Assignment 1: POKEMON!

NAME	NAVJEET KAUR
Student Id:	C0741942

The current game contains 150 Pokémon. Each Pokémon is given a 4- digit id number, starting with 0001. The ids increment by 1.

In January 2020, Nintendo plans on releasing a new version of the Pokémon video game. The new version has a feature that lets players add custom Pokémon to the game. Once a custom Pokémon is added, the Pokémon is available to all players. The custom Pokémon cannot be updated or removed from the game.

You work as a Test Engineer on Nintendo's API team. Your job is to test Nintendo's Pokémon APIs

1. **What endpoints should you test to ensure that players can use the game's new feature? Write your answer in the space below:**

   Let Poke's API url is https://pokeapi.co/

Endpoint	**Request Type**
https://pokeapi.co/api/v2/pokemon ?Id=0151 Pokémon/{id}	GET (Get a specific pokemon.)
https://pokeapi.co/api/v2/pokemon?pokemonName=pocke pokemon/PokemonName	GET (Get a specific pokemon with name)

https://pokeapi.co/api/v2/pokemon-color/1	GET (Get color)
/pokemon/{id}	DELETE (Delete existing pokemon)
/pokemon	POST (Add new pokemon to the database.)
/pokemon/{id}	PUT (Update existing pokemon.)

2. What are the 3 things must you test to ensure that you have properly tested the endpoints? Remember - you are testing endpoints that allow the player to use the new game feature!

The main three things are given below here:

- The custom Pokémon is available to all players.
- The custom Pokémon cannot be updated.
- The custom Pokémon cannot be removed from the game.

3. Suppose you are using Postman to test the endpoint for adding a new custom Pokémon to the database. Write the Postman test case code for the endpoint. In your test case, you should include all appropriate assertions needed to properly test the endpoint. As a reminder, the built-in Postman functions include.

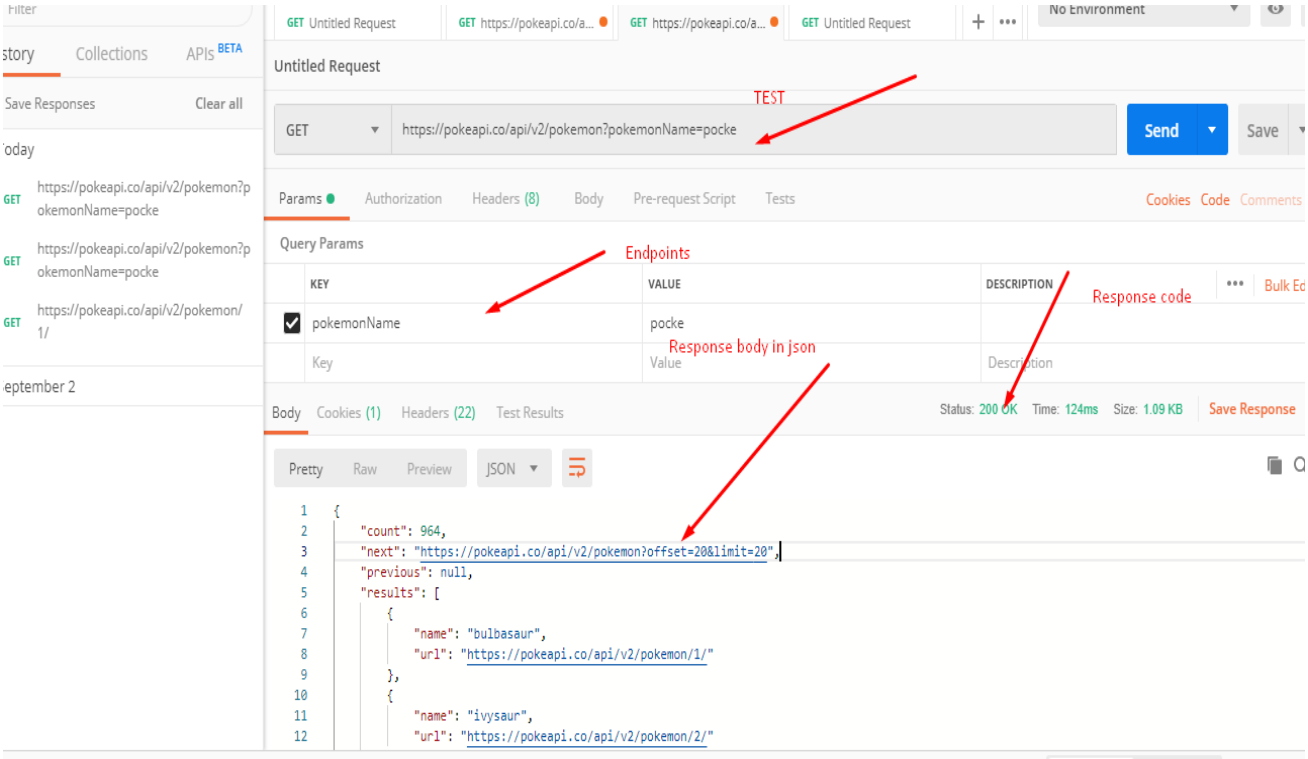
As a reminder, the built-in Postman functions include:

● tests

● responseCode[“code”]

● responseBody

● JSON.parse()



JSON RESPONSE HERE

https://pokeapi.co/api/v2/pokemon?pokemonName=pocke


```
{
    "count": 964,
    "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
    "previous": null,
    "results": [
        {
            "name": "bulbasaur",
            "url": "https://pokeapi.co/api/v2/pokemon/1/"
        },
        {
            "name": "ivysaur",
            "url": "https://pokeapi.co/api/v2/pokemon/2/"
        },
        {
            "name": "venusaur",
            "url": "https://pokeapi.co/api/v2/pokemon/3/"
        },
        {
            "name": "charmander",
            "url": "https://pokeapi.co/api/v2/pokemon/4/"
        },
        {
```

```
    "name": "charmeleon",
    "url": "https://pokeapi.co/api/v2/pokemon/5/"
},
{
    "name": "charizard",
    "url": "https://pokeapi.co/api/v2/pokemon/6/"
},
{
    "name": "squirtle",
    "url": "https://pokeapi.co/api/v2/pokemon/7/"
},
{
    "name": "wartortle",
    "url": "https://pokeapi.co/api/v2/pokemon/8/"
},
{
    "name": "blastoise",
    "url": "https://pokeapi.co/api/v2/pokemon/9/"
},
{
    "name": "caterpie",
    "url": "https://pokeapi.co/api/v2/pokemon/10/"
},
{
    "name": "metapod",
    "url": "https://pokeapi.co/api/v2/pokemon/11/"
},
{
    "name": "butterfree",
    "url": "https://pokeapi.co/api/v2/pokemon/12/"
},
{
    "name": "weedle",
    "url": "https://pokeapi.co/api/v2/pokemon/13/"
},
{
    "name": "kakuna",
    "url": "https://pokeapi.co/api/v2/pokemon/14/"
},
{
    "name": "beedrill",
    "url": "https://pokeapi.co/api/v2/pokemon/15/"
},
{
    "name": "pidgey",
    "url": "https://pokeapi.co/api/v2/pokemon/16/"
},
{
    "name": "pidgeotto",
    "url": "https://pokeapi.co/api/v2/pokemon/17/"
},
{
    "name": "pidgeot",
    "url": "https://pokeapi.co/api/v2/pokemon/18/"
```

```
        },
        {
            "name": "rattata",
            "url": "https://pokeapi.co/api/v2/pokemon/19/"
        },
        {
            "name": "raticate",
            "url": "https://pokeapi.co/api/v2/pokemon/20/"
        }
    ]
}
```


4. As discussed in class, an important part of API testing is to ensure that a command actually gets executed on the backend database. What strategy can you follow to create a test case that checks if a database operation actually occurs?

Answer: - Test case that checks if a database operation actually occurs are:-

- **Structural Testing:** it involves validation of elements inside the data repository that are used for data storage.
- **Functional Testing:** it checks all the functional requirements of applications.
- **Non-functional Testing:** it checks others aspect of applications performance, reliability etc.

According to given question, for checking database operations if it actually works or not, we can use GET Request to get the whole detail of newly added pokemon into database. Apart from this, we can also check the response of response body via using:

tests[“Response contains a ‘pokemonName’ key”] = responseBody.has(“pokemonName”)

5. Examine the Architecture diagram. What test cases should you create to ensure proper functional coverage of the system?

- Input of Pokemon name as empty or spaces to check the validation.
- Input of Pokemon name as number type or digits only.
- Input of Pokemon name as special characters only e.g.!@#$%^&*()_+.
- Input of Pokemon Current Level as empty to check the validation.
- Input of Pokemon Current Level as special characters only e.g.! @ #$ %^&*()_+.

- Input of Pokemon Current Level as only Alphabets or alpha numeric.

6. What equivalence partitions do you need to create?

Testing by Splitting Data into Equivalence Classes a black-box test design technique in which test cases are designed to execute representatives from equivalence partitions. Moreover, Equivalence partitioning is a testing technique. The main motive is to divide the input data into equivalent (similar) classes and have a test case covering each partition at least one time. This technique is used to reduce the amount of test cases and save time.

Partitions for Fast growth:

Invalid	Valid	Invalid
1251 Points	1250 Points	1236 Points

Partitions for Medium growth:

Invalid	Valid	Invalid
221 Points	216 Points	520 Points

Partitions for Slow growth:

Invalid	Valid	Invalid
583.4 Points	409.6 Points	171.6 Points

7. What inputs do you need to select to test using partitions? Fill in the details of your inputs in the chart below:

Equivalence Class	Input	Valid/Invalid Input	Expected Result
Fast	0.8(-2 power 3)	invalid	-6.4
Fast	0.8(1 power 3)	valid	0.8
Medium	L power3(-2power 3)	invalid	8
Medium	L power 3(5 power 3)	valid	125