**AXUR Technical assessment: Quality Assurance Analiyst**

This document outlines the test strategy for validating the project requirements designed for a specific application. This app browses a website in search of a user-supplied term and lists the URLs where the term was found.

Considering the Rest API contract and other instructions on how the user should interact with the application as described in the project requirements, a list of positive and negative/destructive test scenarios were designed in order to map the features to be validated into executable test cases.

The test cases were detailed to individually validate each test scenario within the context of positive/negative scenarios. For example: tests case TC01 to TC06 validates both S01 and S02 individually; and TC07 to TC11 test cases were designed to validate test scenarios S03 to S09, individually.

The functional and performance test cases are categorized into Status Code Validations, Response Payload Validations, Correct Application State Validations, Baseline Response Time, and Caching Performance.

In the absence of front-end elements, to execute this test plan it was decided that the best approach for manual testing would be to assemble a Test Collection using Postman. All JSON structures are detailed in this document.

All documents and assets generated for this assessment are also available on the GitHub repository.

*Important considerations:*

*Although the project requirements document does not specify the responses for requests with errors, negative scenarios were designed considering the directives recommended by OpenAPI and HTTP Status Code, even though they cannot be tested.*

**API Testing Test Plan**

|  |  |  |
| --- | --- | --- |
| **Test Scenarios (SC) by Scope (positive / negative)** | **Test Case (TC) Description (validate individually each SC)** | **Test Validation Category** |
| **Positive testing** | | |
| Execute API operations with valid required parameters:  S01: Initiate a search (POST) informing valid required parameters  S02: Query for search results (GET) for a valid path parameter | TC01: Validate HTTP status code on requests response after API operation  Expected results:  – GET and POST requests should return 200 OK | API Functional Testing - Status Code Validation |
| TC02: Validate response structure is a JSON object formed according to data model (schema validation, field types, and mandatory types)  Expected results: – for POST requests: Body displays a key 'id' formed by an 8-character alphanumeric code automatically generated – for GET requests: Body displays a key 'id' formed by an 8-character alphanumeric code automatically generated; a key 'status' with values 'active' / 'done'; a key 'urls' formed by a list of links related to the keyword searched. | API Functional Testing - Response Payload Validation |
| TC03: Validate request Content-Type in HTTP headers  Expected results:  – for GET and POST: the Content-Type in HTTP headers ‘application/json’ | API Functional Testing - Response Payload Validation |
| TC04: Validate that ‘urls’ field content’s on search result response is base URL related  Expected results:  – for GET requests: the links returned in the 'urls' list on search result response must comply with the base URL, either relative or absolute.  *– for POST requests: Not Applicable* | API Functional Testing - Response Payload Validation |
| TC05: Validate response is received in a timely manner (as defined in the requisites/user story). | API Performance Testing - Baseline Response Time |
| TC06: Validate response mandatory ‘urls’ field for simultaneous identical searchs (GET)  Expected results:  – for GET requests: when key ‘status’ value is equals ‘done’ for both search results responses, the values on ‘urls’ list must be identical.  *– for POST requests: Not Applicable* | API Performance Testing - Caching Performance |
| **Negative/destructive testing – invalid input** | | |
| Execute API operations with invalid/wrong inputs:  SC03: Attempting to initiate a search without filling 'keyword' key  SC04: Attempting to initiate a search informing 'keyword' key formed by less than 4 / over than 32 characters  SC05: Attempting to initiate a search removing 'keyword' key from the request payload  SC06: Attempting to initiate a search adding invalid key in the request payload  SC07: Attempting to initiate a search results informing wrong Content-Type in HTTP headers  SC08: Attempting to query search results without informing invalid UUID in path  SC09: Attempting to execute unsupported methods for endpoints, such as PUT, DELETE, PATCH | TC07: Validate HTTP status code on requests response after API operation  Expected results:  – for GET and POST: an erroneous HTTP status code is sent in accordance with error case as defined in spec (NOT 2XX)  – Missing specific information on spec (review documentation) | API Functional Testing - Status Code Validation |
| TC08: Validate a error response structure is received and is a JSON object formed according to data model  Expected results:  - Missing information on spec (review documentation) | API Functional Testing - Response Payload Validation |
| TC09: Verify error response description is correct for this error case and in accordance with spec  Expected results:  - Missing information on spec (review documentation) |
| TC10: Verify that there is a clear and friendly descriptive error response message  Expected results:  - Missing information on spec (review documentation) |
| TC11: Ensure error is received in a timely manner (as defined in the requisites/user story) | API Performance Testing - Baseline Response Time |

**API Execution Collections (POSTMAN)**

|  |  |  |
| --- | --- | --- |
| **Collection’s Test Case** | **Scenario Validated** | **JS Script** |
| TC01-Status Code Validation | S01: Initiate a search (POST) informing valid required parameters | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); |
| TC02-Status Code Validation | S02: Query for search results (GET) for a valid path parameter | pm.test("Status code is 200", function () {  pm.response.to.have.status(200);  }); |
| TC02-Response Payload Validation | S01: Initiate a search (POST) informing valid required parameters | var expectedSchema = {  "type": "object",  "properties": {  "id": {  "type": "string",  "pattern": "^[a-zA-Z0-9]{8}$"  }  },  "required": ["id"]  }  pm.test('response matches JSON schema', () => {  pm.response.to.have.jsonSchema(expectedSchema);  }); |
| TC02-Response Payload Validation | S02: Query for search results (GET) for a valid path parameter | var expectedSchema = {  "type": "object",  "properties": {  "id": {  "type": "string"  },  "status": {  "type": "string"  },  "urls": {  "type": "array",  "items": {  "type": "string",  "format": "uri"  }  }  },  "required": ["id", "status", "urls"]  };  pm.test('response matches JSON schema', () => {  pm.response.to.have.jsonSchema(expectedSchema);  }); |
| TC03-Response Payload Validation | S01: Initiate a search (POST) informing valid required parameters | pm.test("Content-Type is present and have expected values", function () {  pm.response.to.have.header("Content-Type", "application/json");  }); |
| TC03-Response Payload Validation | S02: Query for search results (GET) for a valid path parameter | pm.test("Content-Type is present and have expected values", function () {  pm.response.to.have.header("Content-Type", "application/json");  }); |
| TC04-Response Payload Validation | S02: Query for search results (GET) for a valid path parameter | const baseURL = "http://hiring.axreng.com/";  const urls = pm.response.json().urls;  urls.forEach((url, index) => {  pm.test(`Link have base URL`, function () {  pm.expect(url).to.have.string(baseURL);  });  }); |
| TC05-Baseline Response Time | S01: Initiate a search (POST) informing valid required parameters | pm.test("Verify response time is less than 5 seconds", function () {  const fiveSecondsMs = 5\_000; // 5 seconds in milliseconds  pm.expect(pm.response.responseTime).to.be.below(fiveSecondsMs);  }); |
| TC05-Baseline Response Time | S02: Query for search results (GET) for a valid path parameter | pm.test("Verify response time is less than 5 seconds", function () {  const fiveSecondsMs = 5\_000; // 5 seconds in milliseconds  pm.expect(pm.response.responseTime).to.be.below(fiveSecondsMs);  }); |