9. Data Driven Testing using Excel, Json & Csv

Data-Driven Testing in API

- ➔ Running the same API test multiple times with different sets of input data to validate the behavior of the API under various conditions is called Data Driven testing
- ➔ We need DDT in API Testing because of
  - ◆ To test positive and negative scenarios.
  - ◆ To cover boundary values, invalid inputs, or empty values.
  - ◆ To avoid writing duplicate test code for different inputs.
  - ◆ To increase coverage and reusability.
- ➔ We mostly prefer below files for DDT
  - ◆ JSON files
  - ◆ CSV files
  - ◆ Excel files
  - ◆ YAML files
  - ◆ Database

test_submit_delete_order.py

```
import requests
import json
class TestOrderAPI:
  AUTH_TOKEN = "Bearer 7c66b9692dc2df833a33cc1a04a507e872a4ab70f3a7165eea47ad43"
  BASE_URL = "https://simple-books-api.glitch.me/orders"
  def test_submit_and_delete_order(self):
                                        Submit order
    payload = {
      "bookId": 1,
      "customerName": "John"
    }
    headers = {
      "Authorization": self.AUTH_TOKEN,
      "Content-Type": "application/json"
    }
    response = requests.post(self.BASE_URL, headers=headers, data=json.dumps(payload))
    assert response.status_code == 201
    order_id = response.json().get("orderId")
    print("Order submitted:", order_id)
                                        Delete order
    delete_response = requests.delete(f"{self.BASE_URL}/{order_id}", headers=headers)
    assert delete_response.status_code == 204
    print("Order deleted")
```

Parameterization

- ➔ Parameterization allows a test method to run multiple times with different data sets without using

loops.Instead of writing loop statements, we can use **@pytest.mark.parametrize** to pass different sets of input values to the test method, making test repetition simple and efficient.

- **Example**
  - `@pytest.mark.parametrize('num1,num2',[(1,1),(3,5),(10,10),(5,20)])`

test_Parameterization.py

```
import pytest
class TestClass:
  @pytest.mark.parametrize('num1,num2',[(1,1),(3,5),(10,10),(5,20)])
  def test_calculation(self,num1,num2):
    assert num1==num2
```

test_data_providers.py

```
import json
import csv
from openpyxl import load_workbook
```

Excel Data Provider

```
def read_excel_data(filepath, sheetname):
  wb = load_workbook(filepath)  # Load the Excel workbook.
  sheet = wb[sheetname]  # Access the specified sheet by name.
  data = []  # Empty list to store data rows
  for row in sheet.iter_rows(min_row=2, values_only=True):  # Loop through all rows from the 2nd row onwards (to skip the header)
    data.append(row)  # Add each row (as a tuple) to the list
  return data  # Return list of tuples
```

JSON Data Provider

```
def read_json_data(filepath):
 with open(filepath, 'r') as f:
   data_list = json.load(f)  # List of dicts
 return [(item,) for item in data_list]    # Wrap each dict as a tuple
```

CSV Data Provider

```
def read_csv_data(filepath):
  data = []  # List to hold CSV data
  with open(filepath, 'r') as f:
    reader = csv.reader(f)  # Create a CSV reader object
    next(reader)  # Skip the header row
    for row in reader:
      data.append(tuple(row))  # Convert each row to a tuple and add to list
  return data  # Return list of tuples
```

test_data_driven_testing.py

```
import requests
import pytest
import json
```

```
from data_providers import read_excel_data, read_json_data, read_csv_data
AUTH_TOKEN = "Bearer 7c66b9692dc2df833a33cc1a04a507e872a4ab70f3a7165eea47ad433e31f32c"
BASE_URL = "https://simple-books-api.glitch.me/orders"
```

Reusable function to submit and delete an order

```
def submit_and_delete_order(book_id, customer_name):
```

Submit order

```
  payload = {
     "bookId": int(book_id),
     "customerName": customer_name
  }
  headers = {
     "Authorization": AUTH_TOKEN,
     "Content-Type": "application/json"
  }
  response = requests.post(BASE_URL, headers=headers, data=json.dumps(payload))
  assert response.status_code == 201
  order_id = response.json().get("orderId")
  print("Order submitted:", order_id)
```

Delete order

```
  delete_response = requests.delete(f"{BASE_URL}/{order_id}", headers=headers)
  assert delete_response.status_code == 204
  print("Order deleted")
```

Excel Test

```
@pytest.mark.parametrize("book_id, customer_name",
read_excel_data("./testdata/orders_excel_data.xlsx", "Sheet1"))
def test_with_excel_data(book_id, customer_name):
  submit_and_delete_order(book_id, customer_name)
```

JSON Test

```
@pytest.mark.parametrize("order_data", read_json_data("./testdata/orders_json_data.json"))
def test_with_json_data(order_data):
  order_data = order_data[0]  # Unpack the tuple
  submit_and_delete_order(order_data["BookID"], order_data["CustomerName"])
```

CSV Test

```
@pytest.mark.parametrize("book_id, customer_name",
read_csv_data("./testdata/orders_csv_data.csv"))
def test_with_csv_data(book_id, customer_name):
  submit_and_delete_order(book_id, customer_name)
```

Note

- ➔ **Serialization** means converting a Python object (like a dictionary or list) into a format like JSON, so it can be saved or sent over a network.
  - ◆ json_data = json.dumps(data) → Serialize to JSON string

➔ **Deserialization** means converting back from JSON (or other formats) to a Python object.

- python_data = json.loads(json_data) → Deserialize JSON string back to Python dict

Method	When to Use
Dictionary	Used when data is simple and already in key-value pairs.
JSON Library	Used when we need to ensure the payload is strictly formatted as a JSON string before sending it
Custom Python Class	Used when data has complex logic, custom behavior, or needs inheritance.
Dataclass	Used when our class is mainly for storing data with fixed structure and easy conversion to dict.
External JSON File	Use when data is static or reused across multiple requests.

➔ **We use them**

- When sending/receiving data via APIs
- When saving data to files or databases
- When caching data
- During inter-process communication

➔ **Why is it Needed?**

- Python objects cannot be directly stored or sent.
- We need to convert them to formats like JSON so they become portable and shareable.