Poquedéquis API

Bem-vindo a Poquedéquis API, aqui você encontrará a melhor e mais simples API sobre Pokémon da internet, sendo de fácil entendimento e utilização, além de ser de livre acesso pelos usuários para ser utilizada em aplicações e serviços, disponibilizando informações sobre os Pokémon, o documento a seguir explicará sua construção a partir de métodos e funções, da linguagem de programação PHP.

Visão Geral

A API foi desenvolvida em PHP e oferece acesso completo aos métodos HTTP (GET, POST, PUT e DELETE), junto disso existe uma rica base de dados com os dados de todos os 1025 Pokémon (excluindo variações como Mega Evolução e Gigantamax), sendo possível acessar as informações detalhadas sobre o Nº da Pokedex (ID), nome, altura, peso, tipo primário, tipo secundário, fraquezas, pré-evolução, evolução, geração, região e imagem (URL).

Uso e Aplicações

A API é ideal para desenvolvedores que desejam criar aplicações relacionadas a Pokémon, como jogos, aplicativos educativos ou ferramentas de análise.

Documentação

Esta documentação fornece detalhes sobre todos os endpoints disponíveis, parâmetros de consulta, exemplos de requisição e resposta, e códigos de status. Utilize-a para compreender como interagir com a API de forma eficaz e integrar seus recursos em sua aplicação.

SUMÁRIO

1. Base de Dados

A tabela **pokemon** armazena dados básicos sobre todos os 1025 pokémon contendo os campos:

- **id** do tipo **int** sendo a Primary Key– Nº da pokedex do pokémon;
- **nome** do tipo **varchar(100)** – Nome do pokémon;
- **altura** do tipo **decimal(3,1)** – Altura do pokémon;
- **peso** do tipo **decimal(5,1)** – Peso do pokémon;
- **tipo1** do tipo **varchar(20)** – Primeiro tipo do pokémon;
- **tipo2** do tipo **varchar(20)** – Segundo tipo do pokémon;
- **fraquezas** do tipo **varchar(255)** – Tipos de ataques que o pokémon tem fraquezas;
- **pre_evolucao** do tipo **varchar(100)** (Esse campo pode ser nulo) – Evolução anterior de um pokémon já evoluído;
- **evolucao** do tipo **varchar(200)** (Esse campo pode ser nulo) – Próxima evolução do pokémon;
- **genero** do tipo **varchar(20)** – Se o pokémon é macho ou fêmea;
- **geracao** do tipo **int** – De qual entre as 9 gerações que o pokémon pertence;
- **regiao** do tipo **varchar(20)** – Evolução anterior de um pokémon já evoluído;
- **imagem_url** do tipo **varchar(255)** – Imagem do pokémon;

2. Conexão com o Banco de Dados

O **bdconnecta.php** realiza a conexão com um banco de dados MySQL e fornece uma resposta em formato JSON para indicar se a conexão foi bem-sucedida ou falhou. Ele utiliza a função **mysqli_connect** para estabelecer a conexão, fornecendo os seguintes parâmetros: **servidor**, **nome de usuário**, **senha** e **nome do banco de dados**.

O código estabelece conexão com um banco MySQL e retorna uma resposta JSON indicando "error" em caso de falha ou "success" se a conexão for bem-sucedida, exibindo mensagens correspondentes ao usuário.

```php
<?php
    $conn = mysqli_connect("localhost", "22086", "", "22086");
    if (!$conn) {
        $json = json_encode([
            'status' => 'error',
            'message' => 'Falha na conexão com o banco de dados'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
        echo $json;

    }
    else {
        $json = json_encode([
            'status' => 'sucess',
            'message' => 'Conexão bem sucedida'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
        echo $json;
    }
    date_default_timezone_set('Brazil/East');
    mysqli_query($conn, "SET NAMES 'utf8'");
?>
```

3. Programação da API

Para realizar a configuração certa, o arquivo “bdconnecta.php” deve estar presente e corretamente configurado para conectar ao banco de dados. **Com isso,** certifique-se de que a tabela pokemon esteja criada no seu banco de dados com a estrutura apropriada e faça o upload do script PHP para o servidor web. A API será acessada através da URL onde o script está hospedado, por exemplo: http://localhost/pokemon/api.php

3.1 Parâmetros de URL

O tipo de conteúdo da resposta HTTP foi definido como JSON, então a resposta será feita no formato JSON. O **$request_method** processará qual será o método de requisição HTTP (GET, POST, PUT, DELETE). As variáveis criadas são os parâmetros que serão responsáveis por realizar uma ação a serem enviados via método **GET**.

Os parâmetros:

- **$id:** procura dados pelo id do Pokémon definido na URL.
- **$nome:** procura dados pelo nome do Pokémon definido na URL.
- **$altura:** procura dados pela altura do Pokémon definido na URL.
- **$peso:** procura dados pelo peso do Pokémon definido na URL.
- **$tipo1:** procura dados pelo tipo primário do Pokémon definido na URL.
- **$tipo2:** procura dados pelo tipo secundário do Pokémon definido na URL.
- **$fraquezas:** procura dados pelas fraquezas do Pokémon definido na URL.
- **$geracao:** procura dados pela geração do Pokémon definido na URL.
- **$regiao:** procura dados pela região do Pokémon definido na URL.
- **$delete:** definida com o propósito de excluir o Pokémon pelo id definido na URL.
- **$ordenarPor:** define o campo (id, nome, altura, peso ou geração) pelo qual os resultados devem ser ordenados. Se não estiver presente, define o valor padrão como 'id'.
- **$ordenarDirecao:** define a direção da ordenação ('asc' para ascendente ou 'desc' para descendente). Se não estiver presente, define o valor padrão como 'asc'.

```php
require('bdconnecta.php');

ob_clean();
header('Content-Type: application/json; charset=utf-8');

$request_method = $_SERVER["REQUEST_METHOD"];

$id = (isset($_GET['id']) ? $_GET['id'] : '');
$nome = (isset($_GET['nome']) ? $_GET['nome'] : '');
$altura = (isset($_GET['altura']) ? $_GET['altura'] : '');
$peso = (isset($_GET['peso']) ? $_GET['peso'] : '');
$tipo1 = (isset($_GET['tipo1']) ? $_GET['tipo1'] : '');
$tipo2 = (isset($_GET['tipo2']) ? $_GET['tipo2'] : '');
$fraquezas = (isset($_GET['fraquezas']) ? $_GET['fraquezas'] : '');
$geracao = (isset($_GET['geracao']) ? $_GET['geracao'] : '');
$regiao = (isset($_GET['regiao']) ? $_GET['regiao'] : '');
// Recupera parâmetros da URL para a exclusão
$delete = (isset($_GET['delete']) ? $_GET['delete'] : '');
// Recupera parâmetros para ordenar os resultados
$ordenarPor = (isset($_GET['ordenar']) ? $_GET['ordenar'] : 'id');
$ordenarDirecao = (isset($_GET['direcao']) ? strtolower($_GET['direcao']) : 'asc');
```

3.2 Roteamento

Para o roteamento, foi usado um switch que trata diferentes métodos HTTP (GET, POST, PUT, DELETE), definindo ações específicas para cada um:

- **GET:** Recupera dados com base nos parâmetros fornecidos.
- **POST:** Cria um novo Pokémon.
- **PUT:** Atualiza um Pokémon existente.
- **DELETE:** Exclui um Pokémon.

3.2.1 Método GET

Quando a requisição é um **GET**, o código realiza a busca de dados com base nos parâmetros fornecidos na URL. O parâmetro **$delete** é verificado para determinar se deve ser feita uma exclusão: se for um ID válido, a função **deletarPoke($id)**. Os parâmetros **$id**, **$nome**, **$altura**, **$peso**, **$tipo1**, **$tipo2**, **$fraquezas**, **$geracao** e **$regiao** são usados para construir condições de filtro na consulta SQL.

Para a função de pesquisa, existem três arrays para construir uma consulta SQL dinâmica:

- **$colunas:** para armazenar as condições de pesquisa na clásula **WHERE** da consulta. Exemplo: id=?
- **$variaveis:** para armazenar os valores dos parâmetros do sql preparado. Exemplo: $id
- **$tipos**: para armazenar os tipos de dados dos parâmetros do sql preparado. Exemplo: i (inteiro)

Cada condição é adicionada com base na presença dos parâmetros na URL, e os valores são preparados para serem usados de forma segura na consulta SQL.

Para refinar a consulta, o código adiciona condições com base nos parâmetros fornecidos, garantindo que a ordenação seja feita corretamente em ordem ascendente (`ASC`) ou descendente (`DESC`).

Na execução da consulta, se houver parâmetros de filtro nos arrays, a função **selectDados** é chamada para realizar a consulta com essas condições. Se não houver parâmetros, a função **selectAll()** é utilizada para recuperar todos os Pokémon, ordenados conforme especificado.

```php
case ('GET'):
    $colunas = [];
    $variaveis = [];
    $tipos = [];

    if ($delete) {
        $id = intval($delete);
        if ($id > 0) {
            deletarPoke($id);
        } else {
            $json = (json_encode(['status' => 'error', 'message' => 'ID inválido.'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo ($json);
        }
    } else {
        $colunas = [];
        $variaveis = [];
        $tipos = [];

        if ($id) {
            $colunas[] = "id = ?";
            $variaveis[] = $id;
            $tipos[] = "i";
        }
        if ($nome) {
            $colunas[] = "nome LIKE ?";
            $variaveis[] = $nome . '%';
            $tipos[] = "s";
        }
        if ($altura) {
            $colunas[] = "altura = ?";
            $variaveis[] = $altura;
            $tipos[] = "d";
        }
        if ($peso) {
            $colunas[] = "peso = ?";
            $variaveis[] = $peso;
            $tipos[] = "d";
        }
```

```php
    if ($tipo1) {
        $colunas[] = "tipo1 = ?";
        $variaveis[] = $tipo1;
        $tipos[] = "s";
    }
    if ($tipo2) {
        $colunas[] = "tipo2 = ?";
        $variaveis[] = $tipo2;
        $tipos[] = "s";
    }
    if ($fraquezas) {
        $colunas[] = "fraquezas LIKE ?";
        $variaveis[] = '%' . $fraquezas . '%';
        $tipos[] = "s";
    }
    if ($geracao) {
        $colunas[] = "geracao = ?";
        $variaveis[] = $geracao;
        $tipos[] = "i";
    }
    if ($regiao) {
        $colunas[] = "regiao = ?";
        $variaveis[] = $regiao;
        $tipos[] = "s";
    }

    $ordenarPor = in_array($ordenarPor, ['id', 'nome', 'altura', 'peso', 'geracao']) ? $ordenarPor : 'id';
    $ordenarDirecao = $ordenarDirecao === 'desc' ? 'DESC' : 'ASC';
    $ordenacao = "ORDER BY $ordenarPor $ordenarDirecao";

    if (!empty($colunas)) {
        selectDados($colunas, $variaveis, $tipos, $ordenacao);
    } else {
        selectAll($ordenacao);
    }
}
break;
```

3.2.1.1 Função selectAll()

A função **selectAll** realiza uma consulta SQL para recuperar todos os dados dos Pokémon armazenados na base de dados, com a opção de ordenação baseada no parâmetro **$ordenacao**.

Inicialmente, prepara a consulta SQL utilizando a função **mysqli_prepare**, em seguida executa a consulta com **mysqli_stmt_execute**, se a execução falhar, retorna uma mensagem de erro em formato JSON, caso contrário, a função vincula os resultados da consulta às variáveis locais usando **mysqli_stmt_bind_result**, e armazena cada registro em um array **$pokemon**.

Após coletar todos os dados, a função converte o array para JSON e o envia como resposta. Se nenhum Pokémon for encontrado, a função retorna um erro 404 com uma mensagem apropriada. Por fim, a função fecha a declaração SQL com **mysqli_stmt_close.**

```php
function selectAll($ordenacao = "") {
    global $conn;
    $sql = ("SELECT id, nome, altura, peso, tipo1, tipo2, fraquezas, pre_evolucao, evolucao, genero, geracao, regiao, imagem_url FROM pokemon
    $ordenacao");
    $stmt = (mysqli_prepare($conn, $sql));
    if (!$stmt) {
        $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    if (!mysqli_stmt_execute($stmt)) {
        $json = (json_encode(['status' => 'error', 'message' => 'Falha ao executar a consulta'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_bind_result($stmt, $id, $nome, $altura, $peso, $tipo1, $tipo2, $fraquezas, $pre_evolucao, $evolucao, $genero, $geracao,
    $regiao, $imagem_url);
    $pokemon = [];
    while (mysqli_stmt_fetch($stmt)) {
        $pokemon[] = [
            'id' => $id,
            'nome' => $nome,
            'altura' => $altura,
            'peso' => $peso,
            'tipo1' => $tipo1,
            'tipo2' => $tipo2,
            'fraquezas' => $fraquezas,
            'pre_evolucao' => $pre_evolucao,
            'evolucao' => $evolucao,
            'genero' => $genero,
            'geracao' => $geracao,
            'regiao' => $regiao,
            'imagem_url' => $imagem_url
        ];
    }
    if (count($pokemon) > 0) {
        $json = (json_encode($pokemon,
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        $json = (str_replace('\\/', '/', $json));
        echo($json);
    } else {
        header("HTTP/1.1 404 Not Found");
        $json = (json_encode(['status' => 'error', 'message' => 'Pokémon não encontrado'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_close($stmt);
}
```

3.2.1.2 Função selectDados()

A função **selectDados** realiza uma consulta SQL dinâmica para recuperar informações de Pokémon do banco de dados.

Ela utiliza os parâmetros **$colunas**, **$variaveis**, **$tipos** e **$ordenacao** para construir uma consulta filtrada e ordenada. Inicialmente, a função gera a cláusula **WHERE** concatenando as condições armazenadas em **$colunas** e constrói a consulta SQL completa com a ordenação especificada.

Em seguida, a função prepara a consulta SQL usando **mysqli_prepare** e verifica se houve falha na preparação, se a consulta exige parâmetros, ela os vincula usando **mysqli_stmt_bind_param**, a consulta é então executada com **mysqli_stmt_execute**.

Após a execução, a função vincula os resultados às variáveis e os coleta em um array **$pokemon**. Os dados são então codificados em JSON e retornados. Se nenhum Pokémon for encontrado, um erro 404 é retornado, indicando que o recurso não foi encontrado. Assim como a função anterior, o recurso é liberaddo com **mysqli_stmt_close**.

```php
function selectDados($colunas, $variaveis, $tipos, $ordenacao) {
    global $conn;
    $condicoes = implode(" AND ", $colunas);
    $sql = ("SELECT id, nome, altura, peso, tipo1, tipo2, fraquezas, pre_evolucao, evolucao, genero, geracao, regiao, imagem_url
            FROM pokemon
            WHERE $condicoes $ordenacao");
    $stmt = (mysqli_prepare($conn, $sql));
    if (!$stmt) {
        $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    if (!empty($variaveis)) {
        mysqli_stmt_bind_param($stmt, implode('', $tipos), ...$variaveis);
    }
    if (!mysqli_stmt_execute($stmt)) {
        $json = (json_encode(['status' => 'error', 'message' => 'Falha ao executar a consulta'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_bind_result($stmt, $id, $nome, $altura, $peso, $tipo1, $tipo2, $fraquezas, $pre_evolucao, $evolucao,
    $genero, $geracao, $regiao, $imagem_url);
    $pokemon = [];
    while (mysqli_stmt_fetch($stmt)) {
        $pokemon[] = [
            'id' => $id,
            'nome' => $nome,
            'altura' => $altura,
            'peso' => $peso,
            'tipo1' => $tipo1,
            'tipo2' => $tipo2,
            'fraquezas' => $fraquezas,
            'pre_evolucao' => $pre_evolucao,
            'evolucao' => $evolucao,
            'genero' => $genero,
            'geracao' => $geracao,
            'regiao' => $regiao,
            'imagem_url' => $imagem_url
        ];
    }
    if (count($pokemon) > 0) {
        $json = (json_encode($pokemon, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        $json = (str_replace('\\/', '/', $json));
        echo($json);
    } else {
        header("HTTP/1.1 404 Not Found");
        $json = (json_encode(['status' => 'error', 'message' => 'Pokémon não encontrado'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo ($json);
    }
    mysqli_stmt_close($stmt);
}
```

3.2.1.3 Função deletarPoke()

A função **deletarPoke** é responsável por excluir um Pokémon do banco de dados com base no seu ID. Primeiro, a função realiza uma consulta para verificar se um Pokémon com o ID fornecido existe.

Isso é feito utilizando **mysqli_prepare** para preparar uma consulta SQL **SELECT** e **mysqli_stmt_bind_param** para vincular o ID como parâmetro. A consulta é então executada com **mysqli_stmt_execute**, e o resultado é vinculado à variável **$nome** com **mysqli_stmt_bind_result** e **mysqli_stmt_fetch**.

Caso o Pokémon seja encontrado, a função prepara e executa uma segunda consulta SQL, desta vez para excluir o Pokémon. Novamente, a consulta é preparada com **mysqli_prepare** e o ID é vinculado como parâmetro. A exclusão é realizada com **mysqli_stmt_execute**.

```php
function deletarPoke($id) {
    global $conn;
    $sql = ("SELECT nome FROM pokemon WHERE id = ?");
    $stmt = (mysqli_prepare($conn, $sql));
    if (!$stmt) {
        $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_bind_param($stmt, 'i', $id);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $nome);
    mysqli_stmt_fetch($stmt);
    if ($nome) {
        mysqli_stmt_close($stmt);
        $sql = ("DELETE FROM pokemon WHERE id = ?");
        $stmt = (mysqli_prepare($conn, $sql));
        if (!$stmt) {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'],
            JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
        mysqli_stmt_bind_param($stmt, 'i', $id);
        if (mysqli_stmt_execute($stmt)) {
            $json = (json_encode(['status' => 'success', 'message' => 'Pokémon excluído com sucesso', 'id' => $id, 'nome' => $nome],
            JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        } else {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha ao excluir Pokémon'],
            JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
    } else {
        $json = (json_encode(['status' => 'error', 'message' => 'Pokémon não encontrado'],
        JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_close($stmt);
}
```

3.2.2 Método POST

No bloco de código para o método POST, a função **criarPokemon** é chamada. Isso indica que, quando uma requisição HTTP com o método POST é feita para a API, essa função é responsável por processar essa requisição e realizar a criação de um novo Pokémon no banco de dados.

```
case ('POST'):
    criarPokemon();
    break;
```

3.2.2.1 Função criarPokemon()

A função **criarPokemon** é responsável por inserir novos registros de Pokémon no banco de dados. Inicialmente, ela obtém os dados de entrada em formato JSON através da leitura do fluxo de entrada **php://input**. A função verifica se todos os campos obrigatórios estão presentes no JSON. Se estiverem, prepara uma consulta SQL **INSERT** para adicionar um novo Pokémon à tabela **pokemon**.

A consulta é preparada usando **mysqli_prepare** para garantir segurança contra injeções SQL. A função, então, verifica e define valores padrão para campos opcionais como **tipo2**, **pre_evolucao** e **evolucao**, substituindo valores vazios por **NULL**. Os parâmetros são vinculados à consulta com **mysqli_stmt_bind_param**, e a consulta é executada com **mysqli_stmt_execute**.

```php
function criarPokemon() {
    global $conn;
    $input = (json_decode(file_get_contents('php://input'), true));
    if (isset($input['id'], $input['nome'], $input['altura'], $input['peso'], $input['tipo1'], $input['fraquezas'], $input['genero'],
    $input['geracao'], $input['regiao'], $input['imagem_url'])) {
        $sql = ("INSERT INTO pokemon (id, nome, altura, peso, tipo1, tipo2, fraquezas, pre_evolucao, evolucao, genero, geracao,
        regiao, imagem_url) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
        $stmt = (mysqli_prepare($conn, $sql));
        if (!$stmt) {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'],
            JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
        $tipo2 = (isset($input['tipo2']) && $input['tipo2'] !== '' ? $input['tipo2'] : NULL);
        $pre_evolucao = (isset($input['pre_evolucao']) && $input['pre_evolucao'] !== '' ? $input['pre_evolucao'] : NULL);
        $evolucao = (isset($input['evolucao']) && $input['evolucao'] !== '' ? $input['evolucao'] : NULL);
        mysqli_stmt_bind_param($stmt, "isddsssssssss",
            $input['id'],
            $input['nome'],
            $input['altura'],
            $input['peso'],
            $input['tipo1'],
            $tipo2,
            $input['fraquezas'],
            $pre_evolucao,
            $evolucao,
            $input['genero'],
            $input['geracao'],
            $input['regiao'],
            $input['imagem_url']
        );
        if (!mysqli_stmt_execute($stmt)) {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha ao executar a consulta'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
        $json = (json_encode(['status' => 'success'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    } else {
        header("HTTP/1.1 400 Bad Request");
        $json = (json_encode(['status' => 'error', 'message' => 'Dados insuficientes'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_close($stmt);
}
```

3.2.3 Método PUT

No bloco de código para o método PUT, a função **atualizarPokemon** é chamada. Isso indica que, quando uma requisição HTTP com o método PUT é feita para a API, essa função é responsável por atualizar os dados de um Pokémon existente no banco de dados.

```php
case ('PUT'):
    atualizarPokemon();
    break;
```

3.2.3.1 Função atualizarPokemon()

A função **atualizarPokemon** é responsável por atualizar registros de Pokémon no banco de dados. Inicialmente, a função decodifica os dados JSON

recebidos na requisição **PUT** através do fluxo de entrada **php://input**. Ela verifica se todos os campos necessários estão presentes e não vazios, assegurando que dados completos e válidos foram fornecidos para a atualização.

Se todos os dados forem válidos, a função prepara uma consulta SQL **UPDATE** para modificar o registro do Pokémon correspondente ao ID fornecido. A consulta SQL é configurada para atualizar os campos **nome**, **altura**, **peso**, **tipo1**, **tipo2**, **fraquezas**, **pre_evolucao**, **evolucao**, **genero**, **geracao**, **regiao**, e **imagem_url**, onde a cláusula **WHERE** garante que apenas o registro com o ID especificado será alterado.

A função usa **mysqli_prepare** para preparar a consulta, e se a preparação falhar, retorna um erro JSON. Para evitar injeções SQL, utiliza **mysqli_stmt_bind_param** para vincular os parâmetros da consulta. Após a execução da consulta com **mysqli_stmt_execute**, a função retorna um status de sucesso ou erro, conforme o resultado da execução. Depois do processo, fecha o statement com **mysqli_stmt_close**, liberando recursos.

```php
function atualizarPokemon() {
    global $conn;
    $input = (json_decode(file_get_contents('php://input'), true));

    if (isset($input['id'], $input['nome'], $input['altura'], $input['peso'], $input['tipo1'], $input['fraquezas'], $input['genero'], $input['geracao'],
    $input['regiao'], $input['imagem_url']) && !empty($input['id']) && !empty($input['nome']) && !empty($input['altura']) && !empty($input['peso']) &&
    !empty($input['tipo1']) && !empty($input['fraquezas']) && !empty($input['genero']) && !empty($input['geracao']) && !empty($input['regiao']) &&
    !empty($input['imagem_url'])) {
        $sql = ("UPDATE pokemon SET
                nome = ?,
                altura = ?,
                peso = ?,
                tipo1 = ?,
                tipo2 = ?,
                fraquezas = ?,
                pre_evolucao = ?,
                evolucao = ?,
                genero = ?,
                geracao = ?,
                regiao = ?,
                imagem_url = ?
                WHERE id = ?");
        $stmt = (mysqli_prepare($conn, $sql));
        if (!$stmt) {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha na preparação da consulta SQL'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
        $tipo2 = (isset($input['tipo2']) && $input['tipo2'] !== '' ? $input['tipo2'] : NULL);
        $pre_evolucao = (isset($input['pre_evolucao']) && $input['pre_evolucao'] !== '' ? $input['pre_evolucao'] : NULL);
        $evolucao = (isset($input['evolucao']) && $input['evolucao'] !== '' ? $input['evolucao'] : NULL);
        mysqli_stmt_bind_param($stmt, "sddsssssissi",
            $input['nome'],
            $input['altura'],
            $input['peso'],
            $input['tipo1'],
            $tipo2,
            $input['fraquezas'],
            $pre_evolucao,
            $evolucao,
            $input['genero'],
            $input['geracao'],
            $input['regiao'],
            $input['imagem_url'],
            $input['id']
        );
        if (!mysqli_stmt_execute($stmt)) {
            $json = (json_encode(['status' => 'error', 'message' => 'Falha ao executar a consulta'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        } else {
            $json = (json_encode(['status' => 'success'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo($json);
        }
    } else {
        header("HTTP/1.1 400 Bad Request");
        $json = (json_encode(['status' => 'error', 'message' => 'Dados insuficientes ou campos vazios'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo($json);
    }
    mysqli_stmt_close($stmt);
}
```

3.2.4 Método DELETE

No **case('DELETE'),** a função **deletarPokemon** é chamada para processar requisições HTTP do tipo **DELETE**. Essa função lida com a exclusão de um Pokémon do banco de dados baseado no ID fornecido. Ela valida o ID, prepara e executa a consulta SQL para exclusão, e retorna uma resposta indicando o sucesso ou falha da operação.

```php
case ('DELETE'):
    deletarPokemon();
    break;
```

3.2.4.1 Função deletePokemon()

A função **deletarPokemon** é responsável por excluir um registro de Pokémon do banco de dados com base no ID fornecido. Ela começa verificando se o ID foi fornecido e é válido. Se o ID não for fornecido, a função retorna um erro JSON indicando que o ID não foi fornecido.

Se o ID estiver presente, a função procede com uma consulta SQL para verificar se um Pokémon com o ID fornecido existe na tabela **pokemon**. Para isso, prepara e executa uma consulta **SELECT** usando **mysqli_prepare**, **mysqli_stmt_bind_param**, e **mysqli_stmt_execute** para buscar o nome do Pokémon correspondente. Se o Pokémon for encontrado, o nome é recuperado usando **mysqli_stmt_bind_result** e **mysqli_stmt_fetch**, e o **statement** é fechado com **mysqli_stmt_close** para liberar recursos.

Em seguida, a função prepara uma nova consulta SQL **DELETE** para excluir o Pokémon com o ID fornecido. O **statement** é novamente preparado com **mysqli_prepare**, os parâmetros são vinculados com **mysqli_stmt_bind_param**, e a consulta é executada com **mysqli_stmt_execute**.

Se a exclusão for bem-sucedida, a função retorna um JSON com status de sucesso, incluindo o ID e o nome do Pokémon excluído, caso contrário, é informado que o Pokémon não foi encontrado.

Após essa função, o **statement** é fechado e a conexão com o banco de dados é encerrada com **mysqli_close**.

```php
function deletarPokemon() {
    global $conn;
    $id = (isset($_GET['id']) ? intval($_GET['id']) : null);
    if ($id !== null) {
        $sql = "SELECT nome FROM pokemon WHERE id = ?";
        $stmt = mysqli_prepare($conn, $sql);
        mysqli_stmt_bind_param($stmt, 'i', $id);
        mysqli_stmt_execute($stmt);
        mysqli_stmt_store_result($stmt);
        if (mysqli_stmt_num_rows($stmt) > 0) {
            mysqli_stmt_bind_result($stmt, $nome);
            mysqli_stmt_fetch($stmt);
            mysqli_stmt_close($stmt);
            $sql = ("DELETE FROM pokemon WHERE id = ?");
            $stmt = (mysqli_prepare($conn, $sql));
            mysqli_stmt_bind_param($stmt, 'i', $id);
            if (mysqli_stmt_execute($stmt)) {
                $json = (json_encode(['status' => 'success', 'message' => 'Pokémon excluído com sucesso!', 'id' => $id, 'nome' => $nome],
                JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
                echo $json;
            } else {
                $json = (json_encode(['status' => 'error', 'message' => 'Erro ao excluir Pokémon.'],
                JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
                echo $json;
            }
        } else {
            $json = (json_encode(['status' => 'error', 'message' => 'Pokémon não encontrado.'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
            echo $json;
        }
        mysqli_stmt_close($stmt);
    } else {
        $json = (json_encode(['status' => 'error', 'message' => 'ID não fornecido.'], JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
        echo $json;
    }
}
mysqli_close($conn);
```

3.2.5 Métodos Não Permitidos

O bloco **default** trata casos onde o método HTTP usado na requisição não é suportado pela API. Ele define o código de status HTTP 405 (Método Não Permitido) e especifica quais métodos são aceitos (**GET**, **POST**, **PUT**, **DELETE**). Em seguida, retorna uma resposta JSON com uma mensagem de erro indicando que o método não é permitido, garantindo que a API só aceite e processe métodos HTTP válidos.

```php
default:
    header("HTTP/1.1 405 Method Not Allowed");
    header("Allow: GET, POST, PUT, DELETE");
    $json = (json_encode(['status' => 'error', 'message' => 'Método não permitido'],
    JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));
    echo($json);
    break;
```

4. Requisições

4.1 GET

Os parâmetros podem ser pesquisados facilmente a partir da url do site, que nem nos exemplos abaixo:

- http://localhost/pokemon/api.php?id=1
- http://localhost/pokemon/api.php?nome=Pikachu
- http://localhost/pokemon/api.php?altura=0.6&peso=6.0
- http://localhost/pokemon/api.php?tipo1=eletrico
- http://localhost/pokemon/api.php?fraquezas=Terra&geracao=1
- http://localhost/pokemon/api.php?ordenar=nome&direcao=desc
- http://localhost/pokemon/api.php?delete=1088

Mas geralmente uma API é feita para um cliente poder acessar dados em uma interface intuitiva e fácil de usar, portanto esse método não é adequado. Para isso, foi feita função **pesquisarPokemon** com ID de parâmetro, que é chamada quando o formulário é enviado. Ela constrói a URL da API com o ID do Pokémon e faz uma requisição **GET** usando **XMLHttpRequest**. Quando a resposta é recebida, a função analisa os dados JSON e exibe as informações do Pokémon em uma **div** com o identificador resultado.

```
function pesquisarPokemon(id) {
    event.preventDefault();
    const encodedId = encodeURIComponent(id);
    const url = `http://localhost/pokemon/api.php?id=${encodedId}`;
    const xhttp = new XMLHttpRequest();
    xhttp.onload = function() {
        console.log('Resposta da API:', this.responseText);
        try {
            let conteudo = JSON.parse(this.responseText);
            let textoPokemon = '';
            if (Array.isArray(conteudo) && conteudo.length > 0) {
                conteudo.forEach(pokemon => {
                    textoPokemon = (`Nome: ${pokemon.nome}<br>
                                     Altura: ${pokemon.altura} m<br>
                                     Peso: ${pokemon.peso} kg<br>
                                     Tipo Primário: ${pokemon.tipo1}<br>
                                     Tipo Secundário: ${pokemon.tipo2 || 'Nenhum'}<br>
                                     Geração: ${pokemon.geracao}<br>
                                     Região: ${pokemon.regiao}<br><br>`);
                });
            } else {
                textoPokemon = 'Pokémon não encontrado ou inválido.';
            }
            document.getElementById('resultado').innerHTML = textoPokemon;
        } catch (e) {
            console.error('Erro ao processar a resposta JSON:', e);
            document.getElementById('resultado').innerHTML = 'Erro ao processar a resposta da API.';
        }
    };
    xhttp.onerror = function() {
        document.getElementById('resultado').innerHTML = 'Erro na conexão com o servidor.';
    };
    xhttp.open("GET", url, true);
    xhttp.send();
}
```

```
function pesquisarPokemon(id) {
    event.preventDefault();
    const encodedId = encodeURIComponent(id);
    const url = `http://localhost/pokemon/api.php?id=${encodedId}`;
    const xhttp = new XMLHttpRequest();
    xhttp.onload = function() {
        console.log('Resposta da API:', this.responseText);
        try {
            let conteudo = JSON.parse(this.responseText);
            let textoPokemon = '';
            if (Array.isArray(conteudo) && conteudo.length > 0) {
                conteudo.forEach(pokemon => {
                    textoPokemon = (`Nome: ${pokemon.nome}<br>
                                    Altura: ${pokemon.altura} m<br>
                                    Peso: ${pokemon.peso} kg<br>
                                    Tipo Primário: ${pokemon.tipo1}<br>
                                    Tipo Secundário: ${pokemon.tipo2 || 'Nenhum'}<br>
                                    Geração: ${pokemon.geracao}<br>
                                    Região: ${pokemon.regiao}<br><br>`);
                });
            } else {
                textoPokemon = 'Pokémon não encontrado ou inválido.';
            }
            document.getElementById('resultado').innerHTML = textoPokemon;
        } catch (e) {
            console.error('Erro ao processar a resposta JSON:', e);
            document.getElementById('resultado').innerHTML = 'Erro ao processar a resposta da API.';
        }
    };
    xhttp.onerror = function() {
        document.getElementById('resultado').innerHTML = 'Erro na conexão com o servidor.';
    };
    xhttp.open("GET", url, true);
    xhttp.send();
}
```

```
<div class="content">
    <form method="get" action="#" onsubmit="pesquisarPokemon(document.getElementById('id').value);">
        <div class="form-group">
            <label>ID do Pokémon:</label>
            <input name="id" type="number" id="id" value="" size="10">
        </div>
        <div class="form-group">
            <button type="submit">Pesquisar</button>
        </div>
        <div id="resultado"></div>
    </form>
</div>
```

4.2 POST

Para esse método de requisição, será usada a mesma lógica, o formulário é enviado e a função **addPokemon** é chamada.

Para não usar a passagem de parâmetros de 13 campos, código coleta os dados do formulário para processá-los de maneira personalizada. A linha **const formData = new FormData(event.target);** cria um objeto **FormData** a partir dos

dados do formulário, permitindo fácil acesso a esses dados. A seguir, **const data = {}**; inicializa um objeto vazio para armazenar os dados do formulário. O método **formData.forEach((value, key) => { data[key] = value; });** percorre cada par de chave e valor no objeto **FormData** e adiciona esses pares ao objeto data.

Esse processo transforma os dados do formulário em um formato JSON, que pode ser enviado para o servidor. Em seguida, a função faz uma requisição **POST** para a URL da API com os dados do Pokémon. O **XMLHttpRequest** é usado para enviar a requisição e lidar com a resposta do servidor.

```
function addPokemon() {
    event.preventDefault();
    const formData = new FormData(event.target);
    const data = {};
    formData.forEach((value, key) => {
        data[key] = value;
    });
    const xhttp = new XMLHttpRequest();
    const url = 'http://localhost/pokemon/api.php';
    xhttp.open("POST", url, true);
    xhttp.setRequestHeader('Content-Type', 'application/json');
    xhttp.onload = function() {
        if (xhttp.status === 200) {
            try {
                const result = JSON.parse(xhttp.responseText);
                if (result.status === 'success') {
                    document.getElementById('resultado').innerText = 'Pokémon adicionado com sucesso!';
                    // Reseta o formulário
                    document.getElementById('insertForm').reset();
                } else {
                    document.getElementById('resultado').innerText = 'Erro ao adicionar Pokémon: ' + result.message;
                }
            } catch (e) {
                document.getElementById('resultado').innerText = 'Erro ao processar a resposta do servidor.';
            }
        } else {
            document.getElementById('resultado').innerText = 'Erro ao comunicar com o servidor.';
        }
    };
    xhttp.onerror = function() {
        document.getElementById('resultado').innerText = 'Erro na conexão com o servidor.';
    };
    const jsonData = JSON.stringify(data);
    xhttp.send(jsonData);
}
```

4.3 PUT

O código JavaScript desse método foi criado para atualizar as informações de um Pokémon ao enviar um formulário, acionando a função **updatePokemon**, que usa **XMLHttpRequest** para fazer uma requisição **PUT** para a API especificada.

Primeiro, a função evita o comportamento padrão do formulário, extrai os dados inseridos e os converte em um objeto JSON. Em seguida, prepara e envia a requisição PUT com o JSON no corpo, definindo o cabeçalho **Content-Type** como **application/json**. A função lida com a resposta da API, exibindo uma mensagem de sucesso ou erro na página com base na resposta do servidor.

```
function updatePokemon(event) {
    event.preventDefault();
    const formData = new FormData(event.target);
    const data = {};
    formData.forEach((value, key) => {
        data[key] = value;
    });
    const xhttp = new XMLHttpRequest();
    const url = 'http://localhost/pokemon/api.php';
    xhttp.open("PUT", url, true);
    xhttp.setRequestHeader('Content-Type', 'application/json');
    xhttp.onload = function() {
        if (xhttp.status === 200) {
            try {
                const result = JSON.parse(xhttp.responseText);
                if (result.status === 'success') {
                    document.getElementById('resultado').innerText = 'Dados do Pokémon atualizados com sucesso!';
                    document.getElementById('insertForm').reset();
                } else {
                    document.getElementById('resultado').innerText = 'Erro ao atualizar Pokémon: ' + result.message;
                }
            } catch (e) {
                document.getElementById('resultado').innerText = 'Erro ao processar a resposta do servidor.';
            }
        } else {
            document.getElementById('resultado').innerText = 'Erro ao comunicar com o servidor.';
        }
    };
    xhttp.onerror = function() {
        document.getElementById('resultado').innerText = 'Erro na conexão com o servidor.';
    };
    const jsonData = JSON.stringify(data);
    xhttp.send(jsonData);
}
```

4.4 DELETE

Para finalizar, a função **deletePokemon** com o parâmetro id pode ser chamada ao enviar o formulário ou ao clicar em um botão.

Essa função utiliza o objeto **XMLHttpRequest** para enviar uma requisição **DELETE** para a URL da API, incluindo o ID do Pokémon como um parâmetro de consulta. O cabeçalho **Content-Type** é definido como **application/json**, embora não seja estritamente necessário para uma requisição DELETE sem corpo.

A função então manipula a resposta do servidor, atualizando o conteúdo da ** ** com mensagens de sucesso ou erro, dependendo do status da resposta.

```javascript
function deletePokemon(id) {
    event.preventDefault();
    if (!id) {
        document.getElementById('resultado').innerHTML = 'Por favor, insira um ID.';
        return;
    }
    const xhttp = new XMLHttpRequest();
    const url = `http://localhost/pokemon/api.php?id=${id}`;
    xhttp.open("DELETE", url, true);
    xhttp.setRequestHeader('Content-Type', 'application/json');
    xhttp.onload = function() {
        if (xhttp.status === 200) {
            try {
                const result = JSON.parse(xhttp.responseText);
                if (result.status === 'success') {
                    document.getElementById('resultado').innerHTML = 'Pokémon excluído com sucesso!';
                } else {
                    document.getElementById('resultado').innerHTML = 'Erro ao excluir Pokémon: ' + result.message;
                }
            } catch (e) {
                document.getElementById('resultado').innerHTML = 'Erro ao processar a resposta do servidor.';
            }
        } else {
            document.getElementById('resultado').innerHTML = 'Erro ao comunicar com o servidor.';
        }
    };
    xhttp.onerror = function() {
        document.getElementById('resultado').innerHTML = 'Erro na conexão com o servidor.';
    };
    xhttp.send();
}
```

```html
<form id="deleteForm" onsubmit="deletePokemon(document.getElementById('id').value);">
    <div>
        <label for="id">ID do Pokémon:</label>
        <input type="number" id="id" name="id" placeholder="ID do Pokémon" required>
    </div>
    <div>
        <button type="submit">Excluir Pokémon</button>
    </div>
</form>
<div id="resultado"></div>
```