-Blog API report-

Testing API Using Postman

Item	*Link*
GitHub Repository	Blog-API-Testing
Postman documentation	Blog API
JSONplaceholder	API Sample

Contents

INTRODUCTION 1

COLLECTION DESCRIPTION 1

API OVERVIEW 3

TESTS 4

NOTES 8

Popis priloga 9

INTRODUCTION

This project demonstrates testing of the Blog API using Postman and JSONplaceholder API sample.

The goal is to show how to test API endpoints, validate responses, and generate reports and documentation.

COLLECTION DESCRIPTION

Collection name: Blog API

Environment name: Blog API Testing

Collection structure:

Blog API /

Positive Endpoints /

- User API Testing
- Posts API Testing
- Comments API Testing

Negative Endpoints /

- User API Testing
- Posts API Testing
- Comments API Testing

All variables:

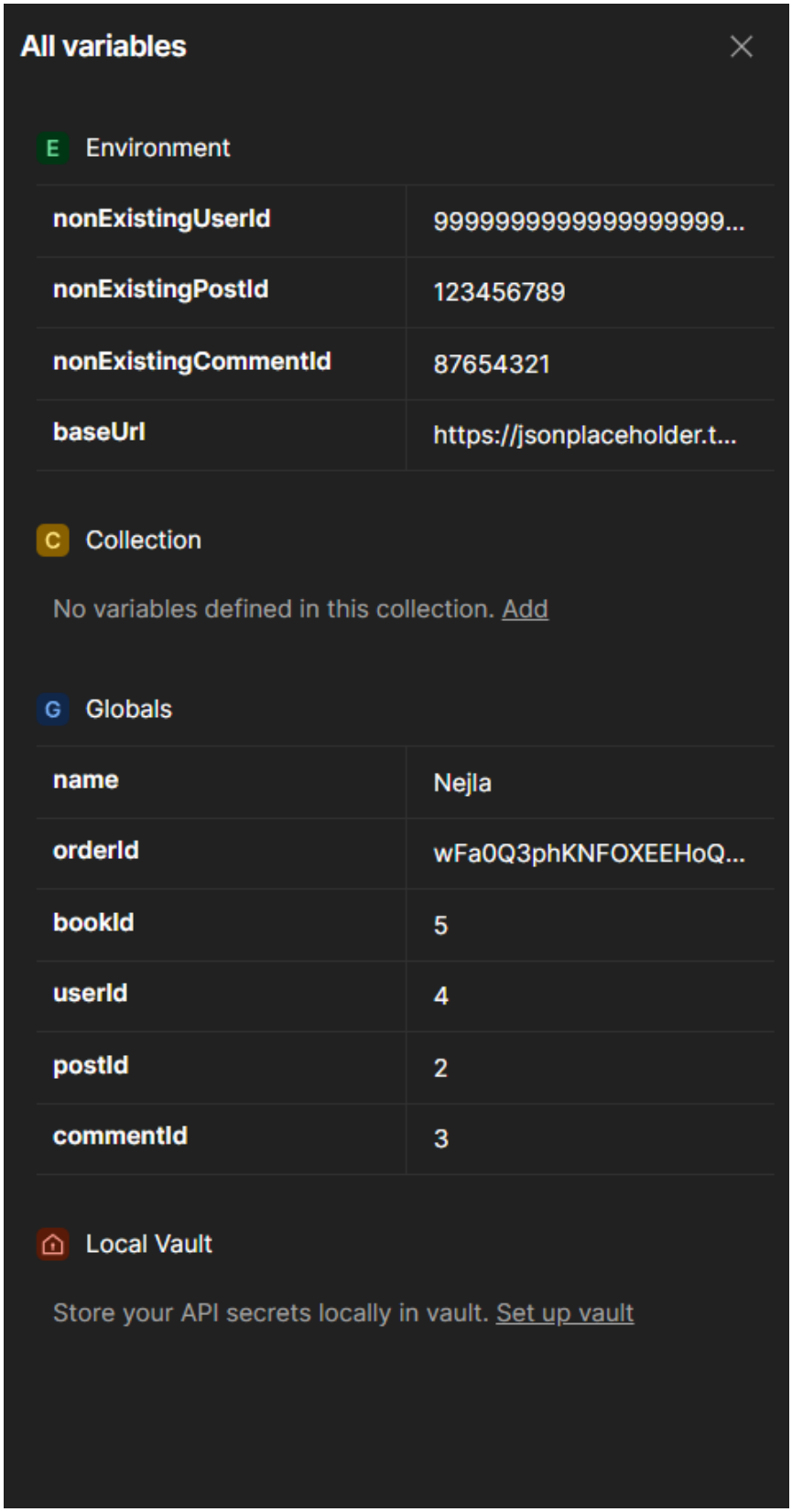

Figure 1 All variables

API OVERVIEW

{{baseUrl}} = https://jsonplaceholder.typicode.com

Endpoint	*Method*	*Description*
/users	GET	Retrieves all blog users.
/users/:userId	GET	Retrieves a single blog user.
/users	POST	Creates a new user.
/users/:userId	PUT	Updates an existing user.
/users/:userId	PATCH	Updates one or more fields of an existing user.
/users/:userId	DELETE	Deletes a specific user.
/posts	GET	Retrieves all blog posts.
/posts/:postId	GET	Retrieves a single post.
/posts	POST	Creates a new post.
/posts/:postId	PUT	Updates an existing post.
/posts/:postId	PATCH	Updates one or more fields of an existing post.
/posts/:postId	DELETE	Deletes a specific post.
/comments	GET	Retrieves all blog comments.
/comments/:commentId	GET	Retrieves a single comment.
/comments	POST	Creates a new comment.
/comments/:commetId	PUT	Updates an existing comment.
/comments/:commetId	PATCH	Updates one or more fields of an existing comment.
/comments/:commetId	DELETE	Deletes a specific comment.

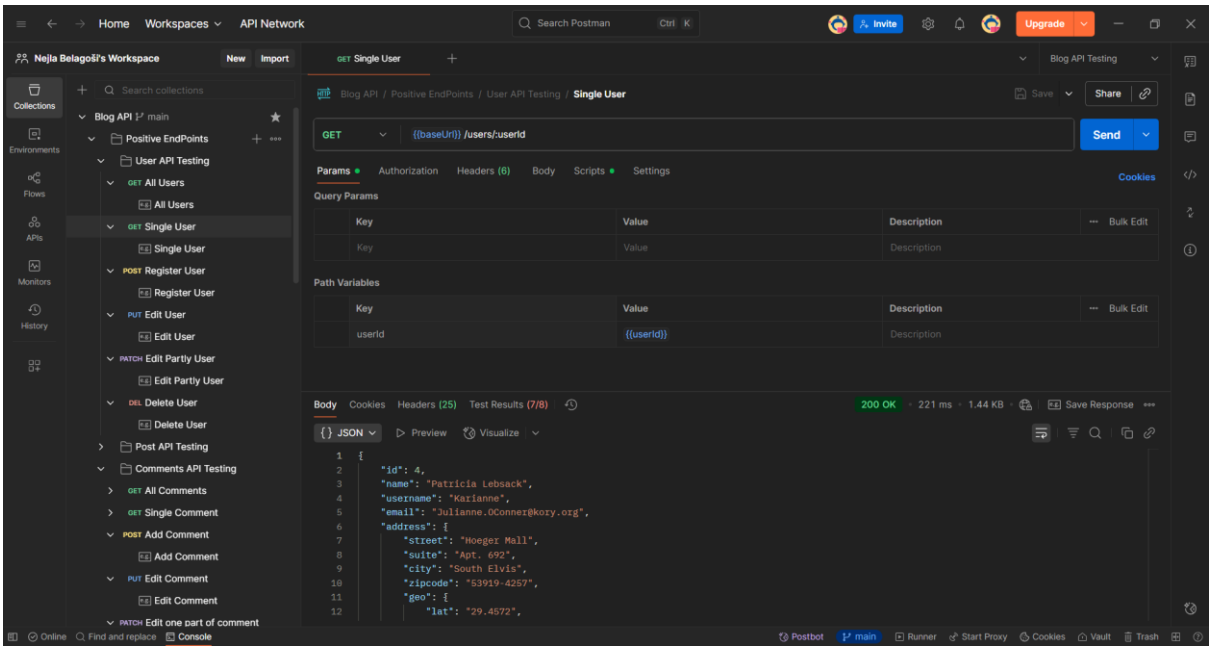

Figure 2 Example of a request and response

TESTS

Script example for Positive Endpoint -> DEL Delete User

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

//checks response time
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});

//checks if API URL Endpoint is valid
pm.test("URL Endpoint is valid", function () {
    pm.expect(pm.request.url.toString()).to.eql(`https://jsonplaceholder.typicode.com/u
sers/${pm.globals.get("userId")}`);
});

//checks header content type
pm.test("Content-Type is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});

//checks if user is really deleted
pm.test("User is deleted", function () {
    let responseData = pm.response.json() ? pm.response.json() : {};
    pm.expect(Object.keys(responseData).length).to.be.at.most(1);
});
```

Script example for Negative Endpoint for DEL Delete User:

```
// Post-response test script for DELETE /users/:userId

// Check if the API is JSONPlaceholder by inspecting the baseURL environment variable
const isJSONPlaceholder = pm.environment.get("baseUrl") &&
pm.environment.get("baseUrl").includes("jsonplaceholder");

// Test for deleting a non-existing user
if (isJSONPlaceholder) {
    // JSONPlaceholder returns 200 OK even if the user does not exist
    pm.test("Delete non-existing user in JSONPlaceholder returns 200 OK", function () {
        pm.expect(pm.response.code).to.eql(200);
        // Optionally, check response body structure if needed
    });
} else {
```

```
    // For other APIs, deleting a non-existing user should return 404 Not Found
    pm.test("Delete non-existing user returns 404 Not Found", function () {
        pm.expect(pm.response.code).to.eql(404);
        // Check if response body contains an error message or structure
        try {
            var jsonData = pm.response.json();
            pm.expect(jsonData).to.be.an('object');
            pm.expect(jsonData.error || jsonData.message).to.exist;
        } catch (e) {
            // Response is not JSON or no error message present
            pm.test("Response body is valid JSON with error message", function () {
                pm.expect.fail("Response body is not valid JSON or missing error
message");
            });
        }
    });
}

// Test for idempotency: deleting the same user twice should yield consistent error
responses
pm.test("Idempotency test: deleting the same user twice returns consistent error
responses", function () {
    // We assume the userId variable is set in the environment or collection
    const userId = pm.variables.get("userId") || "";
    if (!userId) {
        pm.test.skip("No userId provided for idempotency test");
        return;
    }

    // Send the DELETE request again for the same userId
    pm.sendRequest({
        url: pm.request.url.toString(),
        method: 'DELETE',
        header: pm.request.headers.toObject(),
        body: pm.request.body ? pm.request.body.toString() : undefined
    }, function (err, res) {
        pm.test("Second delete request returns expected status code", function () {
            if (isJSONPlaceholder) {
                pm.expect(res.code).to.eql(200);
            } else {
                pm.expect(res.code).to.eql(404);
            }
        });

        if (!isJSONPlaceholder) {
            pm.test("Second delete response contains error message", function () {
                try {
                    var jsonData = res.json();
```

```
                    pm.expect(jsonData.error || jsonData.message).to.exist;
                } catch (e) {
                    pm.expect.fail("Second delete response body is not valid JSON or
missing error message");
                }
            });
        }
    });
});

// Suggested tests for other invalid user IDs
// These tests can be added as separate requests or as part of a collection run
// Examples include malformed userId (e.g., special characters), empty userId, or null
// For demonstration, here is a test for empty userId in this response script context
pm.test("Delete user with empty userId returns error", function () {
    // This test assumes the request was sent with an empty userId path variable
    pm.expect(pm.response.code).to.be.oneOf([400, 404]); // Depending on API design
});

// End of post-response test script
```

Run informations: 13/09/2025, 05:05:43 PM

Source	*Environment*	*Iterations*	*Duration [ms]*	*All tests*	*Avg. Resp. Time [ms]*
Runner	Blog API Testing	1	16s685	1464	189

Passed	*Failed*	*Skipped*
1425	39	0

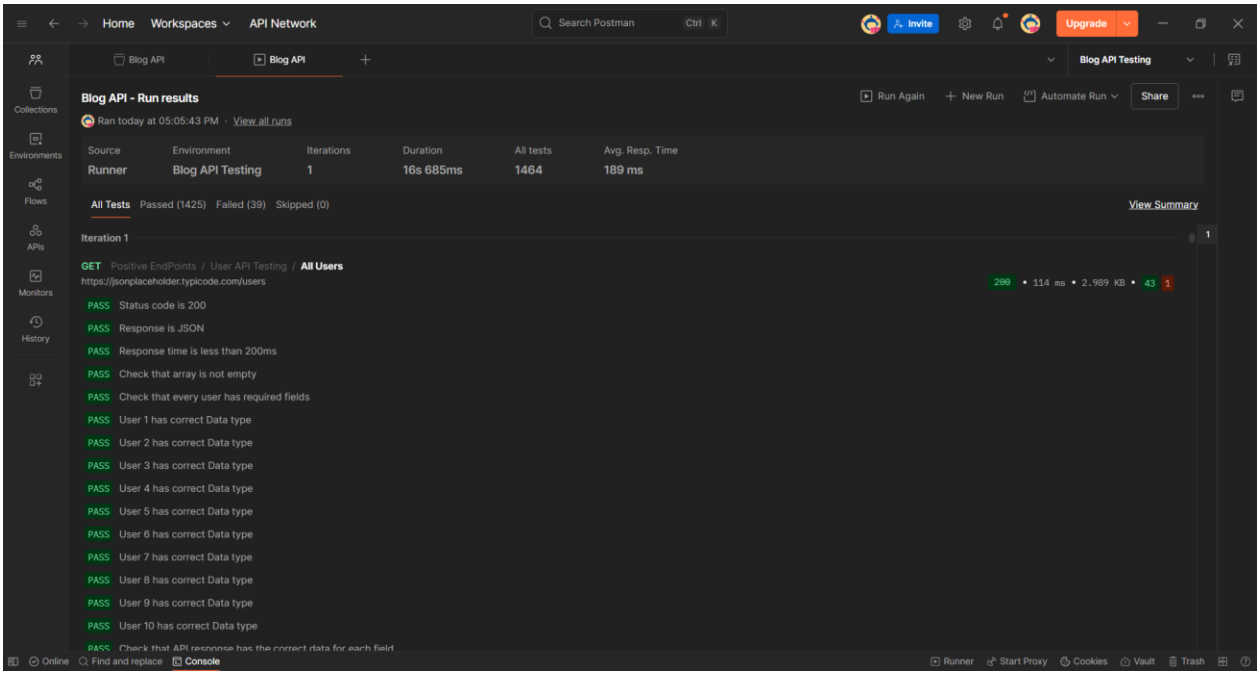

Figure 3 Functional Test Report

Load Profile	*Number of Virtual Users*	*Test duration [min]*
Fixed	50	2

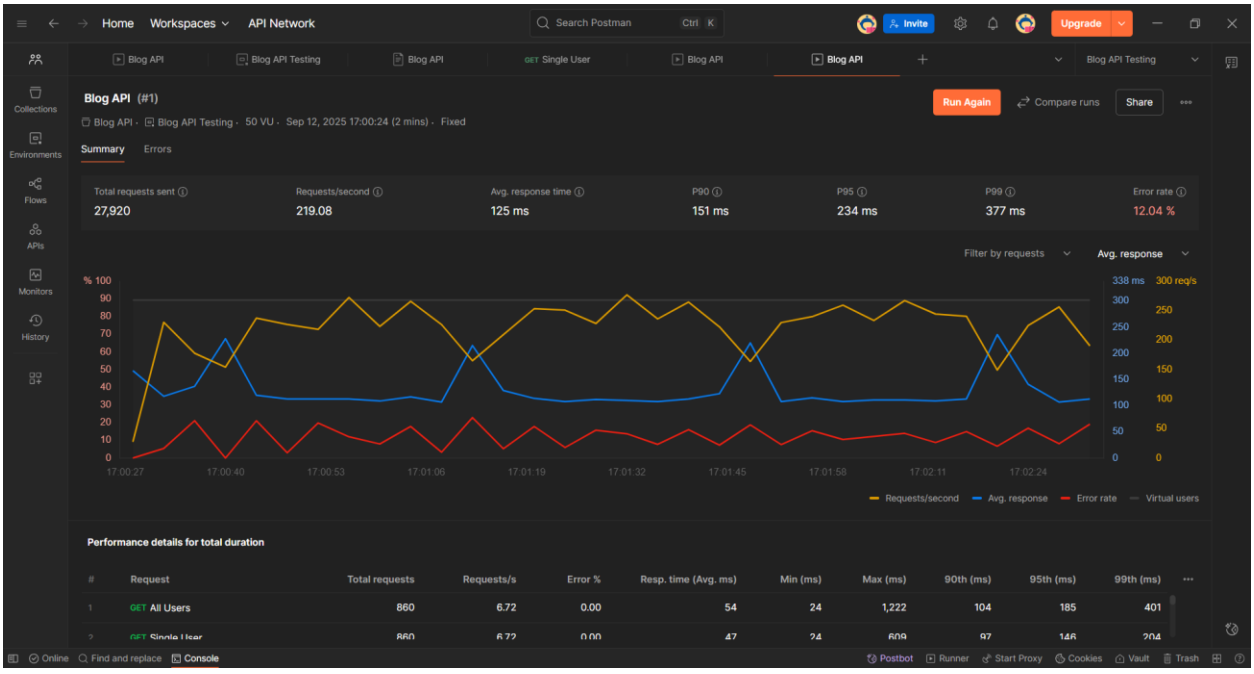

Figure 4 Performance Test report

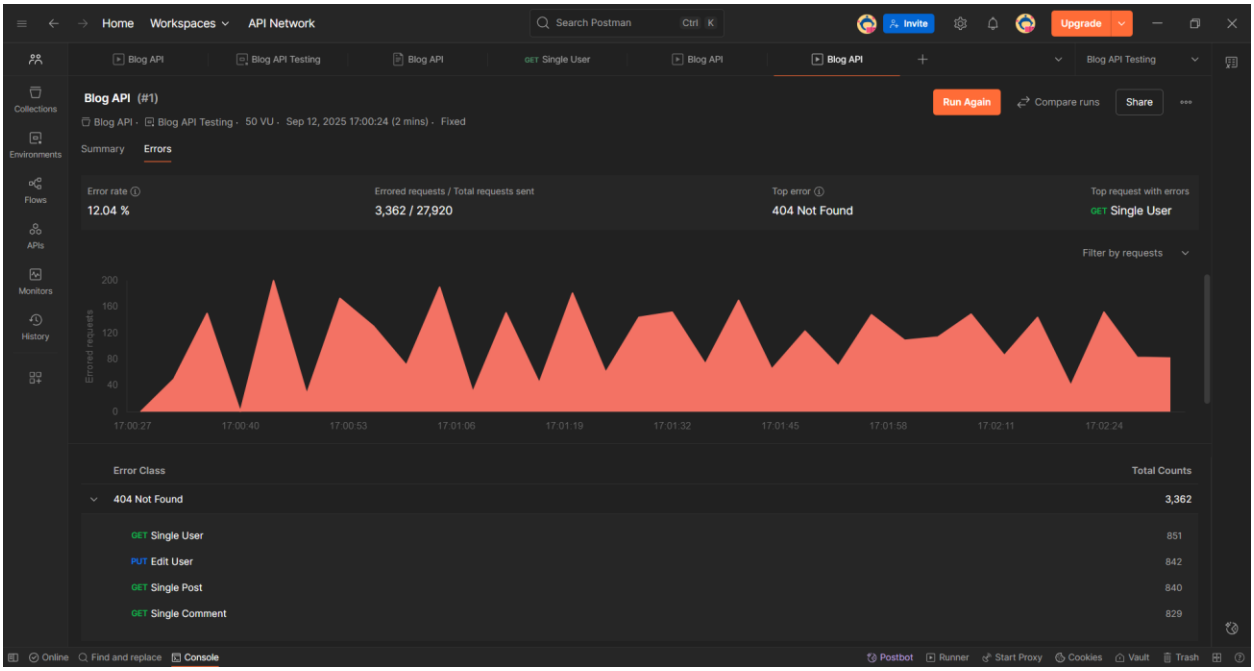

Figure 5 Performance Testing Errors summary

NOTES

JSONplaceholder is a mock API for testing and learning purpose.

Some endpoints don't behave exactly like a real API (e.g., POST, PUT, DELETE may return success but don't actually modify data).

Certain endpoints may sometimes fail or return unexpected results.

This is normal because the service is simulated and not persistent.

Popis priloga

Figure 1 All variables 2
Figure 2 Example of a request and response 3
Figure 3 Functional Test Report 7
Figure 4 Performance Test report 7
Figure 5 Performance Testing Errors summary 8