).json({ error: 'Pokemon no encontrado' });
    }

    if(name) pokemon.name = name;
    if(type_1) pokemon.type_1 = type_1;
    if(type_2) pokemon.type_2 = type_2;
    if(description) pokemon.description = description;
    if(weight) pokemon.weight = weight;
    if(height) pokemon.height = height;

    await pokemon.save();

    res.status(200).json({message: 'Pokemon actualizado', id: pokemon.id});
  } catch (error) {
    res.status(500).json({error: 'Error al actualizar el Pokemon'});
  }
};
```

PokeAPI / API Pokemon

PUT http://localhost:8080/api/pokemons/50

Params Authorization Headers (8) Body Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "name": "Flygon",
3   "type_1": "Ground",
4   "type_2": "Dragon",
5   "description": "FLYAGON is nicknamed \"the elemental spirit of the desert.\" Because its flapping wings whip up a cloud of sand, this POKEMON is always enveloped in a sandstorm while flying.",
6   "weight": "82.0",
7   "height": "2.0"
8 }
9
```

Body Cookies Headers (7) Test Results

200 OK · 725 ms · 276 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Pokemon actualizado",
3   "id": 50
4 }
```

Con la eliminación cuenta con la estructura que ha seguido los otros controladores, cambia con la función para eliminar el pokemon con su id solicitado.

```
// Eliminar registros
export const deletePokemon = async (req, res) => {
  try {
    const pokemonId = req.params.id;
    const pokemon = await Pokemons.findByPk(pokemonId);

    if(!pokemon){
      return res.status(400).json({ error: 'Pokemon no encontrado' });
    }

    await pokemon.destroy();

    res.status(200).json({message: 'Pokemon eliminado', id: pokemon.id});
  } catch (error) {
    res.status(500).json({error: 'Error al eliminar el Pokemon'});
  }
};
```

[HTTP](#) PokeAPI / API Pokemon

**DELETE** ⌵ http://localhost:8080/api/pokemons/153

Params Authorization Headers (8) Body ● Scripts Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (7) Test Results 🕒

Pretty Raw Preview Visualize JSON ⌵ 🔗

```
1 {
2   "message": "Pokemon eliminado",
3   "id": 153
4 }
```

En la carpeta de models se encuentran los modelos, donde se realiza la extracción de la información de las tablas con sus respectivos registros.

En el model de movesPokemons se usa onDelete y onUpdate CASCADE para que al momento de la eliminación y la actualización, los realice de manera global para el registro afectado.

```
JS moves.js U X
src > models > JS moves.js > ...
1  import Sequelize from 'sequelize';
2  import db from "../config/db.js";
3
4  export const Moves = db.define("moves",
5  {
6      id: {
7          type: Sequelize.INTEGER,
8          autoIncrement: true,
9          primaryKey: true,
10     },
11     name: {
12         type: Sequelize.STRING,
13         allowNull: false,
14     },
15     type: {
16         type: Sequelize.STRING,
17     },
18     power: {
19         type: Sequelize.INTEGER,
20     },
21     accuracy: {
22         type: Sequelize.INTEGER,
23     },
24 },
25 {
26     timestamps: false,
27     tableName: 'moves',
28 }
29 );
30
31
32 export default Moves;
```



JS movesPokemons.js U X JS pokemon.js U

src > models > JS movesPokemons.js > MovesPokemons > move\_id

```
1 import Sequelize from 'sequelize';
2 import db from "../config/db.js";
3
4 export const MovesPokemons = db.define("moves_pokemons", {
5   id: {
6     type: Sequelize.INTEGER,
7     autoIncrement: true,
8     primaryKey: true,
9   },
10  pokemon_id: {
11    type: Sequelize.INTEGER,
12    allowNull: false,
13    references: {
14      model: 'pokemons',
15      key: 'id',
16    },
17    onDelete: 'CASCADE',
18    onUpdate: 'CASCADE'
19  },
20  move_id: {
21    type: Sequelize.INTEGER,
22    allowNull: false,
23    references: {
24      model: 'moves',
25      key: 'id',
26    },
27    onDelete: 'CASCADE',
28    onUpdate: 'CASCADE'
29  },
30  timestamps: false,
31  tableName: 'moves_pokemons',
32 });
33
34 export default MovesPokemons;
```

```
JS movesPokemons.js U X JS pokemon.js U
src > models > JS movesPokemons.js > MovesPokemons > move_id
1 import Sequelize from 'sequelize';
2 import db from "../config/db.js";
3
4 export const MovesPokemons = db.define("moves_pokemons", {
5   id: {
6     type: Sequelize.INTEGER,
7     autoIncrement: true,
8     primaryKey: true,
9   },
10  pokemon_id: {
11    type: Sequelize.INTEGER,
12    allowNull: false,
13    references: {
14      model: 'pokemons',
15      key: 'id',
16    },
17    onDelete: 'CASCADE',
18    onUpdate: 'CASCADE'
19  },
20  move_id: {
21    type: Sequelize.INTEGER,
22    allowNull: false,
23    references: {
24      model: 'moves',
25      key: 'id',
26    },
27    onDelete: 'CASCADE',
28    onUpdate: 'CASCADE'
29  },
30  timestamps: false,
31  tableName: 'moves_pokemons',
32 });
33
34 export default MovesPokemons;
```

JS movesPokemons.js U

JS pokemon.js U X

src > models > JS pokemon.js > Pokemons

```
1 import Sequelize from 'sequelize';
2 import db from "../config/db.js";
3
4 export const Pokemons = db.define("pokemons",
5   {
6     id: {
7       type: Sequelize.INTEGER,
8       autoIncrement: true,
9       primaryKey: true,
10     },
11     name: {
12       type: Sequelize.STRING,
13       allowNull: false,
14     },
15     type_1: {
16       type: Sequelize.STRING,
17     },
18     type_2: {
19       type: Sequelize.STRING,
20     },
21     description: {
22       type: Sequelize.TEXT,
23     },
24     weight: {
25       type: Sequelize.DOUBLE,
26     },
27     height: {
28       type: Sequelize.INTEGER,
29     },
30   },
31   {
32     timestamps: false,
33     tableName: 'pokemons',
34   }
35 );
```

Con las rutas se realiza la extracción de los controladores para realizar las rutas con las respectivas peticiones HTTP usando express.

```
JS pokemonRoutes.js U X
src > routes > JS pokemonRoutes.js > ...
1  import express from 'express';
2  import {
3    getAllPokemon,
4    getPokemonById,
5    createPokemon,
6    updatePokemon,
7    deletePokemon
8  } from '../controllers/pokemonController.js';
9
10 const router = express.Router();
11
12 router.get('/', getAllPokemon);
13 router.get('/:id', getPokemonById);
14 router.post('/create', createPokemon);
15 router.put('/:id', updatePokemon);
16 router.delete('/:id', deletePokemon);
17 // Rutas para actualizar y eliminar
18
19 export default router;
20
```

Finalizando con la aplicación, el archivo de arranque el cual se encuentra la ruta, middleware como intermediario de la aplicación, se realiza una relación de mucho a muchos para conectar las tablas Pokemons y Moves.

Se realiza una conexión para verificar si se ha conectado de manera exitosa a la base de datos, con `await db.sync()`, sincroniza los modelos con la base de datos y al último la inicialización del servidor.

```
JS app.js U X
src > JS app.js > ...
1  import express from 'express';
2  import db from './config/db.js';
3  import pokemonRoutes from './routes/pokemonRoutes.js';
4
5  import { Pokemons } from './models/pokemon.js';
6  import { Moves } from './models/moves.js';
7  import { MovesPokemons } from './models/movesPokemons.js';
8
9  const app = express();
10
11  // Establecer la relación muchos a muchos
12  Pokemons.belongsToMany(Moves, { through: MovesPokemons, foreignKey: 'pokemon_id' });
13  Moves.belongsToMany(Pokemons, { through: MovesPokemons, foreignKey: 'move_id' });
14
15  // Middleware
16  app.use(express.json());
17
18  // Rutas
19  app.use('/api/pokemons', pokemonRoutes);
20
21  // Conexión a la base de datos
22  try {
23    await db.authenticate();
24    console.log('Conexion exitosa');
25    await db.sync(); // Sincroniza modelos con la base de datos
26  } catch (error) {
27    console.error('Error en la conexion con la base de datos ...', error);
28  }
29
30  // Inicializar servidor
31  const PORT = 8080;
32  app.listen(PORT, () => console.log(`—— Servidor inicializado en el puerto ${PORT} ——`));
```