@author Bhimashankar Teli

Case Study: Test Plan and Strategy for a SaaS Fintech Product

Overview: A SaaS-based fintech product has been developed that provides external applications with APIs for accessing consumer financial data and insights, including spending from bank accounts, investments (stocks, mutual funds, etc.), EPF balances, and loan liabilities. Initially launched as part of a Minimum Viable Product (MVP), the product underwent manual testing and crowdsourced testing. Following its launch, the product saw a rapid increase in subscribers, tripling in a month. The business now requires frequent updates (at least one update per two days) to address API changes, security patches, and scaling needs. The challenge is to develop a comprehensive testing strategy that supports quick and smooth updates while transitioning to a fully automated testing and deployment process.

Current Challenges:

1. High update frequency (one update every two days).
2. Ensuring robust testing for a fintech product dealing with sensitive financial data.
3. Supporting rapid development and deployment without business disruptions.
4. Transitioning from manual and crowdsourced testing to a fully automated approach.

Objectives:

- Create an automated test strategy to support continuous integration and continuous deployment (CI/CD).
- Ensure high test coverage for API functionality, security, and integration.
- Reduce manual testing efforts and turnaround time for patches and updates.
- Maintain the integrity and security of the platform with each update.
- Enable smooth transitions between manual testing and full automation.

Assumption: All test cases written and refactored. Readily available in the repository.

Testing Strategy Overview

1. **Test Automation Framework:**
    - **Tools to Use**:
        - **API Testing**: Postman for initial explorations; Rest Assured, Wire Mock, Extent Reports, Playwright, Cucumber, JUnit, and TestNG for automation.
        - **CI/CD Integration**: Jenkins, GitHub Actions, or GitLab CI/CD for continuous integration and deployment.
        - **Security Testing**: OWASP ZAP or Burp Suite for automated security scans and use Rest Assured for automation testing
        - **Performance Testing**: JMeter or Gatling for load and stress testing. We can use Rest Assured for performance automation testing.
        - **UI and Mobile testing**: Selenium webdriver, Playwright, Cypress, Appium.
    - **Framework Design**:
        - **Modular Approach**: Create reusable components for test scripts that can be maintained and extended.
        - **Data-Driven Testing**: Use parameterized tests to simulate different scenarios with various data sets.
        - It supports both BDD and TDD.
        - Framework should support API, UI, Performance, Security, Database and Mobile Testing
        - **API Test Cases**:
            - **Basic Functional Testing**: Ensure endpoints return expected responses with correct data formats.
            - **Boundary Testing**: Test for edge cases (e.g., very large transactions).
            - **Negative Testing**: Validate the system's behaviour when given invalid input.
            - **Security Tests**: Include tests for common vulnerabilities (e.g., SQL injection, XSS).
            - **Load Testing**: Simulate concurrent API calls to assess system behaviour under stress.

        - **End-to-End Scenarios**: Simulate complete workflows from data input to final analysis to ensure integration integrity.

2. **Test Coverage Plan**:
    - **API Endpoints**: Coverage should include all public and private API endpoints.
    - **Functional Tests**: Ensure all core features, such as fetching bank spending data, investment portfolios, and liabilities, work as expected.
    - **Integration Tests**: Verify integration with third-party data sources like banks and investment platforms.
    - **Regression Tests**: Automatically run a suite of tests with each release to verify that previous functionalities remain intact.
    - **Security Tests**: Regularly scan APIs for vulnerabilities using tools like OWASP ZAP and manual review for advanced threats.
    - **Performance Tests**: Run benchmarks to measure how new updates affect API response times, scalability, and load handling.
    - **Compliance Checks**: Validate that security updates comply with financial regulations and best practices.
3. **CI/CD Pipeline Design**:
    - **Code Integration**:
        - Developers commit code to the repository. A CI/CD pipeline is triggered upon each commit.
        - **Pipeline Stages**:
            - **Build**: Compile the code and generate the build artifacts.
            - **Unit Tests**: Run basic unit tests to check for individual component integrity.
            - **API Functional Tests**: Execute automated test scripts using RestAssured or Postman.
            - **Security Scans**: Run automated security scans.
            - **Load Tests**: Perform scaled load testing using JMeter to identify potential bottlenecks.
            - **Deployment**: Deploy changes to a staging environment for final testing before production.
        - **Monitoring and Alerts**:
            - Set up monitoring for pipeline runs, with alerts for failures or performance issues.

4. **Test Data Management**:
    - Use anonymized and sanitized data for testing.
    - Implement a data management system for handling test data creation, seeding, and cleanup.
    - Regularly refresh test data to reflect real-world scenarios and edge cases.
5. **Transition Plan from Manual to Automated Testing**:
    - **Phase 1: Hybrid Testing**:
        - Start with running automated regression tests alongside selected manual tests.
        - Focus on automating high-priority and frequently used test cases.
    - **Phase 2: Incremental Automation**:
        - Gradually automate other parts of the testing cycle, such as integration and security tests.
        - Train the QA team to create and maintain automation scripts.
    - **Phase 3: Full Automation**:
        - Complete the transition by automating all test cases.
        - Introduce nightly builds with full test suites that cover functional, performance, and security tests.

6. Best Practices

- **Shift Left Testing**: Start testing earlier in the development cycle to catch issues sooner.
- **Continuous Feedback**: Integrate testing feedback with version control and monitoring tools.
- **Test Data Management**: Use synthetic and anonymized data to maintain data privacy and compliance.
- **Collaboration**: Ensure alignment between development, QA, and operations teams for smoother deployment.

Performance Monitoring and Feedback Loop:

- **User Feedback**: Gather feedback from clients to identify any issues after deployment.
- **Monitoring**: Use tools like New Relic or Datadog for real-time performance monitoring and alerting.
- **Incident Response**: Set up an incident response plan for any critical failures or security incidents.

Conclusion: To meet the demand for rapid updates, the product must transition to a fully automated test strategy integrated within the CI/CD pipeline. **This strategy will ensure that updates can be deployed with minimal manual intervention while maintaining the highest standards of functionality, security, and performance**. The hybrid approach initially allows for the smooth integration of automation with existing manual tests, scaling gradually until full automation is achieved.