Allister Totong
Andi Muhammad Imam Akbar
Christopher Gerard Lissants
Ryufath Alief Adhyaksa Putera Soepeno

Testing and Validation Plan (Week 14)

Testing Plan

A successful, complete, and consistent integration plan requires not only identifying the gaps or strategies toward systemness but also putting the entirety and concept of the integration into fruition. Therefore, a testing plan is needed to fulfill the integration plan, both in terms of knowledge and planning.

The objectives of this testing plan are to verify that data flows correctly between each subsystem, making sure that the system performs well under expected conditions, validate interoperability and seamless communication between the subsystems and address both functional and non-functional requirements. Here below are the plans allocated for the integration:

1. Test individual modules like an API to synchronize the inventory stock for customer data updates in isolation.
2. Integration Testing:
    - Verify data flow between connected subsystems (e.g., Inventory to CRM).
    - Test middleware performance and data transformation.
3. Perform System Testing by assessing the entire integrated system for expected behavior. Then validate business processes (e.g., automated stock updates).
4. Performance Testing:
    - Measure response times of APIs and data synchronization.
    - Load and stress testing for scalability.
5. Verify that new changes do not disrupt existing functionality through regression testing
6. Security Testing: Test for vulnerabilities in data transfer protocols and interfaces.
7. Validate that the system meets end-user requirements through User Acceptance Testing (UAT)

Test Cases

A. Data Flows
    1. Data Exchange between Inventory and CRM system
    Test Scenario:
        - Input: Add new stock to the inventory system

        - Expected Result: CRM reflects updated inventory in real time
    2. Financial System Integration
       Test Scenario:
        - Input: Record a sale in CRM
        - Expected Result: Sale data synchronized with the Financial System for Revenue Tracking
    3. Middleware Transformation
       Test Scenario:
        - Input: Legacy system sends outdated data format
        - Expected Result: Middleware standardized data format and transmits to modern systems

B. System Performance
    1. API Load Test
       Test Scenario:
        - Input: 1,000 concurrent users accessing inventory data.
        - Expected Result: The system maintains a response time of < 2 seconds.
    2. Stress Test for Scaling
       Test Scenario:
        - Input: Increase input to 10x normal load.
        - Expected Result: The system processes data without crashing.

C. Interoperability
    1. Legacy and Modern System Communication
       Test Scenario:
        - Input: Trigger a financial report request from the legacy system.
        - Expected Result: Middleware bridges communication and generates an accurate report.
    2. Event-Driven Communication via Webhooks
       Test Scenario:
        - Input: Inventory level falls below threshold.
        - Expected Result: Notification triggers the supply management system.

Functional and non-functional requirements in the test scenarios

Functional Requirements:

- Data synchronization and accuracy

- Automated workflows such as automated inventory updates and automatic payroll processing
- Real-time and scheduled data transfer

Non-Functional Requirements:

Non-functional requirements for the integrated system include performance, reliability, security, compatibility, and usability. Performance involves ensuring that API response times are quick and the system is scalable to handle increased loads. Reliability focuses on error handling in middleware and APIs, ensuring the system can recover gracefully from failures. Security addresses protocol compliance and the system's ability to withstand attacks, protecting sensitive data during integration. Compatibility ensures smooth communication between legacy and modern systems, allowing seamless data transfer and functionality. Lastly, usability emphasizes creating intuitive interfaces that facilitate ease of use for end-users, enhancing the overall user experience.

Available Tools for Testing:

- Postman for API Testing
- Selenium for UI Testing
- Middleware log files for Error Detection