Assignment No. 4 – API Testing using Postman

Domain: QA (Quality Assurance)
Organization: 10Pearls Pakistan
Name: Zainab Arif
Submission Date: 3rd Oct 2025

This document provides a detailed step-by-step guide for creating and executing an API testing collection in Postman. The assignment demonstrates the use of variables, requests, pre-request scripts, test scripts, and verification methods.

Open Postman

- Launch Postman application (Windows/Mac) or open Postman Web.
- Make sure you are signed in (optional).

Step 1: Create the Collection & Variables

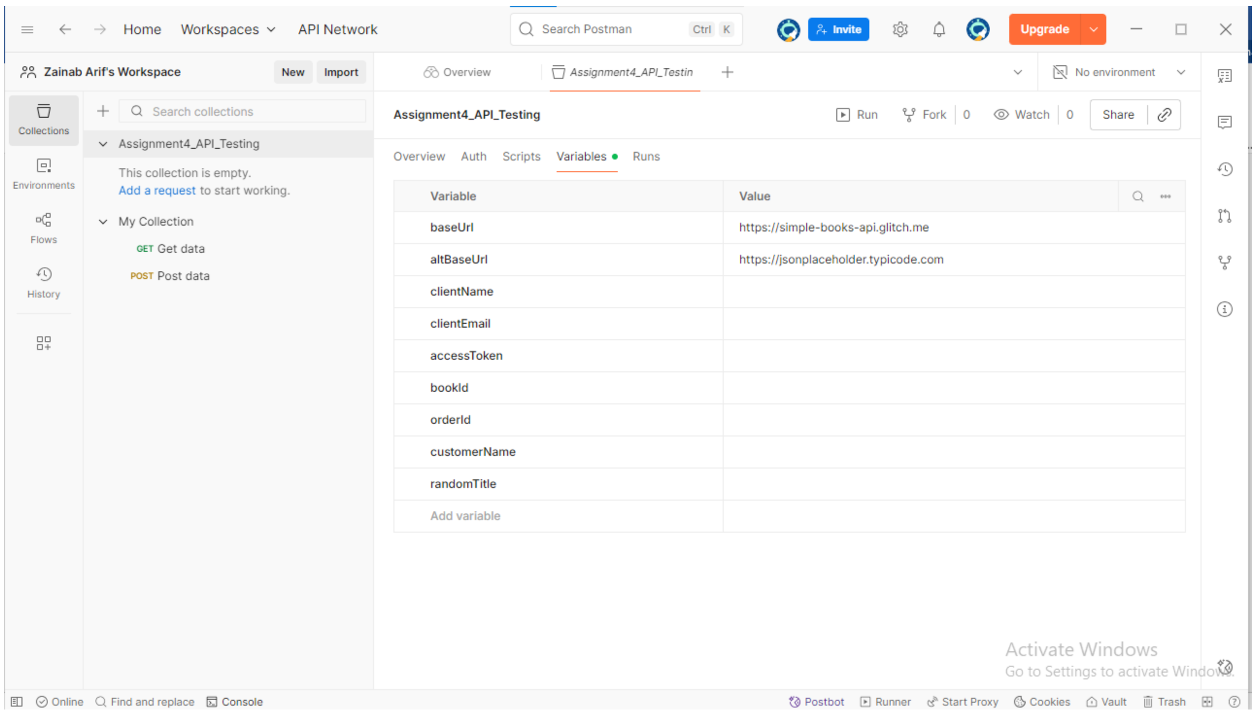
- Click New → Collection. Name: Assignment4_API_Testing → Create.
- Click the collection name in the left sidebar → Variables tab → Add these variables:
- Variables: baseUrl, altBaseUrl, clientName, clientEmail, accessToken, bookId, orderId, customerName, randomTitle
- Click Save (top right in collection variables).

1. Click **New** → **Collection**.
   - Name: Assignment4_API_Testing → **Create**.
2. Click the collection name in left sidebar → **Variables** tab → **Add** these variables exactly:

Variable	Initial Value
baseUrl	https://simple-books-api.glitch.me
altBaseUrl	https://jsonplaceholder.typicode.com
clientName	*(leave empty)*
clientEmail	*(leave empty)*
accessToken	*(leave empty)*
bookId	*(leave empty)*
orderId	*(leave empty)*
customerName	*(leave empty)*

Variable	Initial Value
randomTitle	*(leave empty)*

3. Click **Save** (top right in collection variables).



Assignment4_API_Testing

Variable	Value
baseUrl	https://simple-books-api.glitch.me
altBaseUrl	https://jsonplaceholder.typicode.com
clientName	
clientEmail	
accessToken	
bookId	
orderId	
customerName	
randomTitle	
Add variable	

Step 2: Create Request 01 — GET /status

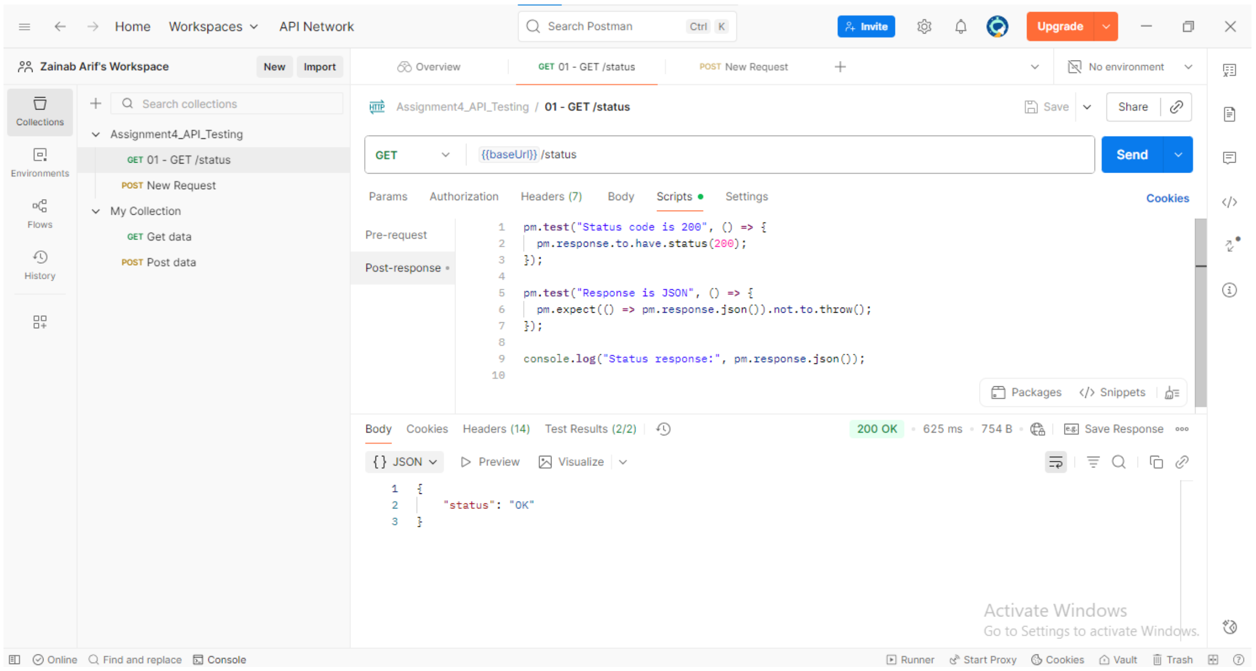
- New → Request. Name it: 01 - GET /status. Save into Assignment4_API_Testing.
- Set Method = GET. URL: {{baseUrl}}/status
- Tests Tab:
- pm.test("Status code is 200", () => { pm.response.to.have.status(200); });
- pm.test("Response is JSON", () => { pm.expect(() => pm.response.json()).not.to.throw(); });
- Check console for output.

1. Click **New** → **Request**. Name it: 01 - GET /status. Save into Assignment4_API_Testing.
2. Set Method = **GET**. URL:
3. {{baseUrl}}/status
4. No Body. Tests tab → paste:

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});
```

```
pm.test("Response is JSON", () => {
  pm.expect(() => pm.response.json()).not.to.throw();
});

console.log("Status response:", pm.response.json());
```

4. Click **Save** → **Send**. Check response and Postman Console (View → Show Postman Console).



GET {{baseUrl}} /status

```
pm.test("Status code is 200", () => {
  pm.response.to.have.status(200);
});

pm.test("Response is JSON", () => {
  pm.expect(() => pm.response.json()).not.to.throw();
});

console.log("Status response:", pm.response.json());
```

200 OK · 625 ms · 754 B

```
{
    "status": "OK"
}
```

Activate Windows
Go to Settings to activate Windows.

Step 3: GET /books (List of Books + Capture bookId)

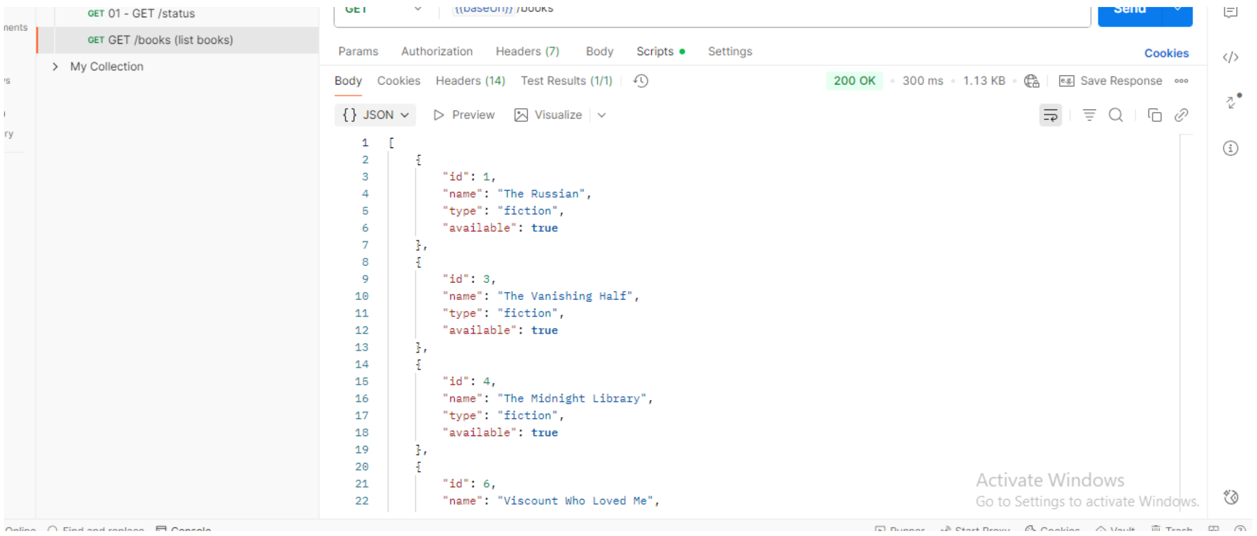
- Create new GET request: URL = {{baseUrl}}/books
- Expected 200 OK with list of books.
- Add Test Script to capture bookId.
- Store first bookId in a variable.

1. **Set the Base URL**
   - Create a variable named baseUrl.
   - Assign this value to it:
   - https://simple-books-api.glitch.me
2. **Create a new GET request**
   - Request URL:
   - {{baseUrl}}/books
3. **Send the request**
   - If everything is correct, you should receive a **200 OK** response.
   - The response body will contain a **list of books** in JSON format. Example:
   - [

   - { "id": 3, "name": "The Vanishing Half", "available": true },
   - { "id": 1, "name": "The Russian", "type": "fiction" }
   -]
4. **Add a Test Script to capture bookId**
   - Go to the **Tests** (or Scripts) tab and paste this code:
   - pm.test("Books list retrieved", () => {
   -     pm.response.to.have.status(200);
   - });
   -
   - let books = pm.response.json();
   - console.log("Books:", books);
   -
   - // Save first bookId in a variable
   - if (books.length > 0) {
   -     pm.collectionVariables.set("bookId", books[0].id);
   -     console.log("Saved bookId:", books[0].id);
   - }

5. **Check the Console Output**
   - You should see:
   - Saved bookId: 3

     (or whichever book ID came first in the list).



```
[
    {
        "id": 1,
        "name": "The Russian",
        "type": "fiction",
        "available": true
    },
    {
        "id": 3,
        "name": "The Vanishing Half",
        "type": "fiction",
        "available": true
    },
    {
        "id": 4,
        "name": "The Midnight Library",
        "type": "fiction",
        "available": true
    },
    {
        "id": 6,
        "name": "Viscount Who Loved Me",
```

Step 4: Create Request 03 — GET /books/{{bookId}}

- New Request: 03 - GET /books/:id
- Tests include validating ID matches and adding an intentional failing test.

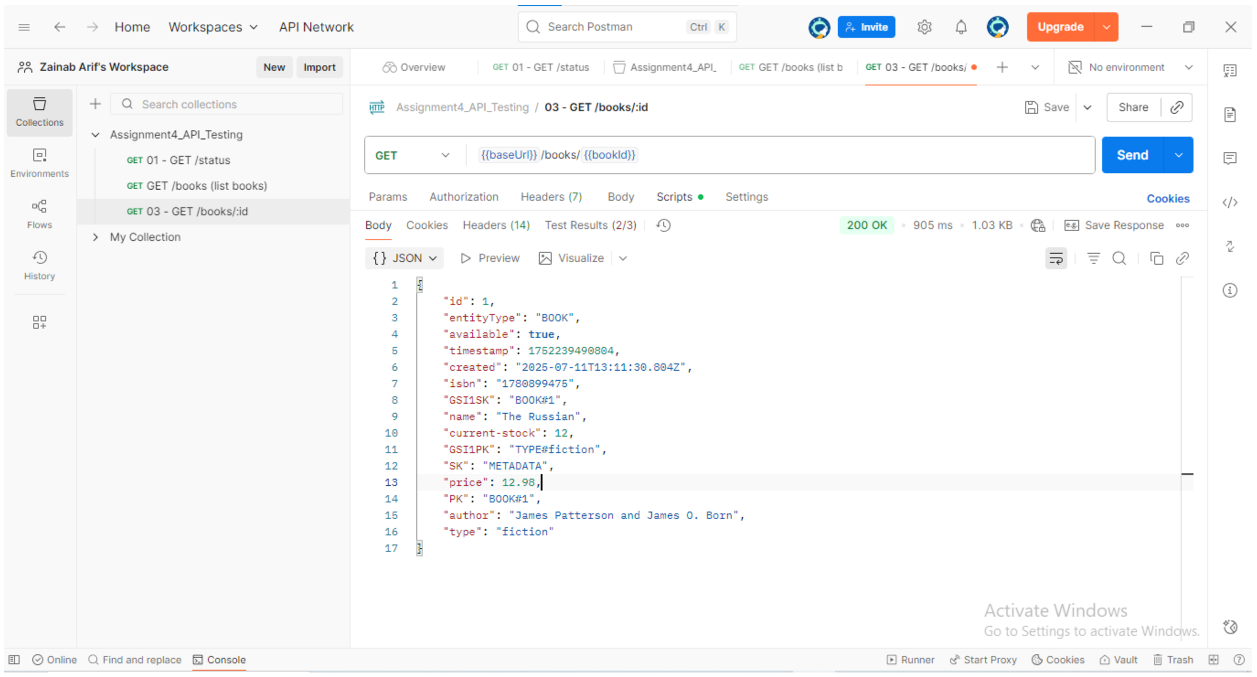
1. New Request → 03 - GET /books/:id → save.
2. Method **GET**, URL:

{{baseUrl}}/books/{{bookId}}

3. Tests tab → paste:

```
pm.test("Status is 200", () => pm.response.to.have.status(200));

pm.test("Returned id matches requested bookId", () => {
  const json = pm.response.json();
  const returnedId = json.id;
  const expectedId = pm.collectionVariables.get("bookId");
  pm.expect(String(returnedId)).to.eql(String(expectedId));
});

console.log("Single book response:", pm.response.json());

// Intentionally failing test (task requires a failing assertion)
pm.test("Deliberate failing test — book type is 'science'", () => {
  pm.expect(pm.response.json().type).to.eql("science");
});
```

4. Save → Send. Expect the last test to **fail** (that's okay — it's by design). Check which assertions pass/fail in the Tests tab.



```
{
    "id": 1,
    "entityType": "BOOK",
    "available": true,
    "timestamp": 1752239490804,
    "created": "2025-07-11T13:11:30.804Z",
    "isbn": "1780899475",
    "GSI1SK": "BOOK#1",
    "name": "The Russian",
    "current-stock": 12,
    "GSI1PK": "TYPE#fiction",
    "SK": "METADATA",
    "price": 12.98,
    "PK": "BOOK#1",
    "author": "James Patterson and James O. Born",
    "type": "fiction"
}
```

Step 5: Create Request 04 — POST /api-clients

- Registers client and retrieves accessToken.
- Pre-request script generates random clientName and clientEmail.
- Tests check response status and token presence.

1. New Request → 04 - POST /api-clients → save.
2. Method **POST**, URL:

```
{{baseUrl}}/api-clients
```

3. Headers → Content-Type: application/json
4. Body → raw JSON:

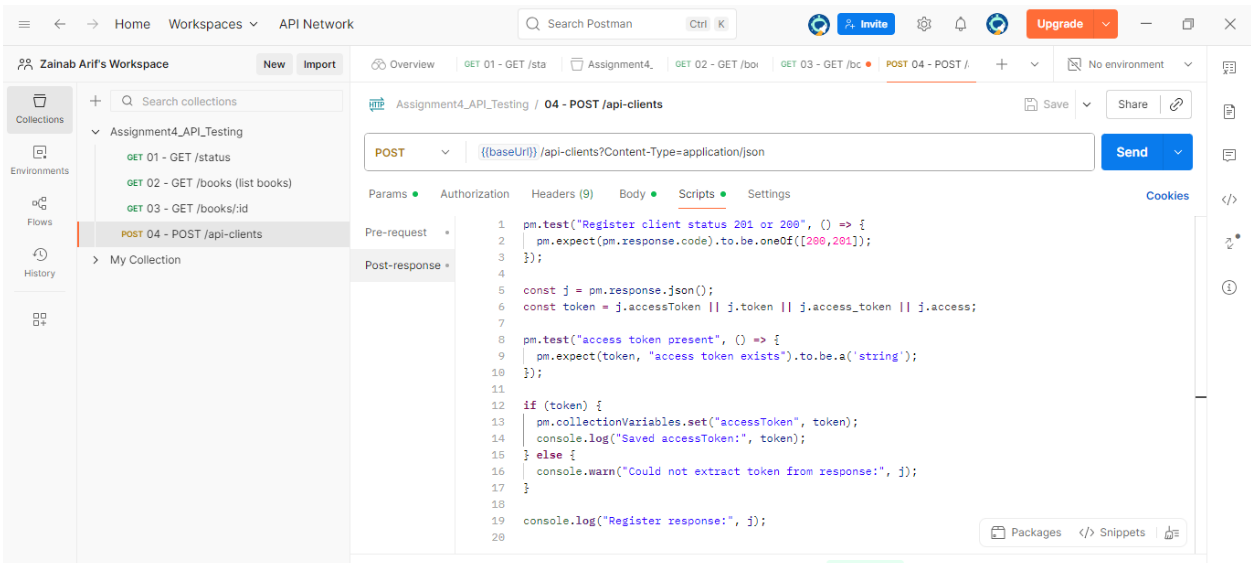
```
{
  "clientName": "{{clientName}}",
  "clientEmail": "{{clientEmail}}"
}
```

5. **Pre-request Script** tab → paste:

```
const r = Math.floor(Math.random() * 1000000);
pm.collectionVariables.set("clientName", `PostmanClient-${r}`);
pm.collectionVariables.set("clientEmail", `client${r}@example.com`);
console.log("clientName & clientEmail set:", pm.collectionVariables.get("clientName"),
pm.collectionVariables.get("clientEmail"));
```

6. **Tests** tab → paste:

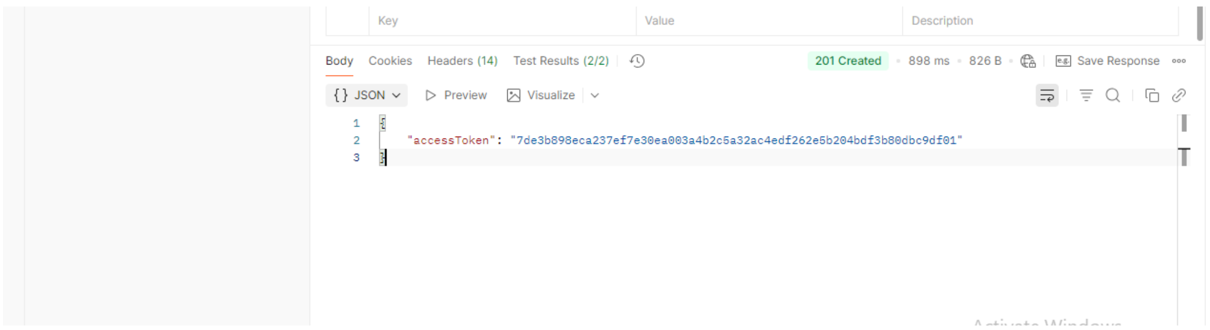
```
pm.test("Register client status 201 or 200", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

const j = pm.response.json();
const token = j.accessToken || j.token || j.access_token || j.access;
pm.test("access token present", () => {
  pm.expect(token, "access token exists").to.be.a('string');
});

if (token) {
  pm.collectionVariables.set("accessToken", token);
  console.log("Saved accessToken:", token);
} else {
  console.warn("Could not extract token from response:", j);
}

console.log("Register response:", j);
```



POST {{baseUrl}} /api-clients?Content-Type=application/json

```
pm.test("Register client status 201 or 200", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

const j = pm.response.json();
const token = j.accessToken || j.token || j.access_token || j.access;

pm.test("access token present", () => {
  pm.expect(token, "access token exists").to.be.a('string');
});

if (token) {
  pm.collectionVariables.set("accessToken", token);
  console.log("Saved accessToken:", token);
} else {
  console.warn("Could not extract token from response:", j);
}

console.log("Register response:", j);
```

7. Save → Send. After success check collection variable accessToken.



201 Created · 898 ms · 826 B

```
{
    "accessToken": "7de3b898eca237ef7e30ea003a4b2c5a32ac4edf262e5b204bdf3b80dbc9df01"
}
```

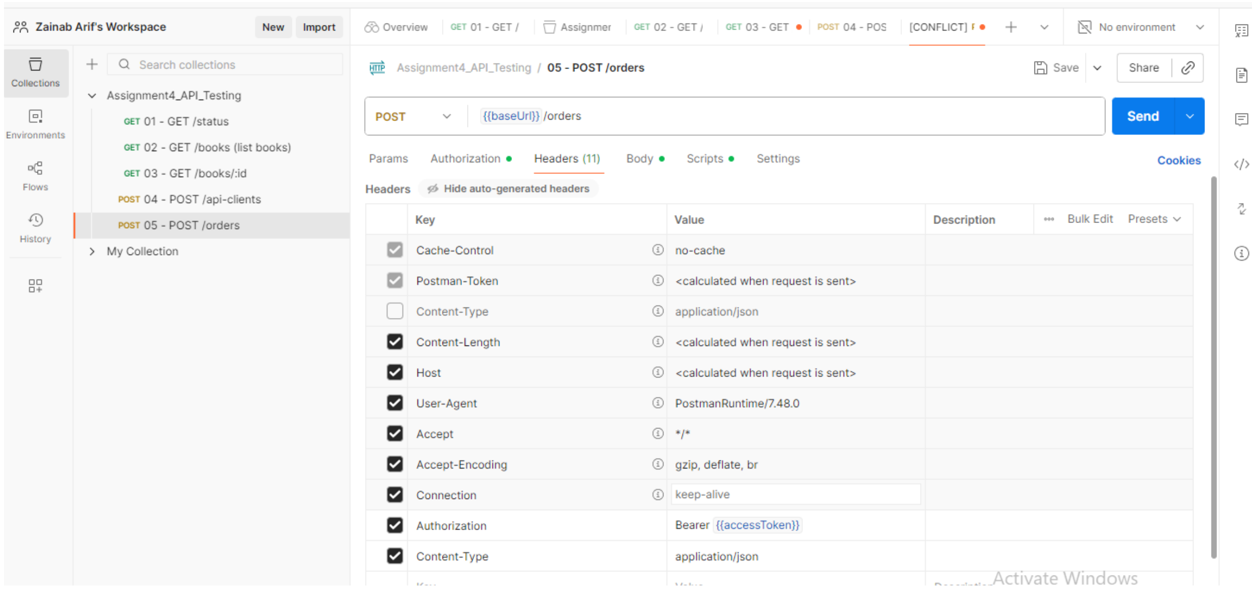
Step 6: Create Request 05 — POST /orders

- Creates an order with bookId and customerName.
- Authorization → Bearer Token with {{accessToken}}.
- Tests capture orderId.

1. New Request → 05 - POST /orders → save.
2. Method **POST**, URL:

{{baseUrl}}/orders

3. Authorization tab → select **Bearer Token** → Token: {{accessToken}}
4. Headers → Content-Type: application/json



Zainab Arif's Workspace

Assignment4_API_Testing / 05 - POST /orders

POST {{baseUrl}} /orders

Headers (11)

	Key	Value	Description
✓	Cache-Control	no-cache	
✓	Postman-Token	<calculated when request is sent>	
☐	Content-Type	application/json	
✓	Content-Length	<calculated when request is sent>	
✓	Host	<calculated when request is sent>	
✓	User-Agent	PostmanRuntime/7.48.0	
✓	Accept	*/*	
✓	Accept-Encoding	gzip, deflate, br	
✓	Connection	keep-alive	
✓	Authorization	Bearer {{accessToken}}	
✓	Content-Type	application/json	

5. Body → raw JSON:

```
{
  "bookId": {{bookId}},
  "customerName": "{{customerName}}"
}
```

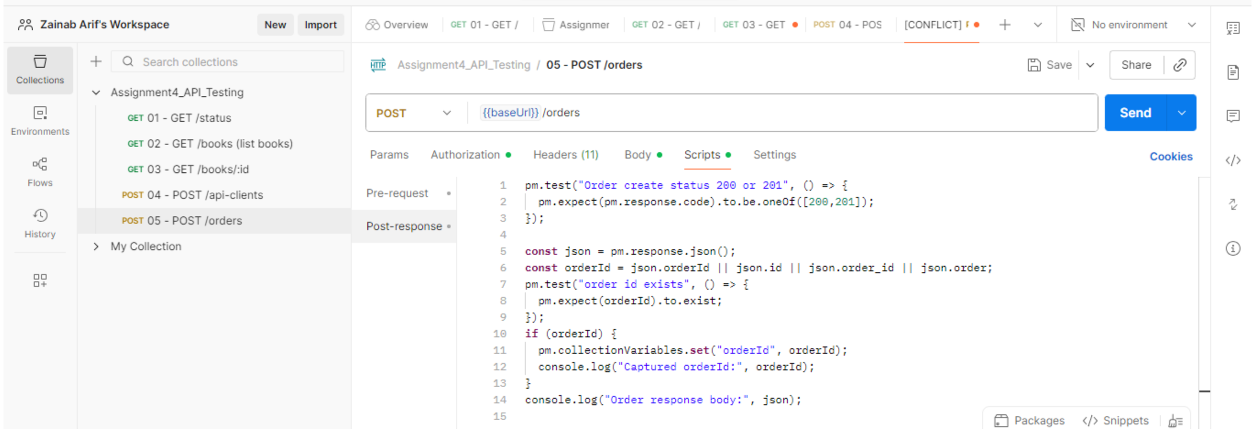
6. Pre-request Script:

```
const r = Math.floor(Math.random() * 1000000);
pm.collectionVariables.set("customerName", `Cust-${r}`);
console.log("customerName:", pm.collectionVariables.get("customerName"));
```

7. Tests:

```
pm.test("Order create status 200 or 201", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

const json = pm.response.json();
const orderId = json.orderId || json.id || json.order_id || json.order;
pm.test("order id exists", () => {
  pm.expect(orderId).to.exist;
});
if (orderId) {
  pm.collectionVariables.set("orderId", orderId);
  console.log("Captured orderId:", orderId);
}
```

console.log("Order response body:", json);



Zainab Arif's Workspace · New · Import

Assignment4_API_Testing
- GET 01 - GET /status
- GET 02 - GET /books (list books)
- GET 03 - GET /books/:id
- POST 04 - POST /api-clients
- POST 05 - POST /orders

My Collection

Assignment4_API_Testing / 05 - POST /orders

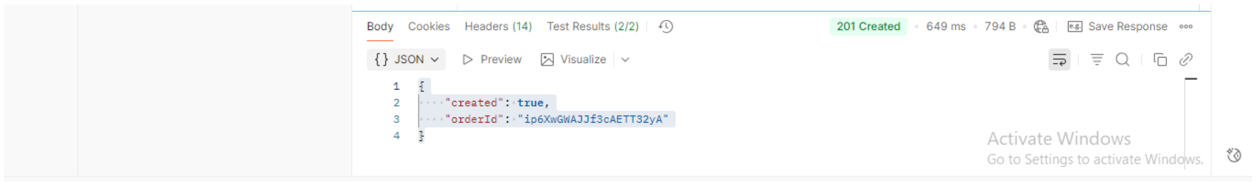
POST {{baseUrl}}/orders · Send

Params · Authorization · Headers (11) · Body · Scripts · Settings

Pre-request · Post-response

```
pm.test("Order create status 200 or 201", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,201]);
});

const json = pm.response.json();
const orderId = json.orderId || json.id || json.order_id || json.order;
pm.test("order id exists", () => {
  pm.expect(orderId).to.exist;
});
if (orderId) {
  pm.collectionVariables.set("orderId", orderId);
  console.log("Captured orderId:", orderId);
}
console.log("Order response body:", json);
```

8. Save → Send. After success check orderId.



Body · Cookies · Headers (14) · Test Results (2/2) · 201 Created · 649 ms · 794 B · Save Response

```
{
    "created": true,
    "orderId": "ip6XwGWAJJf3cAETT32yA"
}
```

Step 7: Create Request 06 — GET /orders/{{orderId}}

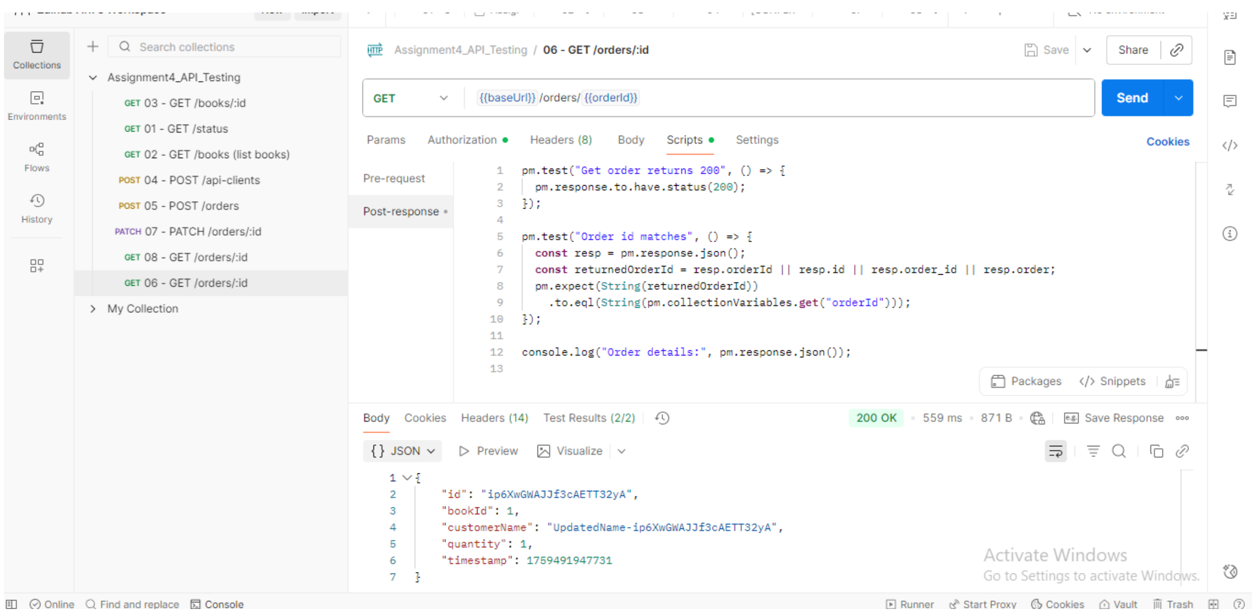
- Fetches order details and validates orderId.

1. New Request → 06 - GET /orders/:id → save.
2. Method **GET**, URL:

{{baseUrl}}/orders/{{orderId}}

3. Authorization → Bearer {{accessToken}}
4. Tests:

```
pm.test("Get order returns 200", () => pm.response.to.have.status(200));
pm.test("Order id matches", () => {
  const resp = pm.response.json();
  const returnedOrderId = resp.orderId || resp.id || resp.order_id || resp.order;
  pm.expect(String(returnedOrderId)).to.eql(String(pm.collectionVariables.get("orderId")));
});
console.log("Order details:", pm.response.json());
```

5. 

Assignment4_API_Testing / 06 - GET /orders/:id

GET {{baseUrl}} /orders/ {{orderId}}

```
pm.test("Get order returns 200", () => {
  pm.response.to.have.status(200);
});

pm.test("Order id matches", () => {
  const resp = pm.response.json();
  const returnedOrderId = resp.orderId || resp.id || resp.order_id || resp.order;
  pm.expect(String(returnedOrderId))
    .to.eql(String(pm.collectionVariables.get("orderId")));
});

console.log("Order details:", pm.response.json());
```

200 OK • 559 ms • 871 B

```
{
    "id": "ip6XwGWAJJf3cAETT32yA",
    "bookId": 1,
    "customerName": "UpdatedName-ip6XwGWAJJf3cAETT32yA",
    "quantity": 1,
    "timestamp": 1759491947731
}
```

Step 8: Create Request 07 — PATCH /orders/{{orderId}}

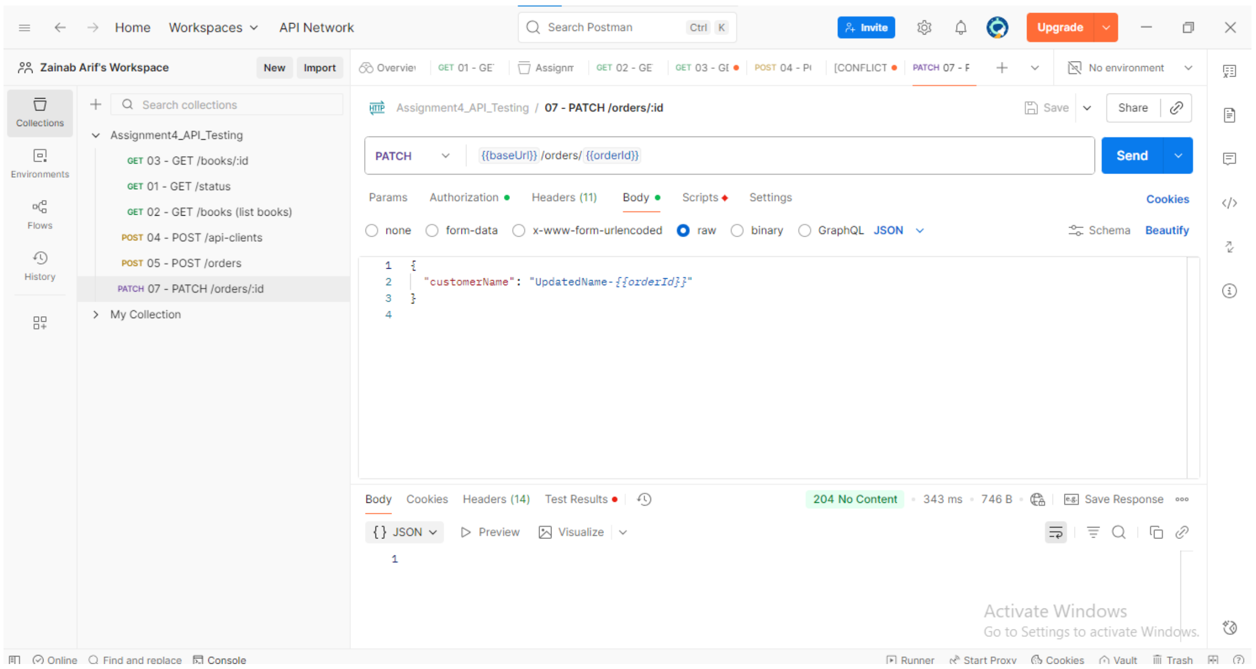
- Updates order partially (customerName).

1. New Request → 07 - PATCH /orders/:id → save.
2. Method **PATCH**, URL:

{{baseUrl}}/orders/{{orderId}}

3. Auth → Bearer {{accessToken}}, Headers → Content-Type: application/json
4. Body:

```
{
  "customerName": "UpdatedName-{{orderId}}"
}
```



Assignment4_API_Testing / 07 - PATCH /orders/:id

PATCH {{baseUrl}} /orders/ {{orderId}}

Body: raw JSON

```
{
  "customerName": "UpdatedName-{{orderId}}"
}
```

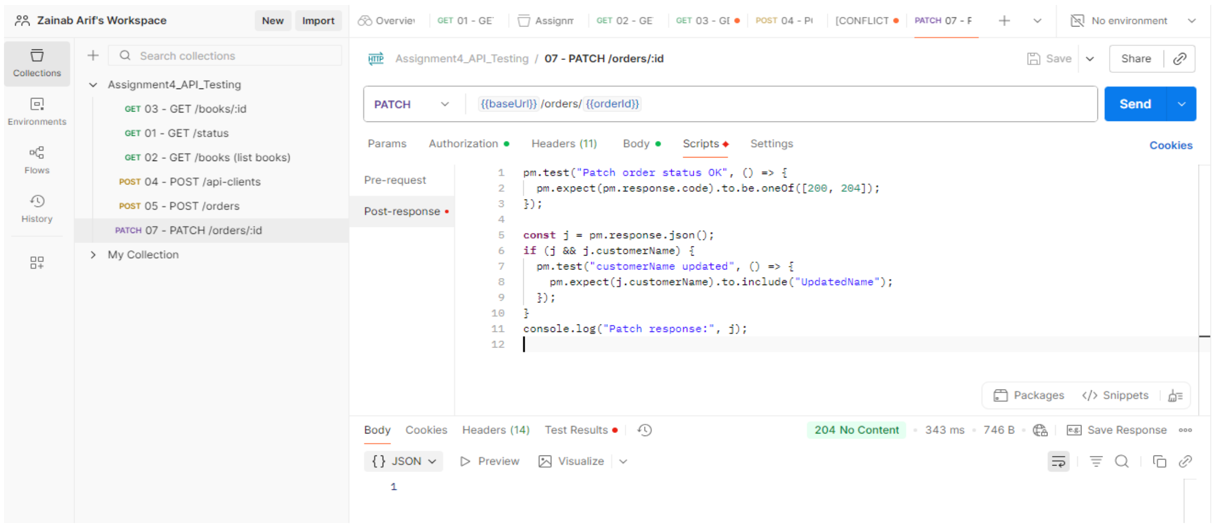
Response: 204 No Content · 343 ms · 746 B

```
1
```

5. Tests:

```
pm.test("Patch order status OK", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,204]);
});

const j = pm.response.json();
if (j && j.customerName) {
  pm.test("customerName updated", () => {
    pm.expect(j.customerName).to.include("UpdatedName");
  });
}
console.log("Patch response:", j);
```

6. Save → Send.



Assignment4_API_Testing / 07 - PATCH /orders/:id

PATCH {{baseUrl}} /orders/ {{orderId}}

Scripts: Post-response

```
pm.test("Patch order status OK", () => {
  pm.expect(pm.response.code).to.be.oneOf([200, 204]);
});

const j = pm.response.json();
if (j && j.customerName) {
  pm.test("customerName updated", () => {
    pm.expect(j.customerName).to.include("UpdatedName");
  });
}
console.log("Patch response:", j);
```

Response: 204 No Content · 343 ms · 746 B

```
1
```

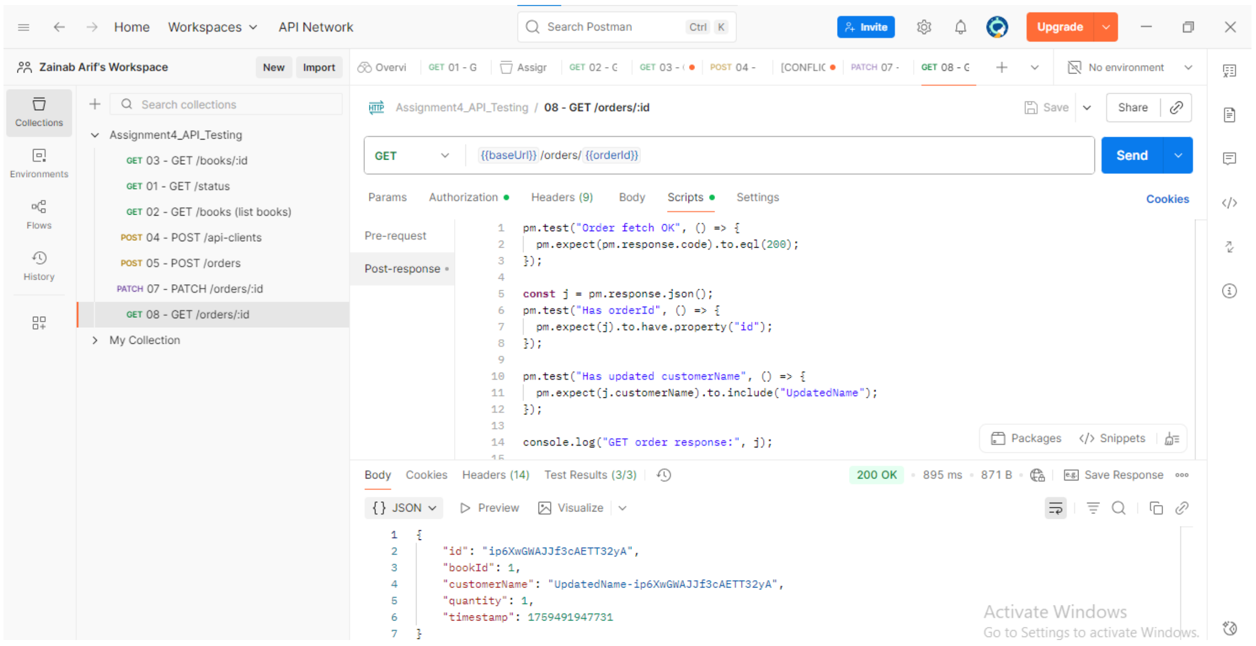
Verification Step: GET after DELETE — GET /orders/{{orderId}}

Steps:

1. Create a new request named 08 - GET /orders/:id and save it.
2. Select method **GET**.
3. Enter the URL:
4. {{baseUrl}}/orders/{{orderId}}
5. Go to **Authorization** tab → select **Bearer Token** and use {{accessToken}}.
6. Add header:
7. Content-Type: application/json
8. Leave the body empty.

Tests (in Tests tab):

```
pm.test("Order fetch OK", () => {
  pm.expect(pm.response.code).to.eql(200);
});

const j = pm.response.json();
pm.test("Has orderId", () => {
  pm.expect(j).to.have.property("id");
});

pm.test("Has updated customerName", () => {
  pm.expect(j.customerName).to.include("UpdatedName");
});

console.log("GET order response:", j);
```


GET {{baseUrl}} /orders/ {{orderId}}

```
pm.test("Order fetch OK", () => {
  pm.expect(pm.response.code).to.eql(200);
});

const j = pm.response.json();
pm.test("Has orderId", () => {
  pm.expect(j).to.have.property("id");
});

pm.test("Has updated customerName", () => {
  pm.expect(j.customerName).to.include("UpdatedName");
});

console.log("GET order response:", j);
```

200 OK · 895 ms · 871 B

```
{
    "id": "ip6XwGWAJJf3cAETT32yA",
    "bookId": 1,
    "customerName": "UpdatedName-ip6XwGWAJJf3cAETT32yA",
    "quantity": 1,
    "timestamp": 1759491947731
}
```

Step 9: Create Request 08 — DELETE /orders/{{orderId}}

- Deletes order.

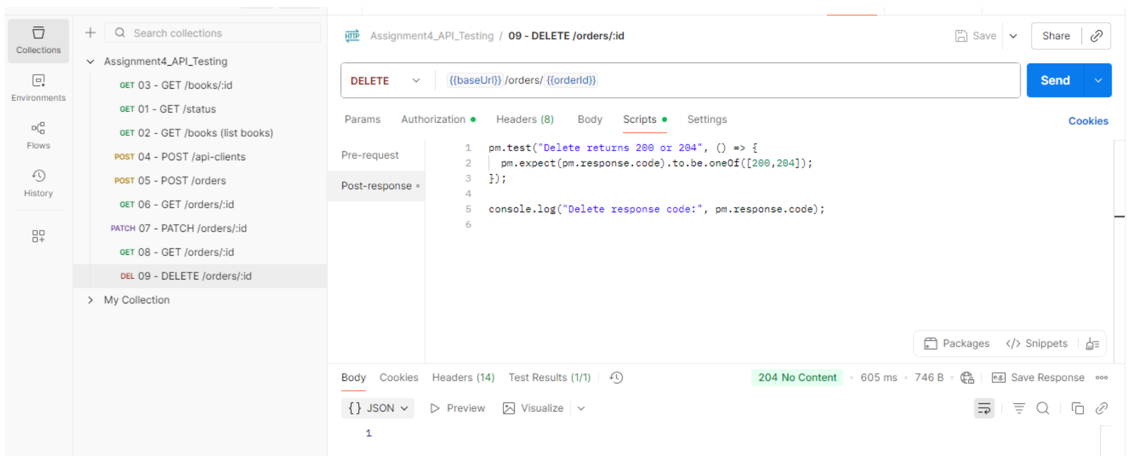
1. New Request → 08 - DELETE /orders/:id → save.
2. Method **DELETE**, URL:

{{baseUrl}}/orders/{{orderId}}

3. Auth → Bearer {{accessToken}}
4. Tests:

```
pm.test("Delete returns 200 or 204", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,204]);
});
console.log("Delete response code:", pm.response.code);
```

5. Save → Send.


Assignment4_API_Testing / 09 - DELETE /orders/:id

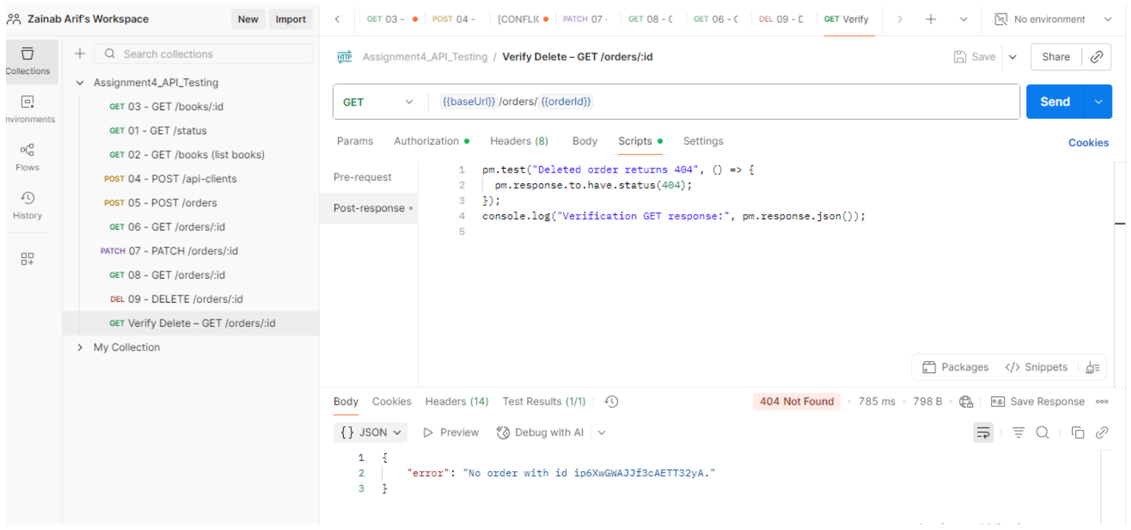
DELETE {{baseUrl}} /orders/ {{orderId}}

Collections: Assignment4_API_Testing
- GET 03 - GET /books/:id
- GET 01 - GET /status
- GET 02 - GET /books (list books)
- POST 04 - POST /api-clients
- POST 05 - POST /orders
- GET 06 - GET /orders/:id
- PATCH 07 - PATCH /orders/:id
- GET 08 - GET /orders/:id
- DEL 09 - DELETE /orders/:id
- My Collection

Post-response:
```
pm.test("Delete returns 200 or 204", () => {
  pm.expect(pm.response.code).to.be.oneOf([200,204]);
});

console.log("Delete response code:", pm.response.code);
```
204 No Content · 605 ms · 746 B

Verification: GET after DELETE should return 404.

New Request → Verify Delete – GET /orders/:id → Save.
Method: **GET**, URL:
{{baseUrl}}/orders/{{orderId}}
Authorization: **Bearer {{accessToken}}**
Test script:
pm.test("Deleted order returns 404", () => {
  pm.response.to.have.status(404);
});
console.log("Verification GET response:", pm.response.json());
Send → Expected output:
{
  "error": "No order with id {{orderId}}"


Zainab Arif's Workspace

Assignment4_API_Testing / Verify Delete – GET /orders/:id

GET {{baseUrl}} /orders/ {{orderId}}

Collections: Assignment4_API_Testing
- GET 03 - GET /books/:id
- GET 01 - GET /status
- GET 02 - GET /books (list books)
- POST 04 - POST /api-clients
- POST 05 - POST /orders
- GET 06 - GET /orders/:id
- PATCH 07 - PATCH /orders/:id
- GET 08 - GET /orders/:id
- DEL 09 - DELETE /orders/:id
- GET Verify Delete – GET /orders/:id
- My Collection

Post-response:
```
pm.test("Deleted order returns 404", () => {
  pm.response.to.have.status(404);
});
console.log("Verification GET response:", pm.response.json());
```
404 Not Found · 785 ms · 798 B
```
{
    "error": "No order with id ip6XwGWAJJf3cAETT32yA."
}
```

Step 10: Create Request 09 — PUT (JSONPlaceholder demo)

- Demonstrates PUT using altBaseUrl.
- Updates title with random value.
- Validates response contains updated title.

(Simple Books API has no PUT — we demonstrate PUT using JSONPlaceholder.)

1. New Request → 09 - PUT /posts/1 (alt) → save.
2. Method **PUT**, URL:

```
{{altBaseUrl}}/posts/1
```

3. Headers → Content-Type: application/json
4. Pre-request Script:

```
pm.collectionVariables.set("randomTitle", "Title-" + Math.floor(Math.random() * 100000));
```

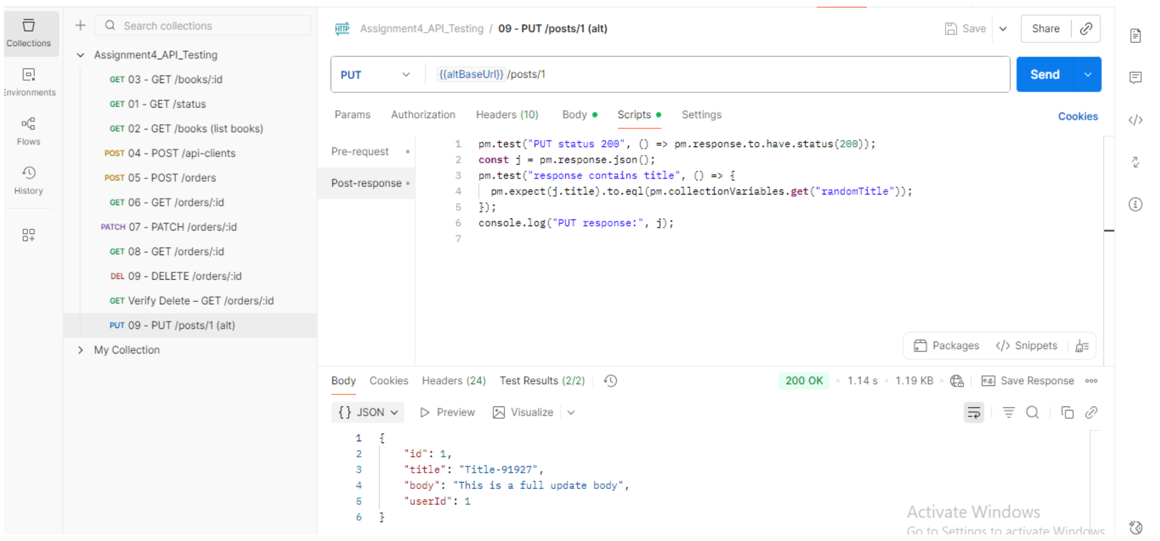
5. Body:

```
{
  "id": 1,
  "title": "{{randomTitle}}",
  "body": "This is a full update body",
  "userId": 1
}
```

6. Tests:

```
pm.test("PUT status 200", () => pm.response.to.have.status(200));
const j = pm.response.json();
pm.test("response contains title", () => {
  pm.expect(j.title).to.eql(pm.collectionVariables.get("randomTitle"));
});
console.log("PUT response:", j);
```

7. Save → Send.



Assignment4_API_Testing / 09 - PUT /posts/1 (alt)

PUT {{altBaseUrl}} /posts/1

Post-response:

```
pm.test("PUT status 200", () => pm.response.to.have.status(200));
const j = pm.response.json();
pm.test("response contains title", () => {
  pm.expect(j.title).to.eql(pm.collectionVariables.get("randomTitle"));
});
console.log("PUT response:", j);
```

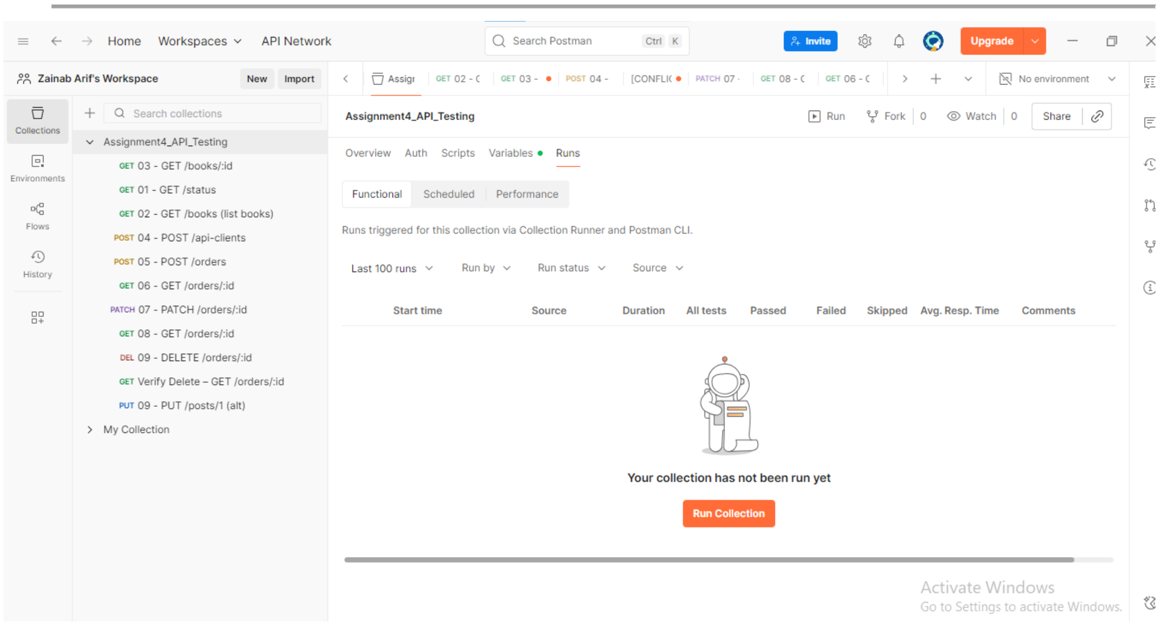
Test Results (2/2) — 200 OK · 1.14 s · 1.19 KB

```
{
    "id": 1,
    "title": "Title-91927",
    "body": "This is a full update body",
    "userId": 1
}
```

Collection Assignment4_API_Testing:
- GET 03 - GET /books/:id
- GET 01 - GET /status
- GET 02 - GET /books (list books)
- POST 04 - POST /api-clients
- POST 05 - POST /orders
- GET 06 - GET /orders/:id
- PATCH 07 - PATCH /orders/:id
- GET 08 - GET /orders/:id
- DEL 09 - DELETE /orders/:id
- GET Verify Delete – GET /orders/:id
- PUT 09 - PUT /posts/1 (alt)

Step 11: Run the entire Collection/ Results History

- Open Collection Runner.
- Run Assignment4_API_Testing with all requests in order.
- Verify test results for pass/fail outcomes.



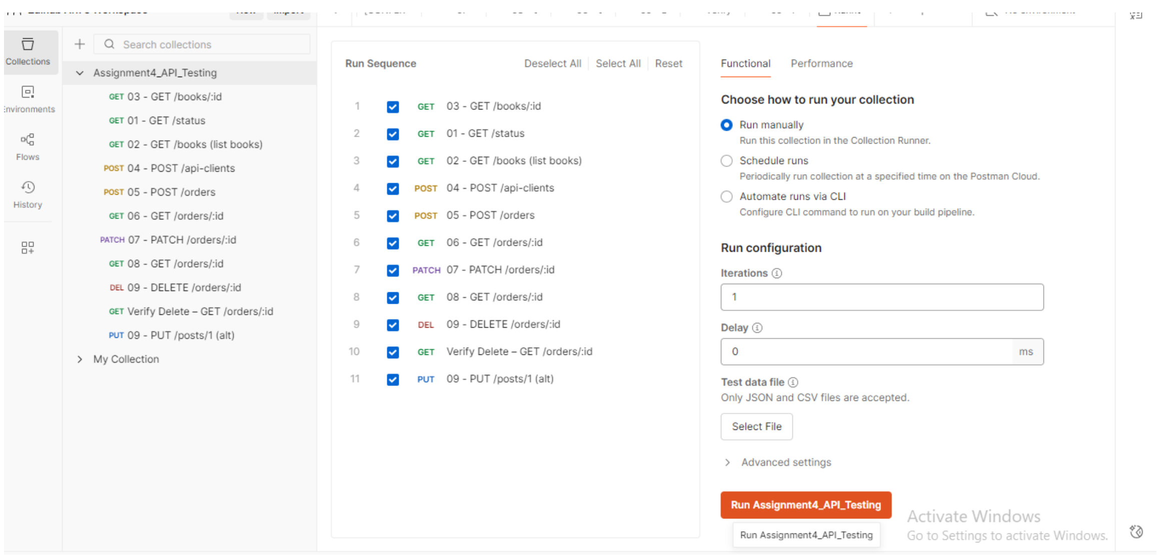
Assignment4_API_Testing — Runs (Functional)

Runs triggered for this collection via Collection Runner and Postman CLI.

Start time	Source	Duration	All tests	Passed	Failed	Skipped	Avg. Resp. Time	Comments

Your collection has not been run yet

Run Collection



Assignment4_API_Testing
- GET 03 - GET /books/:id
- GET 01 - GET /status
- GET 02 - GET /books (list books)
- POST 04 - POST /api-clients
- POST 05 - POST /orders
- GET 06 - GET /orders/:id
- PATCH 07 - PATCH /orders/:id
- GET 08 - GET /orders/:id
- DEL 09 - DELETE /orders/:id
- GET Verify Delete – GET /orders/:id
- PUT 09 - PUT /posts/1 (alt)

My Collection

Run Sequence — Deselect All | Select All | Reset

1	✓	GET	03 - GET /books/:id
2	✓	GET	01 - GET /status
3	✓	GET	02 - GET /books (list books)
4	✓	POST	04 - POST /api-clients
5	✓	POST	05 - POST /orders
6	✓	GET	06 - GET /orders/:id
7	✓	PATCH	07 - PATCH /orders/:id
8	✓	GET	08 - GET /orders/:id
9	✓	DEL	09 - DELETE /orders/:id
10	✓	GET	Verify Delete – GET /orders/:id
11	✓	PUT	09 - PUT /posts/1 (alt)

Functional | Performance

Choose how to run your collection

- [✓ Run manually] — Run this collection in the Collection Runner.
- Schedule runs — Periodically run collection at a specified time on the Postman Cloud.
- Automate runs via CLI — Configure CLI command to run on your build pipeline.

Run configuration

Iterations: 1

Delay: 0 ms

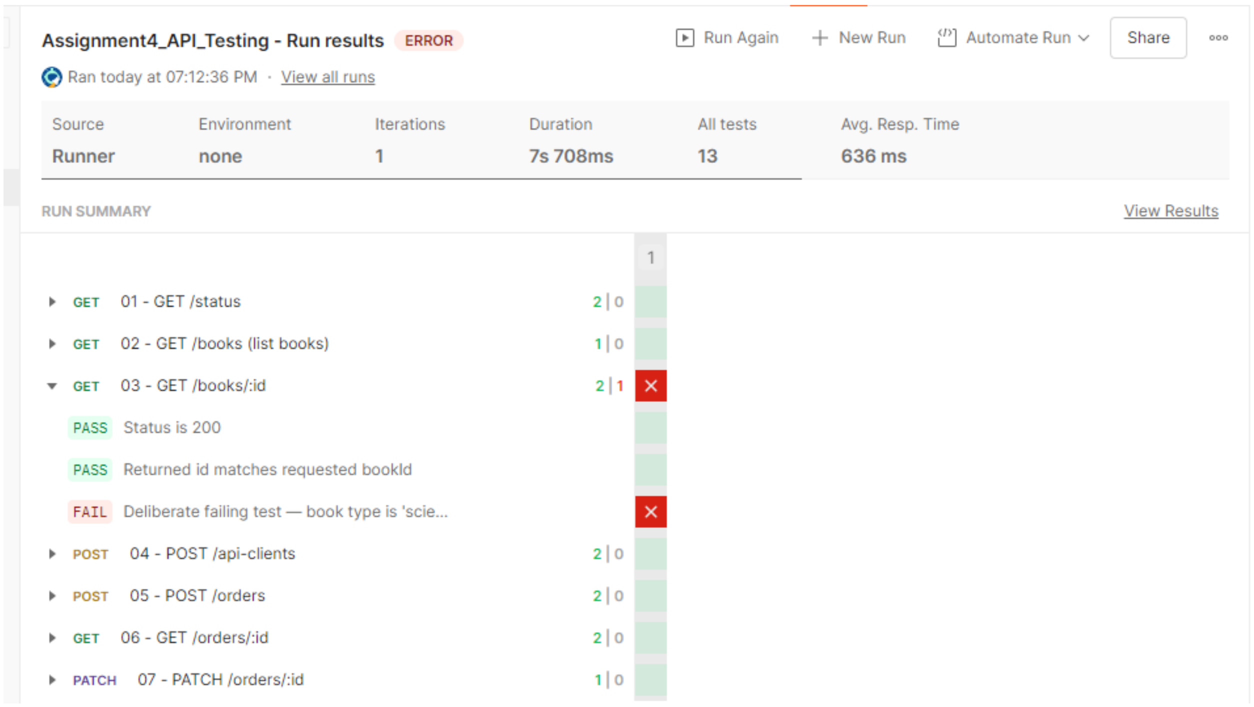
Test data file — Only JSON and CSV files are accepted.

Select File

Advanced settings

Run Assignment4_API_Testing

Activate Windows
Go to Settings to activate Windows.



Assignment4_API_Testing - Run results ERROR

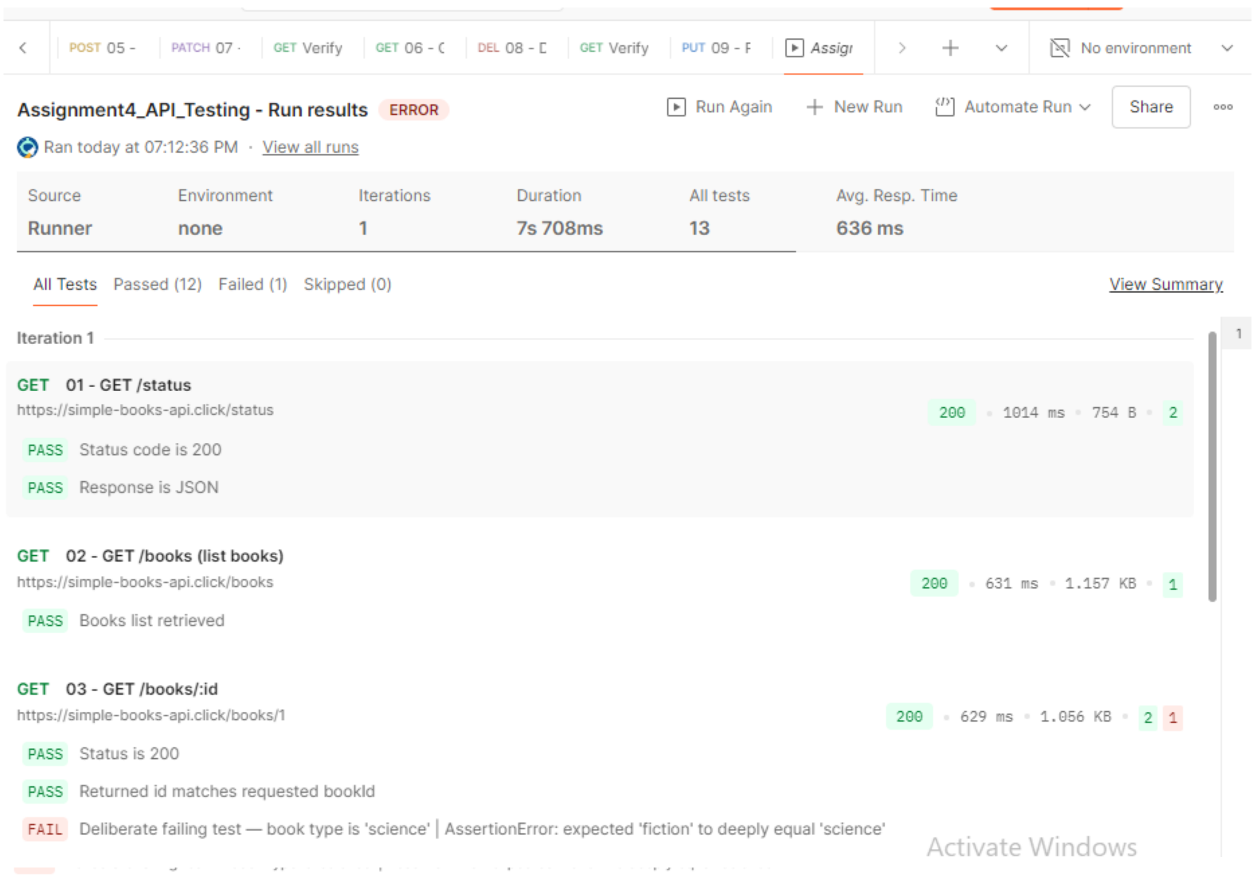
Run Again | New Run | Automate Run | Share

Ran today at 07:12:36 PM · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	7s 708ms	13	636 ms

RUN SUMMARY — View Results

			1
GET	01 - GET /status	2 | 0	🟩
GET	02 - GET /books (list books)	1 | 0	🟩
GET	03 - GET /books/:id	2 | 1	✕
PASS	Status is 200		🟩
PASS	Returned id matches requested bookId		🟩
FAIL	Deliberate failing test — book type is 'scie...		✕
POST	04 - POST /api-clients	2 | 0	🟩
POST	05 - POST /orders	2 | 0	🟩
GET	06 - GET /orders/:id	2 | 0	🟩
PATCH	07 - PATCH /orders/:id	1 | 0	🟩



Assignment4_API_Testing - Run results ERROR

Ran today at 07:12:36 PM · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	7s 708ms	13	636 ms

All Tests Passed (12) Failed (1) Skipped (0) View Summary

Iteration 1

GET 01 - GET /status
https://simple-books-api.click/status 200 · 1014 ms · 754 B · 2

PASS Status code is 200

PASS Response is JSON

GET 02 - GET /books (list books)
https://simple-books-api.click/books 200 · 631 ms · 1.157 KB · 1

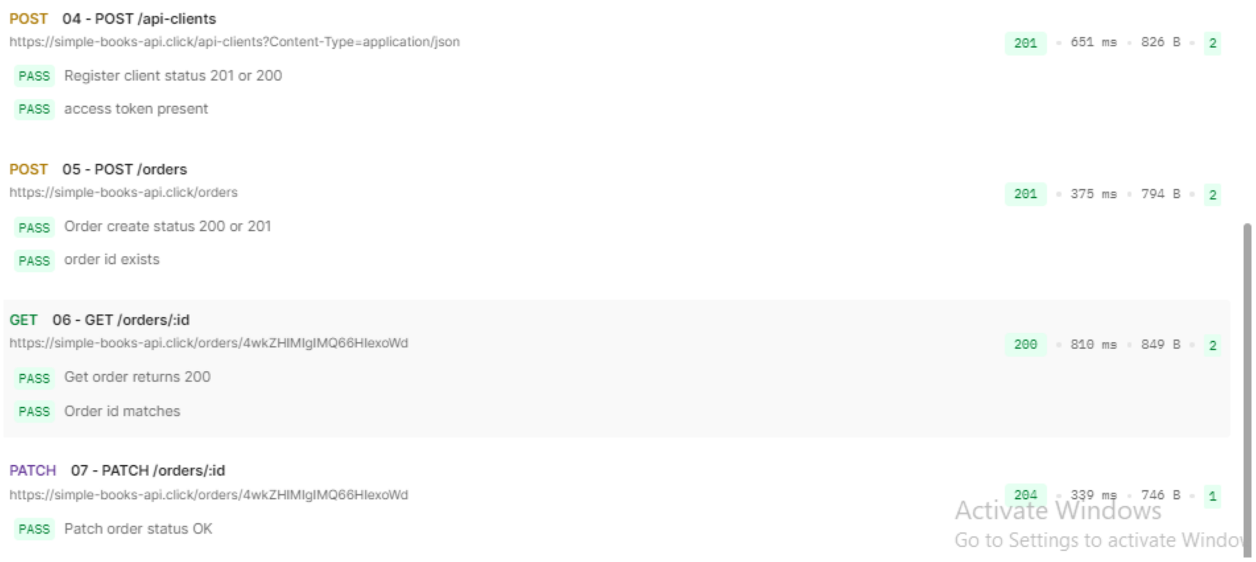
PASS Books list retrieved

GET 03 - GET /books/:id
https://simple-books-api.click/books/1 200 · 629 ms · 1.056 KB · 2 1

PASS Status is 200

PASS Returned id matches requested bookId

FAIL Deliberate failing test — book type is 'science' | AssertionError: expected 'fiction' to deeply equal 'science'



POST 04 - POST /api-clients
https://simple-books-api.click/api-clients?Content-Type=application/json 201 · 651 ms · 826 B · 2

PASS Register client status 201 or 200

PASS access token present

POST 05 - POST /orders
https://simple-books-api.click/orders 201 · 375 ms · 794 B · 2

PASS Order create status 200 or 201

PASS order id exists

GET 06 - GET /orders/:id
https://simple-books-api.click/orders/4wkZHIMIgIMQ66HlexoWd 200 · 810 ms · 849 B · 2

PASS Get order returns 200

PASS Order id matches

PATCH 07 - PATCH /orders/:id
https://simple-books-api.click/orders/4wkZHIMIgIMQ66HlexoWd 204 · 339 ms · 746 B · 1

PASS Patch order status OK