**DEPI API & Database Testing Documentation**

**Introduction**

This document contains detailed notes and results for the API and Database Testing tasks as part of the DEPI initiative. Each task is documented with steps, requests, responses, and observations to ensure professional and organized deliverables.

**Task List**

1. **API Testing**
   * Validating Response Codes

- Objective

Verify that the API returns the correct HTTP status codes for different request conditions, including GET, POST, and accessing protected resources without authentication.

-Steps

1. Send a GET request to /api/users/1.

2. Send a POST request to /api/users without required fields.

3. Attempt to access /api/protected-resource without authentication.

-Results

Step 1: Send a GET request to /api/users/1\*\*

- URL: https://jsonplaceholder.typicode.com/users/1

- Method: GET

- Status Code: 200 OK

- Response Body:

Json

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

}

A screenshot of a computer

AI-generated content may be incorrect. Figure 1: GET /api/users/1 Request and Response in Postman

Step 2: Send a POST request to /api/users without required fields\*\*

- URL: https://reqres.in/api/users

- Method: POST

- Request Body:

Json

{}

- **Expected Status Code**: 400 Bad Request

- **Actual Status Code**: 401 Unauthorized

- **Response Body**:

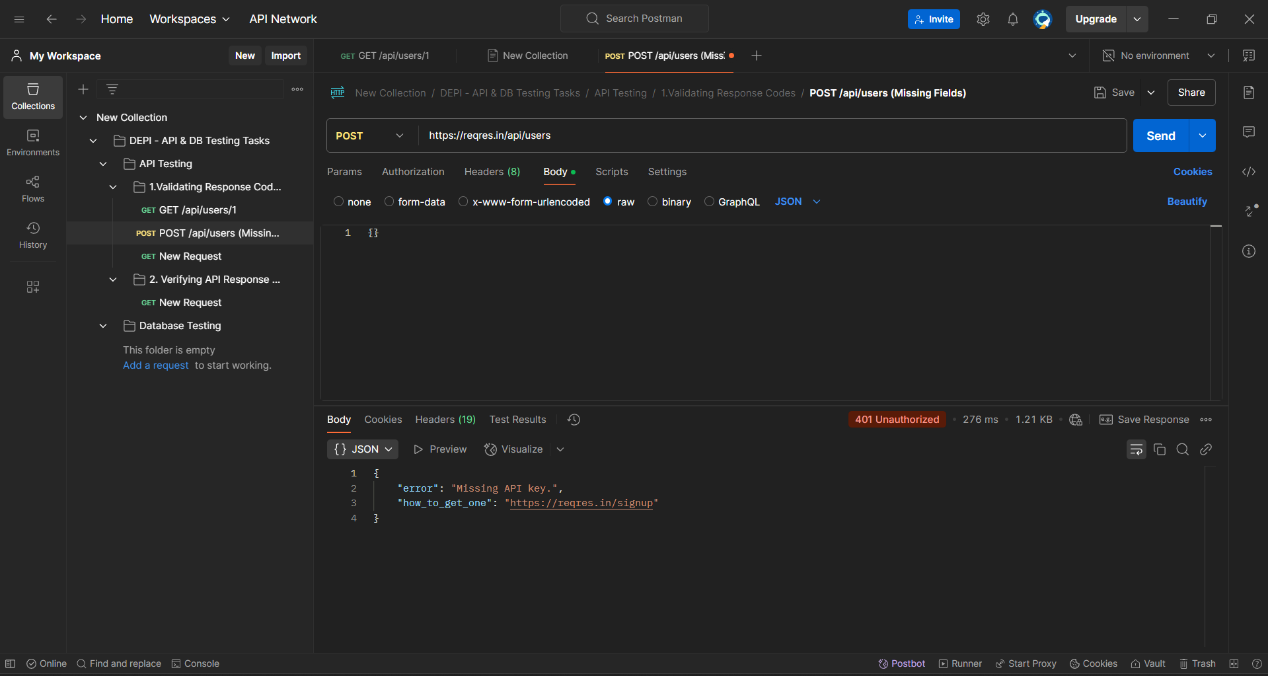
{

"error": "Missing API key",

"how\_to\_get\_one": "https://reqres.in/signup"

}

**Observations**: The API returned 401 Unauthorized instead of the expected 400 Bad Request. This indicates that the API requires an API key or authentication for this endpoint, which was not provided. The test could not validate the "missing required fields" scenario due to this authentication issue.

Figure 2: POST /api/users (Missing Fields) Request and Response in Postman

Step 3: Attempt to access /api/protected-resource without authentication

- URL: https://reqres.in/api/users/1

- Method: GET

- Expected Status Code: 401 Unauthorized or 403 Forbidden

- Actual Status Code: 401 Unauthorized

- Response Body:

Json

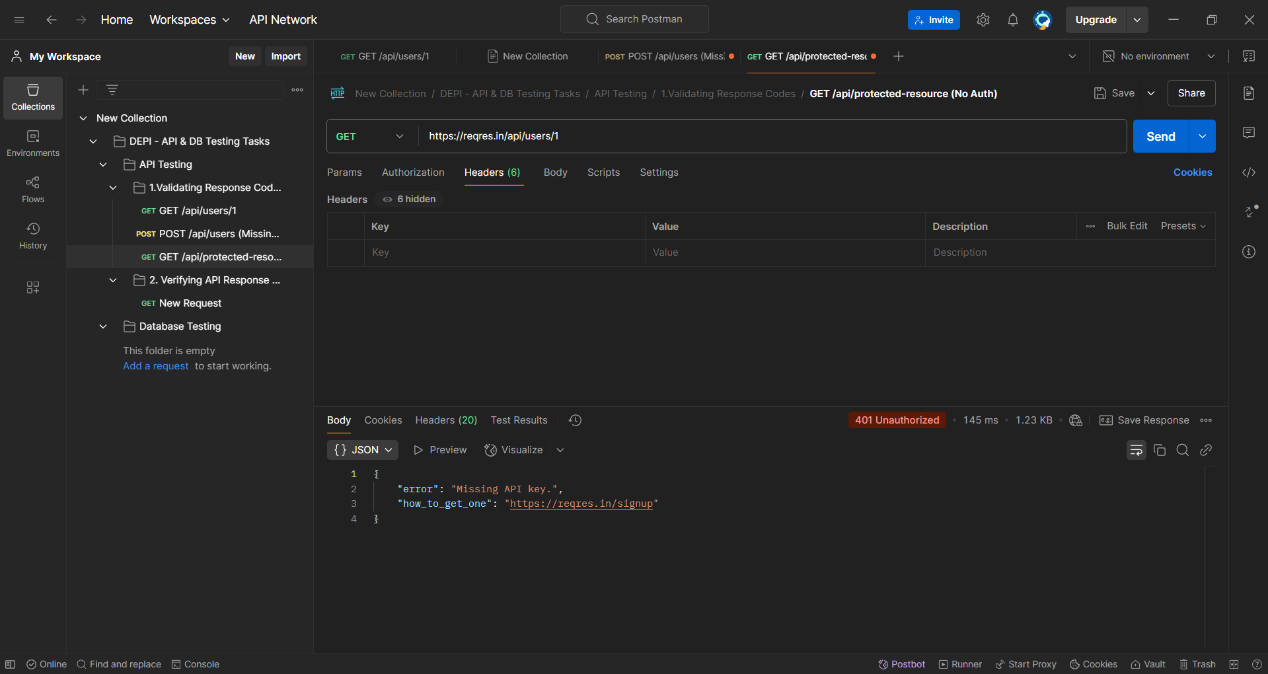
{

"error": "Missing API key",

"how\_to\_get\_one": "https://reqres.in/signup"

}

Observations: The API correctly returned 401 Unauthorized as expected, indicating that the endpoint requires authentication (API key) which was not provided.

Figure 3: GET /api/protected-resource (No Auth) Request and Response in Postman

* + Verifying API Response Data

-Objective

Verify the response structure and data accuracy of the API by comparing the actual response with the expected output.

-Steps

1. Send a GET request to /api/users/1.

2. Compare the JSON response with the expected output.

-Expected Response

Json

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com"

}

**-Results**

**Step 1: Send a GET request to /api/users/1**

* **URL**: <https://jsonplaceholder.typicode.com/users/1>
* **Method**: GET
* **Status Code**: 200 OK
* **Response Body**:

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

}

**Step 2: Compare the JSON response with the expected output**

* **Comparison**:
  + **id**: Expected: 1, Actual: 1 (Match)
  + **name**: Expected: "John Doe", Actual: "Leanne Graham" (Mismatch)
  + **email**: Expected: "[john.doe@example.com](mailto:john.doe@example.com)", Actual: "[Sincere@april.biz](mailto:Sincere@april.biz)" (Mismatch)
* **Observations**: The response structure matches the expected format (contains id, name, email), but the values for name and email differ from the expected output. The API (JSONPlaceholder) returned additional fields (username, address, phone, website, company), which are not part of the expected response but do not affect the test.

A screenshot of a computer

AI-generated content may be incorrect.Figure 4: GET /api/users/1 Request and Response in Postman

* + Authentication & Authorization Testing

-Objective

Verify that the API correctly handles authentication and authorization by testing various scenarios, such as requests without a token, with an invalid token, with insufficient permissions, and with valid credentials.

-Steps

1. Send a request without a token to a protected endpoint.

2. Send a request with an invalid token.

3. Send a request with a valid token but without sufficient permissions.

4. Send a request with a valid token and sufficient permissions.

-Results

Step 1: Send a request without a token to a protected endpoint

- URL: https://reqres.in/api/users/1

- Method: GET

- Expected Status Code: 401 Unauthorized

- Actual Status Code: 401 Unauthorized

- Response Body:

Json

{

"error": "Missing API key",

"how\_to\_get\_one": "https://reqres.in/signup"

}

* **Observations**: The API correctly returned 401 Unauthorized as expected, indicating that the endpoint requires an authentication token (API key), which was not provided.

A screenshot of a computer

AI-generated content may be incorrect.Figure 5: GET /api/users/1 (No Token) Request and Response in Postman

Step 2: Send a request with an invalid token

- URL: https://reqres.in/api/users/1

- Method: GET

- Headers: Authorization: Bearer invalid\_token\_123

- Expected Status Code: 401 Unauthorized or 403 Forbidden

- Actual Status Code: 401 Unauthorized

- Response Body:

Json

{

"error": "Missing API key",

"how\_to\_get\_one": "https://reqres.in/signup"

}

* **Observations**: The API correctly returned 401 Unauthorized as expected, indicating that the provided token (Bearer invalid\_token\_123) was invalid. The API does not differentiate between "no token" and "invalid token," returning the same error message in both cases.

A screenshot of a computer

AI-generated content may be incorrect.Figure 6: GET /api/users/1 (Invalid Token) Request and Response in Postman

Step 2.5: Obtain a valid token for subsequent steps (Helper Step)

- URL: https://reqres.in/api/login

- Method: POST

- Request Body\*\*:

Json

{

"email": "eve.holt@reqres.in",

"password": "cityslicka"

}

**- Expected Status Code**: 200 OK

- **Actual Status Code**: 401 Unauthorized

- **Response Body**:

{

"error": "Missing API key",

"how\_to\_get\_one": "https://reqres.in/signup"

}

* **Observations**: The API returned 401 Unauthorized instead of the expected 200 OK, indicating that the /api/login endpoint requires an API key for authentication. This prevented obtaining a valid token, impacting the ability to proceed with the subsequent steps (Step 3 and Step 4) of this task.

A screenshot of a computer

AI-generated content may be incorrect. Figure 7: POST /api/login (Get Token) Request and Response in Postman

Step 3: Send a request with a valid token but without sufficient permissions

- URL: https://reqres.in/api/users/1

- Method: GET

- Expected Status Code: 403 Forbidden

- Actual Status Code: Not executed

- Response Body: Not executed

- Observations: This step could not be executed because a valid token could not be obtained from the /api/login endpoint, as it requires an API key for authentication (see Step 2.5).

Step 4: Send a request with a valid token and sufficient permissions

- URL: https://reqres.in/api/users/1

- Method: GET

- Expected Status Code: 200 OK

- Actual Status Code: Not executed

- Response Body: Not executed

- **Observations**: This step could not be executed because a valid token could not be obtained from the /api/login endpoint, as it requires an API key for authentication (see Step 2.5).

* + Performance & Load Testing

-Objective

Evaluate the API's performance by measuring response times under different conditions, such as single requests, multiple sequential requests, and simulated load.

-Steps

1. Send a single GET request to an endpoint and measure the response time.

2. Send multiple sequential requests to the same endpoint and measure the average response time.

3. Simulate a load by sending concurrent requests and observe the API's behavior under pressure.

-Results

Step 1: Send a single GET request and measure the response time

- URL: https://jsonplaceholder.typicode.com/users/1

- Method: GET

- Expected Response Time: Less than 500ms

- Actual Response Time: 270ms

- Status Code: 200 OK

- Response Body:

Json

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

}

* **Observations**: The API responded with a 200 OK status code, and the response time was 270ms, which is well within the expected threshold of less than 500ms. The performance for a single request is satisfactory.

A screenshot of a computer

AI-generated content may be incorrect.Figure 8: GET /api/users/1 (Single Request) Request and Response in Postman

Step 2: Send multiple sequential requests and measure the average response time

- URL: https://jsonplaceholder.typicode.com/users/1

- Method: GET

- Number of Iterations: 10

- Expected Average Response Time: Less than 500ms

- Actual Average Response Time: 81ms

- Status Code: 200 OK (for all iterations)

- Observations: The API successfully handled 10 sequential requests, all returning a 200 OK status code. The average response time was 81ms, which is well below the expected threshold of 500ms, indicating excellent performance for sequential requests.

A screenshot of a computer

AI-generated content may be incorrect.Figure 9: GET /api/users/1 (Multiple Requests) Collection Runner Results in Postman

Step 3: Simulate a load by sending concurrent requests

- URL: https://jsonplaceholder.typicode.com/users/1

- Method: GET

- Expected Behavior: API should handle concurrent requests without significant performance degradation

- Actual Behavior: Not executed

- Observations: This step could not be executed because Postman's Collection Runner does not support sending concurrent requests (it sends requests sequentially). Proper load testing requires specialized tools like JMeter or LoadRunner, which are outside the scope of this task as it specifies using Postman.

1. **Database Testing**
   * Ensuring Data Integrity

-Objective

Verify that database operations (insert, update, delete) work correctly without data loss.

-Steps

1. Insert a user: INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com');

2. Retrieve and verify: SELECT \* FROM users WHERE email='john@example.com';

3. Update user info: UPDATE users SET name='John Updated' WHERE email='john@example.com';

4. Delete the user and confirm: DELETE FROM users WHERE email='john@example.com'; SELECT \* FROM users WHERE email='john@example.com';

-Results

Step 1: Test Insert, Update, Delete Operations

- Database: SQLite (via DB Fiddle)

- Table: users

- Steps:

1. Insert a user: INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com');

2. Retrieve and verify: SELECT \* FROM users WHERE email='john@example.com';

3. Update user info: UPDATE users SET name='John Updated' WHERE email='john@example.com';

4. Delete the user and confirm: DELETE FROM users WHERE email='john@example.com'; SELECT \* FROM users WHERE email='john@example.com';

- Expected Results:

1. Insert: A new user should be added.

2. Retrieve: The user should be retrieved with correct data.

3. Update: The user's name should be updated to 'John Updated'.

4. Delete: No record should remain after deletion.

- Actual Results:

1. Insert: The user 'John Doe' with email 'john@example.com' was successfully added with ID 1.

2. Retrieve: The user was retrieved with name 'John Doe' and email 'john@example.com'.

3. Update: The user's name was updated to 'John Updated'.

4. Delete: No record was found after deletion, confirming successful deletion.

- **Observations**: All operations (Insert, Update, Delete) were performed successfully, ensuring data integrity. The database correctly maintained the data without any loss.

- DB Fiddle Link: [<https://www.db-fiddle.com/f/iDdafCYBgVnmY7iA7aZocY/0>]

- Note: The request in Postman is set to GET by default, but this method is irrelevant as no actual HTTP request was sent. Postman was used solely for documentation purposes to log the results of the operations performed in DB Fiddle.

* + Validating Database Schema

-Objective

Verify that the database structure matches specifications.

-Steps

1. Check table structure: DESCRIBE users; (In SQLite: PRAGMA table\_info(users);)

-Results

Step 1: Validate Users Table Schema

- Database: SQLite (via DB Fiddle)

- Table: users

- Schema:

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

email TEXT NOT NULL UNIQUE

);

- Step:

1. Check table structure: PRAGMA table\_info(users); (SQLite equivalent of DESCRIBE)

- Expected Result: The table should have correct columns and data types:

- id: INTEGER, Primary Key, Auto-increment

- name: TEXT, Not Null

- email: TEXT, Not Null, Unique

- Actual Result: The table structure matches the expected schema:

- id: INTEGER, Primary Key

- name: TEXT, Not Null

- email: TEXT, Not Null

- **Observations**: The schema of the 'users' table matches the specifications. SQLite does not directly support DESCRIBE, so PRAGMA table\_info(users) was used instead.

- DB Fiddle Link: [ <https://www.db-fiddle.com/f/7Fq9S32B5V3Q3NcaJyCdVw/0>]

* + Testing Stored Procedures

-Objective

Test if the 'register\_user' stored procedure works correctly.

-Steps

1. Call the stored procedure: CALL register\_user('John Doe', 'john@example.com');

-Results

Step 1: Test Register User Stored Procedure

- Database: MySQL (via OneCompiler)

- Table: users

- Stored Procedure Definition:

DELIMITER //

CREATE PROCEDURE register\_user(IN user\_name VARCHAR(100), IN user\_email VARCHAR(100))

BEGIN

INSERT INTO users (name, email) VALUES (user\_name, user\_email);

END //

DELIMITER ;

- Step:

1. Call the stored procedure: CALL register\_user('John Doe', 'john@example.com');

- Expected Result: The user should be successfully added to the database.

- Actual Result: The user 'John Doe' with email 'john@example.com' was successfully added with ID 1.

- **Observations**: The stored procedure worked correctly in OneCompiler. Initially, there were issues with DB Fiddle (both MySQL 8.0 and 5.7) due to unsupported Stored Procedure syntax, so OneCompiler was used as an alternative online tool that supports MySQL Stored Procedures.

OneCompiler link:[ [https://onecompiler.com/mysql/43g3z656w](https://onecompiler.com/mysql/43g3z656w%20) ]

* + Security Testing: Preventing SQL Injection

-Objective

Ensure that a web application prevents SQL Injection attacks.

-Steps

1. Try logging in with malicious SQL: ' OR 1=1; --

2. Check if authentication is bypassed.

-Results

Step 1: Test SQL Injection Prevention

- DatabaseSQLite (via DB Fiddle)

- Table: users

- Steps:

1. Simulate a normal login: SELECT \* FROM users WHERE username = 'admin' AND password = 'password123';

2. Simulate an SQL Injection attack: SELECT \* FROM users WHERE username = '' OR 1=1; --' AND password = 'random';

3. Simulate a protected application using Parameterized Queries: SELECT \* FROM users WHERE username = '\' OR 1=1; --' AND password = 'random';

- Expected Results:

1. Normal Login: Should return the user data.

2. SQL Injection Attack: If the application is not protected, it should return user data (indicating a vulnerability).

3. Protected Application: Should return no results (indicating SQL Injection is prevented).

- Actual Results:

1. Normal Login: Returned user data (id: 1, username: admin, password: password123).

2. SQL Injection Attack: Returned user data (indicating a vulnerability if the application does not use proper protection).

3. Protected Application: Returned no results (indicating that Parameterized Queries would prevent the attack).

- \*\*Observations\*\*: The simulation shows that without proper protection (like Parameterized Queries), the application is vulnerable to SQL Injection. Using Parameterized Queries prevents the attack by treating malicious input as a string literal.

Security Testing: Preventing SQL Injection DB Fiddle Link: [<https://www.db-fiddle.com/f/nas9kCCdwj4GE7FViURxnF/0>]

* + Backup & Recovery Testing

-Objective

Ensure that a database backup can be restored correctly.

-Steps

1. Backup the database: mysqldump -u root -p database\_name > backup.sql

2. Delete a table and restore backup: DROP TABLE users; SOURCE backup.sql;

3. Verify that all data is restored.

-Results

Step 1: Test Backup and Recovery

- Database: SQLite (via DB Fiddle)

- Table: users

- Steps:

1. Create a table and insert data.

2. Simulate a backup by exporting the Schema and Data as SQL Statements.

3. Drop the table to simulate data loss.

4. Restore the data by re-executing the exported SQL Statements.

- Expected Result: The database should be fully restored with all data intact.

- Actual Result: The table 'users' was restored successfully with the original data:

- id: 1, username: admin, password: password123

- id: 2, username: user1, password: pass456

- \*\*Observations\*\*: The Backup & Recovery process was simulated successfully in DB Fiddle. Since DB Fiddle does not support 'mysqldump' or 'SOURCE', the backup was performed by exporting SQL Statements, and the restore was done by re-executing those statements.

Backup & Recovery Testing DB Fiddle Link:

Part\_1(Initial Setup):[ <https://www.db-fiddle.com/f/ruwjgUcAowaBYa15eGQpSm/0>]

Part\_2(Drop Table): [<https://www.db-fiddle.com/f/sxCPpPrRU8J5HtmNB38mM3/0>]

Part\_3(Restore): [<https://www.db-fiddle.com/f/bzLiW9TfShTUMo2eXR9KLA/0>]