y.
Pokémon API

Yolk Studio

1 Overview

Your task is to build a Pokémon management system that handles basic CRUD operations for trainers and their Pokémon collections.

2 Core Requirements

2.1 Required Functionality

Design and implement a RESTful API that supports the following operations:

2.1.1 Trainer Management

- Create a new trainer
- Retrieve all trainers
- Retrieve a specific trainer along with their Pokémons
- Update trainer information
- Delete a trainer

2.1.2 Pokémon Management

- Add a Pokémon to a specific trainer
- Retrieve all Pokémons in the system

2.2 API Response Format

All API responses should follow this consistent structure:

```
{
  "success": true,
  "statusCode": 200,
  "message": "Pokemon found successfully",
  "data": [
    {
      "id": 1,
      "name": "Pikachu",
      "owner": "Ash Ketchum",
      ... // additional fields
    }
  ]
}
```

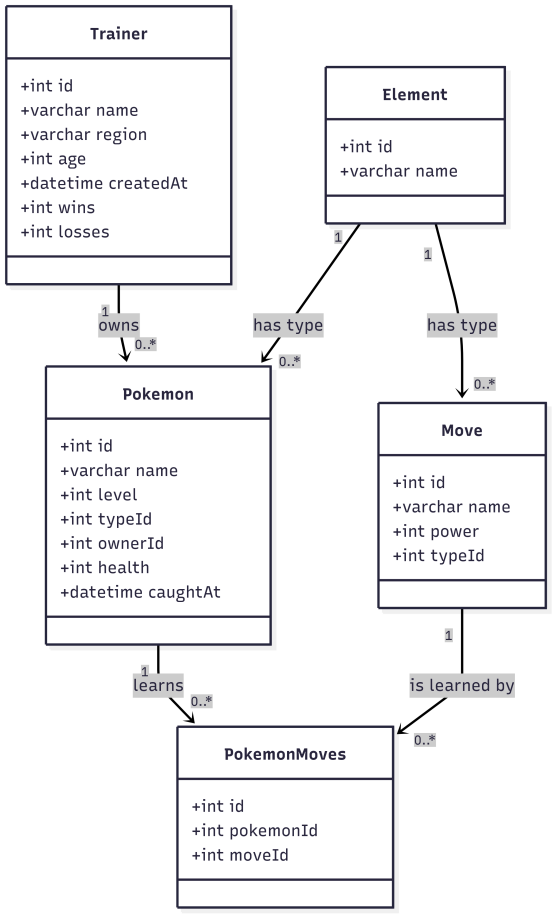
Listing 1: Standard API Response Format

3 Data Model & Database

3.1 Database Schema

The database structure follows the provided UML class diagram. Feel free to adjust the database model as needed - we are happy to discuss any modifications!

Note: For simplicity, assume that each Pokémon is unique (no case where two trainers have the same Pokémon).



3.2 Database Setup & Sample Data

We have already prepared a SQLite database for you. You can find it in the project under `data/pokemondb.sqlite`. It is fully initialized with tables and sample Pokémon data, so you can use it directly in your implementation.

4 Technology Stack

- **.NET 9** - Primary framework
- **Entity Framework Core** - Data persistence
- **Swagger/OpenAPI** - API documentation

5 Bonus Features

Showcase your skills with these optional enhancements:

- **Advanced Search & Filtering**
  - Implement search functionality for Pokémon by name
  - Handle typos and partial matches for name of the Pokemon (e.g., "pikacu" → "Pikachu", "char" → "Charizard")
  - Support multiple search criteria (name, type, level range)
- **Advanced Query Features**
  - Filter by type, level range, trainer region
  - Pagination with page size and page number
  - Sorting by name, level, catch date
  - Design clean and intuitive query parameter conventions
- **Deployment & Containerization**
  - Provide Docker configuration (Dockerfile + docker-compose.yml)
  - Deploy to cloud provider of your choosing (Azure, AWS, Google Cloud, Railway or any other provider you are comfortable with)
  - Provide live demo URL if deployed

Feel free to showcase additional skills and creativity beyond these suggestions!

6 Hints

- **Keep it simple:** Focus on clear and correct implementation of CRUD operations rather than adding unnecessary complexity.
- **Project structure matters:** Organize your code into logical layers to keep it clean and maintainable.
- **Validation:** Never trust your input data
- **Error handling:** Return meaningful error responses when something goes wrong
- **Clean code:** Write readable, well-structured, and self-explanatory code. Use meaningful naming, avoid duplication, and follow standard coding conventions.

7 Submission Guidelines

1. You will be invited to a private GitHub repository containing the starter code and task description.
2. Fork this repository to your own GitHub account (your fork will remain private by default).
3. Complete the assignment according to the requirements above in your fork.
4. Commit and push your changes regularly
5. Once finished, create a Pull Request from your fork back to the original repository.
6. Include any observations or design decisions you made in the README or in the Pull Request description.

We wish you the best of luck and look forward to your solution!

— Yolk Development Team