QUALITY ASSURANCE FOR URL SHORTENER

Positive test cases validate that the system behaves correctly under normal conditions and expected inputs. They ensure that the application functions as intended. In the context of a URL shortening service, positive test cases might include scenarios where users provide valid URLs and expect the service to successfully generate shortened versions.

Negative test cases, on the other hand, aim to identify how well the system handles invalid or unexpected inputs and error conditions. Negative testing helps ensure that the application gracefully handles errors and provides meaningful feedback to users. For a URL shortening service, negative test cases could involve attempting to shorten invalid URLs, using special characters in the input, or checking how the system responds to attempts at abusing the service.

Here's a breakdown of positive and negative test cases for the URL shortening service:

Positive Test Cases:

1. Valid URL Shortening:

- Input: Valid URL (e.g., "https://www.example.com").

- Expected Outcome: Shortened URL is successfully generated.

2. Custom Alias Usage:

- Input: Valid URL with a custom alias (e.g., "https://www.example.com" with custom alias "mylink").

- Expected Outcome: Shortened URL with the custom alias is generated.

3. Redirection Check:

- Input: Clicking on the shortened URL.

- Expected Outcome: User is redirected to the original URL.

Negative Test Cases:

1. Invalid URL:

- Input: Invalid URL (e.g., "invalid-url").

- Expected Outcome: System displays an error message indicating an invalid URL.

2. Duplicate Custom Alias:

- Input: Attempting to use an already existing custom alias.

- Expected Outcome: System displays an error message about the alias already being in use.

3. Malicious Content in URL:

- Input: URL with potentially malicious content.

- Expected Outcome: System rejects the URL and displays a security-related error message.

These examples cover both positive scenarios where the service is expected to work correctly and negative scenarios where the service should gracefully handle errors or potential abuse. The test cases ensure that the URL shortening service is reliable, user-friendly, and secure.

**Test Strategy for URL Shortening Service:**

1. **Functional Testing:**

* *Positive Tests:*
  + Check if you can shorten a regular link.
  + Verify that the short link takes you to the right place.
  + Test customizing short links and tracking statistics.
* *Negative Tests:*
  + Try to shorten an invalid link.
  + Check if using the same custom name causes an issue.
  + Ensure the service rejects harmful links.
* *Edge Cases:*
  + Test really long links.
  + See how the service handles special characters.
  + Check what happens when the list of links is empty.

2. **User Experience Testing:**

* Make sure it's easy to use.
* Check that error messages make sense.
* Test it on different devices and browsers.

3. **Security Testing:**

* Look for common security problems.
* Ensure the service protects against certain types of attacks.
* Check if sensitive data is handled securely.

4. **Automation Testing:**

* Use automated tools to run tests.

5. **Functional Testing:**

* Positive test cases

import requests

import pytest

BASE\_URL = "https://your-url-shortening-service.com"

def valid\_url\_to\_shorten():

return "https://www.example.com"

def test\_successful\_url\_shortening(valid\_url\_to\_shorten):

response = shorten\_url(valid\_url\_to\_shorten)

assert response.status\_code == 200

assert "shortened\_url" in response.json()

assert response.json()["original\_url"] == valid\_url\_to\_shorten

def shorten\_url(url):

endpoint = f"{BASE\_URL}/shorten"

payload = {"url": url}

response = requests.post(endpoint, json=payload)

return response

* Negative Test cases:

import requests

import pytest

BASE\_URL = "https://your-url-shortening-service.com"

def invalid\_url\_to\_shorten():

return "invalid-url"

def test\_invalid\_url\_shortening(invalid\_url\_to\_shorten):

response = shorten\_url(invalid\_url\_to\_shorten)

assert response.status\_code == 400

assert "error" in response.json()

assert "Invalid URL" in response.json()["error"]

def shorten\_url(url):

endpoint = f"{BASE\_URL}/shorten"

payload = {"url": url}

response = requests.post(endpoint, json=payload)

return response

* Edge cases:

import requests

import pytest

BASE\_URL = "https://your-url-shortening-service.com"

def test\_user\_friendly\_shortening\_process():

def test\_clear\_error\_messages():

def test\_consistency\_across\_browsers\_and\_devices():

* Performance Test cases:

import subprocess

import pytest

def test\_simulate\_high\_traffic():

jmeter\_script\_path = "/path/to/your/jmeter/script.jmx"

result = subprocess.run(["jmeter", "-n", "-t", jmeter\_script\_path], capture\_output=True)

assert result.returncode == 0

* Handling high traffic and service abuse:

import requests

import pytest

BASE\_URL = "https://your-url-shortening-service.com"

def test\_rate\_limiting\_effectiveness():

def test\_abnormal\_activity\_monitoring():

def test\_captcha\_mechanism():

def test\_use\_of\_cdns():

* Considerations for maintaining test cases:

def test\_regular\_update\_of\_test\_cases():

def test\_version\_control\_for\_test\_scripts():

def test\_continuous\_integration():