TECHNICAL TEST: TECHNICAL SPECIALIST

Rules

- Resolution time: 10 calendar days
- Technological stack: For the development you can use the most convenient technologies with which you feel more comfortable. In case there are examples in the statement, they will be in *Python* and/or *Bash*, but there is no reason to use them.
- The result should be delivered in a ZIP package, documented and with examples of execution and use. It shall be sent to the following e-mail address: EXAMPLE@EXAMPLE.COM
- There is no single correct solution. This is a test in which we will try to evaluate not only your knowledge but also your ability to solve. There are optional exercises, and we recommend that you take a look at them at the beginning. In fact, we recommend that you take a look at the whole document before you get started.
- If you have any questions at any time, please write to us (EXAMPLE@EXAMPLE.com) and we will try to resolve them as soon as possible.

Statement

Develop an API (REST) through which we can access and create a series of data. Those data, initially will be the Pokémons with ID from 1 to 151.
We want to work with the following values for each Pokémon:

field_name	**id**	**name**	**weight**	**types**
field_type	int	string	int	list of strings

(id, name, weight and types)

This API, initially, will have the following verbs:

- GET: GET of a Pokémon given its ID. For example, if we were to do a GET to /mypokeapi/1 should return a document similar to:

```
{
  "id": 1,
  "name": "bulbasaur",
  "weight": 69,
  "types": [
    "grass",
    "poison"
  ]
}
```

- POST: Create new Pokémon. The document structure must be the same as the GET. There cannot be 2 Pokémons with the same ID.

Optional 1: Databases

We know that what makes more sense for this case is to use a *json* or *yaml* file to store this information. But we would like everything to be stored in a database system (*Postgres, mariaDB, SQLite* ... the one you consider more suitable or the one you like the most). It doesn't matter if it is relational or NoSQL.

Optional 2: Containers

Dockerize the solution.

Optional 3: Complete the CRUD

Complete the CRUD with data modification and deletion verbs:

- PATCH: Modify a Pokémon that already exists and whose ID is known. All fields except the ID can be modified.
- DELETE: Delete an existing Pokémon whose ID is known.

FAQ

- **Where can I get the info about these Pokémons?** There are hundreds of wikis to search. Now, since we are working with APIs, we recommend using https://pokeapi.co/.

  Some examples, using BASH and JQ would be:

  -Obtain the name:

  ```
  $ curl -X GET https://pokeapi.co/api/v2/pokemon/1 | jq '.name'
  "bulbasaur"
  ```

  -Obtaining the weight:

  ```
  $ curl -X GET https://pokeapi.co/api/v2/pokemon/1 | jq '.weight'
  69
  ```

  -Obtain rates:

  ```
  curl -X GET https://pokeapi.co/api/v2/pokemon/1 | jq
  '.types[].type.name' $ curl -X GET
  https://pokeapi.co/api/v2/pokemon/1 | jq '.types[].type.name'
  "grass"
  "poison"
  ```

  If you are going to obtain the data to automatically populate the database, don't forget to document the process ;)

- **There are several approaches to the solution, which one should I choose?** The one you think is the most correct or the one you feel most comfortable with. We recommend not to *over-engineer*. For example: If you are going to use *Python* and you are hesitating between using *Django* or *Flask* for the API, it would be easiest to use *Flask*.

- **Where should it be deployed?** Locally. Everything should work locally (although an internet connection is probably necessary for the database to be popular).