Project Overview

Project Name: PetStore API Testing

Description:

The PetStore API Testing project focuses on validating the functionality and reliability of the PetStore API, a fictional API used to manage pet-related operations. This project aims to ensure that the API endpoints perform as expected under various conditions and scenarios. The testing suite covers a comprehensive range of functionalities, including CRUD operations, chaining of requests, and data-driven testing.

Goals:

1. **Functionality Validation:** Ensure that the API endpoints work correctly for different operations like creating, retrieving, updating, and deleting resources.
2. **Error Handling:** Verify that the API handles invalid inputs and errors gracefully.
3. **Data Integrity:** Confirm that the API maintains data integrity throughout different operations.
4. **Performance Testing:** Assess the performance of the API under various conditions to ensure it meets performance expectations.

Key Features:

1. **CRUD Operations:**
    - **Create:** Test cases validate the creation of new resources, such as pets, orders, and users.
    - **Read:** Test cases ensure that retrieving resources by valid and invalid identifiers functions as expected.
    - **Update:** Test cases verify that updating existing resources is handled correctly.
    - **Delete:** Test cases check the deletion of resources by valid and invalid identifiers.
2. **Chaining Requests:**
    - Chained operations to test the sequential flow of creating, retrieving, and deleting resources to ensure data consistency and correctness.
3. **Data-Driven Testing:**
    - Utilize external data files to drive test cases for creating, retrieving, and deleting resources, allowing for a wide range of inputs and scenarios without hardcoding values.
4. **Error Handling and Edge Cases:**
    - Test cases designed to handle and validate responses to invalid inputs and edge cases, ensuring the API behaves as expected under these conditions.
5. **Variable Creation and Management:**
    - Dynamic variable creation and management using environment variables and collection variables to handle test data and responses, enabling flexible and reusable test scenarios.

6. **Comprehensive Test Coverage:**
    - A suite of 48 test cases covering a wide array of functionalities, including pets, orders, and users. This includes positive and negative scenarios, as well as data-driven tests.

Testing Methodologies:

- **Functional Testing:** Ensure that each API endpoint performs its intended function correctly.
- **Integration Testing:** Verify that different API endpoints work together seamlessly in a chain of operations.
- **Data-Driven Testing:** Execute test cases using various data sets to validate different scenarios and edge cases.
- **Error Handling Testing:** Check how the API handles invalid requests and edge cases.
- **Performance Testing:** Evaluate the API's performance under different loads and conditions.

This project aims to provide a thorough validation of the PetStore API, ensuring that it meets functional requirements and handles various scenarios effectively.

Test Cases and Scenarios:

Pets

GETPets

- GET TC1_findByStatusAvailable: Retrieve pets with the status "available."

- GET TC2_findByStatusPending:Retrieve pets with the status "pending."

- GET TC3_findByStatusSold: Retrieve pets with the status "sold."

- GET TC4_findByStatusPendingAndSold:Retrieve pets with the status "pending" and "sold."

- GET TC5_findByStatusInvalid: Attempt to retrieve pets with an invalid status.

- GET TC6_findByStatusEmpty: Attempt to retrieve pets with an empty status.

- GET TC7_findByStatusAvailableAndPending: Retrieve pets with the status "available" and "pending."

findByPetID

- GET TC8_pet_petId:Retrieve a pet by a valid pet ID.

- GET TC9_pet_invalid_petId: Attempt to retrieve a pet by an invalid pet ID.

Createpets

- POST TC10_CreatePet: Create a pet with valid details.
- POST TC11_CreatePet_invalidDetails: Attempt to create a pet with invalid details.

UpdatePets:

- PUT TC13_UpdatePet: Update an existing pet with new details.

DeletePets:

- DEL TC14_DeletePetById: Delete a pet by a valid pet ID.

Pet_Chaining:

- POST TC15_CreatePet:Create a pet.
- GET TC16_pet_petId: Retrieve the pet by ID to verify creation.
- DEL TC17_DeletePetById:Delete the pet by ID and verify deletion.

Store

Order:

- GET TC21_storeInventory: Retrieve the store's inventory.

GetOrderByID

- GET TC22_PurchaseOrderByValidId: Retrieve an order by a valid ID.
- GET TC23_PurchaseOrderByInvalidId: Attempt to retrieve an order by an invalid ID.

CreateOrder

- POST TC24_PlaceOrderPet:Place an order for a pet with valid details.
- POST TC25_PlaceOrderByIdInvalidId: Attempt to place an order with an invalid pet ID.
- POST TC26_PlaceOrderByIdEmptyId: Attempt to place an order with an empty pet ID.

DeleteOrder

- DEL TC27_DeletePurchaseOrder: Delete an order by a valid ID.
- DEL TC28_DeletePurchaseOrderInvalidId: Attempt to delete an order with an invalid ID.

OrderChaining

- POST TC29_PlaceOrderPet: Place an order for a pet.
- GET TC30_PurchaseOrderByID: Retrieve the order by ID to verify creation.
- DEL TC31_DeletePurchaseOrderByID: Delete the order by ID and verify deletion.

User:

CreateNewUser

- POST TC35_CreateNewUser: Create a new user with valid details.
- POST TC36_CreateNewUserInvalid: Attempt to create a new user with invalid details.

LoginUser

- GET TC37_LoginUser: Log in a user with valid credentials.

LogOutUser

- GET TC38_LogOutUser: Log out a user.

Get_ByUsername

- GET TC39_GET_Username: Retrieve user details by a valid username.
- GET TC40_GET_InvalidUsername: Attempt to retrieve user details by an invalid username.

UpdateUser

- PUT TC41_UpdateUser: Update user details with valid information.

DeleteUser

- DEL TC42_DeleteUser: Delete a user by a valid username.

User Chaining

- POST TC43_CreateNewUser: Create a new user.
- GET TC44_GET_Username: Retrieve the user by username to verify creation.
- DEL TC45_DeleteUser: Delete the user by username and verify deletion.

DataDriven

Pet_DataDriven

- POST TC18_CreatePet Create a pet using data-driven inputs.

- GET TC19_pet_petId: Retrieve the pet by ID to verify creation.

- DEL TC20_DeletePetByPetId: Delete the pet by ID and verify deletion.

OrderDataDriven

- POST TC32_PlaceOrderPet Copy: Place an order using data-driven inputs.

- GET TC33_PurchaseOrderByID Copy: Retrieve the order by ID to verify creation.

- DEL TC34_DeletePurchaseOrderByID Copy: Delete the order by ID and verify deletion.

UserDataDriven

- POST TC46_CreateNewUser Copy: Create a new user using data-driven inputs.

- GET TC47_GET_Username Copy: Retrieve the user by username to verify creation.

- DEL TC48_DeleteUser Copy: Delete the user by username and verify deletion.

Traceability Matrix

Requirement ID	Requirement Description	Test Case IDs	Status	Comments
R1	Retrieve pets by status (available, pending, sold)	TC1, TC2, TC3, TC4, TC5, TC6, TC7	Incomplete	TC5 and TC6 do not satisfy requirements
R2	Retrieve pet by pet ID	TC8, TC9	Completed	
R3	Create pet with valid and invalid details	TC10, TC11	Completed	
R4	Update pet details	TC13	Completed	
R5	Delete pet by pet ID	TC14	Completed	
R6	Create, retrieve, and delete pets (chaining)	TC15, TC16, TC17	Completed	
R7	Retrieve store inventory	TC21	Completed	
R8	Retrieve order by valid and invalid IDs	TC22, TC23	Completed	
R9	Place and delete orders with valid and invalid IDs	TC24, TC25, TC26, TC27, TC28	Completed	
R10	Create, retrieve, and delete orders (chaining)	TC29, TC30, TC31	Completed	
R11	Create, login, logout,	TC35, TC36, TC37,	Completed	

Requirement ID	Requirement Description	Test Case IDs	Status	Comments
	retrieve, update, and delete users	TC38, TC39, TC40, TC41, TC42, TC43, TC44, TC45		
R12	Data-driven testing for pets, orders, and users	TC18, TC19, TC20, TC32, TC33, TC34, TC46, TC47, TC48	Completed	TC5 and TC6 do not satisfy requirements

Notes:

- **Requirement R1**: Test cases TC5 and TC6 are designed to check the behavior of the API when an invalid or empty status is provided. However, these test cases do not fully meet the requirement specifications for error handling and edge cases. They need to be reviewed and updated to cover the necessary scenarios comprehensively.
- **Requirement R2-R12**: All other test cases meet their respective requirements, covering the specified functionalities and error conditions.