@author Bhimashankar Teli

Functional Testing Strategy for API Testing: -

1. Objective: To ensure that the API behaves as expected and meets its functional requirements by validating the core features and business logic.

2. Functional Test Types:

- **Basic Functional Tests< Request/Response Validation>:**
    - Verify that each API endpoint returns the correct HTTP status code (2xx).
    - Validate that the response body contains the correct data based on the request.
    - Check that API responses match the expected JSON schema, including data types and required fields.
    - Ensure that request payloads are accepted as specified and returned in the expected format.
    - Validate that all required fields are included in API responses.
- **CRUD Operations Verification<** Pass different datasets from the CSV/Excel file to validate different scenarios – Both Negative and Positive>**:**
    - **Create**: Test POST requests for creating new records ().
    - **Read**: Test GET requests to retrieve records (e.g., spending summaries, account balances).
    - **Update**: Test PUT/PATCH requests to modify existing records (e.g., updating investment details).
    - **Delete**: Test DELETE requests to remove records or cancel transactions.
- **Edge Case Testing:**
    - Test for boundary conditions like maximum input length and data limits.
    - Check how the API handles invalid input (e.g., invalid account number or unsupported formats).
- **Security and Authorization Tests:**
    - Verify that the API enforces authentication and authorization properly.
    - Ensure secure access controls and that only authorized users can access sensitive data.
- **Error Handling Tests:**
    - Ensure that the API returns appropriate error codes and messages when provided with invalid input.

- Test for specific error codes such as 400 Bad Request, 401 Unauthorized, 403 Forbidden, and 404 Not Found.

Non-Functional Testing Strategy for API Testing: -

1. Objective: To evaluate the non-functional aspects of the API, ensuring that the product performs well under different scenarios and meets requirements for security, usability, and scalability.

2. Non-Functional Test Types:

- **Performance Testing:**
  - **Load Testing**: Use tools like **JMeter** or **Gatling** to simulate a large number of concurrent users to test the API's response time and stability under load.
  - **Stress Testing**: Push the API beyond its limits to determine how it handles extreme traffic and to identify failure points.
  - **Scalability Testing**: Assess the API's ability to scale when subjected to increasing loads. Verify that performance does not degrade as more requests are processed.
- **Security Testing:**
  - **Vulnerability Scanning**: Use automated tools like **OWASP ZAP** or **Burp Suite** to check for common vulnerabilities such as SQL injection, XSS, and insecure API endpoints.
  - **Data Encryption**: Verify that data transmitted via the API is encrypted using protocols such as HTTPS and TLS.
  - **Access Controls**: Ensure role-based access control (RBAC) and secure API token management.
  - **Authentication & Authorization**: Confirm that the API enforces user authentication and access restrictions using standards like OAuth2 or JWT.
- **Usability Testing:**
  - **Error Messages and Documentation**: Validate that API error responses are user-friendly and provide meaningful information to clients.
  - **Ease of Integration**: Ensure that the API can be integrated seamlessly by third-party applications, with clear and accurate documentation.

- **Compliance Testing:**
    - Ensure that the API follows relevant industry regulations (e.g., GDPR (General Data Protection Regulation), PCI DSS (Payment Card Industry Data Security Standard) for financial data security).
    - Verify that data privacy and security measures are in place to protect sensitive user data.
- **Compatibility Testing:**
    - Ensure that the API works across different platforms, browsers, and client applications.
    - Verify that API responses are consistent regardless of client location or device type.
- **Reliability and Availability Testing:**
    - **Uptime Monitoring**: Verify that the API maintains high availability and minimize downtime through automated monitoring tools like **Pingdom** or **Datadog**.
    - **Failover Testing**: Simulate failure scenarios to ensure that the API can recover and maintain service availability without data loss.

Test Plan and Execution for Functional and Non-Functional Testing

1. Test Setup:

- **Environment**: Ensure that testing environments are configured to mimic production as closely as possible.
- **Test Tools**: Set up **Postman**/ **Insomnia** for manual testing and **Rest Assured**, **JUnit**, **TestNG, Playwright** for automated functional tests. Use **JMeter** for performance and load testing, and **OWASP ZAP** for security tests.
- **CI/CD Integration**: Include API testing in the CI/CD pipeline using **Jenkins**, **GitHub Actions**, or **GitLab CI/CD** to automatically run tests on each deployment.

2. Test Execution:

- **Functional Testing Execution**:
    - Run functional tests as part of pre-commit or nightly builds.
    - Use data-driven testing to cover various scenarios with different data sets.
- **Non-Functional Testing Execution**:

- Schedule performance tests during off-peak hours to simulate high load and analyse system behaviour.
- Implement continuous security scans as part of the pipeline.

3. Monitoring and Reporting:

- **Real-Time Monitoring**: Integrate with tools like **New Relic** and **Datadog** to monitor API health.
- **Test Reporting**: Use **Allure** or **Extent Reports** for comprehensive reporting on test outcomes.
- **Feedback Loop**: Collect feedback from test reports to refine test cases and update the strategy as needed.

Conclusion:

A comprehensive API testing plan that encompasses both functional and non-functional testing ensures that the fintech product meets user expectations, performs reliably under various conditions, and adheres to security and compliance standards. By integrating these testing practices with an automated CI/CD pipeline, the product can be maintained with high quality while supporting frequent updates.