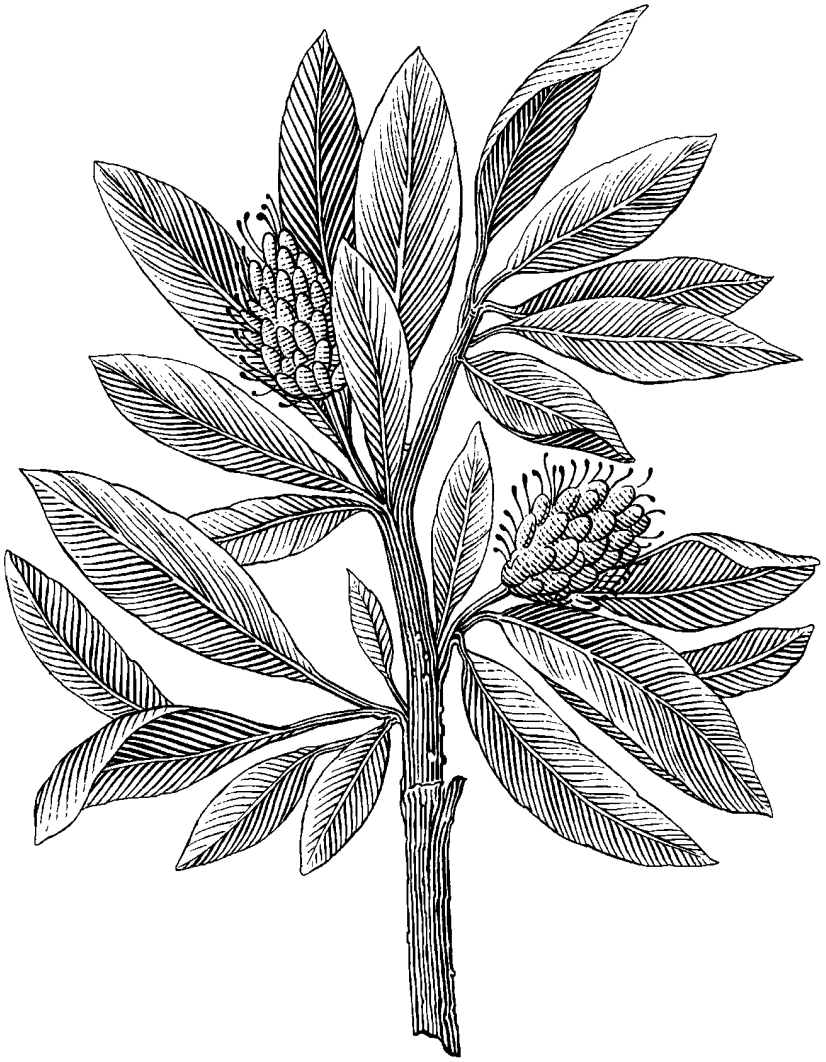
Linnéuniversitetet
Kalmar Växjö

Report

Assignment 3

2DV604



Semester: Spring 2024

Author1: Katarina Simakina
Email es225hi@student.lnu.se

Author2: Zejian Wang
Email zw222bb@student.lnu.se

Task1

Stakeholders for logging and Purposes:

System Administrators and Developers: To monitor systems, diagnose issues, and perform troubleshooting. They are interested in logs related to system errors, performance and the overall condition of the application.
Purpose: for error diagnosis and system improvement, which identifies the causes of system failures or performance issues to find resolution and future prevention. For this purpose, they have access to levels of log for TRACE, DEBUG, ERROR, FATAL.

Security Analysts: To detect and investigate security incidents and potential vulnerabilities. They require access to logs that track access controls, transaction logs, and any anomalies indicating a breach.
Purpose: for security monitoring and incident response, which detects unauthorized access, fraud, and assessing the impact of security incidents. For this purpose, they have access to levels of log for WARN, ERROR, FATAL

Compliance Officers: To ensure the system adheres to legal and regulatory requirements such as GDPR. They need logs that document data access, user consent, and data deletion requests.
Purpose: for regulatory compliance, which demonstrates compliance with data protection regulations through audit trails of data access and manipulation. For this purpose, they have access to levels of logs for INFO, WARN, ERROR.

Customer Support Staff: To resolve user issues effectively. They benefit from logs that capture user actions, transaction failures, and system errors encountered by users.
Purpose: User Support, which provides insights into user issues for effective troubleshooting and customer support. For this purpose, they have access to levels of logs for INFO, WARN, ERROR.

Marketing and Business Analysts: To understand user behavior, product popularity, and system usage patterns for better business decision-making. They are interested in logs that contain user interactions, transaction completions, and feature usage.
Purpose: for business insights, which analyze user behavior and system interactions for marketing insights. For this purpose, they have access to levels of logs for INFO.

Requirements with Architectural Significance Arising:
1. The logging mechanism should allow for structured logging, adjustable log levels, and support outputting logs in an analyzable format, such as JSON;
2. The logging mechanism must not impact system performance. Asynchronous logging or buffering is required to handle high-volume logs without causing delays in user transactions;
3. The system should be able to handle logging at scale, accommodating increases in users, transactions, and interactions without degradation in performance or log integrity;
4. The logging mechanism must incorporate failover and redundancy strategies for log storage, safeguarding against data loss in the event of system failures and ensuring continuous log integrity and availability.

Task2

Architectural strategies are composed of a set of tactics and patterns. ASRs have been defined in task1.

Strategy for Modifiability (ASR1):

Increase Cohesion tactic: Organize logging functionality to minimize overlap and redundancy. This might involve modularizing logging based on system components or log types (error, debug, info, etc).

The Plug-in Pattern is chosen to enhance LnUShop's logging system, allowing for dynamic feature extensions and user-customizable options without altering the core system.

Strategy for Performance (ASR2):

Manage Event Rate tactic: It involves controlling the number of log entries generated, possibly through dynamic log level adjustments or selective logging based on system state or event importance.

Introduce Concurrency tactic: Utilize asynchronous logging to allow the main application threads to proceed without waiting for log entries to be fully processed or written to the storage medium.

Load Balancer Pattern can be adapted for logging by balancing logging load across multiple handlers or processes, especially in distributed systems where logs can be routed to different storage or processing nodes based on current load.

Strategy for Scalability (ASR3):

Efficient Resource Utilization tactic: Optimize the use of system resources (CPU, memory, network bandwidth) to ensure logging processes are lean and do not overwhelm the system as load increases.

Data Partitioning tactic: Partition log data across different storage systems or databases to improve write and read efficiency, facilitating faster access to logs and reducing the impact on performance.

Microservices Pattern involves breaking down the logging system into smaller, independently scalable services. Each service can handle different aspects of logging (e.g., error logs, transaction logs, audit logs), and can be scaled out independently based on demand.

Strategy for Availability (ASR4):

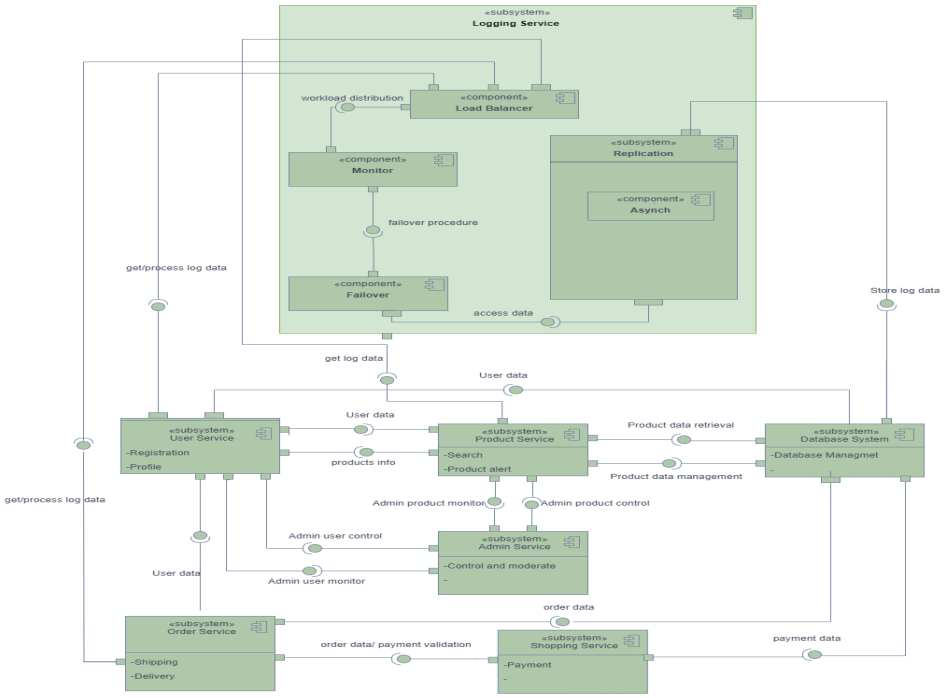
Detect Faults tactic: Utilize system monitors, ping/echo mechanisms, and heartbeat signals to detect and diagnose issues promptly.

Recover from Faults tactic: Implement strategies like redundant spares (hot, warm, and cold), rollback mechanisms for reverting to known good states, and exception handling for graceful degradation or recovery.

Prevent Faults tactic: Engage in removal from service for maintenance, use transactions to ensure ACID properties in log data management, and employ predictive models to foresee and mitigate potential faults.

Redundant Spare Pattern (Active and Passive Redundancy) is vital for ensuring high availability and reliability by having backup components ready to take over immediately or within a minimal delay after a fault is detected.

Outlined strategy: Seamlessly integrating logging across subsystems for consistency; designing logging for easy integration and future expansion; optimizing logging for speed without sacrificing usability, ensuring swift fault detection and recovery for high availability, designing logging to efficiently scale with demand and resources.



Description of Subcomponents of Logging Service:

Load Balancer component: Efficiently distributes logging tasks across available processing resources to prevent overloading, enhancing system performance and scalability.

Interfaces Required:

User Service: Captures user activities and events, tagging logs for enhanced traceability and analysis.

Order Service: Logs the full lifecycle of orders such as encompassing transactions and user interactions.

Product Service: Tracks product-related queries and interactions.

Interface Provided:

Monitoring Component: Transmits logging workload data and performance metrics for real-time monitoring, alerting on anomalies for proactive system management and swift issue resolution.

Monitor Component: It contributes to system Availability by alerting the Failover Component about system's issues.

Requires interface from Load Balancer: to receive updates on the logging load and system performance.

Provides interface to Failover Component: sends alerts, notifications about detected system issues, performance anomalies. It allows the Failover Component to initiate contingency protocols to maintain system availability.

Failover Component: It maintains system availability by the switch to backup systems during failures; improves scalability by efficiently handling increased demands and system strains; collaborates with the Monitor Component for alerts and triggers the Replication Subsystem for seamless data synchronization and failover operations.

Requires interface from Monitor Component: receives alerts on system issues, triggering failover procedures when needed.

Provides interface to Replication Subsystem: coordinates the activation and management of data replication processes during failover. It ensures data is not lost.

Replication component: It maintains data consistency, replicates data during failovers, collaborating with the Database System to ensure log data integrity and availability, particularly during primary database failures.

Requires interface from Failover Component: activating, starting the data replication if failover event is detected.

Provides interface to Database System for synchronizing and storing log data.

Asynchronous Component: Processes log entries asynchronously to prevent interference with the main application's performance. Enhances overall system performance and responsiveness, crucial for user experience, and supports scalability by managing higher volumes of logs as user activities and transactions grow.

The logging system would operate within the existing architecture in the following way:

User Service, Order Service, Product Service, Shopping Service send log data to the logging system. It includes user activities, order processing details, product queries, transactions. Load Balancer within the logging system receives the incoming log data streams. It ensures that the log data is efficiently distributed across the processing units of the logging system. It prevents any single process from being overwhelmed by high volumes of log data, so maintaining system performance and responsiveness. Monitor Component analyzes issues and performance of the logging system. It uses metrics and logs provided by the Load Balancer to detect anomalies. If an issue is detected, it sends an alert to the Failover Component. Failover Component activates the Replication Subsystem which is responsible for ensuring that logs are consistently replicated to maintain data availability. In the event of a system failure, it manages the failover process to minimize disruption and data loss. Asynchronous Component processes log entries without impacting the main application's performance. It is essential for ensuring that the logging system can handle large volumes of data without negatively impacting the user experience on the LnUShop platform. Finally, processed logs are stored in the Database System, which is set up to handle the current logging needs and allow for quick retrieval for analysis, reporting, and troubleshooting. In summary, the logging system works in a way with the existing architecture by collecting logs from various subsystems, distributing the logging load, monitoring system issues, and processing logs asynchronously to maintain overall system performance and reliability.

The technology choices and alternatives for LnUShop's Logging System :

For ASR1.*Technology Choice:* Use a logging framework that supports structured logging in JSON format, such as Serilog for .NET applications or Logback in combination with Jackson for Java-based applications.

Alternative: Fluentd or Logstash can be used to collect logs in various formats and transform them into a structured JSON format before storing.

Rationale: Structured Logging facilitates easier analysis and automation, addressing the requirement for logs to be in an analysable format.

For ASR2. *Technology Choice*: Implement asynchronous logging to ensure that the logging operations don't block the main application threads.

Alternative: Use a dedicated logging service or agent, like Fluentd or Vector, which can buffer and asynchronously process and transmit log data to the storage or analysis backend, minimizing the impact on the application's performance.

Rationale: Asynchronous Logging for Performance: Ensures logging does not impact user experience by delaying transactions or other user actions.

For ASR3. *Technology Choice***:** Elastic Stack (ELK - Elasticsearch, Logstash, Kibana) for log storage and analysis.

Alternative: Use a managed cloud-based logging solution like AWS CloudWatch or Azure Monitor, which can scale automatically with the increase in log data volume and offers integrated analysis and alerting features.

Rationale: Scalable Storage Solutions enables the system to handle an increasing volume of logs, supporting LnUShop's growth and increased transaction rates.

For ASR4.*Technology Choice***:** Implement replication and sharding in Elasticsearch to ensure that log data is replicated across multiple nodes and can survive node failures.

Alternative: Utilize a cloud-based service's built-in redundancy and failover capabilities, such as Amazon Elasticsearch Service or Google Cloud's Operations suite (formerly Stackdriver), which manage the underlying infrastructure's resilience.

Rationale: It guarantees high availability of the logging system, crucial for maintaining operational integrity and compliance with data protection regulations.

Task 3

Step 1: Review Business Goals

The primary business goals for LnUShop include:

Facilitates Academic Transactions: Serve the university community by providing an efficient platform for buying and selling academic materials.

Ensures System Accessibility and Usability: The system should be easy to use for students, employees, and other staff, encouraging widespread adoption.

Maintains High Security and Privacy Standards: Protect user information and transaction data to build trust and comply with regulatory requirements.

Step 2: Review the Architecture

The review focuses on understanding how the architectural components, such as the User Service, Product Service, Order Service, and more, interact with the logging system to ensure comprehensive coverage and integration for logging activities across the platform. The logging system emphasis strategies for modifiability, performance, scalability, and availability. The architecture employs various patterns and tactics such as the Plug-in Pattern, Load Balancer, Microservices, and Redundant Spare Patterns to meet these strategies.

Step 3: Review the Architectural Approaches

Review how the identified architectural approaches address the scenarios:

Modifiability: The introduction of functionalities like cancel/undo commands and modularizing logging functionality indicates a focus on making the system user-friendly and easily modifiable, adhering to the Plugin Pattern for flexibility.

Performance: Strategies like managing event rate and introducing concurrency through asynchronous logging are designed to ensure the system's performance does not suffer due to logging activities, employing the Load Balancer Pattern to distribute logging load effectively.

Scalability: Efficient resource utilization and data partitioning strategies support scalability, with the Microservices Pattern facilitating the independent scaling of logging services as demand increases.

Availability: Fault detection, recovery strategies, and preventive measures aim to maintain high availability, utilizing the Redundant Spare Pattern for logging system reliability.

Step 4: Review the quality attributes utility tree

Quality Attribute	Attribute Refinement	ASR Scenario
Modifiability	Structured Logging	In the event of an unexpected system behavior during a critical operation, a developer can immediately escalate the log level to capture detailed debug information in real-time without deploying new code or stopping the system. (M, L)
Performance	Asynchronous Logging	During high traffic periods, the system continues to operate with optimal response times, while log entries are queued and processed in the background without delaying user transactions. (H, M)
Scalability:	Volume Handling	Ensure the logging system can scale to process double the current peak log volume while maintaining current performance for response times. (H, M)
Availability:	Failover Strategy	The system must automatically replicate log data to a secondary storage location in real-time to prevent any data loss in case the primary storage fails.(H, L)

Step 5: Brainstorm and Prioritize Scenarios

Since we're assuming a lack of direct stakeholder input, we brainstorm potential scenarios that could challenge the architecture's ability to meet its quality attributes, focusing on scalability and availability. These scenarios should reflect possible real-world situations that the system may encounter.

Scalability:

Test the scalability of the logging system when the number of product listings and associated log items grows significantly due to a major expansion in the catalog.

Priority: Medium (M), as it is important for long-term growth but may not be an immediate concern.

Availability:

Verify the logging system's capability to maintain operation and prevent data loss during and after a critical system failure, such as a database crash or server failure.

Priority: High (H), because it is crucial for the logging service to be resilient and ensure data integrity and availability at all times.

Step 6: Analyze the Architectural Approaches

Availability Scenario Analysis:

Architectural Component: Redundant Spare Pattern along with the Replication Subsystem.

Approach: The system utilizes active and passive redundancy to ensure continuous operation. The Replication Subsystem is tasked with real-time data synchronization to secondary storage.

Analysis: The system's resilience during critical failures, such as a database crash, needs to be analyzed. *Considerations include:*

The failover process is automatic and seamless, ensuring zero data loss.

The system recovers quickly from a failure, the impact on performance during recovery is not affected on the system.

The Monitor Component effectively detects failures and triggers the Replication Subsystem without delay.

The goal is to map the high-priority scenarios onto the architectural decisions and identify any gaps or potential improvements. This may result in actions such as upgrading infrastructure, optimizing services, implementing new monitoring tools, or improving the failover and replication mechanisms to better support the architecture's availability.

Step 7: Capture Results

Results from Scalability Scenario Analysis:

Risks Identified: There may be a risk of bottlenecking in the log data processing if the Load Balancer cannot efficiently distribute a dramatically increased load.

Non-Risks: Modularization of the logging system appears to be non-risk due to the well-implemented Microservices Pattern.

Sensitivities: The system is sensitive to the configuration of the Load Balancer and the capability of each microservice to handle an increased load.

Tradeoffs: There is a tradeoff between the complexity of managing multiple logging services and the benefit of independent scaling.

Results from Availability Scenario Analysis:

Risks Identified: A potential risk is the time-to-recovery in the Replication Subsystem could be longer than acceptable if not properly optimized.

Non-Risks: The use of Redundant Spare Patterns and proactive monitoring presents a non-risk for immediate fault detection.

Sensitivities: The architecture is sensitive to the performance of the failover mechanisms and the efficiency of the monitor component.

Tradeoffs: There might be a tradeoff between the cost of maintaining active redundancy and the benefit of high availability.