

TEST CASES

Business API Tests

Postman API Testing Guide

Environment Setup

First, create a new environment in Postman with these variables:

- `base_url`: <http://localhost:5000>
- `token`: (leave empty initially, will be filled after login)

Test:

1. User Registration Test

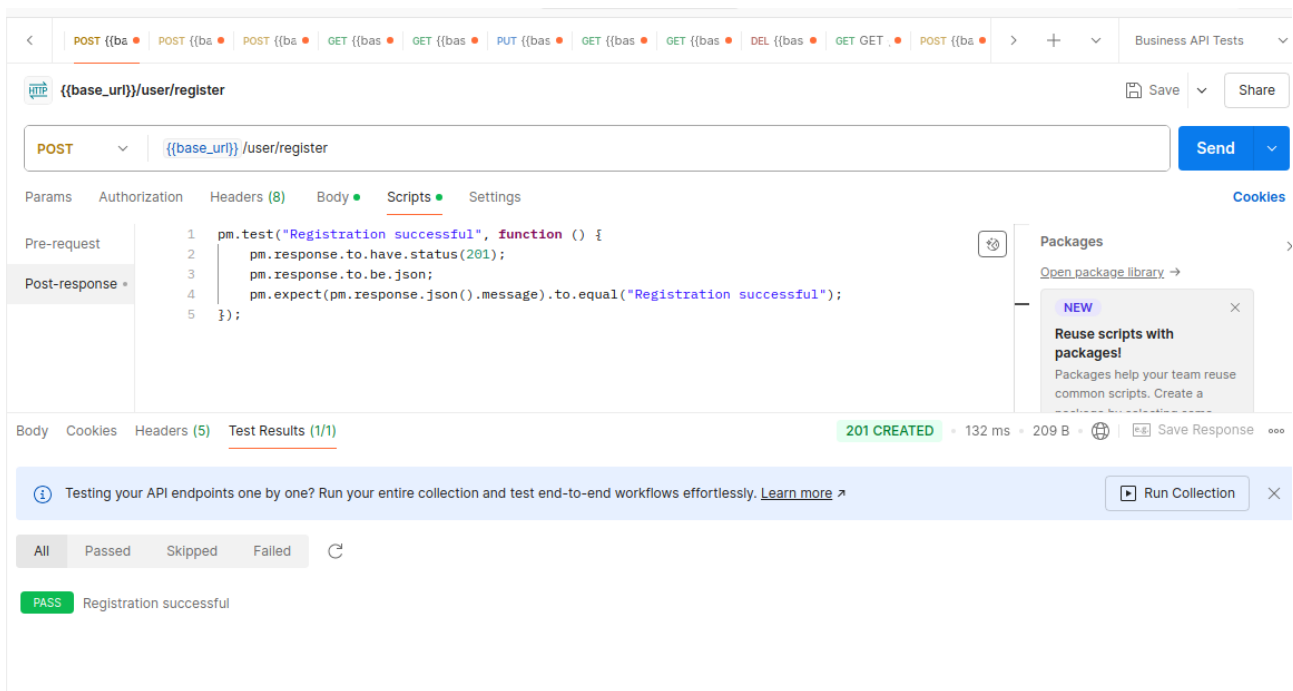
POST `{{base_url}}/user/register`

Content-Type: application/json

```
{  
  "username": "testuser",  
  "password": "test123"  
}
```

// Test Scripts

```
pm.test("Registration successful", function () {  
  pm.response.to.have.status(201);  
  pm.response.to.be.json;  
  pm.expect(pm.response.json().message).to.equal("Registration successful");  
});
```



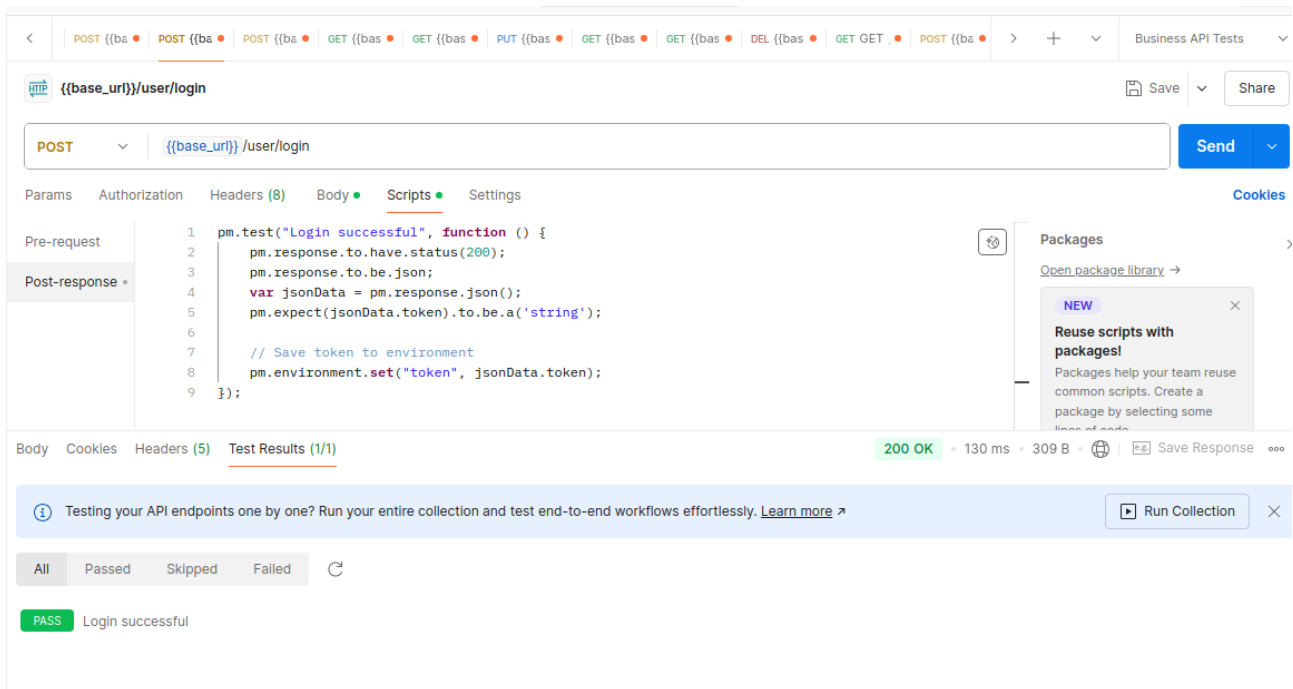
2. User Login Test

POST `{{base_url}}/user/login`
Content-Type: application/json

```
{  
  "username": "testuser",  
  "password": "test123"  
}
```

// Test Scripts

```
pm.test("Login successful", function () {  
  pm.response.to.have.status(200);  
  pm.response.to.be.json;  
  var jsonData = pm.response.json();  
  pm.expect(jsonData.token).to.be.a('string');  
  
  // Save token to environment  
  pm.environment.set("token", jsonData.token);  
});
```



3. Add Business Test

POST `{{base_url}}/businesses`
Authorization: Bearer `{{token}}`
Content-Type: application/json

```
{  
  "name": "Test Business",  
  "address": "123 Test Street",  
  "type": "Retail"  
}
```

```
// Test Scripts  
pm.test("Business added successfully", function () {  
  pm.response.to.have.status(201);  
  pm.response.to.be.json;  
  pm.expect(pm.response.json().message).to.equal("Business added");  
});  
  
// Save business ID for later tests  
if (pm.response.status === 201) {  
  pm.environment.set("business_id", pm.response.json()._id);  
}
```

The screenshot shows a Postman interface for a POST request to `{{base_url}}/businesses`. The request body is empty. The test scripts section contains the following code:

```
1 pm.test("Business added successfully", function () {
2   pm.response.to.have.status(201);
3   pm.response.to.be.json;
4   pm.expect(pm.response.json().message).to.equal("Business added");
5 });
6
7 // Save business ID for later tests
8 if (pm.response.status === 201) {
9   pm.environment.set("business_id", pm.response.json()._id);
10 }
```

The response is a JSON object: `{ "message": "Business added" }`. The status is 201 CREATED, with a response time of 50 ms and a body size of 200 B.

4. Get All Businesses Test

GET `{{base_url}}/businesses`

Authorization: Bearer `{{token}}`

// Test Scripts

```
pm.test("Get businesses successful", function () {
  pm.response.to.have.status(200);
  pm.response.to.be.json;
  pm.expect(pm.response.json()).to.be.an('array');
});
```

The screenshot shows a Postman interface for a GET request to `{{base_url}}/businesses`. The response is a JSON array. The test scripts section contains the following code:

```
1 pm.test("Get businesses successful", function () {
2   pm.response.to.have.status(200);
3   pm.response.to.be.json;
4   pm.expect(pm.response.json()).to.be.an('array');
5 });
```

The response is a JSON array. The status is 200 OK, with a response time of 7 ms and a body size of 316 B.

Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)

Run Collection

All Passed Skipped Failed

PASS Get businesses successful

5. Get Single Business Test

GET {{base_url}}/businesses/{{business_id}}
Authorization: Bearer {{token}}

```
// Test Scripts
pm.test("Get single business successful", function () {
  pm.response.to.have.status(200);
  pm.response.to.be.json;
  pm.expect(pm.response.json()).to.have.property('name');
  pm.expect(pm.response.json()).to.have.property('address');
  pm.expect(pm.response.json()).to.have.property('type');
});
```

The screenshot displays the Postman interface for a GET request. The URL is {{base_url}}/businesses/{{business_id}} and the Authorization header is Bearer {{token}}. The Scripts tab is selected, showing the following test script:

```
1 pm.test("Get single business successful", function () {
2   pm.response.to.have.status(200);
3   pm.response.to.be.json;
4   pm.expect(pm.response.json()).to.have.property('name');
5   pm.expect(pm.response.json()).to.have.property('address');
6   pm.expect(pm.response.json()).to.have.property('type');
7 });
```

The Test Results tab shows a successful response (200 OK) with a status of PASS. The response body is empty. A notification banner at the bottom states: "Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)".

6. Update Business Test

PUT {{base_url}}/businesses/{{business_id}}
Authorization: Bearer {{token}}
Content-Type: application/json

```
{
  "name": "Updated Test Business",
```

```
"address": "456 Test Avenue",  
"type": "Retail"  
}
```

// Test Scripts

```
pm.test("Update business successful", function () {  
  pm.response.to.have.status(200);  
  pm.response.to.be.json;  
  pm.expect(pm.response.json().message).to.equal("Business updated");  
});
```

The screenshot shows the Postman interface for a PUT request. The URL is `{{base_url}}/businesses/{{business_id}}`. The method is PUT. The request body is empty. The 'Scripts' tab is active, showing a test script that checks the status is 200, the response is JSON, and the message is 'Business updated'. The 'Test Results' tab shows a 'PASS' result for 'Update business successful'. A 'Packages' sidebar is visible on the right with a 'NEW' notification about reusing scripts with packages.

7. Search Businesses Test

GET `{{base_url}}/businesses/search?type=Retail&location=Test`
Authorization: Bearer `{{token}}`

// Test Scripts

```
pm.test("Search businesses successful", function () {  
  pm.response.to.have.status(200);  
  pm.response.to.be.json;  
  pm.expect(pm.response.json()).to.be.an('array');  
});
```

The screenshot shows the Postman interface for a REST client. At the top, a collection of requests is visible, with the current request selected: `GET {{base_url}}/businesses/search?type=Retail&location=Test`. The interface includes tabs for Params, Authorization, Headers (7), Body, Scripts, and Settings. The Scripts tab is active, showing a test script:

```
1 // Test Scripts
2 pm.test("Search businesses successful", function () {
3     pm.response.to.have.status(200);
4     pm.response.to.be.json;
5     pm.expect(pm.response.json()).to.be.an('array');
6 });
```

The response status is **200 OK** with a response time of 6 ms and a body size of 324 B. A notification banner at the bottom states: "Testing your API endpoints one by one? Run your entire collection and test end-to-end workflows effortlessly. [Learn more](#)" with a "Run Collection" button.

8. Aggregate Businesses Test

GET `{{base_url}}/businesses/aggregate`
Authorization: Bearer `{{token}}`

```
// Test Scripts
pm.test("Aggregate businesses successful", function () {
    pm.response.to.have.status(200);
    pm.response.to.be.json;
    pm.expect(pm.response.json()).to.be.an('array');
    if (pm.response.json().length > 0) {
        pm.expect(pm.response.json()[0]).to.have.property('count');
        pm.expect(pm.response.json()[0]).to.have.property('businesses');
    }
});
```

The screenshot shows a Postman interface for a GET request to `{{base_url}}/businesses/aggregate`. The test script in the 'Scripts' tab is as follows:

```
1 pm.test("Aggregate businesses successful", function () {
2   pm.response.to.have.status(200);
3   pm.response.to.be.json;
4   pm.expect(pm.response.json()).to.be.an('array');
5   if (pm.response.json().length > 0) {
6     pm.expect(pm.response.json()[0]).to.have.anyty('count');
7     pm.expect(pm.response.json()[0]).to.have.property('businesses');
8   }
9 });
```

The test results show a **PASS** for "Aggregate businesses successful". The response status is **200 OK** with a response time of 10 ms and a size of 272 B.

9. Delete Business Test

DELETE `{{base_url}}/businesses/{{business_id}}`

Authorization: Bearer `{{token}}`

// Test Scripts

```
pm.test("Delete business successful", function () {
  pm.response.to.have.status(200);
  pm.response.to.be.json;
  pm.expect(pm.response.json().message).to.equal("Business deleted");
});
```

The screenshot shows a Postman interface for a DELETE request to `{{base_url}}/businesses/{{business_id}}`. The test script in the 'Scripts' tab is as follows:

```
1 pm.test("Delete business successful", function () {
2   pm.response.to.have.status(200);
3   pm.response.to.be.json;
4   pm.expect(pm.response.json().message).to.equal("Business deleted");
5 });
```

The test results show a **PASS** for "Delete business successful". The response status is **200 OK** with a response time of 5 ms and a size of 197 B.

