Tour and Travel BiH API report

-API Testing using Postman-

Item	*Link*
GitHub Repository	Tour and Travel BiH API Testing
Postman Documentation	Tour and Travel BiH
Tour and Travel BiH API Repository	API Sample

Contents

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/

- Positeve Tests/
  - Create User & Login/
  - Favorite Items/
  - Reservations/
  - Reviews/
  - Account/
- Negative Tests/
  - Create User & Login/
  - Favorite Items/
  - Reservations/
  - Reviews/
  - Account/

All variables:

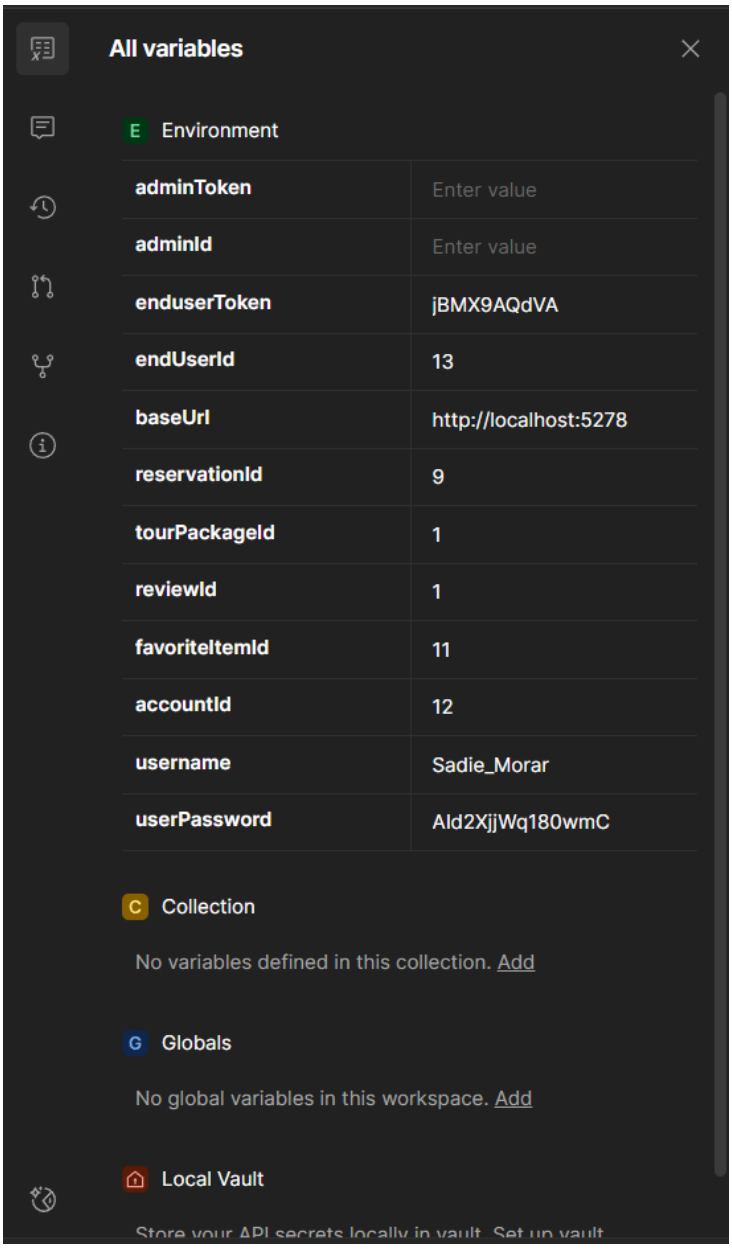

Figure 1 All variables

API Overview

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

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

/api/Account/UpdateAccount/{{accountId}}	PUT	Updates account information for a specific user account.
/auth/logout	DELETE	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 generic alphanumeric token, generated using a cryptographically secure random number generator (RNGCryptoServiceProvider), 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.

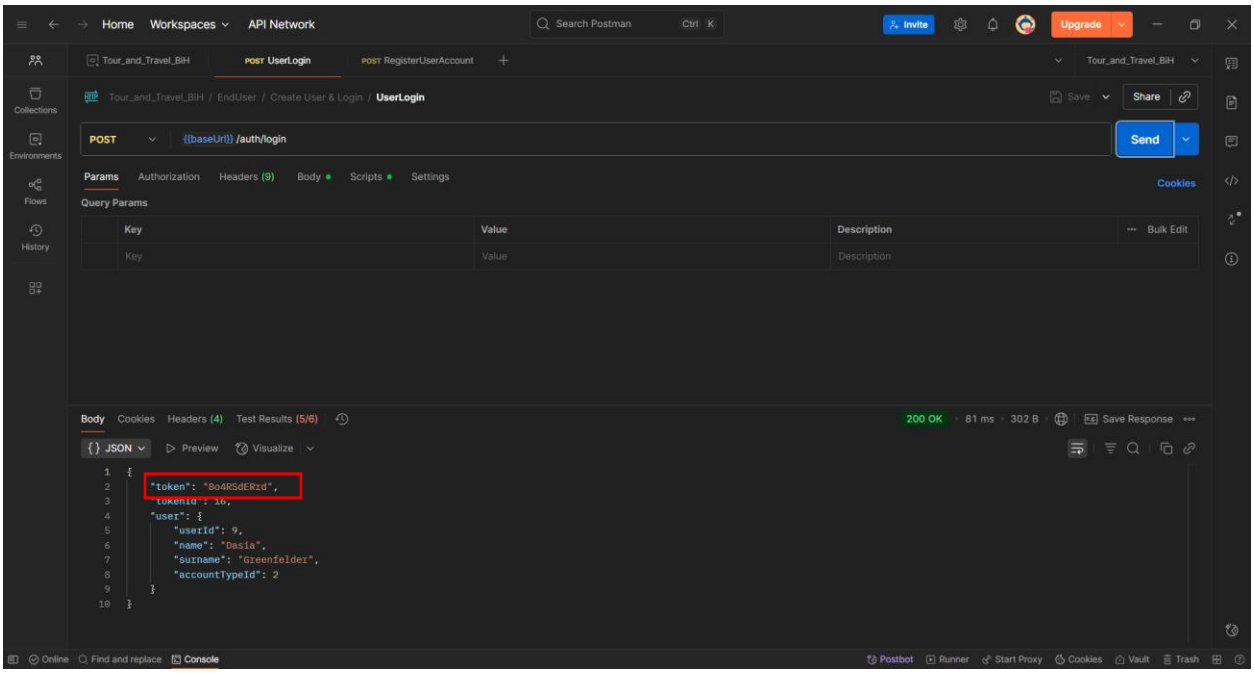

Figure 2 Login Process

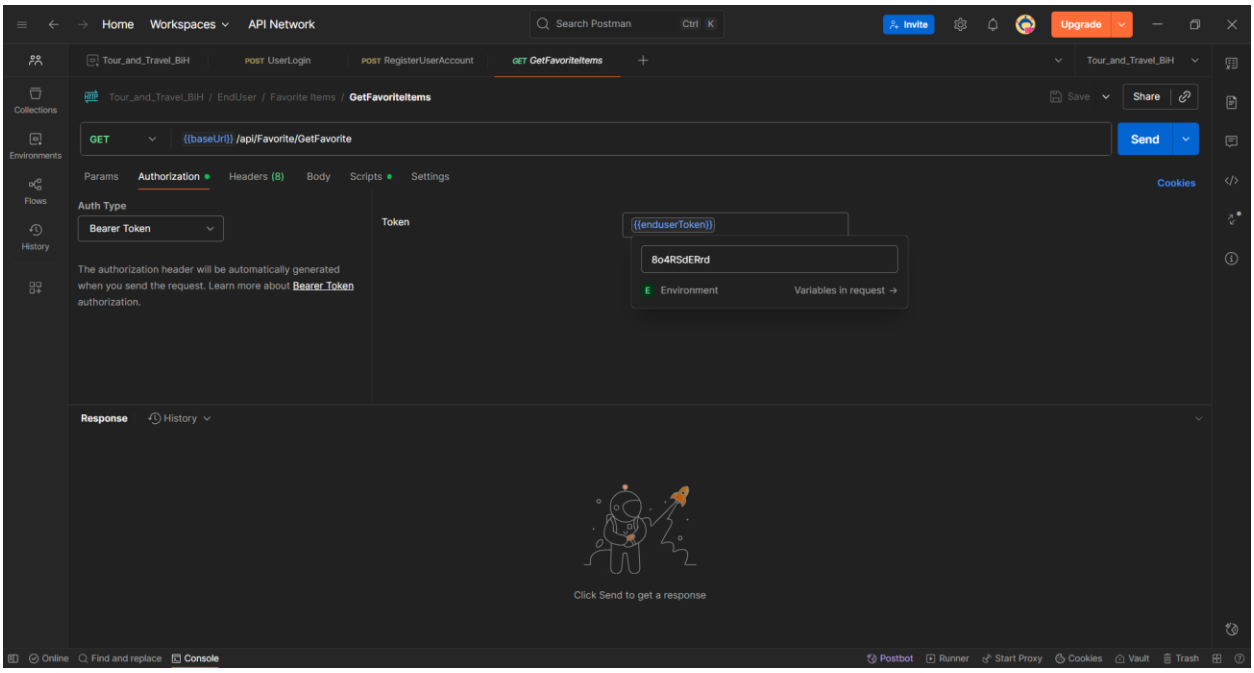

Figure 3 Authentication using a token

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

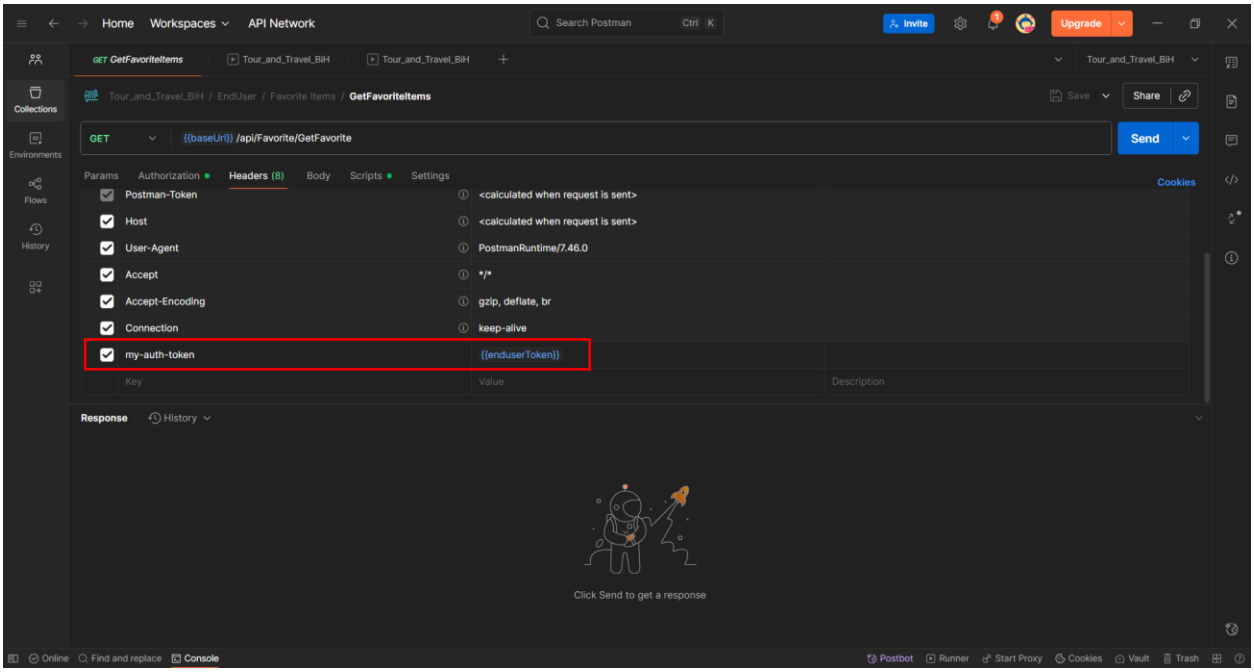

Figure 4 Adding a new key-value pair in the request headers in Postman

Tests

Body:

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

Script example for Enduser -> User Login (positive test)

```
//authorization token automaticaly added to the environment variable
const 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:

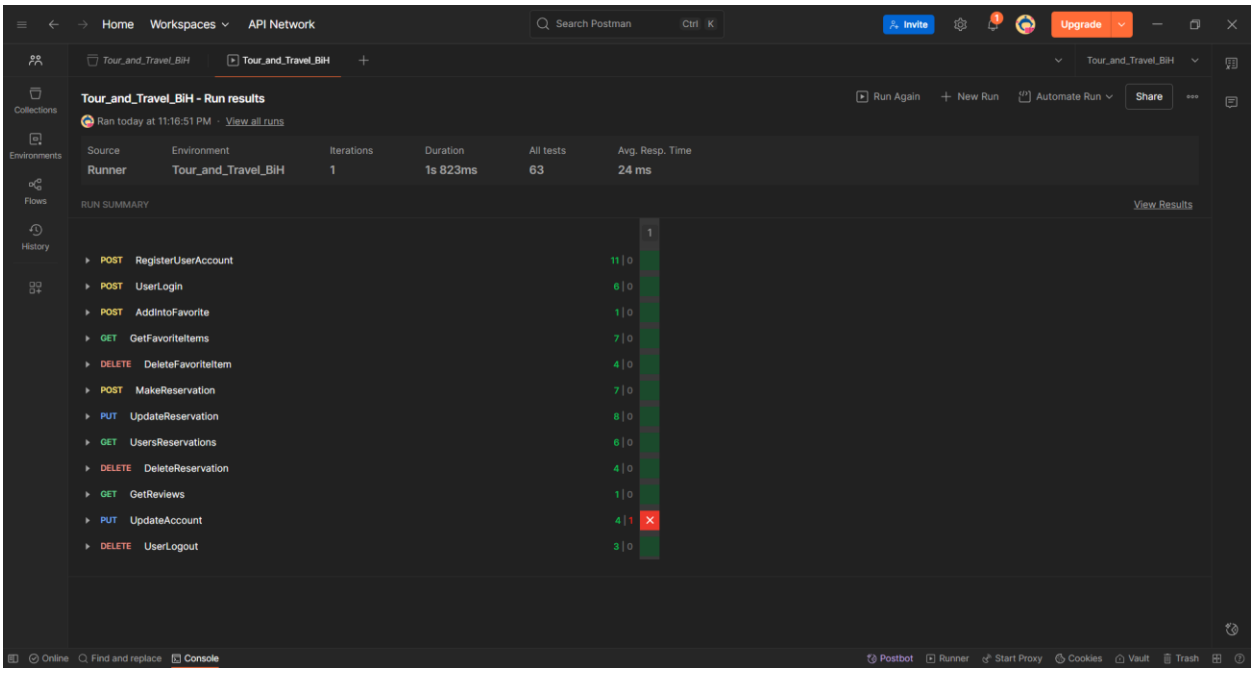

Figure 5 Functional Testing Summary

Failed test: Update User Account

Description: User updated successfully | AssertionError: expected 'Account edited' to include 'Account not found'.

The test failed because the response message was ‘account edited’, while the assertion expected ‘account not found’. This indicates that the update operation succeeded instead of returning an error for non-existent account.

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

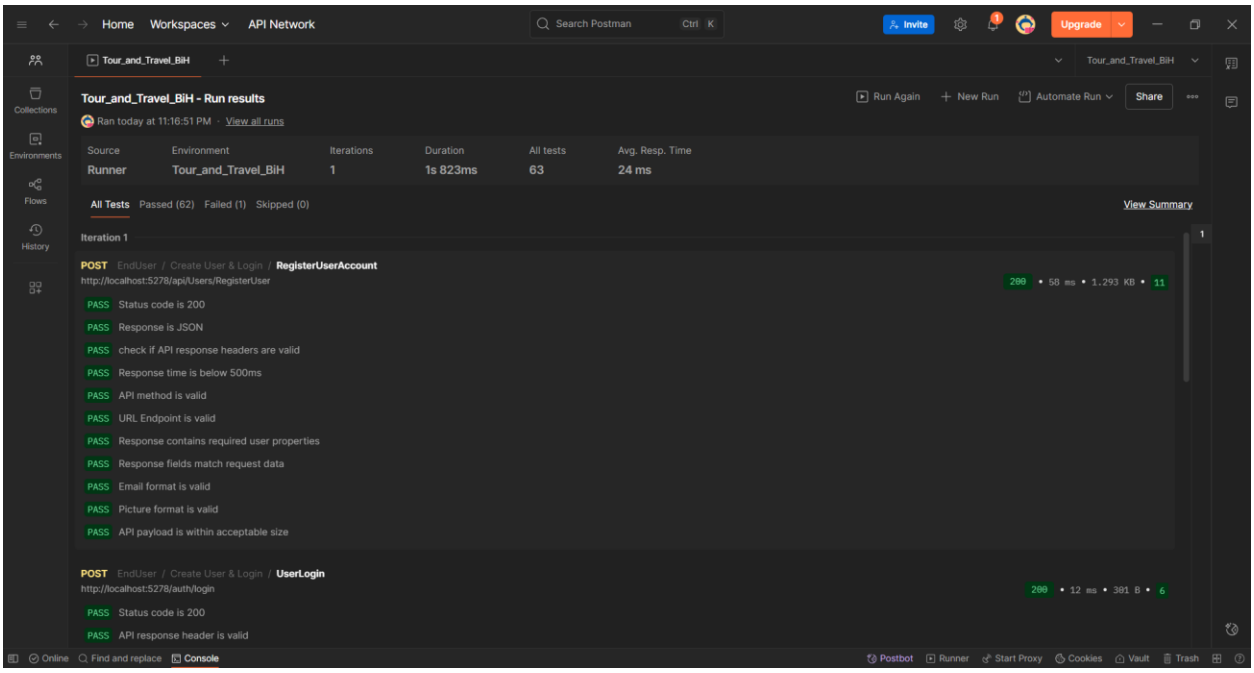

Figure 6 Functional Testing Report

Functional Testing: **Register User ---> Login ---> Delete Account**

Source	*Environment*	*Iterations*	*Duration*	*Avg.Resp.Time*
Runner	Tour_and_Travel_BiH	1	2s212ms	547ms

Passed	*Failed*	*Skip*
20	1	0

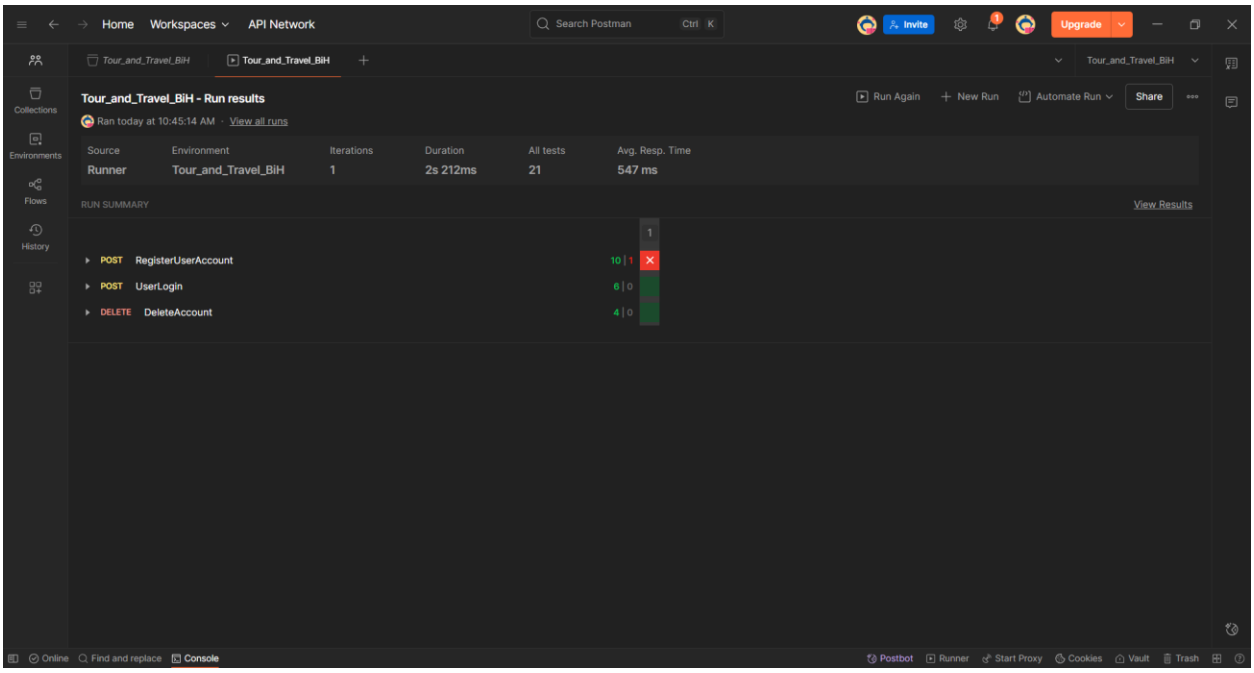

Figure 7 Functional Testing create account, login, delete account

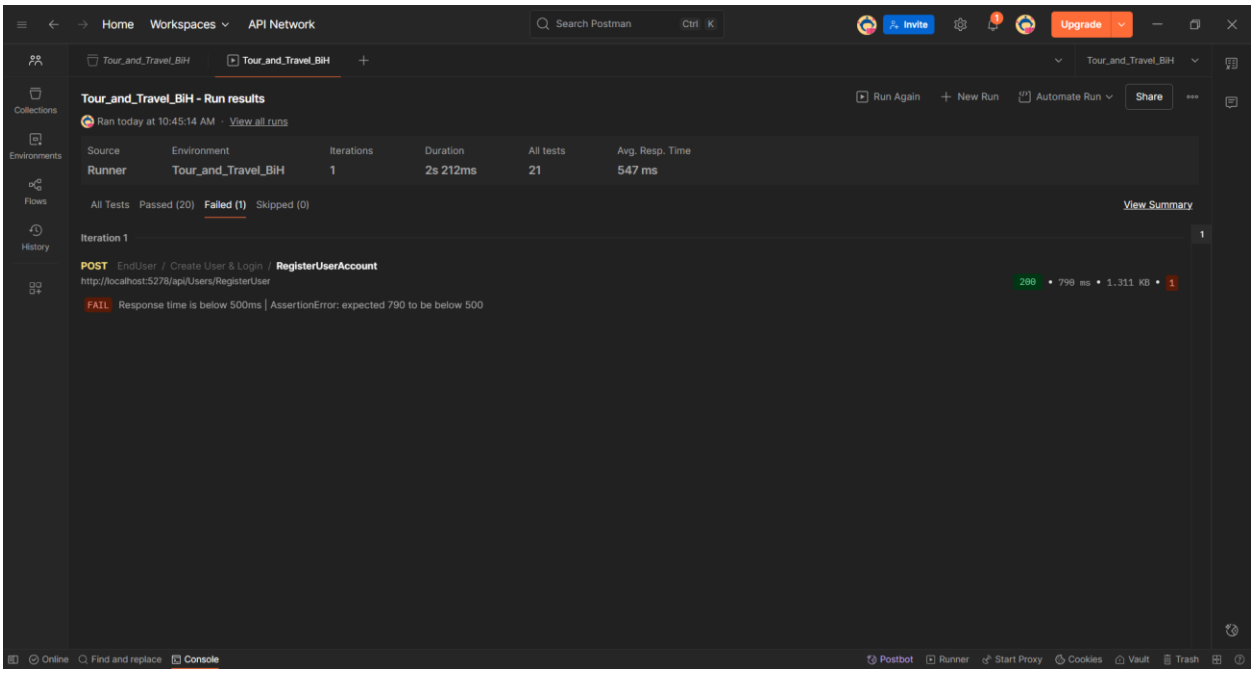

Figure 8 Failed Test

Failed test: User registration.

The test failed because the avg. response time was 547 [ms] , exceeded the defined threshold of 500 [ms]. This indicates that the API's performance is slightly below the expected standard and may require optimization.

Performance Testing

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

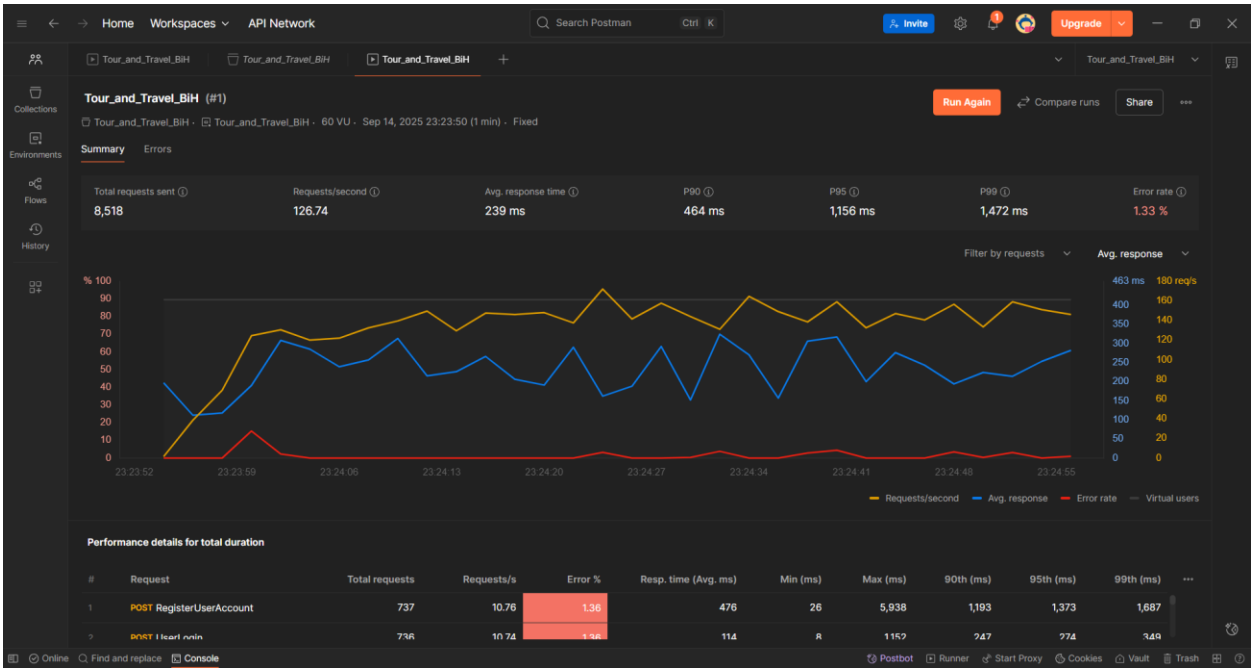

Figure 9 Performance Testing

Most request were handled within acceptable response times, averaging below 250 [ms]. Peak latencies reached over 1 second for the slowest 5-10% of requests.

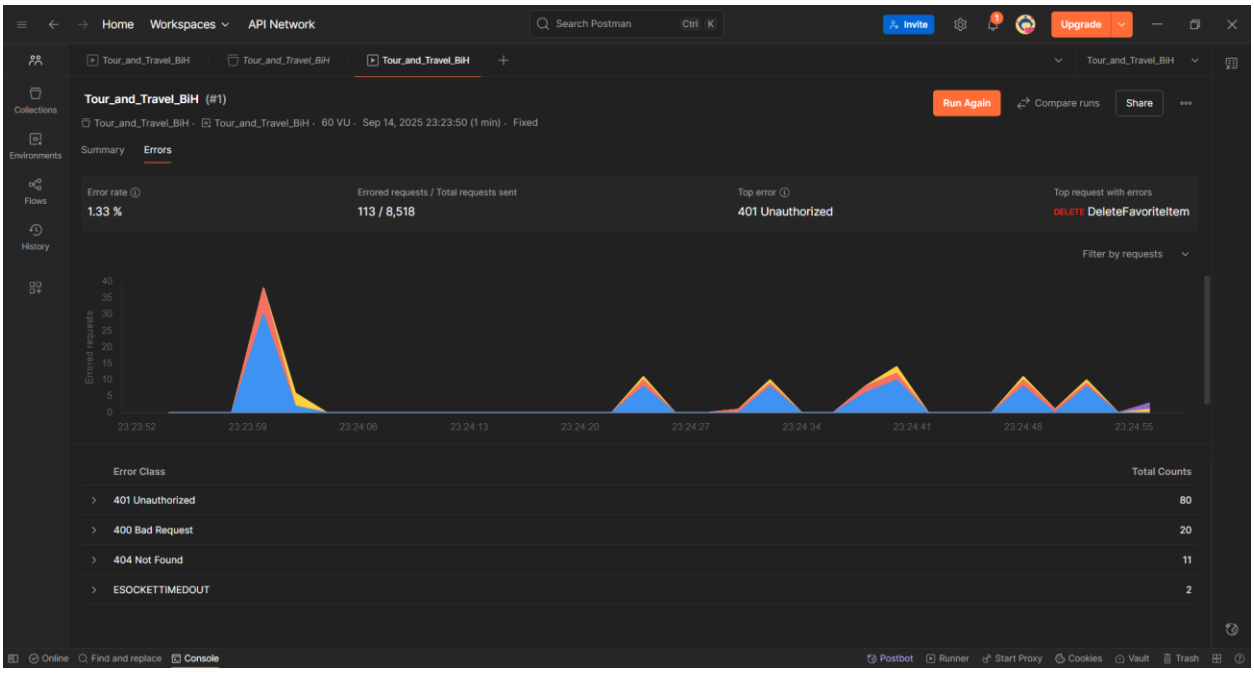

Figure 10 Performance Testing Errors

Total errors: 113 out of 8518 requests.

Top error: 401 Unauthorized (80 occurences). Top request error is from the DeleteFavoriteItem endpoint. Other errors included 400 Bad Request, 404 Not Found and ESOCKETTIMEDOUT.

Negative test example: POST RegisterUser – duplicate username

```
//Expecting a 400 for duplicated username

//status code
pm.test("Response statuse code is 400", function(){
    pm.response.to.have.status(400);
});

const jsonData = pm.response.text();

//check the error message in the response
pm.test("Error message is present", function(){
    pm.expect(pm.response.text()).to.not.be.empty;
});
```

Response example:

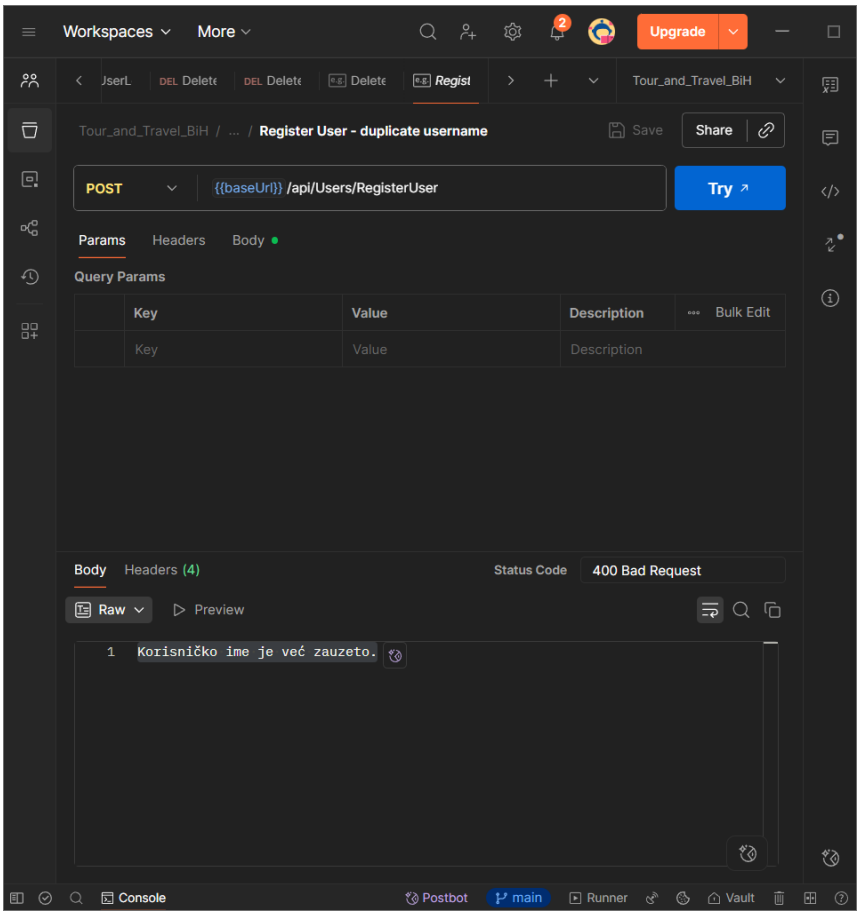

Figure 11 Response for negative test

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.

Payment's method will be added in the future.

Findings

Case: **Get Reviews by Non-Existent tourPackageId**

Expeced behavior: Response status 400 Bad Request or 404 Not Found, since the package does not exist.

Actual behavior: Response status 200 OK with an empty array [ ] .

Impact: This may confuse clients, since an invalid *{{tourPackageId}}* is treated the same as a valid package with zero reviews.

Since *{{tourPackageId}}* is passed as a path parameter, the API should ideally return 404 Not Found when the resource does not exist. Along with a 404 status, it can also return a minimal body with details, e.g. in JSON format:

{ "error": Package not found", "id": "100000" }

And in the extended form, the body with additional details could look like this:

{

"status": 404,

"error": "Not Found",

"code": "PACKAGE_NOT_FOUND",

"message": "PackageID with id 100000 not found",

"path": "/api/Review/GetReviewByPackageId/100000",

"timestamp": "2025-09-23T14:19:00Z",

"details": null

}

Which of the variants will be implemented depends on the detailed design requirements. Furthermore, the detailed requirement depends on the design, and that in turn on the functional/non-functional requirements, which should answer the question: 'who needs such a specific response from the API, and why.'

It may be needed by two different parties:

1) the API provider and

2) the API consumer,

and within these parties there are different types of users: developers, end users, etc.

Returning 200 OK with an empty array may mislead clients into thinking the request was successful for an existing package.

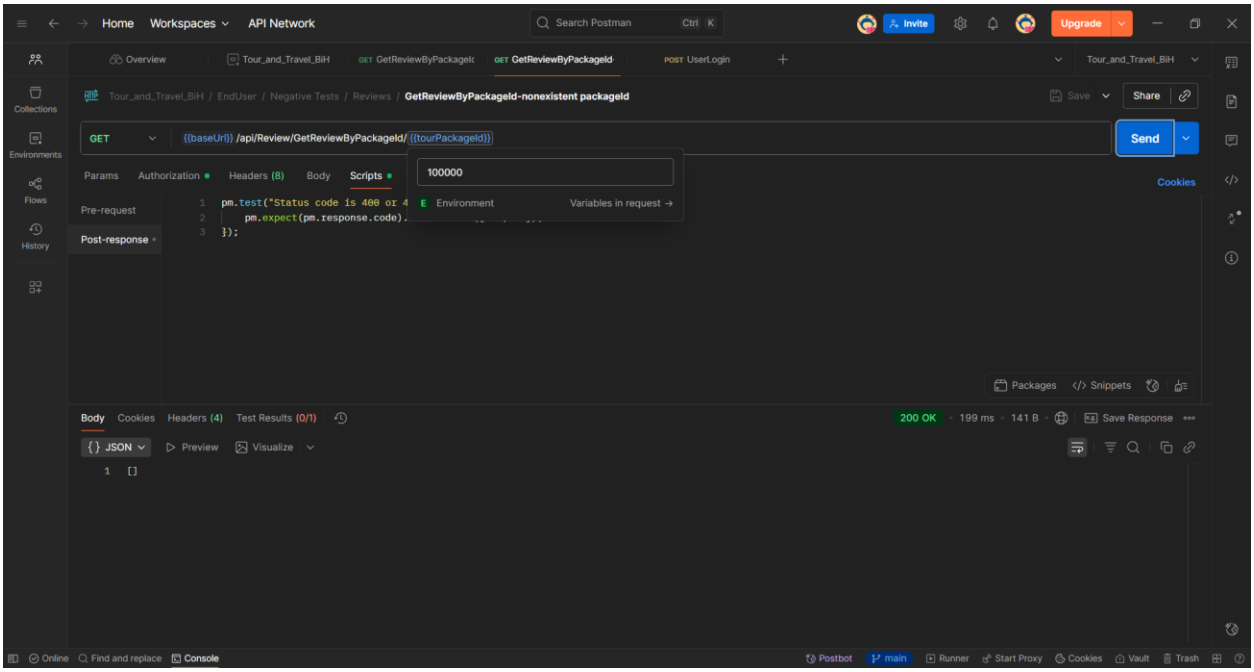

Figure 12 Response when requesting review for non-existent package

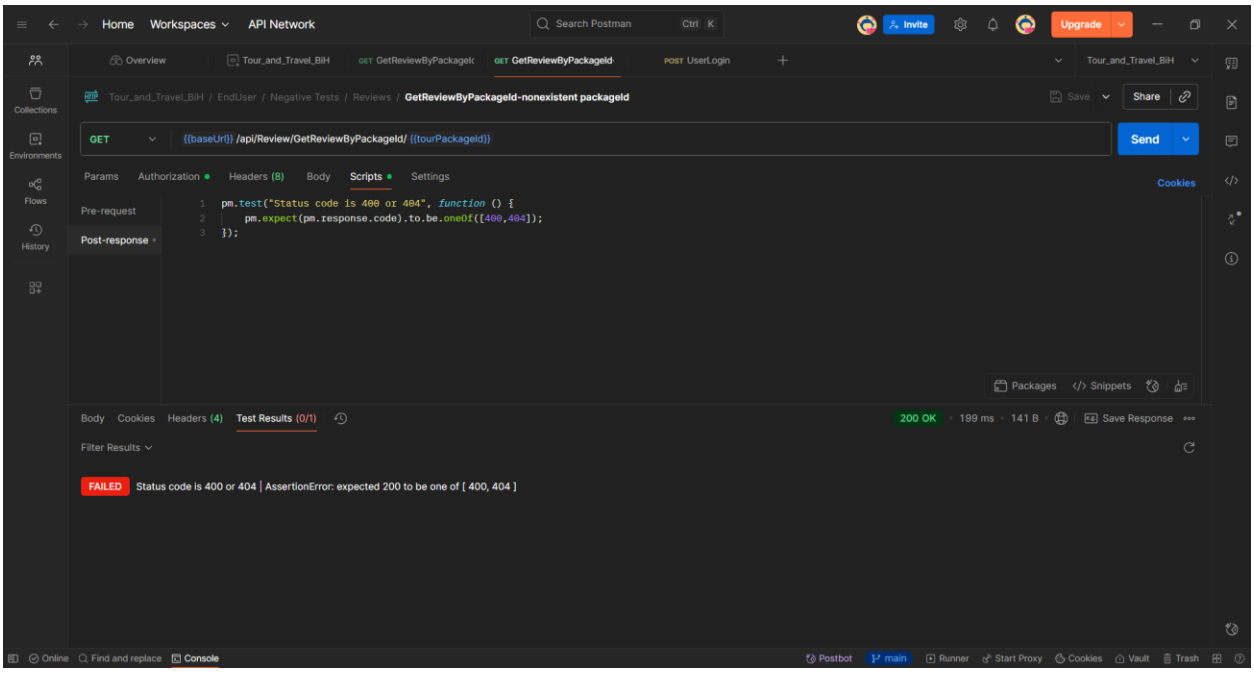

Figure 13 Test result showing failed assertion (expected 400/404, got 200)

Attachments