

Práctica 03

Prototipo de APP de escritorio

Por Francisco Crespo Martín

De: 2º de DAM Bilingüe (Horario de tardes)

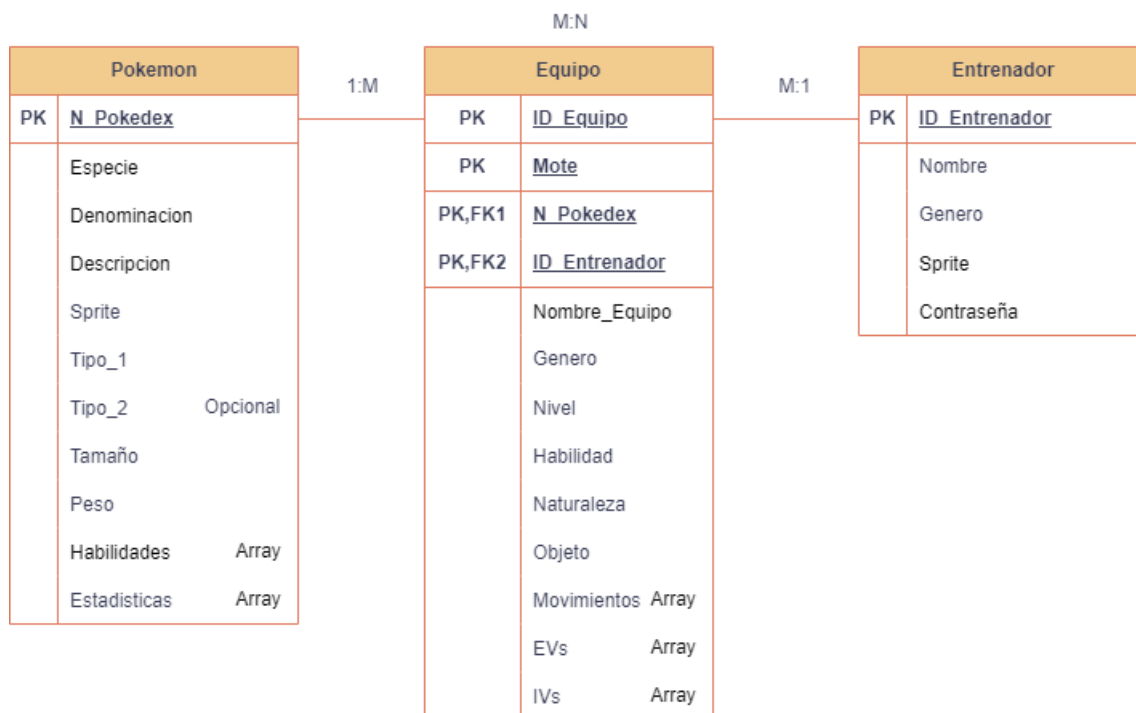
Índice

- 1) [Tabla](#)
- 2) [Docker](#)
- 3) [Creación BBDD y datos](#)
- 4) [Páginas php](#)
- 5) [Pruebas](#)

1. TABLA

La tabla que he usado es la tabla “Pokemon” en la siguiente imagen, con esos campos exactos. Esta tabla almacena los datos generales de los pokemon antes de crear personalizados en la tabla “Equipo”.

Entidad Relación



Los campos de la tabla son:

- **Pokemon**

- N Pokedex:

Es el número de la pokedex e identificador único de cada entrada.

- Especie:

El nombre de la especie del pokemon, define que pokemon es (EJ: bulbasaur, pikachu, etc).

- Denominación:

Breve frase/oración que define de forma rápida a la especie (EJ: pokemon semilla).

- Descripción:

Una descripción extensa de la especie.

- Sprite:

La imagen de la especie.

- Tipo_1:

El elemento de la especie.

- Tipo_2:

El segundo (y opcional) elemento de la especie.

- Tamaño:

El tamaño de la especie (determina el efecto y daño de algunos movimientos).

- Peso:

El peso de la especie (determina el efecto y daño de algunos movimientos).

- Habilidades:

Es un array de las distintas habilidades que puede tener una especie.

- Estadísticas:

Es un array de los valores de las estadísticas base de la especie (Salud, Ataque, Defensa, Ataque Especial, Defensa Especial y Velocidad)

2. DOCKER

La instalación de docker es bastante sencilla. Podemos encontrar el instalador buscando “docker desktop” en un navegador y entrando a este enlace:

<https://www.docker.com/products/docker-desktop/>

Elegimos la opción apropiada para nuestro chipset y SO (En mi caso Windows AMD64) para descargar el instalador. Tras eso lo ejecutamos cómo administrador y seguimos el asistente. Tras ello solo tenemos que reiniciar el equipo y ya tenemos docker desktop instalado.

Para instalar LAMP he usado una de las imágenes de la siguiente página:

<https://hub.docker.com/r/matrayner/lamp>

Concretamente la “latest-1804”. Desde una terminal y con docker desktop abierto se tiene que ejecutar el siguiente comando para bajarse la imagen:

```
docker pull matrayner/lamp:latest-1804
```

Tras eso, tendremos la imagen a partir de la cual se crearan los contenedores. Solo debemos ejecutar el siguiente comando para que se creen:

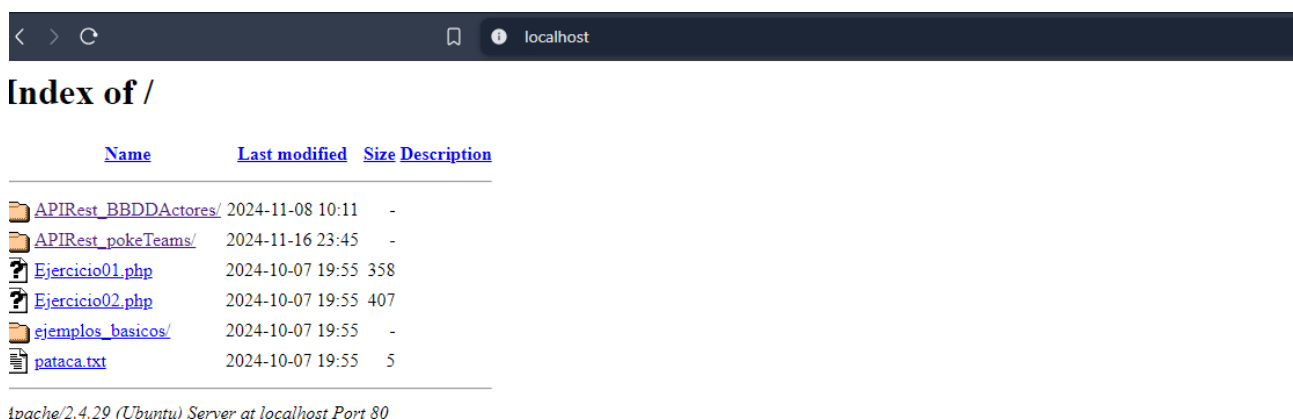
```
docker run --name LAMP --restart=always -p "80:80" -v ${PWD}/app:/app  
matrayner/lamp:latest-1804
```

El anterior sería para linux.

```
docker run --name LAMP --restart=always -p "80:80" -v C:/app:/app  
matrayner/lamp:latest-1804
```

Y este para Windows.

Una vez creados los contenedores, podremos acceder mediante localhost en el navegador a LAMP:



| Name | Last modified | Size | Description |
|----------------------|------------------|------|-------------|
| APIRest_BBDDActores/ | 2024-11-08 10:11 | - | |
| APIRest_pokeTeams/ | 2024-11-16 23:45 | - | |
| Ejercicio01.php | 2024-10-07 19:55 | 358 | |
| Ejercicio02.php | 2024-10-07 19:55 | 407 | |
| ejemplos_basicos/ | 2024-10-07 19:55 | - | |
| pataca.txt | 2024-10-07 19:55 | 5 | |

Apache/2.4.29 (Ubuntu) Server at localhost Port 80

3. CREACIÓN DE LA BBDD Y LOS DATOS

La base de datos originalmente fue creada en heidiSQL sobre un contenedor de mariaDB, pero al tener LAMP un mariaDB, no hay problemas de incompatibilidad.

Desde heidiSQL creé una nueva BBDD “pokeTeams”, y dentro de ella las 3 tablas que explico en el apartado 1. Una vez dentro de la tabla “Pokemon”, creo las columnas con sus tipos de datos.

Los tipos de datos asignados a cada columna son los de la siguiente captura:

| # | Nombre | Tipo de datos | Longitud/Co... | Sin signo | Permitir... | Relle... | Predeterminado |
|----|------------------|-----------------|-----------------|-------------------------------------|--------------------------|-------------------------------------|------------------------|
| 1 | N_Pokedex | SMALLINT | 3 | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | AUTO_INCREME... |
| 2 | Especie | VARCHAR | 50 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 3 | Denominacion | VARCHAR | 50 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 4 | Descripcion | TEXT | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 5 | Sprite | LONGTEXT | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 6 | Tipo_1 | ENUM | 'Acero','Agu... | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 7 | Tipo_2 | ENUM | 'Ninguno','A... | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 'Ninguno' |
| 8 | Tamaño | DECIMAL | 4,1 | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | '0.0' |
| 9 | Peso | DECIMAL | 4,1 | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | '0.0' |
| 10 | Habilidades | LONGTEXT | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |
| 11 | Estadísticas | LONGTEXT | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Sin valor predeter... |

“Habilidades” y “Estadísticas” son campos que solo admiten archivos json.

4. PÁGINAS PHP

Mi API consta de 6 páginas php.

1. PokeTeams.php

Esta página crea la conexión con la base de datos. Hay que indicarle el host, el usuario de LAMP, la contraseña que se encuentra en docker desktop > logs:

```
You can now connect to this MySQL Server with SXH6obmADJAM
```

Y por último se le indica la bbdd a la que vamos a acceder.

2. Pokemon.php

Esta página es el “objeto” pokemon que se va a crear para hacer las operaciones sobre la tabla.

Se le indican la tabla a usar de la bbdd y los campos de esa tabla. Tras eso se crea una variable para almacenar la conexión.

```
13 references
private $tabla = "pokemon";
13 references
public $N_Pokedex;
12 references
public $Especie;
11 references
public $Denominacion;
8 references
public $Descripcion;
8 references
public $Sprite;
11 references
public $Tipo_1;
11 references
public $Tipo_2;
8 references
public $Tamaño;
8 references
public $Peso;
12 references
public $Habilidades;
12 references
public $Estadisticas;
14 references
private $conn;
```

Justo después se crea una función para asignar a la variable “conn” la conexión a la bbdd (En mi caso se llama `_construct()`).

Tras eso se definen los métodos que tendrá la clase, tales como leer, insertar, actualizar y borrar.

- Leer

En las funciones de leer, se comprueba si el atributo a usar no está en blanco, tras eso se prepara una sentencia sql para seleccionar de la tabla las entradas que coincidan. Se debe indicar con “`bind_param`” el tipo de los campos usados para seleccionar.

Para recoger los datos se ejecuta la sentencia y se devuelve el resultado.

- Insertar

Si se tienen campos en json hay que codificarlos para que puedan ser insertados por la api.

```
$this->Habilidades = json_encode(value: $this->Habilidades);  
$this->Estadisticas = json_encode(value: $this->Estadisticas);
```

Tras eso, se prepara la sentencia insert, dando los campos que queremos insertar (Añadir todos menos los “`auto_increment`”), y en valores ponemos tantas interrogaciones como campos hemos puesto justo antes.

Tras eso, por cada campo se ejecuta y se almacena la función “`strip_tags`” que se asegura que no haya etiquetas html o inserciones maliciosas.

```
$this->Especie = strip_tags(string: $this->Especie);
```

Finalmente se indica el tipo de los campos a insertar con “`bind_param`” antes de ejecutar la sentencia y devolver “`true`”:

```
$stmt->bind_param("ssssssddss", $this->Especie, $this->Denominacion, $this->Descripcion, $this->Sprite, $this->Tipo_1, $this->Tipo_2,  
    $this->Tamaño, $this->Peso, $this->Habilidades, $this->Estadisticas);  
if ($stmt->execute()) {  
    return true;  
}  
return false;
```

- Actualizar

Aquí también se tienen que codificar los campos json como en la función “insertar”.

Tras eso se prepara la sentencia “`update`” con todos los campos a modificar indicados en la parte “`set`” seguidos de “`= ?`” y finalmente el “`where`” con el campo que define que entrada se actualiza (No se pueden actualizar campos “`auto_increment`”):


```
$stmt = $this->conn->prepare("
UPDATE " . $this->tabla . "
SET Especie = ?, Denominacion = ?, Descripcion = ?, Sprite = ?, Tipo_1 = ?, Tipo_2 = ?, Tamaño = ?, Peso = ?, Habilidades = ?, Estadísticas = ?
WHERE N_Pokedex = ?");
```

Se ejecuta la función “strip_tags” por cada campo indicado en la sentencia para eliminar etiquetas e inserciones maliciosas y se asigna el tipo de datos a cada campo para después ejecutar la sentencia y devolver “true” si es posible:

```
$stmt->bind_param("ssssssddssi", $this->Especie, $this->Denominacion, $this->Descripcion, $this->Sprite, $this->Tipo_1, $this->Tipo_2,
$this->Tamaño, $this->Peso, $this->Habilidades, $this->Estadísticas, $this->N_Pokedex);

if ($stmt->execute()) {
    return true;
}

return false;
```

- Borrar

Se prepara la sentencia “delete” con el campo que indica que entrada se borrará en el “where”:

```
$stmt = $this->conn->prepare("
DELETE FROM " . $this->tabla . "
WHERE N_Pokedex = ?");
```

Se ejecuta la función “strip_tags” para deshacerse de etiquetas e inserciones maliciosas y se le indica el tipo de dato con “bind_param”. Tras eso se ejecuta la sentencia y se devuelve “true” si es posible:

```
$stmt = $this->conn->prepare("
DELETE FROM " . $this->tabla . "
WHERE N_Pokedex = ?");

$this->N_Pokedex = strip_tags(string: $this->N_Pokedex);
$stmt->bind_param("i", $this->N_Pokedex);
if ($stmt->execute()) {
    return true;
}

return false;
```

3. leer.php

Se indican los headers, se incluyen los archivos necesarios y se crean variables con la bbdd, la conexión y el pokemon a usar:

```
//Se han activado los ERRORES (displayError) en el servidor!!!
header(header: "Access-Control-Allow-Origin: *");
header(header: "Access-Control-Allow-Headers: Content-Type");
header(header: "Content-Type: application/json; charset=UTF-8");

include_once '../basedatos/PokeTeams.php';
include_once '../tablas/Pokemon.php';

//A) Se crea conexión y objeto act
$databse = new PokeTeams();
$conex = $databse->dameConexion();
$pkm = new Pokemon(db: $conex);
```

Se comprueba individualmente que el campo para filtro de cada función de búsqueda está correctamente asignado. Si lo están, se ejecuta la función:

```
if (isset($_GET['Especie'])) {
    $pkm->Especie = $_GET['Especie'];
    $result = $pkm->leerSegunEspecie();
}
```

En el caso de la función de leer por “N_Pokemon”, si no está bien asignado, se leen todos los datos:

```
if (isset($_GET['N_Pokedex']))
    $pkm->N_Pokedex = $_GET['N_Pokedex'];
else
    $pkm->N_Pokedex = -1; //Mostrará todo

$result = $pkm->leer();
```

Se comprueba si se han encontrado datos y si se ha encontrado solo una entrada:

```
if ($result->num_rows > 0) {
    if ($result->num_rows == 1)
        $unsolopokemon = true; //E
```

Tras eso, se crea una array para almacenar las entradas leídas y se va actualizando con cada entrada:

```
$listaPokemon = array();
while ($pokemon = $result->fetch_assoc()) { //Crea un array asociativo
    extract(array: &$pokemon); //Exporta las variables de un array
    $datosExtraidos = array(
        "N_Pokedex" => $N_Pokedex,
        "Especie" => $Especie,
        "Denominacion" => $Denominacion,
        "Tipo_1" => $Tipo_1,
        "Tipo_2" => $Tipo_2,
        "Tamaño" => $Tamaño,
        "Peso" => $Peso,
        "Habilidades" => $Habilidades,
        "Estadísticas" => $Estadísticas,
    );
    array_push(array: &$listaPokemon, values: $datosExtraidos);
}
```

Si solo se ha leído un pokemon, se muestra la variable que almacena ese pokemon, si se han leído más, se muestran todas, codificándolas en json:

```
//D) Se envía respuesta y se envían los datos codificados
http_response_code(response_code: 200);
if ($unsolopokemon) //Con este cambio funcionará también para la API con JAVAEX
//donde solo queremos sacar los actores de uno en uno, no todos
    echo json_encode(value: $datosExtraidos);
else
    echo json_encode(value: $listaPokemon);
} else { //E) En caso de no recibir datos, informa
    http_response_code(response_code: 404);
    echo json_encode(
        value: array("info" => "No se encontraron datos")
    );
}
```

4. insertar.php

Se indican los headers, se incluyen los archivos necesarios y se crean variables con la bbdd, la conexión y el pokemon a usar, todo como en el archivo "leer.php".

Se decodifican los datos recibidos mediante una solicitud http de json a un objeto php:

```
$datos = json_decode(json: file_get_contents(filename: "php://input"));
```

Tras eso se comprueba que las variables se han pasado correctamente y se asignan las recibidas por http al objeto "pkm":

```
//C) Se comprueba que le pasamos las variables correctamente
if ([isset($datos->Especie) && isset($datos->Denominacion) && isset($datos->Descripcion) && isset($datos->Sprite) && isset($datos->Tipo_1) &&
    isset($datos->Tipo_2) && isset($datos->Tamaño) && isset($datos->Peso) && isset($datos->Habilidades) && isset($datos->Estadísticas)]) {

    //D) Se rellena el objeto pokemon con datos salvo el N_Pokedex
    $pkm->Especie = $datos->Especie;
    $pkm->Denominacion = $datos->Denominacion;
    $pkm->Descripcion = $datos->Descripcion;
    $pkm->Sprite = $datos->Sprite;
    $pkm->Tipo_1 = $datos->Tipo_1;
    $pkm->Tipo_2 = $datos->Tipo_2;
    $pkm->Tamaño = $datos->Tamaño;
    $pkm->Peso = $datos->Peso;
    $pkm->Habilidades = $datos->Habilidades;
    $pkm->Estadísticas = $datos->Estadísticas;
```

Finalmente se llama a la función “insertar()” y se devuelven los errores pertinentes si no se puede ejecutar:

```
//E) Llamamos a la base de datos
if ($pkm->insertar()) {
    //F) Se envía respuesta y se envían los datos codificados
    http_response_code(response_code: 201);
    echo json_encode(value: array("info" => "Pokemon Creado!"));
} else {
    http_response_code(response_code: 503);
    echo json_encode(value: array("info" => "No se puede crear"));
}
} else { //D) En caso de no recibir datos, informa
    http_response_code(response_code: 400);
    echo json_encode(value: array("info" => "No se puede crear, falta algo!"));
    echo $pkm->Especie;
}
```

5. borrar.php

Se indican los headers, se incluyen los archivos necesarios y se crean variables con la bbdd, la conexión y el pokemon a usar, todo cómo en el archivo “leer.php”.

Se decodifican los datos recibidos mediante una solicitud http de json a un objeto php cómo en el archivo “insertar.php”.

Tras eso, se comprueba si el campo “N_Pokedex” tiene valor asignado para después hacer que el campo necesario del objeto pokemon sea igual al recibido mediante http, e intentar ejecutar el borrado mediante la función “borrar()” del objeto pokemon, dando cómo salida los errores pertinentes:

```

if (isset($datos->N_Pokedex)) {
    $pkm->N_Pokedex = $datos->N_Pokedex; //D) Se rellena el objeto pokemon con datos necesarios
    if ($pkm->borrar()) { //E) Llamamos a la base de datos
        //F) Se envía respuesta y se envían los datos codificados
        http_response_code(response_code: 200);
        echo json_encode(value: array("info" => "Pokemon borrado con éxito o no está en el sistema!"));
    } else {
        http_response_code(response_code: 503);
        echo json_encode(value: array("info" => "No se puede borrar"));
    }
} else { //G) En caso de no recibir datos, informa
    http_response_code(response_code: 400);
    echo json_encode(value: array("info" => "No se puede borrar, datos incompletos"));
}

```

6. actualizar.php

Se indican los headers, se incluyen los archivos necesarios y se crean variables con la bbdd, la conexión y el pokemon a usar, todo cómo en el archivo “leer.php”.

Se decodifican los datos recibidos mediante una solicitud http de json a un objeto php cómo en el archivo “insertar.php”.

Tras eso se comprueba que las variables se han pasado correctamente y se asignan las recibidas por http al objeto “pkm”:

```

if(isset($datos->N_Pokedex) && isset($datos->Especie) && isset($datos->Denominacion) && isset($datos->Descripcion) && isset($datos->Sprite) &&
isset($datos->Tipo_1) && isset($datos->Tipo_2) && isset($datos->Tamaño) && isset($datos->Peso) && isset($datos->Habilidades) && isset($datos->Estadisticas)) {
    //D) Se rellena el objeto pokemon con datos
    $pkm->N_Pokedex = $datos->N_Pokedex;
    $pkm->Especie = $datos->Especie;
    $pkm->Denominacion = $datos->Denominacion;
    $pkm->Descripcion = $datos->Descripcion;
    $pkm->Sprite = $datos->Sprite;
    $pkm->Tipo_1 = $datos->Tipo_1;
    $pkm->Tipo_2 = $datos->Tipo_2;
    $pkm->Tamaño = $datos->Tamaño;
    $pkm->Peso = $datos->Peso;
    $pkm->Habilidades = $datos->Habilidades;
    $pkm->Estadisticas = $datos->Estadisticas;
}

```

Finalmente se llama a la función “actualizar()” y se devuelven los errores pertinentes si no se puede ejecutar:

```

if ($pkm->actualizar()) { //E) Llamamos a la base de datos
    //F) Se envía respuesta y se envían los datos codificados
    http_response_code(response_code: 200);
    echo json_encode(value: array("info" => "Pokemon actualizado"));
} else {
    http_response_code(response_code: 503);
    echo json_encode(value: array("info" => "No se ha podido actualizar"));
}

} else { //G) En caso de no recibir datos, informa
    http_response_code(response_code: 400);
    echo json_encode(value: array("info" => "No se ha podido actualizar. Datos incompletos."));
}

```

5. PRUEBAS

- LEER

GET http://localhost/APIRest_pokeTeams/crud/leer.php?N_Pokedex=3 Send

Params • Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

| Key | Value | Bulk Edit |
|---|-------|-----------|
| <input checked="" type="checkbox"/> N_Pokedex | 3 | |
| Key | Value | |

Body Cookies Headers (8) Test Results Status: 200 OK Time: 9 ms Size: 1.07 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "N_Pokedex": 3,
3   "Especie": "Venusaur",
4   "Denominacion": "Semilla",
5   "Tipo_1": "Planta",
6   "Tipo_2": "Veneno",
7   "Tamaño": "2.0",
8   "Peso": "100.0",
9   "Habilidades": "{\n\"habilidades\": [{\n\"nombre\": \"Espesura\", \n\"descripcion\": \"Si el Pokémon con esta habilidad tiene igual o menos del 33% de su PS máximo, el poder de sus ataques de tipo Planta aumenta en un 50%.\", \n\"nombre\": \"Clorofila\", \n\"descripcion\": \"La Velocidad del Pokémon aumenta un 100% (se duplica) en Día Soleado. No se incluye el turno en el que se activa el sol.\"}]",
10  "Estadísticas": "{\n\"stats\": [{\n\"estadística\": \"HP\", \n\"valor\": 80}, {\n\"estadística\": \"Atk\", \n\"valor\": 82}, {\n\"estadística\": \"Def\", \n\"valor\": 83}, {\n\"estadística\": \"SpA\", \n\"valor\": 100}, {\n\"estadística\": \"SpD\", \n\"valor\": 100}, {\n\"estadística\": \"Spe\", \n\"valor\": 80}]"
```

GET http://localhost/APIRest_pokeTeams/crud/leer.php?Tipo_1=Fuego Send

Params • Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

| Key | Value | Bulk Edit |
|--|-------|-----------|
| <input checked="" type="checkbox"/> Tipo_1 | Fuego | |
| Key | Value | |

Body Cookies Headers (8) Test Results Status: 200 OK Time: 12 ms Size: 2.74 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "N_Pokedex": 4,
4     "Especie": "Charmander",
5     "Denominacion": "Lagartija",
6     "Tipo_1": "Fuego",
7     "Tipo_2": "Ninguno",
8     "Tamaño": "0.6",
9     "Peso": "8.5",
10    "Habilidades": "{\n\"habilidades\": [{\n\"nombre\": \"Mar llamas\", \n\"descripcion\": \"Si el Pokémon con esta habilidad tiene igual o menos del 33% de su PS máximo, el poder de sus ataques de tipo Fuego aumenta en un 50%.\", \n\"nombre\": \"Poder solar\", \n\"descripcion\": \"En Día Soleado, un Pokémon con esta habilidad ve aumentado su Ataque Especial en un 50%, pero pierde un 12.5% de PS máximo al final de cada turno.\"}]",
11    "Estadísticas": "{\n\"stats\": [{\n\"estadística\": \"HP\", \n\"valor\": 39}, {\n\"estadística\": \"Atk\", \n\"valor\": 52}, {\n\"estadística\": \"Def\", \n\"valor\": 43}, {\n\"estadística\": \"SpA\", \n\"valor\": 60}, {\n\"estadística\": \"SpD\", \n\"valor\": 58}, {\n\"estadística\": \"Spe\", \n\"valor\": 65}]"
```

- INSERTAR

The screenshot shows a REST client interface with the URL `http://localhost/APIRest_pokeTeams/crud/insertar.php`. The request method is `GET` (though the body suggests a POST). The request body is a JSON object representing a Pokémon: `{ "N_Pokedex": 18, "Especie": "Pikachu", "Denominacion": "El ratón eléctrico", "Descripcion": "Pikachu es un Pokémon de tipo eléctrico, conocido por su agilidad y su habilidad para generar electricidad. Es uno de los Pokémon más emblemáticos de la franquicia.", "Sprite": "pikachu_sprite.png", "Tipo_1": "Eléctrico", "Tipo_2": "Ninguno", "Tamaño": 0.4, "Peso": 6.0, "Habilidades": ["Estática", "Absorción eléctrica"], "Estadísticas": { "HP": 35, ... } }`. The response status is `201 Created` with a message `"info": "Pokemon Creado!"`.

- ACTUALIZAR

The screenshot shows a REST client interface with the URL `http://localhost/APIRest_pokeTeams/crud/actualizar.php`. The request method is `GET`. The request body is a JSON object representing a Pokémon, similar to the one in the previous screenshot but with `"N_Pokedex": 14`. The response status is `200 OK` with a message `"info": "Pokemon actualizado"`.

- BORRAR

The screenshot shows a REST client interface with the URL `http://localhost/APIRest_pokeTeams/crud/borrar.php`. The request method is `GET`. The request body is a simple JSON object: `{ "N_Pokedex": 14 }`. The response status is `200 OK` with a message `"info": "Pokemon borrado con éxito o no está en el sistema!"`.