Introduction to Pokemon API

Welcome to the Pokemon API documentation. This API provides an interface for managing Pokemon data, enabling developers to create applications that can search, retrieve, update, and manipulate Pokemon records with ease. Designed with scalability and efficiency in mind, the API interacts seamlessly with a MongoDB database and the external PokémonAPI to ensure comprehensive and accurate data.

Base URL: All API endpoints are accessible at:

`https://finalprojectpublic.onrender.com`

1. Get a Pokemon (/getPokemon)

Purpose:
Retrieve detailed information about a specific Pokemon by its unique ID. This endpoint ensures that the requested Pokemon is available in your database; if not, it fetches the data from an external Pokemon API, saves it locally for future requests, and returns the data to the client.

- **Functionality:**
  - **Input Validation:** Validates the `id` query parameter to ensure it's a positive integer.
  - **Database Check:** Searches the local database for a Pokemon with the given ID.
    - **If Found:** Returns the Pokemon data from the database.
    - **If Not Found:** Fetches the data from the external Pokemon API, saves it to the database, and then returns it.
  - **Error Handling:** Handles cases where the ID is invalid or the Pokemon doesn't exist in the external API.
- **Usage Scenario:**
  Ideal for applications that need to display detailed information about a specific Pokemon, such as in a Pokedex app or for game mechanics.

Example Request:
https://finalprojectpublic.onrender.com/getPokemon?id=1

Example Response:
```
{
   "message": "Fetched Pokémon bulbasaur from the database.",
   "data": {
       "id": "1",
       "abilities": [
           "overgrow",
           "chlorophyll"
```

```
        ],
        "height": 7,
        "image":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/po
kemon/1.png",
        "name": "bulbasaur",
        "types": [
            "grass",
            "poison"
        ],
        "weight": 69
    }
}
```

2. Search for Pokemon (/searchPokemon)

Purpose:

Allows users to search for Pokemon based on various criteria such as name, ID, or type.

- **Functionality:**
  - **Input Validation:** Ensures that the search parameter is provided in the request body.
  - **Search Logic:** Performs a case-insensitive search across the name, id, and types fields in the database.
  - **Result Compilation:** Collects all matching Pokemon and returns them in the response.
  - **Error Handling:** Returns an appropriate message if no matches are found or if the input is invalid.
- **Usage Scenario:**
  Useful for features like search bars or filters in applications where users can look up Pokemon by different attributes.

Example Request:
https://finalprojectpublic.onrender.com/searchPokemon

Example Body:
```
{
“search”: “bulbasaur”
}
```

Example Response:
```
{
    "status": "Success.",
    “description”: “Pokemon found!”
    "results": {
        "id": "1",
        "abilities": [
            "overgrow",
            "chlorophyll"
        ],
        "height": 7,
        "image":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/po
kemon/1.png",
        "name": "bulbasaur",
        "types": [
            "grass",
            "poison"
        ],
        "weight": 69
    }
}
```

3. Get All Pokémon(/getAllPokemons)

Purpose:
Populate your local database with a bulk set of Pokemon data by fetching details for Pokemon IDs ranging from 1 to 100. This range could change if it's needed!

- **Functionality:**
  - **Batch Processing:** Iterates through Pokemon IDs 1 to 100.
  - **Database Check and Fetching:**
    - **If a Pokémon is not in the database:** Fetches from the external API and saves it.
    - **If it already exists:** Skips to the next ID.
  - **Performance Considerations:** Processes multiple requests efficiently, with concurrency control to prevent overloading the external API.
  - **Feedback:** Provides a summary of how many Pokemon were fetched and saved.
- **Usage Scenario:**
  Ideal for initial setup or when you need an own database to get the info from.

Example Request:
`https://finalprojectpublic.onrender.com/getAllPokemons`

Example Response:
```
{
    "message": "Fetched and saved Pokemon data for IDs 1 to 100."
}
```

4. Paginated Pokemon Page(/getPokemonPage)

Purpose:
Retrieve a specific page of Pokemon data from the database, with each page containing 10 Pokemon. This makes it easier to display a pagination format.

- **Functionality:**
  - **Input Validation:** Validates the `page` query parameter to ensure it's a positive integer.
  - **Pagination Logic:**
    - Calculates the offset (`skip`) based on the page number.
    - Retrieves a maximum of 10 Pokemon per page from the database.
  - **Sorting:** Typically sorts the results by `id` in ascending order.
  - **Error Handling:** Returns an appropriate message if the page number is out of range or if no Pokemons are found.
- **Usage Scenario:**
  Useful for applications with a user interface that displays Pokemon in a list or grid format with pagination controls.

Example Request:
`https://finalprojectpublic.onrender.com/getPokemonPage?page=1`

Example Response:
```
{
  "message": "Pokemon found!",
  "data": [
    {
      "id": 11,
      "name": "metapod",
      "types": ["bug"],
      "image":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/po
kemon/11.png"
    },
    {
      "id": 12,
      "name": "butterfree",
      "types": ["bug", "flying"],
      "image":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/po
kemon/12.png"
```

```
    }
  ]
}
```

Delete All Pokemon (/deleteAllPokemons)

Purpose: Deletes all Pokemon from the database.

- **Method:** DELETE
- **Response:**
  - **200:** All Pokémon deleted successfully.
  - **500:** Internal server error.
- **Usage Scenario:**
  - When you want to reset all the system with new info!

Example Request:
`https://finalprojectpublic.onrender.com/deleteAllPokemons`

Example response:

```
{

"message": "All Pokémon have been deleted successfully.",
"deletedCount": 101

}
```

6. Delete Specific Pokémon (/deleteSpecificPokemon)

Purpose:
Delete a single Pokemon from your database using its unique ID. This is helpful when you need to remove outdated or incorrect data for a specific Pokemon.

- **Functionality:**
  - **Input Validation:** Validates the `id` query parameter.
  - **Database Check:** Searches for the Pokemon with the specified ID.
    - **If Found:** Deletes the Pokemon from the database.
    - **If Not Found:** Returns a message indicating the Pokemon doesn't exist.
  - **Error Handling:** Manages any exceptions during the deletion process.
- **Usage Scenario:**
  Useful for data maintenance tasks where individual entries need to be removed.

Example Request:
`https://finalprojectpublic.onrender.com/deleteSpecificPokemon?id=201`

Example Response:

```
{
  "message": "Pokémon with ID 201 has been successfully deleted."
}
```

7. Edit Pokémon (/editPokemon)

Purpose:
Update the details of an existing Pokemon in the database. This allows for corrections or updates to Pokemon data without the need to delete and re-add entries.

- **Functionality:**
  - **Input Validation:** Ensures all required fields are present in the request body.
  - **Database Operation:**
    - Searches for the Pokemon by `id`.
    - Updates the specified fields with new values.
  - **Fields That Can Be Updated:** `name`, `abilities`, `weight`, `height`, `types`, `image`.
  - **Error Handling:**
    - Returns an error if the Pokemon doesn't exist.
    - Validates data types and required fields.
- **Usage Scenario:**
  Useful for administrators or applications where users can modify Pokemon data, such as adding custom notes or corrections.

Example Request:
```
https://finalprojectpublic.onrender.com/updatePokemon?id=201
```

Example Body:

```
{
"name": "new-bulbasaur",
"weight": 80,
"abilities": ["overgrow", "new-ability"],
"types": ["grass", "water"]
}
```

Example Response:

```
{
  "message": "Pokemon with ID 201 has been updated successfully."
}
```

Add Pokemon (/addPokemon)

Purpose: Adds a new Pokemon to the database by providing detailed information about the Pokémon.

- **Functionality:**
  - **Input Validation:** Ensures all required fields are present in the request body.
  - **Database Operation:**
    - Searches the dataBase to see if it already exists.
    - Updates the specified fields with new values.
  - **Fields That Can Be Updated:** `name`, `abilities`, `weight`, `height`, `types`, `image`.
  - **Error Handling:**
    - Returns an error if the Pokemon doesn't exist.
    - Validates data types and required fields.
- **Usage Scenario:**
  Useful for apps where users can modify Pokemon data, such as adding custom notes or corrections.

`Example REquest:`

```
https://finalprojectpublic.onrender.com/addPokemon
```

Example Body:

```
{
"id": 201,
"name": "electabuzz",
"abilities": ["static", "vital-spirit"],
"weight": 300,
"height": 11,
"types": ["electric"],
"image":"https://raw.githubusercontent.com/PokeAPI/sprites/master/sp
rites/pokemon/101.png"
}
```

- **Response:**
  - **201:** Pokémon added successfully.
  - **400:** Validation error (e.g., missing or incorrect fields).
  - **409:** Pokémon with the same ID already exist.
  - **500:** Internal server error.

Example Response:

```
{
"message": "Pokémon added successfully!",
"data": {
"id": 201,
"name": "electabuzz",
"abilities": ["static, "vital-spirit"],
"weight": 300,
"height": 11,
"types": ["electric],
"image":"https://raw.githubusercontent.com/PokeAPI/sprites/master/sp
rites/pokemon/101.png"}
}
```