Enhancing Piranha CMS for Enterprise Editorial Workflows

Architecture, Workflow and Strategic Design

Software Architecture 2024/2025

Cláudio Teixeira
University of Aveiro, DETI

Hugo Correia, 108215 – hf.correia@ua.pt
André Oliveira, 107637 – andreaoliveira@ua.pt
Alexandre Cotorobai, 107849 – alexandrecotorobai@ua.pt
Joaquim Rosa, 109089 – joaquimvr15@ua.pt

Table of Contents

1. System Analysis & Project Vision

1.1. Current State Analysis

Piranha CMS provides a lightweight content management system built on ASP.NET Core with a clean architectural foundation. However, its current capabilities fall short when evaluated against enterprise-level editorial workflow requirements:

- **Limited Content Lifecycle**: Content progresses through only four basic states (New, Draft, Unpublished, Published), lacking the sophistication required for multi-stage editorial processes;
- **Constrained Permission Model**: The basic role-based permission system cannot support state-specific content permissions needed for complex approval hierarchies;
- **Absent Workflow Engine**: No mechanism exists for orchestrating content through approval pathways based on content type or classification;
- **Minimal Collaboration Features**: Lacks integrated commenting, change tracking, version comparison and task management required for effective team review.

1.2. Strategic Goals & Key Quality Attributes Prioritization

Our vision is to transform Piranha CMS into an enterprise-grade editorial platform that orchestrates complex content approval workflows while maintaining performance excellence. The platform will empower organizations with sophisticated governance requirements by providing strategic goals and key quality attributes.

1.2.1. Strategic Goals

1. Advanced Multi-stage Content States and Permissions
   - Implement a comprehensive state machine for content with configurable workflow stages;
   - Create dynamic permissions that adjust based on content state and user role;
   - Establish conditional state transitions with validation rules.

2. Seamless, Context-aware Content Handoffs
   - Create personalized dashboards for pending review tasks;
   - Enable contextual feedback and commenting system;
   - Establish notifications for workflow transitions.

3. Enterprise-grade Scalability and Performance
   - Optimize for horizontal scalability and high concurrency;
   - Implement multi-level caching strategies;
   - Develop optimistic concurrency control for simultaneous editing.

1.2.2. Key Quality Attributes Prioritization

The evolution of Piranha CMS into an enterprise editorial platform requires meticulous attention to non-functional requirements that will determine its success in demanding organizational environments. These quality attributes shape user adoption, operational efficiency and overall platform viability in content-intensive operations.
The following quality attributes will serve as foundational pillars guiding our architectural decisions.

1. **Modifiability**: Support customizable workflow definitions without technical expertise;
2. **Usability**: Balance sophisticated capabilities with intuitive interfaces for diverse users;
3. **Security**: Implement granular permissions based on state and user role;
4. **Performance**: Ensure editorial productivity with responsive content operations;
5. **Auditability**: Maintain comprehensive records of all content modifications and transitions;
6. **Scalability**: Support growth in both user base and content volume;
7. **Reliability**: Guarantee transactional integrity during all workflow processes.

2. Architectural Design Methodology

2.1. The Strategic Importance of Design Methodology Selection

Selecting an appropriate architectural design methodology is a critical decision that fundamentally shapes the evolution of any software system. This decision establishes not only the framework for technical implementation but also determines how effectively the architecture will align with business objectives and stakeholder needs. In the context of evolving Piranha CMS, this selection carries particular weight as it must facilitate the introduction of complex new capabilities while preserving system stability and performance. The right methodology provides structured guidance for navigating complex architectural decisions, ensures consistent quality across implementations and creates a shared language that bridges technical and business domains. Furthermore, it establishes governance mechanisms that maintain architectural integrity throughout the system's evolution.

2.2. Attribute-Driven Design as Our Selected Methodology

The selection of Attribute-Driven Design (ADD) methodology for evolving Piranha CMS represents a strategic approach that aligns architectural decisions with business goals through systematic quality attribute prioritization. This refined methodology provides several advantages for our project while addressing the specific challenges of enhancing Piranha CMS with new editorial workflow capabilities.

2.2.1. Why Attribute-Driven Design is Optimal

ADD provides a quality-attribute-focused framework that directly supports our strategic vision. By emphasizing quality attributes from the outset, we ensure that critical non-functional requirements like scalability, maintainability and performance guide our architectural decisions rather than being afterthoughts.

The iterative nature of ADD enables progressive evolution of Piranha CMS's architecture, allowing us to introduce complex features like editorial workflows incrementally while continuously validating against stakeholder needs. This approach minimizes disruption to existing functionality while facilitating architectural growth.

ADD's stakeholder integration mechanisms ensure continuous alignment between technical architecture and business objectives. Through structured quality attribute workshops and regular validation sessions, stakeholders maintain visibility into architectural decisions and their business implications.

For an evolving system like Piranha CMS, ADD offers particular advantages in preserving the integrity of existing architectural layers while introducing new capabilities. This evolutionary compatibility reduces risk and maintains system stability throughout the transformation process.

2.2.2. Comparative Analysis with Alternative Methodologies

While other architectural methodologies were considered, ADD provides distinct advantages over alternatives for our specific context.

The Architecture-Centric Design Method (ACDM) offers strong technical rigor but lacks ADD's explicit focus on quality attributes as primary drivers. ACDM tends to emphasize functional decomposition first, with quality attributes considered secondarily. For Piranha CMS evolution, where enhancing specific qualities (flexibility, scalability) is paramount, ADD's quality-first approach provides better alignment with our goals.

TOGAF Architecture Development Method (TOGAF ADM) offers comprehensive enterprise architecture coverage but introduces excessive formality and documentation overhead for our medium-scale project. The ceremony required by TOGAF ADM would extend our timeline considerably without proportional benefits. ADD provides a more targeted approach that maintains rigor while fitting within our 4-week timeline constraints.

2.3. Implementation Process

Our implementation of the ADD methodology unfolds across seven well-defined steps, ensuring a systematic and quality-attribute-driven architectural design:

1. **Reviewing Inputs:** The team began the architectural process by thoroughly reviewing all relevant inputs that would guide the design. This included analyzing the current limitations of Piranha CMS, such as its simplistic content state model (New, Draft, Unpublished, Published), lack of support for complex workflows, and insufficient permission granularity. Additionally, the team gathered business goals, chiefly, the need to support enterprise-grade editorial workflows with advanced governance and collaboration features. Quality attribute scenarios were also defined and prioritized, focusing on scalability, auditability, modifiability, usability and performance. These inputs served as the architectural drivers that framed all subsequent decisions and iterations;
2. **Iteration Goal & Input Selection:** With the foundational understanding in place, the team defined a concrete goal for the first design iteration: enabling multi-stage editorial workflows and role-sensitive permission transitions. The primary inputs selected for this iteration included the business requirements for configurable workflows, the need for traceability of transitions and the enforcement of role-based access at each workflow stage. These inputs were tightly coupled with the quality attributes previously identified, especially auditability, security and modifiability, and would guide the design focus moving forward;
3. **Choosing Elements to Refine:** The next step involved identifying which architectural elements would be refined or newly introduced. The team chose to refine and extend several key system components rather than replace existing ones. These included the content model, which was extended through a WorkflowContentExtension to support workflow metadata without altering core CMS logic. Additionally, a new Workflow domain was introduced, complete with entities for managing state transitions and approvals. The User & Permissions

system was extended through dynamic permission checks embedded in the TransitionRule logic. The team also planned to introduce Audit and Notification subsystems to provide comprehensive compliance and traceability mechanisms. This selection aligned with a strategy of progressive enhancement while preserving backward compatibility;

4. **Choosing Design Concepts:** To address the architectural drivers, the team selected a set of proven design concepts and architectural patterns. Chief among these was the use of a state machine pattern to orchestrate editorial workflow transitions, chosen for its natural fit with content approval processes. Event-driven communication was introduced between services (Workflow, Audit, Notification), ensuring loose coupling and scalability. The team also employed Domain-Driven Design (DDD) principles to clearly delineate bounded contexts: Editorial Workflow (core domain), Content Management and Permissions (supporting domains) and Notifications (generic domain). Role-based permissions were enforced not globally, but contextually, using TransitionRule mappings within the workflow. These concepts were justified and formally recorded in ADRs 1 through 4;
5. **Instantiate Architectural Elements:** Having selected the design approach, the team proceeded to instantiate concrete architectural components. They defined and implemented key domain entities such as WorkflowDefinition, WorkflowInstance, WorkflowState, and TransitionRule. The WorkflowService was introduced to encapsulate the logic for state transitions and validation. The CMS's existing content entities were integrated into the workflow engine via the WorkflowContentExtension, ensuring minimal disruption to core logic. RESTful API endpoints were implemented to allow front-end applications to interact with the new workflow features, and services for audit logging and notification delivery were constructed to support traceability and engagement. Clear interface contracts and service responsibilities were defined to support modular development and testing;
6. **Sketching Views & Recording Design Decisions:** To communicate the evolving architecture and ensure consistency, the team created several architectural views. These included a high-level system architecture diagram showing the interaction between the Content Management Service, User & Permissions, Workflow Service, Audit Service, and Notification Service — being the last three coordinated via a central message queue. Each service was documented in terms of its responsibilities and interactions. In parallel, a set of Architecture Decision Records (ADRs) was compiled to document the rationale behind key choices, such as the use of a state machine, event-driven messaging, and the extension-based integration strategy with Piranha CMS. This documentation supported both technical alignment and stakeholder transparency;
7. **Performing Analysis & Review:** The final step in the iteration cycle was a comprehensive analysis and review of the architectural state. The team validated the design against the original quality attribute requirements. Auditability was confirmed through the implementation of AuditLog, StateChangeRecord, and PermissionCheckTrace. Scalability was addressed through asynchronous communication via a message queue, allowing services to operate independently and handle load efficiently. Modifiability was supported through the DDD

approach and the extension of rather than changes to the CMS's core. Observability tools such as Prometheus, Jaeger, and Grafana were integrated for real-time monitoring and metrics collection. This iteration concluded with a review of stakeholder feedback and verification that the architecture remained aligned with strategic and technical goals.

3. Domain-Driven Design

3.1. The Importance

Domain-Driven Design (DDD) serves as the cornerstone of our architectural approach for evolving Piranha CMS into an enterprise-grade editorial workflow platform. By implementing DDD principles, we've created a design that not only addresses the immediate requirements but establishes a sustainable architecture that can evolve alongside business needs.

The primary advantage of applying DDD in this context is the alignment between technical implementation and business objectives. Editorial workflows represent complex business processes with nuanced rules, approval hierarchies and governance requirements. DDD provides a methodology to capture these complexities accurately in our software model, ensuring the system behaves according to stakeholder expectations.

3.2. Strategic Domain Analysis

Following DDD principles, we've conducted a strategic domain analysis to identify the key domains in our editorial workflow system. We've classified these domains into three categories:

- **Core Domains:** Areas that provide competitive advantage and require custom solutions;
- **Supporting Domains:** Important areas that support core domains but aren't differentiators;
- **Generic Domains:** Common functionality that can be implemented using existing solution.

3.2.1. Core Domain

Editorial Workflow is our core domain, the area that provides the most significant business value and competitive differentiation. This domain encompasses:

- Workflow definition and configuration;
- State management and transitions;
- Approval processes and validation rules.

As a core domain, Editorial Workflow receives the highest investment in terms of development resources, modeling effort and architectural attention. We've implemented this domain with custom-built solutions that precisely capture the business requirements and complex rules of editorial processes. This domain represents our primary competitive advantage and requires domain experts' continuous involvement.

3.2.2. Supporting Domains

Supporting domains are essential to the system's operation but don't represent our primary differentiator. They include:

1. **Content Management:** While fundamental to the CMS, this domain leverages existing Piranha capabilities with extensions to support workflow integration. This domain receives careful attention but builds upon established patterns;
2. **User & Permissions:** Authentication and authorization are critical for secure operations but follow known patterns extended to support the workflow-specific requirements. We implement this domain with a combination of existing framework capabilities and custom extensions;
3. **Audit & History:** Tracking content changes and workflow activities is essential for compliance and traceability but follows established patterns for event recording and history management. This domain is implemented using proven audit trail patterns adapted to our specific workflow needs.

3.2.3. Generic Domain

Notification is a generic domain, providing important functionality but following well-established patterns that don't require custom innovation. This domain manages communication about workflow events to stakeholders through various channels.
For this generic domain, we've employed standard notification patterns, potentially leveraging existing libraries or frameworks, while ensuring proper integration with our core domain events. While crucial for user experience, this domain doesn't require the same level of custom development as our core domain.

3.3. Domain Models

3.3.1. Editorial Workflow Domain Model (Core)

The core domain revolves around a finite state machine that governs content progression through editorial stages such as Draft, In Review, Approved and Published. Key aggregates and entities include:

- **WorkflowDefinition:** Defines the structure and rules of a content workflow, including valid states and transitions;
- **WorkflowInstance:** Represents a specific execution of a workflow applied to a content item;
- **WorkflowState:** Captures the current state of the workflow instance;
- **TransitionRule:** Encodes the conditions under which state transitions are allowed, based on user roles and workflow configuration.

This model is highly configurable to support diverse organizational requirements and complex governance processes.

3.3.2. Content Management Domain Model (Supporting)

This domain reuses the existing Content Management module provided by Piranha CMS, which includes the foundational structures for managing content types such as Pages

and Posts. The native content model is preserved without modification to ensure compatibility and maintainability.

To enable integration with the editorial workflow, a dedicated extension called **WorkflowContentExtension** is introduced. This extension serves as a bridge between content items and the workflow engine. It stores workflow-specific metadata such as the current state of the content, the associated workflow instance ID and timestamps of recent transitions.

By isolating workflow logic within this extension, the model ensures that content remains in sync with workflow progression while keeping the core CMS structures untouched. This design approach promotes a clean separation of concerns and allows the editorial workflow capabilities to evolve independently of the base content model.

3.3.3. User & Permissions Domain Model (Supporting)

This domain leverages the existing User & Permissions modules provided by Piranha CMS, without introducing custom user management logic. Instead of implementing a new permission framework, workflow-specific access control is enforced through the TransitionRule component within the Workflow domain.

Each TransitionRule explicitly defines which roles are authorized to perform a given state transition. This approach allows the workflow engine to delegate permission validation to a consistent rule set, maintaining separation of concerns while ensuring flexible, state-aware authorization.

By relying on predefined roles and associating them with transition logic, the system achieves granular access control without altering the underlying identity or role infrastructure.

3.3.4. Audit & History Domain Model (Supporting)

The notification domain follows standard publish/subscribe patterns and is triggered by workflow events:

- **NotificationEvent:** Represents significant editorial actions (e.g., state transitions, task assignments);
- **NotificationChannel:** Defines the delivery medium such as email, in-app alert, or webhook;
- **UserNotification:** A user-specific notification message with contextual metadata and required action prompts.

Notifications are processed asynchronously through a message queue, ensuring scalability and responsive user engagement.

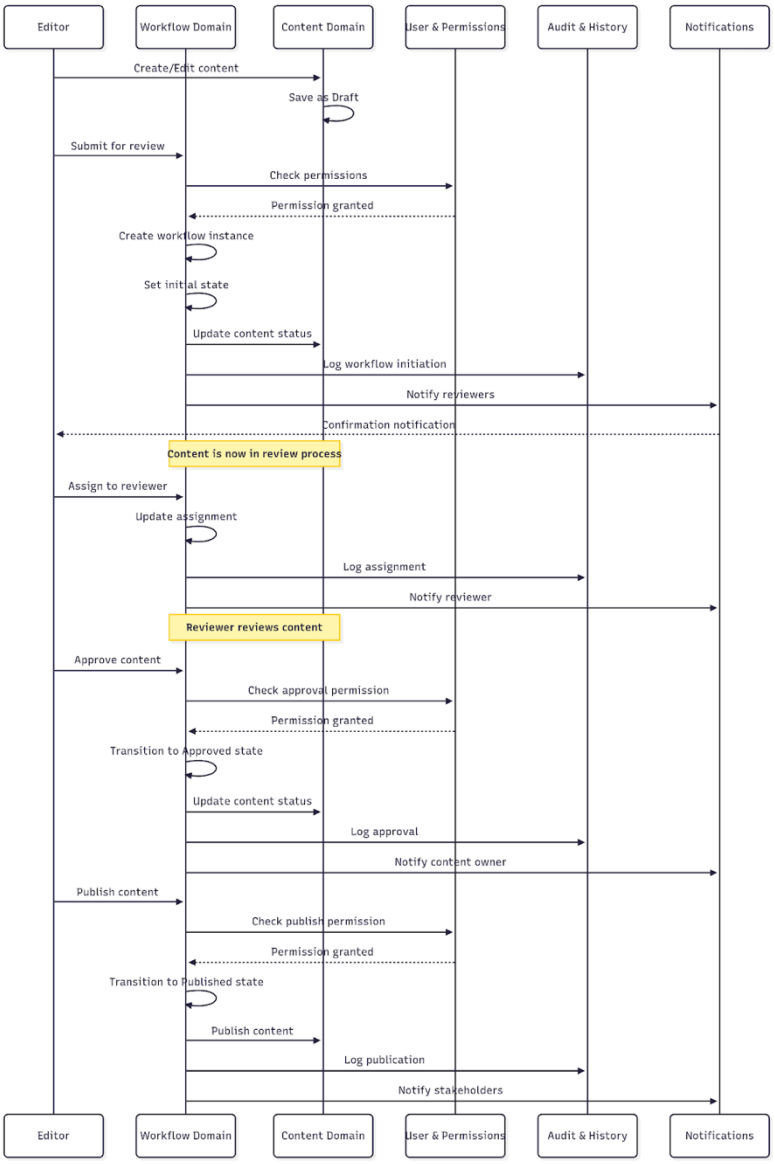
3.3.5. Notification Domain Model (Generic)

This model provides full traceability of user actions and workflow transitions to support governance and regulatory needs:

- **AuditLog:** A persistent record of system actions, including actor, timestamp, action type and content reference;
- **StateChangeRecord:** Specialized logs capturing each workflow state transition for a content item;
- **PermissionCheckTrace:** Stores results of authorization checks performed prior to critical transitions.

This model underpins accountability, enabling detailed compliance reporting and facilitating incident investigation.

3.4. Integration Flow Between Domains



4. Cross-Cutting Concerns

As the system evolves into a modular, enterprise-ready platform, several cross-cutting concerns must be consistently addressed across all modules and services. These concerns are essential for maintaining cohesion, reliability, and operational excellence at a system-wide level.

4.1 Logging and Error Handling

All modules implement unified logging and error-handling mechanisms to ensure traceability and simplify maintenance. This includes standardized log formats, centralized log aggregation, and structured error responses. Regardless of the module, failures and state changes are captured uniformly to support monitoring, debugging, and auditing.

4.2 Security

Security is enforced globally through consistent authentication and authorization policies. Role-based access control, encryption practices and secure API communication are applied across all services. Each module adheres to centralized security policies, ensuring a uniform defense strategy across the platform.

4.3 Performance

To guarantee responsiveness, all components follow shared performance optimization practices. This includes caching where appropriate, minimizing synchronous dependencies and reducing resource contention. Performance metrics are collected from every module to ensure balanced system-wide efficiency.

4.4 Scalability

All modules are designed with scalability in mind, whether through stateless processing, load balancing, or asynchronous communication. The architecture supports horizontal scaling at service boundaries, ensuring that each module can independently scale with increasing demand.

4.5 Observability

Observability is applied uniformly across all modules to provide end-to-end visibility. Metrics, logs and traces are collected from each component and exposed to centralized tools like Prometheus, Jaeger and Grafana. This enables consistent monitoring, alerting and root-cause analysis across the entire platform.

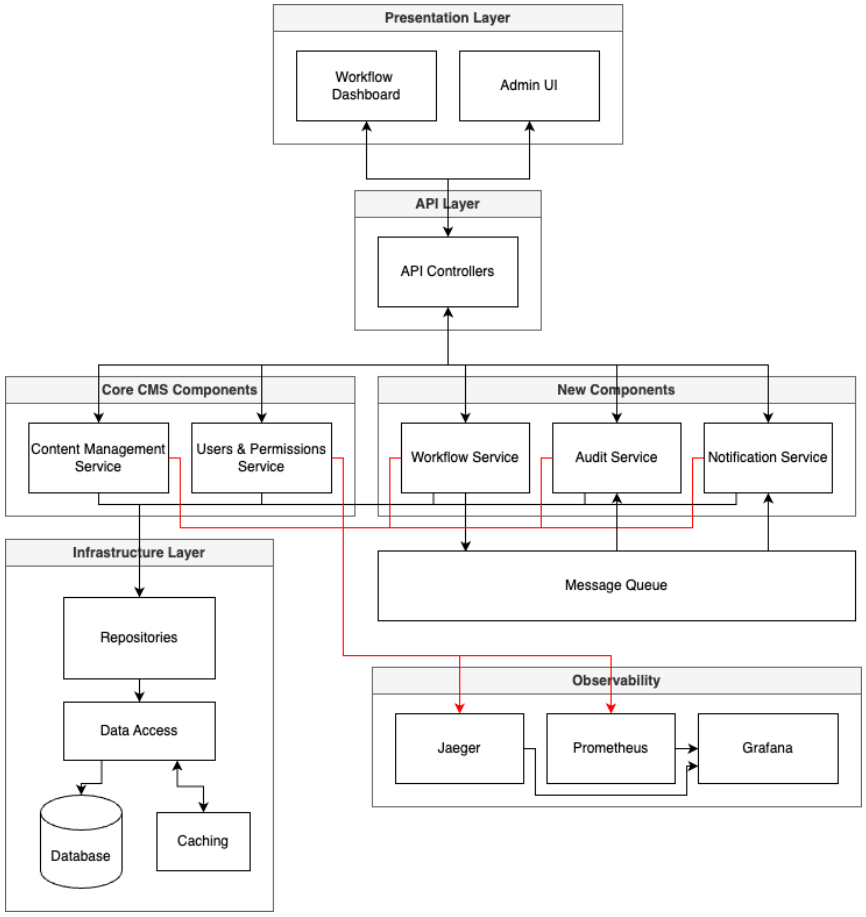
5. Architecture & Roadmap

5.1. Architecture Overview

The architecture adopts a modular, service-oriented design where core CMS functionalities are extended to support enterprise-grade editorial workflows. The existing Content Management Service and Users & Permissions Service are integrated with the new Workflow Service to enforce workflow state transitions and user access control. When content actions are performed (e.g., edits, submissions, approvals), the Content Management Service communicates with the Workflow Service to validate and apply the correct state transitions. Simultaneously, the Users & Permissions Service ensures that each action complies with dynamic, state-aware permission rules defined by the workflow. This coordination guarantees that only authorized users can trigger specific transitions, enabling secure, traceable, and compliant content operations.

Additionally, each time a workflow event occurs, the Workflow Service emits a message to the Message Queue. These messages are asynchronously consumed by the Audit Service, which logs the action for compliance, and the Notification Service, which informs relevant users of required actions or changes. This event-driven architecture promotes decoupling, improves system scalability, and ensures accountability across the editorial process.

5.2 High-Level Architecture



5.3. Architectural Layers and Components

The architecture follows a well-structured layered approach, ensuring clear separation of concerns and supporting scalability and maintainability.

5.3.1. Presentation Layer

Responsible for user interaction and editorial management:

- **Workflow Dashboard:** A dedicated interface for managing editorial workflows, displaying content status, pending tasks, and approval history;
- **Admin UI:** The extended administrative interface of Piranha CMS, now supporting approval workflows within the editorial lifecycle.

5.3.2. API Layer

Provides secure and structured access to system functionalities:

- **API Controllers:** RESTful endpoints that expose content and workflow capabilities to the UI, handling validation, authentication, and authorization.

5.3.3. Core CMS Components

Existing Piranha CMS components that are reused and integrated into the new workflow:

- **Content Management Service:** Manages content entities such as pages and posts, based on the original CMS logic;
- **Users & Permissions Service:** Manages authentication and access control, extended to support state-based workflow permissions.

5.3.4. New Components

New services introduced to support enterprise-grade editorial workflows:

- **Workflow Service:** Manages state machines, workflow instances, transitions, and validation rules;
- **Audit Service:** Logs key events and user actions to ensure traceability and regulatory compliance,
- **Notification Service:** Delivers notifications to users about workflow changes and pending tasks.

5.3.5. Message Queue

An asynchronous communication layer that decouples services and ensures reliable event delivery. It is primarily used by the Workflow Service to emit events, such as state transitions, which are then consumed by the Audit Service for logging and by the Notification Service for sending alerts to relevant users. This design improves system scalability and resilience by enabling loosely coupled, event-driven interactions between services.

5.3.6. Infrastructure Layer

This layer already exists in the current CMS and provides technical support for persistence, performance and data integrity:

- **Repositories:** Encapsulate domain access logic and enforce aggregate boundaries;
- **Data Access:** Manages transactional operations and database queries;
- **Database:** The persistent store for all application data;
- **Caching:** Implements a multi-level strategy to boost read performance using local and distributed caches.

5.3.7. Observability

A cross-cutting layer that provides full visibility into system behavior and performance:

- **Jaeger:** Enables distributed tracing for analyzing service-to-service requests;
- **Prometheus:** Collects and aggregates metrics exposed by each service;
- **Grafana:** Visualizes metrics and traces in real-time dashboards, enabling monitoring and alerting.

5.4. Key Architectural Decisions

Several critical architectural decisions were made to ensure robustness, flexibility and enterprise readiness of the system:

1. **Layered Architecture**
   - We adopted a multi-layered design that promotes modular development and scalability, with clear boundaries between UI, API, services and infrastructure.

2. **CMS Extension Rather than Replacement**
   - Instead of modifying Piranha CMS internals, we extended existing models and services (e.g., content model and permissions), ensuring backward compatibility and upgradability.

3. **State Machine for Workflow Orchestration**
   - The editorial workflow is modeled as a finite state machine, providing predictable state transitions, business rule enforcement and traceability.

4. **Asynchronous Communication via Message Queue**
   - The Workflow Service emits events like approvals and transitions, which are consumed by the Audit and Notification services. This decouples components and improves scalability and resilience through event-driven communication.

5. **Decentralized Observability Implementation**
   - Each module is individually instrumented using OpenTelemetry, exposing its own metrics and traces. These are centrally collected via Prometheus

and Jaeger and visualized in Grafana, providing full observability without coupling the services.

6. **Domain-Driven Design (DDD) Approach**
   - The system is organized around well-defined bounded contexts and aggregates, ensuring that business rules are encapsulated in the domain layer, facilitating testability and maintainability.

5.5. Implementation Roadmap

The implementation roadmap outlines a focused four-week plan to extend Piranha CMS with advanced editorial workflows and content approval capabilities. The goal is to deliver a fully integrated solution that enhances content collaboration, governance and traceability without disrupting the existing CMS experience. Each phase builds on the previous one, progressing from foundational models and core services to user interface development, system integration and performance optimization. By the end of this roadmap, we aim to provide a robust, enterprise-ready workflow module complete with state management, role-based permissions and audit logging for full traceability.

5.5.1. Week 1: Project Setup and Core Domain Modeling

- Set up project structure and architectural foundation;
- Develop core domain models for workflows and content integration.

5.5.2. Week 2: Workflow Engine and Core Application Services

- Develop the workflow state machine engine;
- Implement state transition validation and execution;
- Build core application services (Workflow Service, extended Content Service);
- Set up initial API controllers for workflow operations;

5.5.3. Week 3: Permission System and Audit Features

- Integrate enhanced permission system for workflow roles;
- Develop workflow dashboard components;
- Build audit logging and traceability features.

5.5.4. Week 4: Dashboard, Documentation, and Templates

- Complete UI integration with existing Piranha CMS;
- Finalize documentation and configuration guides;
- Create sample workflow templates for demonstration.

6. Architecture Decision Records (ADR)

6.1. ADR 1: Workflow Domain Modeling Approach

Context
We needed to determine the appropriate modeling approach for the core Editorial Workflow domain, considering how to represent workflow definitions, states, transitions and instances.

Decision
We decided to model the workflow domain using a rich domain model with entities like WorkflowDefinition, WorkflowInstance, WorkflowState and TransitionRule.

Rationale

- The model needed to support flexible workflow definition with arbitrary states and transitions;
- A rich domain model makes complex validation rules and business logic easier to express;
- The separation between workflow definitions and instances allows for reuse of common workflows.

Consequences

- Positive: Highly flexible workflow model that can support varying organizational needs;
- Positive: Clear separation of concerns between workflow definition and execution;
- Negative: More complex model compared to a simpler linear workflow approach;
- Negative: Potential performance impact from the relational complexity;
- Negative: Higher implementation effort required for the complete model.

6.2. ADR 2: State Machine Pattern for Workflow Engine

Context
The editorial workflow system needs to manage complex state transitions with validation rules.

Decision
We decided to implement a state machine pattern for the workflow engine instead of conditional logic or a rules engine.

Rationale

- State machines provide a natural model for workflow transitions;
- The pattern enforces that only valid state transitions are permitted;
- It simplifies the representation of complex workflows visually and in code;
- The approach is extensible to support future workflow requirements;
- The model clearly shows the relationship between states and transitions.

Consequences

- Positive: Clear, maintainable model of workflow states and transitions;
- Positive: Strong validation of permitted operations based on current state;
- Positive: Simplified visualization of workflows for administrators;
- Positive: Natural fit for the domain problem (editorial workflow management);
- Negative: Less flexibility compared to a full rules engine;
- Negative: Potential complexity when implementing conditional transitions;
- Negative: Additional effort required to build workflow visualization tools.

6.3. ADR 3: Event-Driven Communication Between Domains

Context

The workflow system needs to coordinate actions across multiple domains (workflow, notifications and audit) while maintaining loose coupling.

Decision

We decided to implement an event-driven communication pattern between domains rather than direct method calls, as illustrated in the integration flow sequence diagram.

Rationale

- Loose coupling between domains improves maintainability and allows for independent evolution;
- Events provide a natural audit trail of system actions;
- The approach supports asynchronous processing for better scalability;
- Events make it easier to add new features without modifying existing code.

Consequences

- Positive: Domains can evolve independently with minimal impact on each other;
- Positive: Improved scalability through asynchronous processing;
- Positive: Natural audit trail through event history;
- Positive: Easier to extend with new functionality in response to events;
- Negative: Increased complexity in tracking the flow of operations;
- Negative: Potential eventual consistency challenges;
- Negative: Learning curve for developers used to synchronous programming models.

6.4. ADR 4: Role-Based Permission via Transition Rules

Context

Piranha CMS's built-in permission model is static and not sufficient for complex editorial workflows where access must vary depending on the content's workflow state. We needed a mechanism to support dynamic, state-aware permissions.

Decision

We delegated permission checks to the TransitionRule entity within the Workflow domain. Each transition explicitly defines the roles that are allowed to trigger it, enabling role-based access at the state transition level.

Rationale

- Integrates cleanly with the existing CMS role infrastructure;
- Enables fine-grained, dynamic permissions tied to workflow states;
- Maintains a clear separation between workflow logic and user management.

Consequences

- Positive: Flexible and extensible governance model based on real-world editorial processes;
- Positive: Simple integration with existing roles and user structures;
- Negative: Adds configuration overhead, especially in workflows with many transitions and roles;
- Negative: Potential for misconfiguration without proper tooling or validation mechanisms.

6.5. ADR 5: Extending vs. Replacing Piranha CMS Content Model

Context

We needed to determine whether to extend the existing Piranha CMS content model or replace it with a new model that directly incorporates workflow states.

Decision

We decided to extend the existing content model through a complementary WorkflowContentExtension rather than modifying the core content classes.

Rationale

- Extending preserves compatibility with existing code and plugins;
- Reduces risk during implementation and deployment;
- Allows for gradual adoption of workflow features;
- Maintains separation between content and workflow concerns;
- Follows the open/closed principle (open for extension, closed for modification).

Consequences

- Positive: Backwards compatibility with existing content and code;
- Positive: Lower risk implementation approach;
- Positive: Clear separation of concerns;
- Positive: Easier upgrade path for existing Piranha CMS users;
- Negative: Additional joins required when querying content with workflow state;
- Negative: Slightly more complex data model;
- Negative: Potential performance impact from additional queries.

6.6. ADR 6: Comprehensive Audit and History Tracking

Context

Enterprise editorial workflows require comprehensive auditing for accountability, compliance and debugging purposes.

Decision

We decided to implement a detailed audit and history tracking system with specialized entities for different types of audit events.

Rationale

- Regulatory compliance often requires detailed audit trails for content changes;
- Audit records support accountability in the editorial process;
- Specialized audit entities provide domain-specific context for events;
- The audit structure supports both general system events and workflow-specific events.

Consequences

- Positive: Comprehensive traceability of all system actions;
- Positive: Support for regulatory compliance requirements;
- Positive: Ability to reconstruct the sequence of events for troubleshooting;
- Negative: Performance impact from extensive logging;
- Negative: Storage implications for long-term audit retention;
- Negative: Potential complexity in querying and analyzing audit data.