Project Report: Combined Functional and Performance API Testing Automation

1. Project Overview

This project involved the development of a Python-based automated testing framework designed to validate both the functional correctness and performance efficiency of RESTful API endpoints. The goal was to simulate real-world testing scenarios for backend services, evaluate API reliability, and generate meaningful reports to support quality assurance processes.

2. Objectives

- Implement a test suite for verifying API responses, status codes, and data schemas.
- Simulate concurrent user requests to measure response times under load.
- Generate structured reports (CSV and HTML) for test traceability and performance review.
- Provide insights into endpoint stability and responsiveness through automated testing.

3. Tools and Technologies Used

- Python – Core programming language
- requests – Sending HTTP requests
- aiohttp & asyncio – Simulating concurrent API load
- jsonschema – Validating JSON structure and schema
- pandas – Logging and saving test data
- jinja2 – Generating structured HTML reports
- Google Colab – Execution environment for cloud-based testing

4. Functional Testing Implementation

Functional tests were executed using Python's requests library to validate:

- Correct HTTP status codes (200, 404, etc.)
- JSON validity for API responses
- Compliance with predefined JSON schema for selected endpoints

Test outcomes were stored in structured DataFrame and exported to a CSV file for analysis.

The testing framework handled both successful and failed requests, ensuring coverage for expected and edge-case scenarios.

5. Performance Testing Implementation

Performance tests were conducted using aiohttp and asyncio to simulate 20 concurrent users hitting each GET endpoint. Metrics captured include:

- Total requests
- Successful responses
- Average, minimum, and maximum response time (in milliseconds)

This approach helped identify APIs that might degrade under load or show inconsistency in response time.

6. Reporting and Visualization
Two types of reports were generated:

- CSV Reports: For functional and performance test results
- HTML Report: A styled, comprehensive summary of all test cases and performance stats created using jinja2

Reports included response validation, schema validation status, and execution time—providing a QA-friendly view of API behavior.

7. Challenges and Resolutions

- Concurrent Testing in Colab: Addressed with nest_asyncio to enable asyncio.run() in a Jupyter/Colab context.
- Schema Variability: Limited schema checks to known endpoints; handled missing keys gracefully to avoid false negatives.
- Invalid Endpoint Simulation: Incorporated negative tests to validate 404/error handling without halting execution.

8. Outcome and Learnings

- Successfully built a reusable, scalable framework for REST API testing.
- Gained practical experience in API automation, schema validation, and concurrent performance simulation.
- Understood how to interpret performance metrics and detect API bottlenecks or failures.

9. Future Enhancements

- Integrate with pytest for unit-style test cases.
- Extend to support POST/PUT methods with payload validation.
- Include authentication (OAuth or token-based) for protected APIs.
- Deploy reports via email or integrate into CI/CD pipelines for continuous testing.