QA Automation Strategy for API Testing

1. Overview

This document outlines a **QA Automation Strategy** for the API testing framework. The strategy ensures **scalability, coverage, and seamless integration** within the development lifecycle, enabling efficient and reliable automated testing.

2. Goals & Objectives

- Ensure **consistent API reliability** through automated testing.
- Achieve **high test coverage** for both functional and non-functional requirements.
- Provide **fast feedback** to developers during CI/CD processes.
- Maintain **scalability** to support future test growth.
- Ensure smooth **integration** into the development lifecycle.

3. Test Coverage & Scope

Functional Testing (Verifies API behavior)

- **GET /products** → Ensure correct product retrieval.
- **POST /product** → Validate product creation with valid/invalid data.
- **DELETE /products/{id}** → Confirm product deletion and proper error handling.

Non-Functional Testing (Ensures performance, security, and reliability)

- **Performance** → Stress tests with high concurrent API requests.
- **Security Testing** → SQL Injection, XSS, and invalid input handling.
- **Scalability** → Running parallel tests in Docker for faster execution.

4. Test Automation Framework

- **Programming Language**: Python 3.12+
- **Test Framework**: Pytest
- **HTTP Client**: Requests
- **Data Handling**: JSON-based test data files
- **informative comments and improved assertion explanations**

5. Test Execution & CI/CD Integration

- **Local Execution**:
    - Run all tests: `pytest -n auto`

    - Run specific tests: `pytest tests/test_api/test_get_product.py`
    - Run stress tests: `pytest -m stress`
- **Continuous Integration (CI)**:
    - **GitHub/GitLab CI/CD** triggers on new commits.
    - Tests executed automatically on PR merges.
    - Reports generated for each execution.
- **Docker Integration**:
    - Tests run inside Docker for consistency.
    - Command: `docker run --rm api-tests`
    - Supports `TEST_SELECTION` environment variable for test filtering.

6. Scalability & Maintenance

- **Modular & Reusable Code**: Helper functions for API requests.
- **Data-Driven Testing**: Dynamic test case generation (using python decorator)
- **Version Control**: Git & branching strategies.
- **Regular Test Suite Review**: Identify and remove outdated cases.

7. Defect Management & Reporting

- **Defects logged in issue tracking system** (Jira, X-Ray for example).
- **Reports shared with the team** after each CI/CD test run.
- **Failure analysis performed regularly** to improve test reliability.

8. Conclusion

This QA automation strategy ensures **reliable, scalable, and efficient API testing** while integrating seamlessly into the development lifecycle. It enables rapid feedback, improves software quality, and supports future scalability.

✅ **Next Steps:** Integrate QA strategy in CI/CD, enhance test coverage, expand security testing, implement logging for tracking (using python logger), generate stress test reports with Pandas, and add more negative test cases.

🚀 **Goal:** Ensure a **robust, well-tested API** with automation-driven confidence!