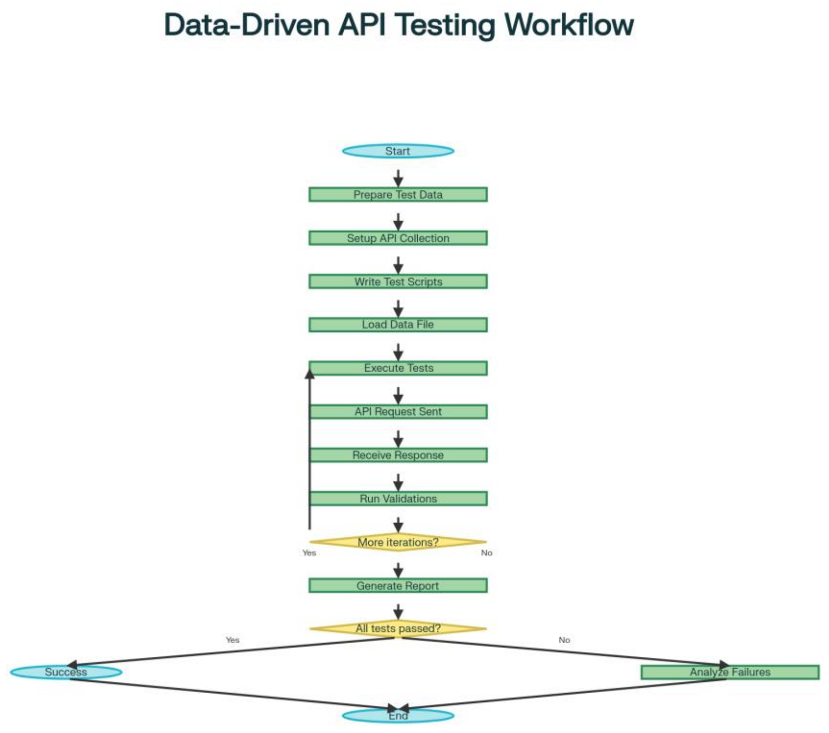
Data-Driven API Testing - Complete Guide

Table of Contents

Data-driven API testing empowers teams to efficiently validate endpoints by leveraging dynamic datasets for automated test execution. This comprehensive guide teaches developers and testers how to create datasets, design test cases, automate with runners, and analyze test results using Postman, CSV/JSON files, and test scripts. This guide also includes special coverage for testing file uploading APIs with a data-driven approach.

Visual Overview

The diagram below summarizes the essential steps in the data-driven API testing workflow:



Key Concepts and Benefits

- **Separates data from logic:** Test scripts use variables, datasets contain test inputs and expected outputs.
- **Efficient and scalable:** Single scripts can be run against hundreds of scenario datasets with minimal effort.

- **Improves coverage:** Covers positive, negative, and edge cases for comprehensive API validation.
- **Fosters maintainability:** Easily update and extend scenarios by editing data files, not code.

Step-by-Step Approach

1. Prepare Your Test Dataset

- Use CSV or JSON files to store test inputs. Each row/object is a test scenario.

CSV Example

username	password	email	newsletter
amit_qa	test123	amit@example.com	true
sara_dev	sar@67	sara@company.org	false

JSON Example

```
[

{"username": "amit_qa", "password": "test123", "email": "amit@example.com", "newsletter
{"username": "sara_dev", "password": "sar@67", "email": "sara@company.org", "newsletter

]
```

2. Write Parameterized Test Scripts

Test scripts reference variables from the imported data file using double curly braces (`{{variable}}`):
In Request body:

```
{
  "username": "{{username}}",
  "password": "{{password}}",
  "email": "{{email}}"
}
```

Tests use these dynamic values for assertions: Write the below script to your post response

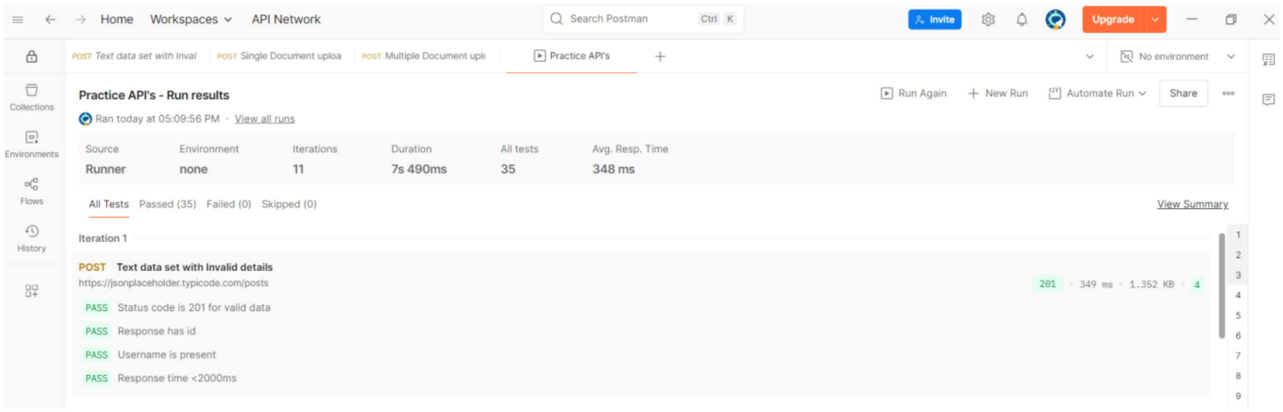
```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

var em = pm.variables.get("email");
pm.test("Check email " + em, function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.email).to.eql(em);
});
```

3. Run Tests with the Collection Runner

How to Execute:

- Open Postman, select your collection.
- Click "Run"/"Runner".
- Upload the CSV or JSON data file.
- Run the collection; tests will iterate over all rows/scenarios.

You can preview file rows, configure iterations, and view pass/fail results with detailed metrics for each test to run.
Example



Practice API's - Run results
Ran today at 05:09:56 PM · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	11	7s 490ms	35	348 ms

All Tests Passed (35) Failed (0) Skipped (0)

Iteration 1

POST Text data set with Invalid details
https://jsonplaceholder.typicode.com/posts
201 · 349 ms · 1.352 KB · 4

PASS Status code is 201 for valid data
PASS Response has id
PASS Username is present
PASS Response time <2000ms

4. Analyze Results and Reporting

- View detailed logs and assertions for each row/request
- Export run results for sharing and further analysis
- Use performance dashboards to monitor status, response time, error rates
- Integration with CI/CD platforms (Jenkins, GitLab, etc.) supported for automation.

Testing File Upload APIs with Data-Driven Approach

Many APIs require file upload functionality for processing documents, images, or other binary data. Testing these APIs requires special handling for multipart form-data and file attachments.

Understanding File Upload Testing

File upload testing validates that your API correctly:

- Accept files in specified formats (PDF, JPG, PNG, DOCX, etc.)
- Validates file size limits.
- Handles multiple file uploads simultaneously.
- Returns appropriate error messages for invalid files.
- Processes uploaded files correctly and stores them securely.

File Upload API Test Cases **Positive Test**

Cases:

- Upload a single file with valid format and size.

- Upload multiple files simultaneously.
- Upload files with different valid extensions (PDF, JPG, PNG).
- Upload files at maximum allowed size limit.
- Upload files with valid metadata/parameters.

Negative Test Cases:

- Upload file with unsupported format (e.g., .exe, .bat)
- Upload files exceeding the size limit.
- Upload empty file (0 bytes).
- Upload without required parameters
- Upload with incorrect content-type header.
- Upload locked/password-protected files.

Setting Up File Upload Tests in Postman

Step 1: Configure Request

1. Set request method to POST
2. Set Content-Type header to `multipart/form-data`
3. In Body tab, select "form-data"
4. Add file field with type "File"

Step 2: Create Test Data File

For file upload testing, create a CSV file with:

- File paths (relative or absolute)
- Expected status codes
- Expected response messages
- File metadata (optional parameters)

CSV Example for File Upload Testing:

file_path	doc_type	expected_status	expected_message
test_files/passport.pdf	passport	200	File uploaded successfully
test_files/eid.jpg	eid	200	File uploaded successfully
test_files/invalid.exe	passport	415	Unsupported file type
test_files/large.pdf	passport	413	File size exceeds limit

Step 3: Parameterize File Upload Request

Use variables in your Postman request:

```
// Pre-request Script
 pm.variables.set("filePath", pm.iterationData.get("file_path"));
 pm.variables.set("docType", pm.iterationData.get("doc_type"));
```

Form-data configuration:

- Key: `file`, Type: File, Value: `{{filePath}}`
- Key: `doc_type`, Type: Text, Value: `{{docType}}`

Step 4: Write Test Assertions

```
// Test Script
pm.test("Status code validation", function () {
    const expectedStatus = pm.iterationData.get("expected_status");
pm.response.to.have.status(expectedStatus);
});

pm.test("Response message validation", function () {
    var jsonData = pm.response.json();
    const expectedMsg = pm.iterationData.get("expected_message");
pm.expect(jsonData.message).to.include(expectedMsg);
});

pm.test("File upload validation", function () {
   if (pm.response.code === 200) {
   var jsonData = pm.response.json();
pm.expect(jsonData).to.have.property("file_id");
pm.expect(jsonData).to.have.property("uploaded_size");     }
});
```

Advanced File Upload Testing Scenarios Testing

Multiple File Uploads:

For APIs that accept multiple files, structure your form-data with multiple file fields:

```
file1: {{filePath1}}
file2: {{filePath2}}
description: {{description}}
```

Testing with Additional Parameters:

Many file upload APIs require additional metadata along with files:

```
{
  "files": ["file:project_files/document.pdf"],
  "user_id": "{{userId}}",
  "category": "{{category}}",
  "description": "{{description}}"
}
```

Testing File Size Limits:

Create test files of varying sizes to validate size restrictions:

- Small files (1-10 KB)

- Medium files (100 KB - 1 MB)
- Large files (5-10 MB)
- Extra large files (exceeding limit)

Using Postman Test Data Storage

Postman provides test data storage for team collaboration:

1. Upload test files to Postman team storage.
2. Files become available to all team members.
3. Files work with monitors and scheduled collection runs.
4. Files accessible from Postman Flows and CLI.

This eliminates the need for each team member to maintain local copies of test files.

REST Assured Example for File Upload Testing

For Java-based automation, use REST Assured:

```
@Test
public void testFileUpload() {
    File testFile = new File("test_files/document.pdf");

    given()
        .multiPart("file", testFile, "application/pdf")
        .formParam("doc_type", "passport")
        .formParam("user_id", "12345")
    .when()
        .post("/api/upload")
    .then()
        .statusCode(200)
        .body("message", containsString("uploaded successfully"))
        .body("file_id", notNullValue());
}
```

Python Example for File Upload Testing

Using Python requests library:

```
import requests
```

```
def test_file_upload():
    url = "http://api.example.com/upload"

    files = {
        'file': open('test_files/document.pdf', 'rb')
    }

    data = {
        'doc_type': 'passport',
        'user_id': '12345'
    }

    response = requests.post(url, files=files, data=data)

    assert response.status_code == 200
    assert 'file_id' in response.json()
    assert response.json()['message'] == 'File uploaded successfully'
```

Best Practices and Enhancements

General Best Practices

- Use simple, descriptive column names and variable placeholders.
- Include edge cases: blank, invalid values, large payloads.
- Refactor tests for reusable logic and maintain script clarity.
- Store test scripts in version control (Git) for collaboration.
- Document the dataset, expected outputs, and validations.
- Visualize test workflows and performance metrics for team review.

File Upload Specific Best Practices

- Test with diverse file types and sizes.
- Validate file integrity after upload.
- Test concurrent uploads from multiple users.
- Monitor upload performance and timeout handling.
- Test security validations (file type restrictions, malicious files).
- Clean up test files after execution.
- Use realistic test data that mirrors production scenarios.

Summary Table: Tools Comparison

Tool/Feature	Data-driven Support	File Upload Support	Batch Execution	Data Sources
Postman	Yes	Yes	Yes	CSV, JSON
REST Assured	Yes	Yes	Yes	Excel, CSV, JSON
PyTest	Yes	Yes	Yes	CSV, JSON, Excel
Selenium	Yes	Yes	Yes	CSV, JSON, Excel

Conclusion

Data-driven API testing transforms manual validation into an automated, scalable routine. By leveraging structured datasets, parameterized scripts, and test runners, teams gain broad coverage, increase automation ROI, and improve API reliability.

The addition of file upload testing capabilities ensures comprehensive validation of document processing APIs, which are critical for banking, e-commerce, and content management systems. The step-by-step process and practical examples make this technique accessible for QA engineers, developers, and teams transitioning to modern automated testing workflows.

Whether testing simple REST APIs or complex file upload endpoints, the data-driven approach provides the framework for efficient, maintainable, and comprehensive test automation.

References

All examples and guidance in this document are informed by current best practices and tutorials from Postman, REST Assured, Testsigma, and practical API testing workflows.

1. https://stackoverflow.com/questions/38395661/how-to-test-rest-api-multiple-file-upload-with-sigoden-apitest
2. https://www.geeksforgeeks.org/node-js/use-of-data-files-csv-json-in-postman-for-data-driven-testing/
3. https://www.echoapi.com/blog/mastering-postman-data-driven-testing-with-csv-and-json-files/
4. https://www.browserstack.com/release-notes/en/csv-import-json-test-case-id-tagging-for-bdd-test-cases-Lv4jyJq 0
5. https://blogs.perficient.com/2022/07/12/api-testing-multipartfile-upload-using-java-spring-framework/
6. https://www.xrmlabs.com/About-Us/XRMs-Blog/Post/8585/Data-Driven-Testing-Using-Postman
7. https://www.contentstack.com/blog/composable/csv-upload-using-automate
8. https://onlinetestcase.com/test-cases-for-upload-file/
9. https://www.linkedin.com/pulse/how-perform-data-driven-testing-postman-jithin-somaraj-8x8fc
10. https://stackoverflow.com/questions/75386097/put-data-from-csv-file-automation-testing
11. https://davidwalsh.name/demo/multiple-file-upload.php
12. https://learning.postman.com/docs/sending-requests/create-requests/test-data/
13. https://github.com/authorjapps/zerocode/wiki/File-Upload-Using-Http-API-Automated-Testing
14. https://www.redline13.com/blog/2019/09/jmeter-multiple-random-file-uploads/
15. https://testsigma.com/blog/test-file-upload/
16. https://docs.loadforge.com/examples/specialized-performance/file-upload-testing
17. https://community.postman.com/t/api-testing-a-beginners-view-data-driven-testing-in-postman/20302
18. https://www.youtube.com/watch?v=mN-tr73ZNtQ
19. https://testsigma.com/blog/test-file-upload/
20. https://www.geeksforgeeks.org/software-testing/how-to-perform-data-driven-testing-from-csv-to-postman-reque sts-scripts/
21. https://community.postman.com/t/data-driven-testing-using-a-csv-file-for-automation/14590