Content Management System

with SOA Principles

Overview

This Content Management System is a SOAP-based application designed for efficient content creation, management, and analysis. The application facilitates a smooth content workflow where users can create and manage content, while managers oversee the process and gain insights. The architecture follows Service-Oriented Architecture (SOA) principles, ensuring schema centralization, loose coupling, and service reusability.

Services

Entity Services

- ***User Service:*** Manages user information and operations.
- ***Content Service:*** Handles content creation, updates, and queries.
- ***Manager Service:*** Manages admin or moderator information and operations.
- ***Blog Service***: Manages blog-specific content and operations.

Utility Services

- ***Logging Service:*** Records log messages for various events.
- ***Content Insights Service:*** Provides analytics and insights for content performance.

Task Services

- ***Manage Content Service:*** Orchestrates other services to manage the content workflow.

Common Service

- ***Common Service:*** Contains common schemas used across the system.

How to Install

To install and run the Content Management System, follow these steps:

1. Clone the Repository:
    - git clone https://github.com/Md-Arif-Hasan/SOAP_ContentManagement.git
    - cd SOAP_ContentManagement

2. Load Each Service into Your IDE:
    - Open your preferred IDE (e.g., IntelliJ IDEA).
    - Load each folder (SOAP_Users, SOAP_Contents, SOAP_ManageContent etc.) as distinct projects.
3. Run the Projects:
    - For each loaded project, configure the run configurations if necessary.
    - Run each project to start the respective service.

Note: Ensure that the required ports are free on your system before running the services.

Content Management Service Workflow

1. **User Operations:**
    - Users can perform CRUD (Create, Read, Update, Delete) operations on blogs through the Blog Service.
    - Users can read content created by Managers through the Content Service.
    - All user actions are logged using the Logging Service.

2. **Manager Operations:**
    - Managers can perform CRUD operations on specific content that is only accessible to managers, using the Content Service.
    - Managers can review and approve content.
    - Upon content approval:
        - The content status is updated.
        - A log message is generated for the content approval using the Logging Service.

3. **Content Management:**

    - The Manage Content Task Service (implemented in ContentClient.java) orchestrates the content management process.
    - It interacts with other services to manage the content lifecycle.

4. **Logging:**

    - The Logging Service (implemented in LoggerClient.java) records all significant events across the system.
    - Log messages are generated for actions such as content creation, updates, and approvals.

5. **Management Overview:**

    - The Management Service (implemented in ManagerClient.java) provides an interface for managers to oversee the system.
    - Managers can view analytics and insights about content performance through this service.

6. **Content Analysis:**

    - The Content Insights Service analyzes content performance and usage.
    - This data is made available to managers through the Management Service.

7. **Common Service Integration:**
    - All services utilize the Common Service for shared schemas and standardized responses.
    - The Common Service provides elements like 'fault' and 'StatusCodeResponse' that are used across the system.

8. **Service Interaction:**
    - Services communicate with each other using SOAP protocols.
    - The ManageContent service acts as an orchestrator, coordinating between User, Content, Blog, and Logging services as needed.

SOA Principles Implementation

Content Management System is built based on the following SOA principles:

1. Service Contract and Standardization

All WSDL definitions are standardized for consistency and clarity.

- ☐ Entity Services (e.g., User Service, Content Service) are named according to the business entities they represent.
- ☐ Task Services (e.g., Manage Content Service) are named based on the processes they automate.
- ☐ Operations follow a verb + noun naming convention (e.g., createContentRequest, updateUserResponse).
- ☐ Schema centralization is implemented through the Common Service, which is reused across all services.

Standardized Service and Data Representation Layers for your Content Management project:

Schema centralization is implemented in our Content Management System by reusing predefined schemas and defining new schemas only when necessary. This approach maintains a centralized repository for all schemas, ensuring consistency and reuse across the application. Some examples of this implementation include:

1. The 'Content' schema is broken down into 'ContentHeadline' and 'ContentDescription' to allow for more flexible content management and retrieval.

2. 'StatusCode' and 'fault' elements are moved to the CommonService and reused across all services, providing a standardized way of handling responses and errors.

3. The ManageContent service reuses schemas defined in the Content and Manager services, demonstrating the principle of schema reuse.

4. User-related schemas are centralized in the User service and reused by other services that require user information, reducing redundancy and ensuring consistency in user data representation.

This centralized approach to schema management not only promotes consistency across the entire system but also simplifies maintenance and updates. When a schema needs to be modified, it can be done in one place, and the changes will be reflected across all services that use that schema.

2. Loose Coupling

- Services exhibit logic-to-contract coupling with standardized service contracts.
- Entity services (User, Content, Blog) and utility services (Logger, ContentInsights) are designed to minimize functional coupling to external processes.
- The ManageContent service, being task-centric, shows targeted functional coupling to the content management business process.

3. Abstraction

- Most services provide ***concise*** functional abstraction with targeted functionality.
- Technology and programmatic details are abstracted away from service consumers.
- The ManageContent service may have a lower level ***(detailed)*** of abstraction due to its complex workflow.

Example ~ ManageContent Service

Abstraction Level	Description
Functional Abstraction (Content Abstraction)	**Detailed:** It has a lover level functional abstraction due to complex functionalities.

4. Reusability

In our Content Management System, entity services (User, Content, Blog) and utility services (Logger, ContentInsights) are designed as self-contained, reusable components. These services provide core functionalities that can be utilized independently or in combination by other parts of the system.

Task Service and Entity Services

The ManageContent service acts as a task service, orchestrating operations that involve multiple entity and utility services. It interacts with several other services to manage content-related operations.

Example: Content Creation

When a user creates new content:

1. The ManageContent service interacts with the User service to verify user permissions.
2. It then uses the Content service to create and store the new content.
3. The Logger service is called to record the content creation event.
    a.
4. This design allows for efficient reuse of core functionalities. For example, the same User service methods used for content creation can also be used for user authentication in other parts of the system. Similarly, the Logger service can be used to log various events across different services.

5. Autonomy

All services in our Content Management System are self-agnostic, maintaining their own data and business logic. This design ensures independent operation, enhancing modularity and maintainability.

1. We applied Contract Denormalization to split operations like 'GetContent' into more specific ones like 'GetContentHeader' and 'GetContentDescription', allowing for more efficient content retrieval.

2. Core services (User, Content, Blog, Logger, ContentInsights) follow Pure Autonomy, operating independently without relying on other services.

3. The ManageContent service is an exception, exhibiting Shared Autonomy. It orchestrates other services (Manager, Content, Logger) to perform complex tasks like content creation, while maintaining its own business logic for content workflow management.

This approach balances individual service independence with the ability to perform coordinated operations, ensuring flexibility and maintainability in our system architecture.

6. Statelessness

1. All services are designed to be stateless, not retaining information between requests.
2. Common operations (getAll, getById, update, delete, create) are implemented in a stateless manner.
3. Authentication and session management are handled separately to maintain service statelessness.

7. Composability

Our Content Management System demonstrates high composability, allowing services to be combined to create complex functionalities. This is primarily showcased in the ManageContent service, which acts as a composite service by integrating capabilities of other core services.

Examples:

1. Content Creation: The *createContent*() function in ManageContent service composes:

- **createContent()** from Content service for storage
- **logger()** from Logger service for event recording

2. Blog Management: The manage function combines:

- createBlogPost() from Blog service for blog-specific operations
- updateContent() from Content service for general content management

3. Content Analytics: The getContentInsights() function integrates:

- gatherAnalyticsData() from ContentInsights service
- getContentDetails() from Content service

These examples demonstrate how the ManageContent service acts as a composite service, orchestrating functions from multiple services to perform complex operations in the Content Management System.

8. Discoverability

1. Services use standard SOAP interfaces, making them easily discoverable.

2. While not currently published to a service registry, the architecture allows for easy integration of service discovery mechanisms in the future.

This adherence to SOA principles ensures that our Content Management System is modular, flexible, and capable of adapting to changing business requirements while maintaining efficiency and reliability.