SWAGGER PETSTORE : API Test Strategy

Table of Contents

Introduction

Swagger Petstore is an API first core petstore and therefore APIs are our most valuable asset. It is QA engineers responsibility to guarantee the quality and reliability of APIs.

Purpose

The purpose of this document is to describe the methodology and strategy that will be followed for Swagger API Testing. This document would serve as a guideline for the Test Approach to the project. It would define a strategy which explains the overall approach the testing team followed for their testing activities and also highlight on the scope of testing.

Scope

Items in Scopes

The following are the list of the API's that are going to be tested:

- PET (https://petstore.swagger.io/#/pet)
- STORE (https://petstore.swagger.io/#/store)
- USER (https://petstore.swagger.io/#/user)

Items Not-in Scope

- Performance Testing
- Security Testing
- User Interface testing
- Database Testing

Test Process

The following sections details out the different steps involved with testing process:

- **Sprint Test Plan:** Involves in creating the test plan in the planning phase, the same should be updated for the Sprint / release if there are any changes to the Testing Scope, and Test Approach & Strategy for the Sprint /Release.
- **Test Case Design:** Involves identifying Test cases for each user story prioritized for the Sprint, gathering the data required for test execution for every test condition. Review and Rework of Test cases

- **Build Verification Testing:** QA team will perform Smoke test to make sure the build is stable for functional testing or not.
- **Test Execution:** Executing all the created for scoped user stories for the sprint /release
    - Validate the status codes
    - Verify and validate the response
    - Validate the Response time and headers
    - Negative case with valid input
    - Negative case with invalid input
- **API Automation:** Automating Regression tests through RestAssured
- **Defect Tracking & Management:** Reporting defects prioritize the defects and verify the fixes to closure with Regression / Retesting.
- **Regression Testing:** Regression Testing will be performed before the release
- **Release:** All the completed user stories for the Sprint would be released for each Sprint.

API Testing

Each test consists of test actions. These are the individual actions a test needs to take per API test flow. For each API request, the test would need to take the following actions:

1. **Verify correct HTTP status code.** For example, getting data should return 200 OK, creating a resource should return 201 CREATED and unpermitted requests should return 403 FORBIDDEN, etc.
2. **Verify response payload.** Check valid JSON body and correct field names, types, and values — including in error responses.
3. **Verify response headers.** HTTP server headers have implications on both security and performance.
4. **Verify correct application state.** This is optional and applies mainly to manual testing, or when a UI or another interface can be easily inspected.
5. **Verify basic performance sanity.** If an operation was completed successfully but took an unreasonable amount of time, the test fails.

API Demonstration using test Matrix

API #	**Endpoint Cluster**	**Method**	**URL path**	**Description**

1	Pet	POST	/pet/{petId}/uploadImage	Uploads an Image
2	Pet	POST	/pet	Adds a new pet
3	Pet	PUT	/pet	Update an existing pet
4	Pet	GET	/pet/findByStatus	Finds pets by status
5	Pet	GET	/pet/{petId}	Get Pet by ID
6	Pet	POST	/pet/{petId}	Updates a pet in the store data with a form
7	Pet	DELETE	/pet/{petId}	Deletes a pet
8	Store	POST	/store/order	Place an order for a pet
9	Store	GET	/store/order/{orderId}	Find purchase order by ID
10	Store	DELETE	/store/order/{orderId}	Deletes purchase order by ID
11	Store	GET	/store/order/{orderId}	Returns pet inventories by status
12	User	POST	/user/createWithArray	Create a list of users with a given input array.
13	User	POST	/user/createWithList	Create a list of users with a given input array.
14	User	GET	/user/{username}	Get user with user name
15	User	PUT	/user/{username}	Updated user

16	User	DELETE	/user/{username}	Delete user
17	User	GET	/user/login	Logs user into the system
18	User	GET	/user/logout	Logs out current logged in user session
19	User	POST	/user	Create user

#	**Test Scenario Category**	**Test Action Category**	**Test Action Description**
1	Positive/Happy cases		
	Execute API call with valid required/mandatory parameters	Validate status code:	Response should return 2XX HTTP code Returned status code is according to spec: 200 OK for GET requests. 201 for POST or PUT requests creating a new resource. 200, 202, or 204 for DELETE operations.
		Validate payload:	Response is a well-formed JSON object. Response structure is according to data model (schema validation: field names and field types are as expected, including nested objects; field values are as expected; non-nullable fields are not null, etc.)
		Validate headers:	Verify that HTTP headers are as expected like content-type etc according to spec.

2	Negative test with valid input		
	Execute API with valid input that attempts illegal operations. i.e.: – Attempt to create resource that already exists (e.g., Attempt to create existing pet again)		
		Validate headers:	As in #1
3	Negative test with invalid input		
	Execute API calls with invalid input, e.g.: Missing desire parameters Payload having wrong data Payload with incomplete data Missing mandatory field Invalid values in HTTP headers Unsupported methods for API		
		Validate status code:	Response should return 4XX HTTP code
		Validate payload:	Validate the payload has a proper error message.
		Validate headers:	

Test Entry and Exit Criteria

Entry Criteria

- Signed-Off User Stories and/or requirements documents are available to the QA team.
- Development team has completed planned feature implementation.
- The Development team has completed Unit Testing.
- The Test environment is available for the QA team.
- Development team provides release notes indicating features being released & any known constraints/issues.
- Test data required for execution has been loaded or generated.
- All Build Verification Tests pass.

Exit Criteria

- Test team completes execution of planned tests.
- No P1 and P2 priority defects.
- Any open P3, P4 and P5 must be triaged with BA Team/Product Owners before release.

Suspension criteria and Resumption

Test Execution would be suspended under the following conditions:

- Smoke Test fails with critical defects in the application that affects further testing.
- Build is unstable & crashes frequently.

Test Execution will be resumed when following conditions are satisfied:

- Critical defect blocking test execution is addressed & new build provided.
- Build is stable for further testing

Test Assumptions

- It is assumed that all user stories and/or requirement documents will be approved by the Business/Product Owner.
- It is understood that modifications to requirements will occur based on the sprint demo, UAT and User feedback sessions. These modifications will be entered in the production backlog and prioritized by the product Owner.
- Application build will be deployed to the QA environment on time and is of the required quality.
- All “Show-Stopper” bugs receive immediate attention from the development team.

- Every function/module will be unit tested by the development team before releasing the build to QA.
- Functionality will be delivered for testing as per the Build Plan.
- Required resources will be available for testing.
- All documentation will be up to date and delivered to the QA team.

Defect Management

The test engineers will log the defect in Jira. The defects will be assigned to the development lead. Once the defect is fixed it will be marked as fixed by the developer and retested by the test engineer before closing.

A defect must contain the Appropriate key fields (and *Optional*) in addition to product/project specific fields.

1. Defect ID
2. Defect Description [Steps to reproduce and required test data]
3. Assigned to
4. Severity
5. Priority
6. Environment (QA, UAT,PROD)
7. Required snapshots for both Fail and Pass
8. Expected vs Actual result

Guidelines for Defect Classification

Priority

- **P1** - Fix by next build (it’s a blocker issue that needs immediate fix).
- **P2** - Should Fix soon, (specific timing based on test/customer “cost” of workaround, if exists.)
- **P3** - Fix (to replace customer workaround by next project milestone with related deliverable.)
- **P4** - Consider fix by upcoming release (somewhat trivial ticket, but may be postponed.)

Severity

- **Severity 1** - System/Application crash, 404 error or data loss. Product/Service instability, major test blockage, broken build or failed new build
- **Severity 2** - Major loss of functionality or other severe problems (feature unusable; product crashes in obscure cases), Bug in major feature with complex workaround, or moderate test blockage.
- **Severity 3** - Minor functionality and feature problems; may affect 'fit and finish'. Feature problem with simple workaround, or small test impact.

- **Severity 4** - Very minor problems such as misspelled words (typos), unclear wording or error messages in low visibility fields, incorrect tab order in GUI, obscure feature broken, etc. Little or no test impact.

Test Scenarios

API #1 (POST: /pet/{petId}/uploadImage)

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
	POSITIVE CASES		
Provide all the information (petId, additionalMetadata and file) and hit the endpoint.	● Response code 200 ● Response message has information related to the values provided in the endpoint. Eg ("additionalMetadata: null\nFile uploaded to ./batman-8510027_640.webp, 375162 bytes)		
	NEGATIVE CASES		
Provide a value to the petId that isn't a number.	Response code: 404		
	EDGE CASES		
Provide only the petId and hit the endpoint.	Response code 415		

API #2 (POST /pet)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide the pet information in the payload and hit the endpoint	● Response code 200 ● Response message: contains information about the created pet.		
NEGATIVE CASES			
Don't provide the mandatory data as payload and hit the endpoint.	Response code 400		
EDGE CASES			
Create a pet with empty json payload (for eg {})	● Response code 200 ● Response message: Has id of the created pet but empty other values.		

API #3 (PUT /pet)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			

Provide the updated information as payload and hit the endpoint	• Response code 200 • Response message: contains information about the updated pet.		
NEGATIVE CASES			
Provide the incorrect data/empty json as payload and hit the endpoint. (negative Id, not a number)	Response code 4XX		
EDGE CASES			
Provide a part of the json body as payload and hit the endpoint.	• Response code 200 • Response message: contains information of the updated pet with the updated information and the other fields remain the same.		

API #4 (GET /pet/findByStatus)

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
POSITIVE CASES			
Provide a valid status and hit the endpoint	• Response code 200 • Response message: Contains an array of pets that have the status provided in the endpoint.		

Provide multiple valid status and hit the endpoint.	● Response code 200 ● Response message: Contains an array of pets that have the statuses provided in the endpoint.		
EDGE CASES			
Provide an invalid status and hit the endpoint	● Response code 200 ● Response message: Contains an empty array.		

API #5 (GET /pet/{petId})

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
POSITIVE CASES			
Provide a valid petId and hit the endpoint	● Response code 200 ● Response message: contains the information of the provided petId.		
NEGATIVE CASES			
Provide an invalid petId and hit the endpoint	● Response code 404 ● Response message: Pet not found		
Provide a string as petId and hit the endpoint.	● Response code 404 ● Response message: java.lang.NumberFor		

	matException: For input string: xyz		

API #6 (POST /pet/{petId})

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid petId(mandatory),status and name and hit the endpoint	● Response code 200 ● Response message: contains the id of the pet updated. ● Pet information gets updated to the ones provided in the form.		
NEGATIVE CASES			
Provide a petId that does not exist.	● Response code 404 ● Response message: not found		
Provide a string as petId and hit the endpoint.	● Response code 404 ● Response message: java.lang.NumberFormatException: For input string: xyz		
EDGE CASES			
Provide a valid petId and leave the other fields empty and hit the endpoint	● Response code 200 ● Response message: Contains an array of pets that have the		

	statuses provided in the endpoint. ● Pet information stays the same since name and status fields were left empty.		

API #7 (DELETE /pet/{petId})

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
POSITIVE CASES			
Provide a valid petId(mandatory) and hit the endpoint	● Response code 200 ● Response message: successful deletion ●		
Provide a valid petId(mandatory) and the apiKey and hit the endpoint	● Response code 200 ● Response message: contains the id of the pet deleted.		
NEGATIVE CASES			
Provide a petId that does not exist.	● Response code 404 ● Response message: not found		
Provide a string as petId and hit the endpoint.	● Response code 404 ● Response message: java.lang.NumberFormatException: For input string: xyz		

EDGE CASES			
Delete a pet and then try to delete it again.	• Response code 404 • Response message: Pet not found since the pet was already deleted.		

API #8 (POST /store/order)

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
POSITIVE CASES			
Provide all the the information in the payload and hit the endpoint	• Response code 200 • Response message: contains the information regarding the placed order.		
NEGATIVE CASES			
Provide empty payload and hit the endpoint.	• Response code 400 • Response message: No data		
Provide the data of a non-existent pet in the payload and hit the endpoint.	• Response code 404 • Response message: Pet not found		
EDGE CASES			

Provide an empty json payload and hit the endpoint.	● Response code 200 ● Response message: Contains information about the placed order but all the other fields other than Id are 0 or false.		
Provide the quantity of the pet that exceeds the available stock in the payload and hit the endpoint.	● Response code 4XX ● Response message: quantity of order placed exceeds the available stock.		

API #9 (GET /store/order/{orderId})

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid orderId(value lying between 1-10) and hit the endpoint.	● Response code 200 ● Response message: contains the information of the orderId provided in the endpoint.		
NEGATIVE CASES			
Provide an invalid orderId(value not lying between 1-10) and hit the endpoint.	● Response code 404 ● Response message: Order not found		

Provide an invalid orderId(any alphabet eg abc) and hit the endpoint.	● Response code 404 ● Response message: java.lang.NumberFormatException: For input string: \"abc\"		

API #10 (DELETE /store/order/{orderId})

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
	POSITIVE CASES		
Provide a valid orderId and hit the endpoint.	● Response code 200 ● Response message: successful deletion.		
	NEGATIVE CASES		
Provide an invalid orderId and hit the endpoint.	● Response code 404 ● Response message: Order not found		
Provide an invalid orderId(any alphabet eg abc) and hit the endpoint.	● Response code 404 ● Response message: java.lang.NumberFormatException: For input string: \"abc\"		
	EDGE CASES		
Delete an order and then try to delete it again.	● Response code 404 ● Response message: Order not found since the order was already deleted.		

API #11 (GET /store/inventory)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Hit the endpoint.	● Response code 200 ● Response message: contains information regarding available quantity for each pet status.		

API #12 (POST /user/createWithArray)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid array of user details as payload and hit the endpoint.	● Response code 200 ● Response message: ok		
NEGATIVE CASES			
Provide a json and not an array of user details in the payload and hit the endpoint.	● Response code 500 ● Response message: something bad happened		
Provide an invalid dataType in the payload(eg. Provide	● Response code 500		

orderId as alphabets “abc”)and hit the endpoint.	● Response message: something bad happened		
EDGE CASES			
Provide a valid array of user data where the password doesn’t meet the criteria for the password as payload and hit the endpoint.	● Response code 400 ● Response message:Password doesn’t match the criteria.		
Provide a valid array of user (username,email Id and phone already linked to a user)data that already exists in the system.	● Response code 400 ● Response message: User already present in the system.		

API #13 (POST /user/createWithList)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid array of user details as payload and hit the endpoint.	● Response code 200 ● Response message: ok		
NEGATIVE CASES			
Provide a json and not an array of user details in the payload and hit the endpoint.	● Response code 500 ● Response message: something bad happened		

Provide an invalid dataType in the payload(eg. Provide orderId as alphabets “abc”)and hit the endpoint.	• Response code 500 • Response message: something bad happened		
EDGE CASES			
Provide a valid array of user data where the password doesn’t meet the criteria for the password as payload and hit the endpoint.	• Response code 400 • Response message:Password doesn’t match the criteria.		
Provide a valid array of user (username,email Id and phone already linked to a user)data that already exists in the system.	• Response code 400 • Response message: User already present in the system.		

API #14 (GET /user/{username})

TEST SCENARIO	**EXPECTED RESULT**	**ACTUAL RESULT**	**PASS /FAIL**
POSITIVE CASES			
Provide a valid username and hit the endpoint.	• Response code 200 • Response message: contains information of the user that matches the username provided.		
NEGATIVE CASES			

Provide an invalid username that doesn't exist and hit the endpoint.	● Response code 404 ● Response message: User not found		

API #15 (PUT /user/{username})

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide valid username and user data as payload and hit the endpoint.	● Response code 200 ● Response message: confirm the information was successfully updated.		
NEGATIVE CASES			
Provide an invalid username/user Id that doesn't exist and hit the endpoint.	● Response code 500 ● Response message: something bad happened		
EDGE CASES			
Provide a subset of the fields to be updated as payload and hit the endpoint.	● Response code 400 ● Response message: The provided fields get updated and the fields that were not provided stay the same.		

API #16 (DELETE /user/{username})

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid username and hit the endpoint.	● Response code 200 ● Response message: successful deletion.		
NEGATIVE CASES			
Provide an invalid username and hit the endpoint.	● Response code 404 ● Response message: User not found		
EDGE CASES			
Delete an user and then try to delete it again.	● Response code 404 ● Response message: User not found since the user was already deleted.		

API #17 (GET /user/login)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Provide a valid username and password and hit the endpoint.	● Response code 200 ● Response message: logged in user session: ${sessionId}		

NEGATIVE CASES			
Provide an invalid username and password and hit the endpoint.	• Response code 401 • Response message: invalid credentials/Unauthorized		
EDGE CASES			
Provide a valid username but provide it in different case..	• Response code 401 • Response message: invalid credentials/Unauthorized		

API #18 (GET /user/logout)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			
Hit the endpoint.	• Response code 200 • Response message: Logout successful		

API #19 (POST /user)

TEST SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL
POSITIVE CASES			

Provide valid user data as payload and hit the endpoint	● Response code 200 ● Response message: user created and userId		
NEGATIVE CASES			
Provide a username/email/phone of an existing user as payload and hit the endpoint.	● Response code 409 ● Response message: Conflict error about the user already existing.		
EDGE CASES			
Provide optional fields of the user as empty as payload and hit the endpoint.	● Response code 200 ● Response message: User created with the fields left as empty in payload created with null values.		