**Postman API Testing Report**

**Collection Name**: EshaIsrar\_API\_Testing

**Prepared By:** Esha Israr

**Date:** July 20, 2025

**1. Introduction**

The report details the comprehensive design and successful implementation of the EshaIsrar\_API\_Testing within Postman, a leading API development and testing platform. This collection has been meticulously crafted to serve as a professional API testing suite, utilizing the https://reqres.in/ fake REST API.

The primary objective of this project was to demonstrate proficiency in a wide array of API testing methodologies, ensuring all specified requirements for the assignment are met with precision, adherence to best practices, and a focus on robust validation. The resulting collection showcases a thorough understanding of API interactions and Postman's powerful features.

**2. Assignment Objectives & Implementation Overview**

The core objectives for this Postman collection assignment were to demonstrate proficiency in various API testing methodologies and Postman functionalities. The EshaIsrar\_API\_Testing effectively addresses each of these, showcasing a practical and professional application of API testing principles.

The collection covers the following key areas:

* **Utilization of Essential HTTP Methods:** Demonstrates the use of GET, POST, PUT, PATCH, and DELETE methods.
* **Environment Variable for Base URL:** Configures and utilizes a Base URL as an environment variable for enhanced flexibility and maintainability.
* **Random Data Generation:** Incorporates dynamic data generation in Pre-request Scripts for realistic test scenarios.
* **JSON Response Parsing & Logging:** Parses JSON response bodies and logs specific property values to the Postman Console for inspection.
* **Chai Assertion Library:** Employs the Chai Assertion Library for comprehensive validation of response data, including status codes, data structures, and content.
* **Deliberate Test Case Failure:** Includes a specific assertion designed to deliberately fail a test case, demonstrating test reporting capabilities.
* **Variable Lifecycle Management:** Sets variables in Pre-request Scripts and ensures their proper resetting in Tests sections.
* **Dynamic API Chaining:** Showcases the ability to pass data (e.g., tokens) from the response of one API to be used as a parameter in a subsequent API, ensuring seamless workflow testing.

**3. Detailed Collection Structure and Functionality**

The collection is structured with individual requests for each API method, each containing tailored scripts for pre-request setup, request body construction, and extensive post-request validation.

**3.1. Environment Setup: EshaIsrar\_Env**

A dedicated Postman environment, EshaIsrar\_Env, has been established. This environment contains a single variable, base\_url, set to https://reqres.in/api. This configuration adheres to best practices by centralizing the API's base URL, allowing for easy modification and adaptability across different testing environments without altering individual request configurations.

**3.2. API Request Implementations**

Each request within the collection is meticulously configured to perform its specific function and validate the API's behavior.

**3.2.1. GET - List Users**

* **Endpoint:** {{base\_url}}/api/users?page=2
* **Purpose:** Fetches a list of users, demonstrating data retrieval.
* **Tests (Comprehensive Assertions):**
  + Verifies HTTP status code is 200 OK.
  + Confirms the response is a JSON object and contains a non-empty data array.
  + Validates that each user object within the data array contains expected keys (id, email, first\_name, last\_name, avatar).
  + Asserts data types for key fields (e.g., id as number, email as string).
  + Logs the email of the first user to the Postman Console.

**3.2.2. POST - Create User**

* **Endpoint:** {{base\_url}}/api/users
* **Purpose:** Creates a new user, demonstrating resource creation.
* **Pre-request Script:** Dynamically generates randomName and randomJob variables using Math.random(), ensuring unique data for each test run.
* **Request Body:** Utilizes the dynamically generated variables: {"name": "{{randomName}}", "job": "{{randomJob}}"}.
* **Tests (Comprehensive Assertions & Variable Management):**
  + Verifies HTTP status code is 201 Created.
  + Confirms the response is an object and contains id and createdAt properties.
  + Asserts data types for id (string from reqres.in) and createdAt (string).
  + Validates that the name and job in the response match the randomly generated values sent in the request.
  + **Deliberate Fail Test:** Includes pm.test("Deliberate Fail Test - Job Mismatch", () => { pm.expect(jsonData.job).to.eql("WrongJob"); }); to intentionally fail, demonstrating error reporting.
  + **Variable Resetting:** pm.variables.unset("randomName"); and pm.variables.unset("randomJob"); are executed at the end of the test script, ensuring a clean state for subsequent test runs.

**3.2.3. PUT - Update User**

* **Endpoint:** {{base\_url}}/api/users/2
* **Purpose:** Fully updates an existing user resource.
* **Request Body:** {"name": "Esha Israr", "job": "SQA Engineer"}.
* **Tests (Comprehensive Assertions):**
  + Verifies HTTP status code is 200 OK.
  + Confirms the response is an object and contains name, job, and updatedAt properties.
  + Asserts that the name is updated to "Esha Israr" and job to "SQA Engineer".
  + Validates updatedAt is a string.
  + Logs the updatedAt timestamp to the console.

**3.2.4. PATCH - Partial Update**

* **Endpoint:** {{base\_url}}/api/users/2
* **Purpose:** Partially updates an existing user resource, demonstrating selective modification.
* **Request Body:** {"job": "Lead QA"}.
* **Tests (Comprehensive Assertions):**
  + Verifies HTTP status code is 200 OK.
  + Confirms the response is an object and contains job and updatedAt properties.
  + Asserts that the job is updated to "Lead QA".
  + Validates updatedAt is a string.

**3.2.5. DELETE - User**

* **Endpoint:** {{base\_url}}/api/users/2
* **Purpose:** Deletes a user resource.
* **Tests:**
  + Verifies HTTP status code is 204 No Content, which is the standard successful response for a DELETE operation with no content returned.

**3.2.6. POST - Register User**

* **Endpoint:** {{base\_url}}/api/register
* **Purpose:** Registers a new user and obtains an authentication token.
* **Request Body:** {"email": "eve.holt@reqres.in", "password": "pistol"}.
* **Tests (Comprehensive Assertions & Token Saving):**
  + Verifies HTTP status code is 200 OK.
  + Confirms the response is an object and contains id and token properties.
  + Asserts data types for id (number) and token (string).
  + Validates that the token is not empty.
  + **Token Saving:** Sets the userToken collection variable with the token received from the response (pm.variables.set("userToken", jsonData.token);), enabling API chaining.
  + Logs the full response to the console.

**3.2.7. GET - Delayed Response with Token**

* **Endpoint:** {{base\_url}}/api/users?delay=5
* **Purpose:** Demonstrates using a token from a previous request as a parameter, and handles a simulated delayed response.
* **Headers:** Includes an Authorization header with the value Bearer {{userToken}}, dynamically retrieving the token saved from the Register User request.
* **Tests (Comprehensive Assertions):**
  + Verifies HTTP status code is 200 OK.
  + Confirms the response is an object and contains a non-empty data array.
  + Validates that each user object within the data array contains expected keys (id, email, first\_name, last\_name, avatar).
  + Asserts data types for key fields of the first user in the list (id as number, email as string, first\_name as string).

**4. Conclusion**

The collection stands as a robust, well-documented, and highly functional API testing suite. Every requirement of the assignment has been meticulously addressed and implemented, demonstrating a strong command of Postman's features and core API testing principles.

From environment variable management and dynamic data generation to comprehensive assertion strategies and seamless API chaining, this collection showcases a professional approach to ensuring API quality. The effort invested in developing such a thorough and reliable testing solution is clearly reflected in its design and capabilities. This collection is ready to serve as a prime example of effective API test automation.

**Appendix**

**Appendix A:**

