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

Integration Strategy and Plan (Week 13)

Step-by-step integration plan detailing how each subsystem will be connected.

1. Define system architecture and integration requirements to see how each subsystem would interact:
    - Identify key endpoints for data exchange
    - Define real-time vs scheduled syncs for data transfer
    - Establish integration priorities (i.e., Inventory-CRM first, Finance later)
    - Document API requirements, data structures and protocols
2. Develop Standardized Integration Interfaces:
    - Build RESTful APIs for real-time data sharing between systems
    - Implement message queues such as Kafka for asynchronous updates
    - Configure webhooks for event-driven communication
3. Implement Data Transformation and Normalization:
    - Use a middleware or ETL (*Extract, Transfer, Load*) tool such as Apache Nifi or Talend to standardize data schemes
    - Validate data formats to prevent discrepancies across systems
    - Create a transformation layer for legacy system data
4. Integrates the components to each of their systems:
    a. Inventory system:
        i. Connect inventory database for the items list, stock level, new stock, and orders.
        ii. Integrate supply management, this includes automating the recording of new stock entries in the database.
        iii. Link order management.
        iv. Add supply chain log to check when stock is replenished
    b. CRM:
        i. Connect transaction records to log sales data from the order management system.
        ii. Build a customer database.
        iii. Integrate customer support.
        iv. Enable marketing automation.
    c. Financial System:

        i. Connect accounting system to monitor expenses from supply management and payroll.
        ii. Integrate payroll to automated payroll processing.
        iii. Enable budgeting to be sync with the inventory database to forecast supply costs.
        iv. Add a financial report generator.
5. Connect Inventory system with CRM with middleware.
6. Connect CRM with financial systems with middleware.
7. Conduct Incremental Integration Module Testing to ensure everything is integrated and running properly:
    - Start with a single subsystem pair (i.e., Inventory-CRM)
    - Perform unit testing for each interface (i.e., Stock Sync API)
    - Validate Data Flow Accuracy and Error Handling
8. Set up Monitoring Tools such as ELK Stack or Grafana to track system performance and Optimize APIs and message queues for latency and load

How legacy systems will be integrated, ensuring compatibility with new technologies.

To ensure the compatibility of the existing system with newer technology, there would be a need for a gradual transition plan in order to keep the system going:

1. Develop APIs that interface with legacy systems, translating between old protocols and modern RESTful endpoints an API, which is referred to as Wrapper API.
2. Middleware platform will be used between the systems to act as the bridge between the systems with newer and older technology
3. Transition legacy system to modern platforms over time, ensuring backward compatibility during migration
4. Break down business functions into discrete services that can be reused across different applications, which is known as Service-oriented architecture (SOA)

Risk assessment and mitigation strategy for the integration process, including potential issues with data migration, interface incompatibility, or system downtimes.

Risk	**Mitigation strategy**
Data migration Data quality is not ready for migration such as duplicates, data being sourced	1. Verify and validate data before and after migration 2. Conduct pilot migrations on small datasets before migrating the whole system

in multiple place, conflicting, and/or incompatible with modern systems	3. Use ETL tools with robust error-handling 4. Data cleansing before migration.
Interface incompatibility APIs of different system not aligning and protocol mismatch happens between subsystems	1. Create wrapper APIs, which are APIs that communicate with ancient systems by converting between outdated protocols and contemporary RESTful endpoints. 2. Test all interfaces with mock data 3. To serve as a link between systems with more recent and older technologies, middleware platforms will be employed.
Security As digital security threats arise, there would be more cyberattacks launched on the systems	1. Develop a durable security scope and requirement to ensure that all updates remain the same 2. Focus on mitigating potential large cyberattacks first before moving on to minor attacks 3. Integrate the API to report and analyze security measures and data
System downtimes Cloud servers may experience downtime, rendering the cloud unavailable for service. This would be a threat to availability of the service.	1. Disaster System Recovery including making backup regularly, backup plan, etc. 2. Employee training to handle and respond to downtime incidents. 3. Conduct monthly maintenance to prevent crashes or at least keep the downtime under control.
Scaling When there are more inputs and quantities that the system will need to maintain, the systems will need to scale it's functionality, else it will overload	1. Upgrade the database and data sources of the system 2. When possible, segment the datasets in order to make the data more feasible
Isolation Systems–particularly legacy systems–often operates in isolation creating data silos that make it difficult to access and integrate data across different systems	• Ensure that the system has a way or source of communicating, therefore giving the indication and possibility of integration