Test Documentation: Simple Book API

Author: *Nikolay Tasev*
Project: *API Testing Portfolio – Simple Book API*
Tool: *Postman*
Date: *05.07.2025*

1. Introduction

This document contains the manual test cases for the **Simple Book API**, a RESTful API for an online bookstore simulation. The goal is to demonstrate a structured and functional approach to testing common API endpoints using Postman.

2. Scope

The test cases cover:

- API availability
- Retrieval of available books
- Placing, retrieving, updating, and deleting orders
- Registering an API client
- Use of dynamic variables and test scripts in Postman

3. Environment Details

Property	***Value***
Base URL	*https://simple-books-api.glitch.me*
Authorization	*Bearer Token (stored as Postman variable)*
Tool	*Postman v11.52*
Collection	*See https://github.com/NTasev/api-testing-simple-book/blob/main/postman_collection.json file*

4. Test Cases:

Test Case 1: Place an Order for a Non-Fiction Book

Test Case ID	TC_API_001
Test Description	Verify that a user can place an order for a non-fiction book successfully via the API.
Preconditions	- The API is online - Valid access token is present - A non-fiction book with available: true exists

Test Steps	1. Send **GET /status** to check **API availability** 2. Send **GET /books?type=non-fiction** 3. Extract **bookId** from available book 4. Send **GET /books/:bookId?limit=1** to confirm stock 5. Send **POST /orders with bookId** and random name 6. Save **orderId** from response
Expected Result	- Status codes **200** and **201** - JSON response includes a valid orderId - Order is placed
Actual Result	Order placed. Received **orderId:** OWCHg5_z9Mkgfo8t-2yug
Status	Passed
Comments/Notes	Chained requests using global variables; dynamic customer name used with {{$randomFullName}}

Test Case 2: Reject Order Without Authorization (Negative)

Test Case ID	TC_API_002
Test Description	Ensure that unauthorized users cannot place book orders
Preconditions	- API is reachable - Authorization header is not included
Test Steps	1. Send **POST /orders without** bearer token 2. Use valid **bookId** and customer name
Expected Result	- Status code: **401 Unauthorized** - Error message indicating missing or invalid token
Actual Result	Received **401 Unauthorized** with message "Missing Authorization header"
Status	Passed
Comments/Notes	Negative test to validate backend access control

Test Case 3: Retrieve Book Details

Test Case ID	TC_API_003
Test Description	Verify book details can be fetched successfully using a valid ID
Preconditions	- API is reachable - Valid bookId is known from previous request

Test Steps	1. Send **GET /books/:bookId?limit=1** 2. Validate fields such as **name, type,** and **current-stock**
Expected Result	- Status code: **200 OK** - **JSON** includes **valid book details** - current-stock > 0
Actual Result	Book details retrieved. Stock is 4 units
Status	Passed
Comments/Notes	Confirmed that the limit parameter is accepted and does not break the request

Test Case 4: Update an Order

Test Case ID	TC_API_004
Test Description	Verify that the customer name in an existing order can be updated
Preconditions	- Valid orderId is known - Token is present
Test Steps	1. Send **PATCH /orders/:orderId** 2. In body, update **customerName** to **John {{$randomLastName}}**
Expected Result	- Status code: **204 No Content** - No error returned
Actual Result	Status 204 received
Status	Passed
Comments/Notes	Used randomized name to test dynamic update

Test Case 5: Delete an Order

Test Case ID	TC_API_005
Test Description	Verify that an order can be deleted with a valid order ID and token
Preconditions	- A previously created orderId exists - Token is active
Test Steps	1. Send **DELETE /orders/:orderId** with valid auth 2. Observe status code

Expected Result	- Status code: **204 No Content** - Order is deleted successfully
Actual Result	Received 204 No Content
Status	Passed
Comments/Notes	Final step in the CRUD flow; order was removed successfully

5. Variables Used:

Variable Name	**Description**
BaseUrl	API base URL
bookId	Stored after retrieving books
orderId	Stored after placing an order
accessToken	Used for authenticated requests

6. Postman Features Used:

- **Chaining requests** using pm.globals.set() and pm.globals.get()
- **Dynamic tests** with pm.test() and pm.expect()
- **Postman variables** (global & environment)
- **Scripted execution** using pm.execution.setNextRequest()
- **Random data** generation: {{$randomFullName}}

7. Conclusion

This example of all test cases covers the core CRUD functionalities of the Simple Book API. It demonstrates a real-world simulation of ordering and managing a book purchase using chained Postman requests, validation scripts, and dynamic data.

The structure follows a clear and maintainable pattern and can be reused or expanded into automated tests in the future.