API concepts

https://youtu.be/-sIODx7CmAk

API Concepts Practice.

For this practice, we are going to define the endpoints needed to make HTTP requests to a fictional API and their possible response codes.

Let's assume we are working with an API that provides information about books and we need to make the following requests (remember to define if any of the endpoints should have a request body):

- Get a list of all available books.
- Get the details of a specific book by providing its unique ID.
- Add a new book.
- Update information for an existing book by providing its unique ID.
- Delete a book using its unique ID.

Additionally, we need to define what would be the possible response codes for each of these requests.

Create a document and define all the required methods with its name, HTTP method, body (or required parameters), and its possible response codes.

Additional Resources

Basic of HTTP **HTTP Methods** **HTTP Status Codes**

API first steps (Postman)

https://youtu.be/NaUjfg2_2rY

API First Steps Practice.

For this practice, we are going to retrieve data from a real API making requests and handling responses using postman. We will use the "PokeAPI" for this exercise, the base URL for the API is **https://pokeapi.co/**

Start by navigating to the API URL on your browser to see the documentation and identify the available endpoints; on the other hand, download and install postman if you haven't already:

PART I:

1. Open postman and create a new collection to organize the requests.
2. Within the collection, add a new request for retrieving Pokemon data.
3. Set the request details to get the data of "Pikachu".
4. Send the request and observe the response returned by the API. The response will contain various details about the Pokemon, such as its name, ID, abilities, types, etc.

PART II:

Use the information obtained from the first request to make other requests types, for example, to obtain pokemons of a specific species or type.

Additional Resources

API response structure

Create and send API requests

Restassured

https://youtu.be/wNa6HrO5-z8?si=dGRcPM9NtEVrziUt

RestAssured Practice

For this practice, we are going to test a real API using JAVA, RestAssured, any test runner and an appropriate framework architecture to organize and reuse the API calls (Do not forget POJOS!).

We will use the Star Wars API (swapi) for this exercise, the base URL is **https://swapi.dev/** and we need to Implement tests (a different test for each request) performing the following actions:

1. Test the endpoint **people/2/** and check the success response, the skin color to be gold, and the amount of films it appears on (should be 6).
2. Request the endpoint of the second movie in which **people/2/** was present (using the response from **people/2/**). Check the release date to be in the correct date format, and the response to include characters, planets, starships, vehicles and species, each element including more than 1 element.
3. Request the endpoint of the first planet present in the last film's response (using the previous response). Check the gravity and the terrains matching the exact same values returned by the request (Use fixtures to store and compare the data of terrains and gravity).
4. On the same response from the planet, grab the url element on the response, and request it. Validate the response being exactly the same from the previous one.
5. Request the **/films/7/** and check the response having a 404 code.

Additional Resources

Restassured API Testing

TDD/BDD

https://youtu.be/IlXJ7xZcOvE?si=Ur4fc5zIV0JFqkhu

TDD/BDD Practice

In this exercise, we will create the possible Gherkin scenarios for the user registration feature of and log in of the **https://www.demoblaze.com/** web app. The goal is to practice writing Gherkin scenarios that capture the behavior of the system regarding the following example:

Scenario: User Registration

Description: As a new user, I want to be able to register on the website so that I can access the platform's features.

On a document, define the required features, its scenarios (at least 5 per feature), their descriptions, their steps, and any background if needed. Use at least one scenario outline on each feature.

Additional Resources

Behaviour-Driven Development (BDD) **BDD Fundamentals**

Best practices

https://youtu.be/_6UncnqNGqU

Best Practices Exercise

For this practice, we are going to implement the best practices to test a real bank API.

Start by creating an account at the following link **https://www.mockapi.io/projects** and set up an endpoint for bank transactions (Add user, log in, deposit money, retrieve money).

Once the endpoint is created, define the best practices to be implemented and then, let's start testing the following scenarios on your automation framework:

- **Test 1:** Verify that the endpoint is empty (If it has any data use the DELETE request to clean and leave it empty, use hooks or annotations).
- **Test 2:** Initialize the POJO with 10 random user data (use the minimal requirements). Also, make a code verification for avoiding duplicate email accounts. Then, perform the POST request to create the user.
- **Test 3:** Make the GET request to get the users, asserting that there are no duplicate email accounts.
- **Test 4:** Add a test to update an existing AccountNumber.
- **Test 5:** Add a test to verify the correct deposit of money.
- **Test 6:** Add a test to verify the correct withdrawal of money

In order to implement the best practices keep in mind the following tips:

- Structure the project by making requests reusable, avoid repeating code or endpoints, use TestNG, create a Readme.md file with the specs and steps to run the exercise, and add a gitignore.

- For each request, make sure to include at least one assertion for the Status Code (implement POJOs to handle the response data, not just the body) and make sure to use JavaDoc.

Additional Resources

10 Automation best practices