

Tour and Travel BiH API report

-API Testing using Postman-

<i>Item</i>	<i>Link</i>
<i>GitHub Repository</i>	Tour and Travel BiH API Testing
<i>Postman Documentation</i>	Tour and Travel BiH
<i>Tour and Travel BiH API Repository</i>	API Sample

Contents

INTRODUCTION	1
Collection description	1
API Overview	2
Authentication	4
Tests.....	5
Functional Testing	7
Performance Testing	9
Notes	10
Popis priloga.....	11

INTRODUCTION

This report summarizes the testing of the main backend APIs of the web application Tour and Travel BiH, which handle user account management, favorite items, reservations, and leaving reviews. The tested APIs allow users to register, authenticate, manage their profile, and manage other functionalities that this application provides.

Collection description

Collection name: Tour_and_Travel_BiH

Environment name: Tour_and_Travel_BiH

Collection structure:

Tour_and_Travel_BiH/

Admin/ (in the future)

Enduser/

- Create User & Login

- Favorite Items

- Reservations

- Reviews

- Account

Negative Endpoints – Enduser/ (in the future)

All variables:

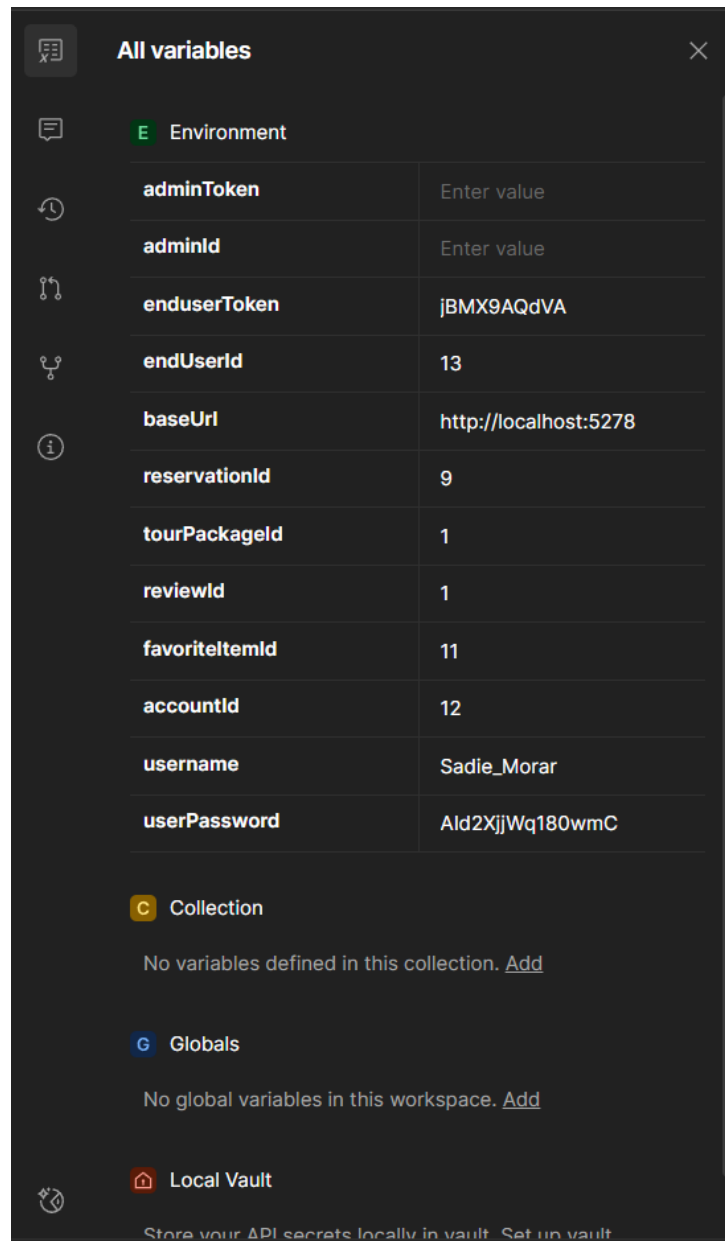


Figure 1 All variables

API Overview

{{baseUrl}} = <http://localhost:5278>

<i>Endpoint</i>	<i>Method</i>	<i>Description</i>
<i>/api/Users/RegisterUser</i>	POST	Creates new user account.
<i>/auth/Login</i>	POST	Authenticates a user and returns a token along with basic user information.
<i>//api/Favorite/GetFavorite</i>	GET	Retrieves all favorite items of a specific user.
<i>/api/Favorite/PostFavorite</i>	POST	Adds an item to the user's favorites section.
<i>/api/Favorite/DeleteFavorite/:id</i>	DELETE	Deletes a specific item from the user's favorites section.
<i>/api/Reservation/GetReservationsByUserId/{{endUserId}}</i>	GET	Retrieves all reservations for a specific user.
<i>/api/Reservation/PostReservation</i>	POST	Creates a new reservation for a user.
<i>/api/Reservation/DeleteReservation/{{reservationId}}</i>	DELETE	Deletes a specific reservation by ID.
<i>/api/Reservation/UpdateReservation/{{reservationId}}</i>	PUT	Updates an existing reservation by ID.
<i>/api/Review/GetReview</i>	GET	Retrieves all reviews in the system.
<i>/api/Review/GetReviewByPackageId/{{tourPackageId}}</i>	GET	Retrieves reviews for a specific tour package.
<i>/api/Review/PostReview</i>	POST	Creates a new review for a tour package.
<i>/api/Review/UpdateReview/{{reviewId}}</i>	PUT	Updates an existing review by ID.
<i>/api/Review/DeleteReview/{{reviewId}}</i>	DELETE	Deletes a review by ID.
<i>/api/Account/DeleteAccount/{{accountId}}</i>	DELETE	Deletes a user account by account ID.
<i>/api/Account/UpdateAccount/{{accountId}}</i>	PUT	Updates account information for a

/auth/logout

DELETE

specific user account.
Logs out the currently authenticated user and invalidates their token.

Authentication

After creating a new user account, the user can authenticate using the POST `/auth/Login` endpoint, which returns a JWT token along with basic user information. This token can be used to access protected resources within the application. Each time the user logs in, a new token is generated, ensuring that a fresh token is issued for every login session.

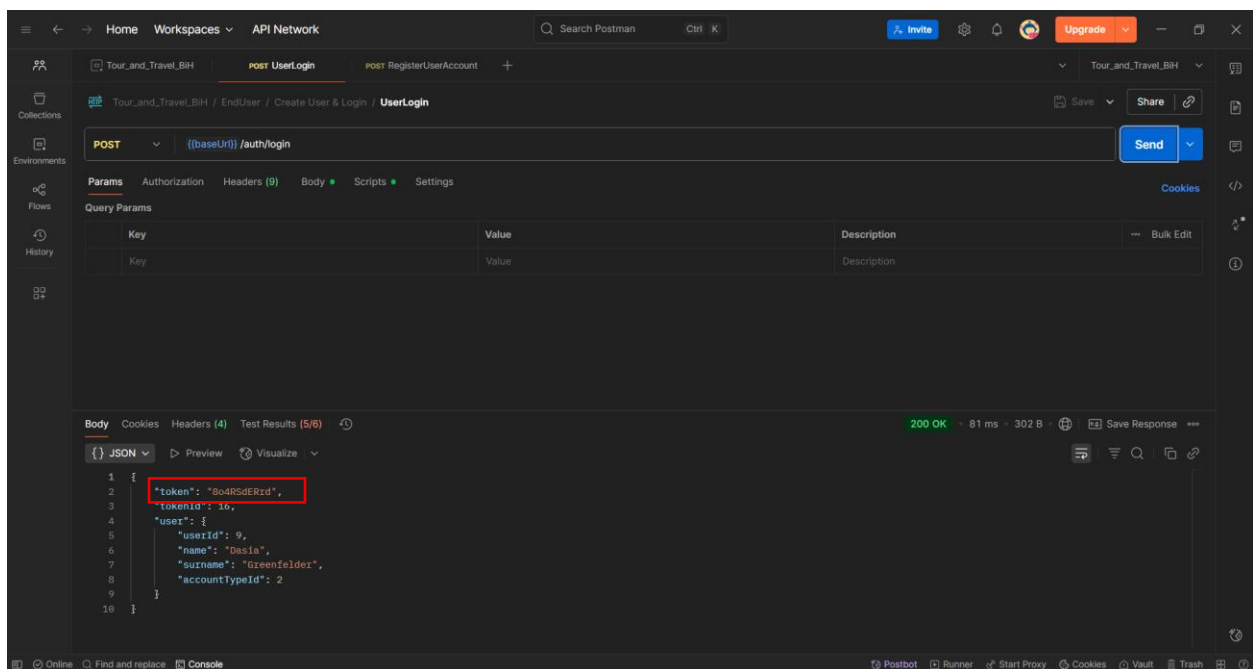


Figure 2 Login Process

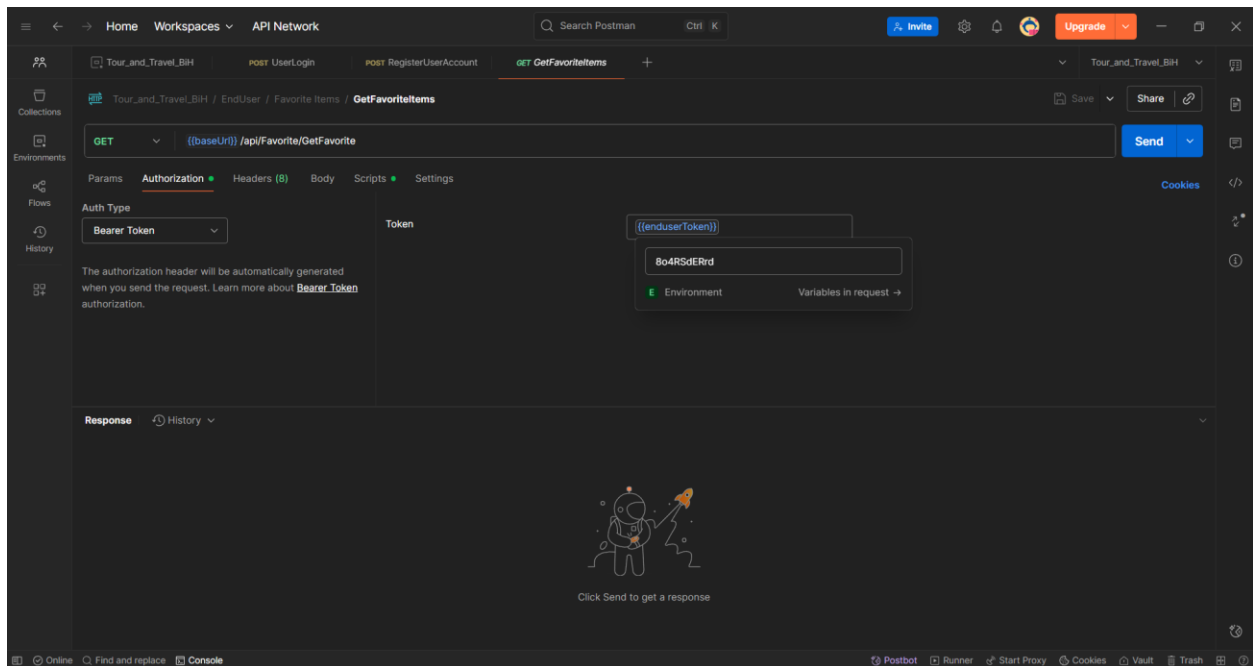


Figure 3 Authentication using a token

Then, include a new key and its corresponding value in the request headers:

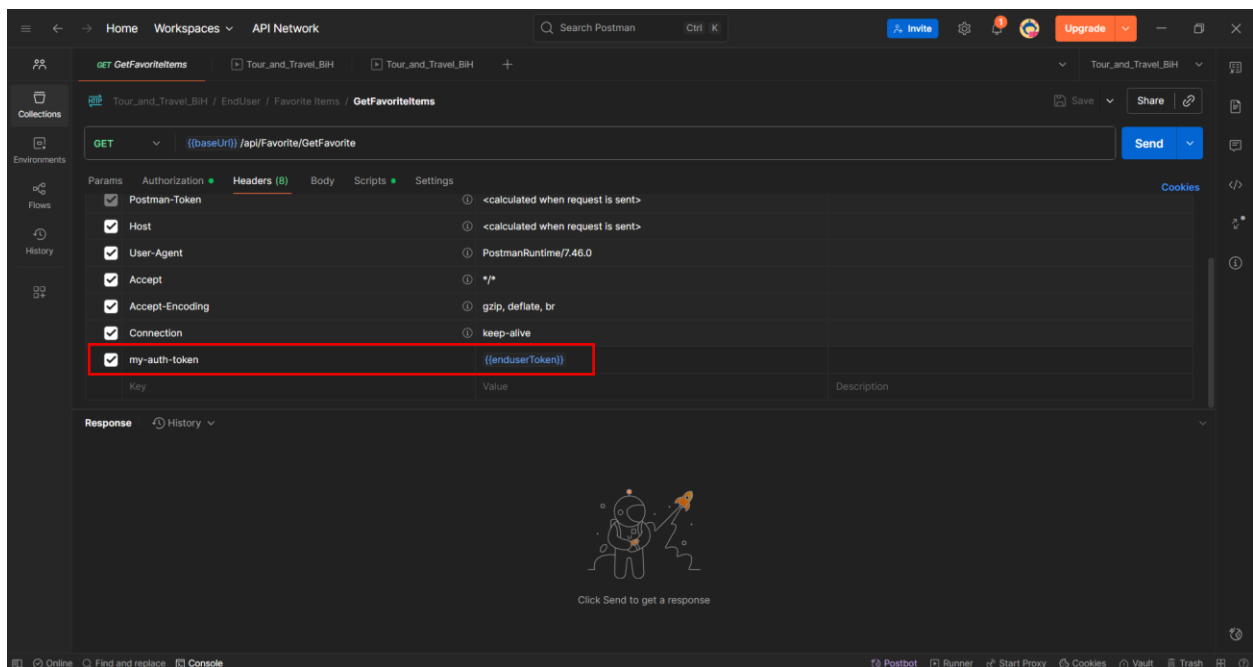


Figure 4 Adding a new key-value pair in the request headers in Postman

Tests

Body:

```
{
  "username": "{{username}}",
  "userPassword": "{{userPassword}}"
}
```

Script example for Enduser -> User Login

```
//authorization token automatically added to the environment variable
let jsonData = pm.response.json();

// checks if response has token value
if (jsonData.token) {
  pm.environment.set("enduserToken", jsonData.token);
  console.log("User token set: " + jsonData.token);
} else {
  console.log("Token not found in response!");
}

//store userId after login
if(jsonData.user && jsonData.user.userId){
  pm.environment.set("endUserId", jsonData.user.userId);
}

//store accountId
if(jsonData.user && jsonData.user.accountId){
  pm.environment.set("accountId", jsonData.user.accountId);
}

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

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

// Token is not empty
pm.test("Token is not empty", function(){
  pm.expect(pm.response.json().token).not.to.be.empty;
})

// response has token
pm.test("Response has token", function(){
  pm.expect(pm.response.json()).to.have.property("token");
});
```



```

    pm.expect(pm.response.json().token).to.be.a("string").and.not.empty;
});

//Response has objects
pm.test("Response has user object", function() {
    pm.expect(pm.response.json().user).to.have.property("userId");
    pm.expect(pm.response.json().user).to.have.property("name");
    pm.expect(pm.response.json().user).to.have.property("surname");
    pm.expect(pm.response.json().user).to.have.property("accountTypeId");
});

// UserId should be a positive number
pm.test("UserId is a positive number", function () {
    pm.expect(jsonData.user.userId).to.be.a("number");
    pm.expect(jsonData.user.userId).to.be.above(0);
});

// Error handling if token is missing
if (!jsonData.token) {
    console.error("Token not found in response!");
    pm.test("Token missing error", function () {
        throw new Error("Token not found in response");
    });
}

```

Functional Testing

Summary:

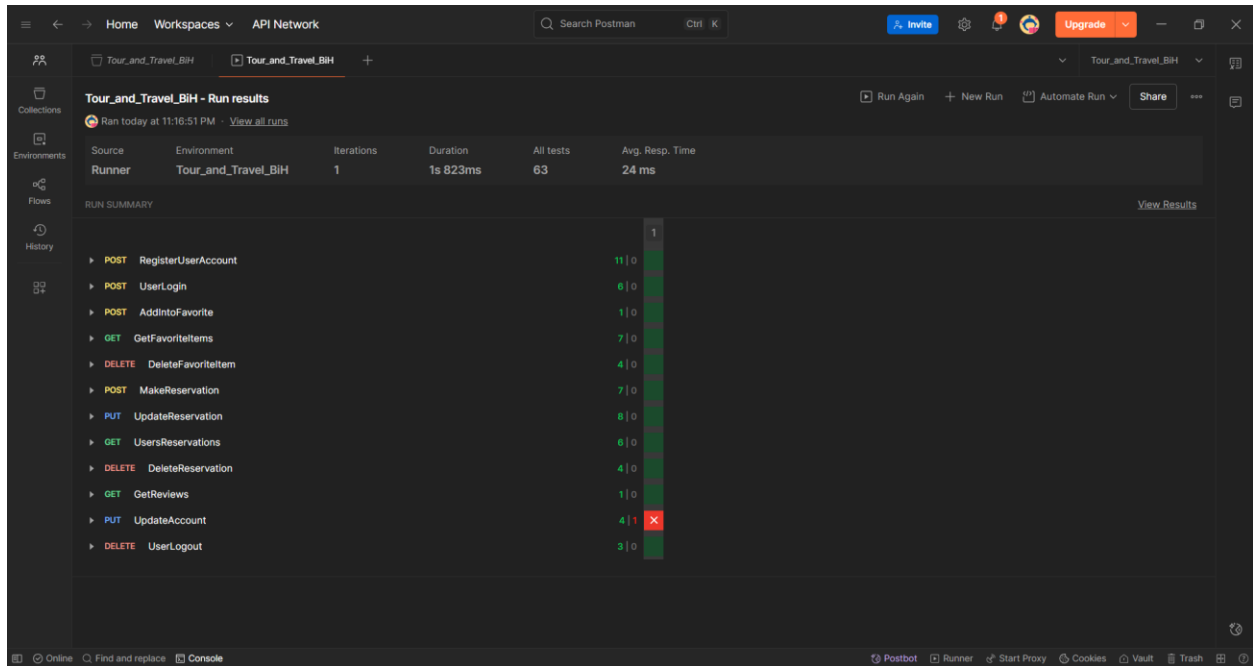


Figure 5 Functional Testing Summary

Run informations: 14/09/2025

Source	Environment	Duration	All tests	Avg.Resp. Time [ms]
Runner	Tour_and_Travel_BiH	1s823ms	63	24

Passed	Failed	Skipped
62	1	0

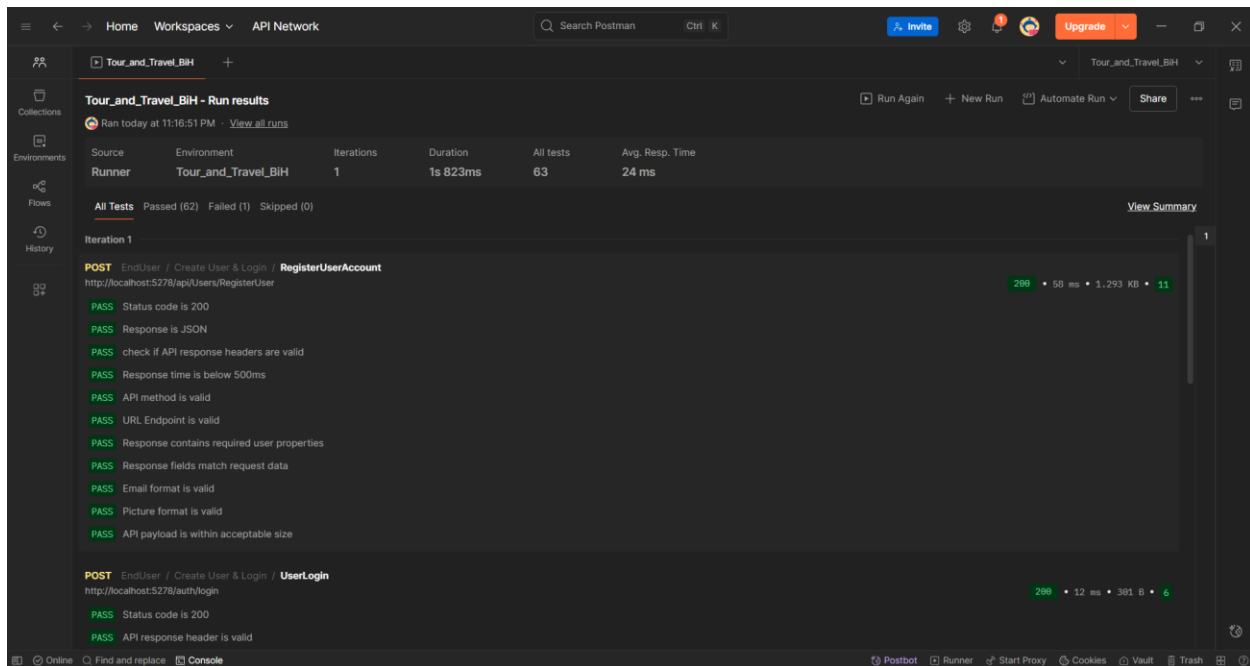


Figure 6 Functional Testing Report

Performance Testing

Load Profile	Number of Virtual Users	Test duration [min]
Fixed	60	1

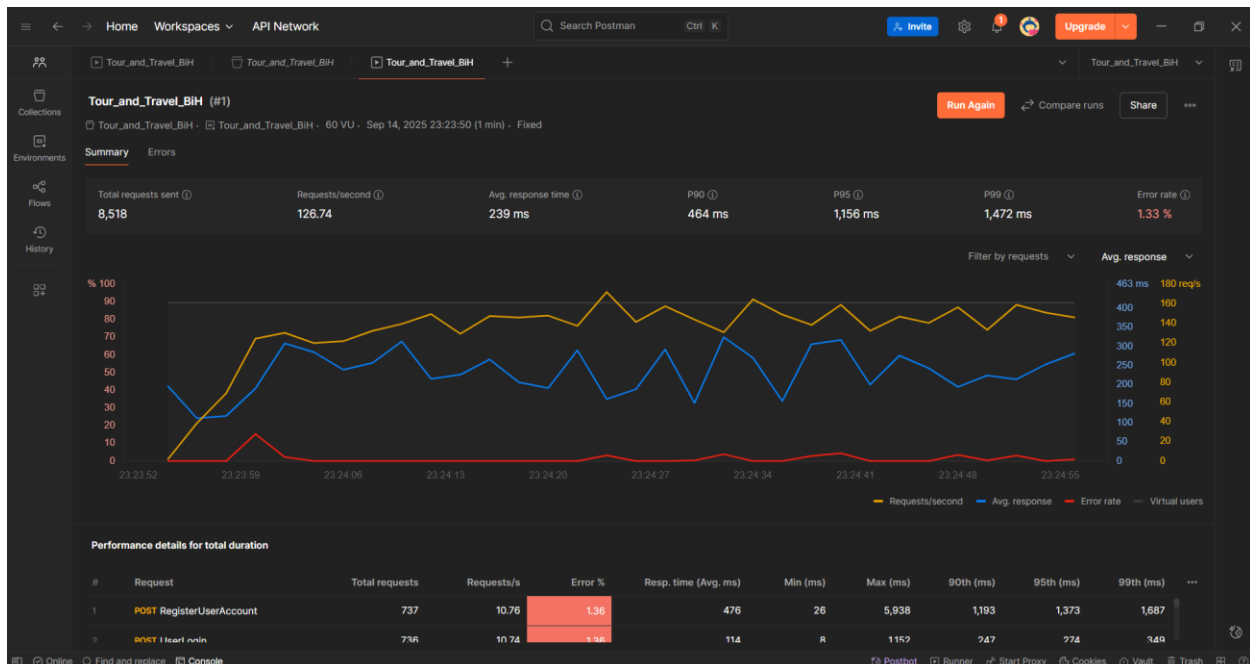


Figure 7 Performance Testing

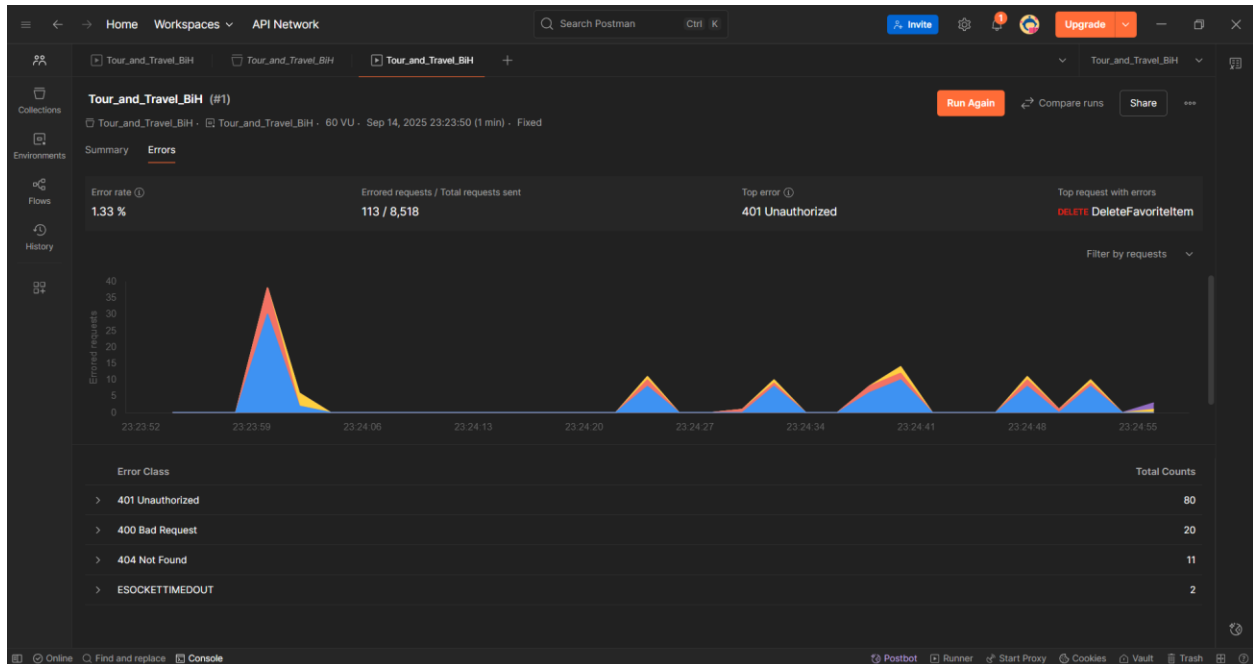


Figure 8 Performance Testing Errors

Notes

Local API

- All tests were executed against a local API (localhost) and a development database.
- URLs (baseUrl) and IDs (userId, accountId) are specific to the local environment and may differ.

Dynamic Data

- Environment variables are used to store dynamic values such as username, userPassword, token, favoriteItemId, reservationId, and reviewId.
- This allows automated use of newly created resources in subsequent requests without manual input.

Authentication and Tokens

- The login endpoint returns a token used to access protected resources.
- Tokens are dynamic and generated on each login.

Payments method, and adding Reviews will be added in the future.

Popis priloga

Figure 1 All variables	2
Figure 2 Login Process.....	4
Figure 3 Authentication using a token	5
Figure 4 Adding a new key-value pair in the request headers in Postman.....	5
Figure 5 Functional Testing Summary	8
Figure 6 Functional Testing Report	9
Figure 7 Performance Testing	9
Figure 8 Performance Testing Errors.....	10