**Journal of Computer Science, Information Technology and Telecommunication Engineering (JCoSITTE)**
Vol. 7, No. 1, March 2026, pp. 1101~1108
ISSN: 2721-3838, DOI: 10.30596/jcositte.v7i1.29362 

# Building a Modular Creative Economy Assessment System with a Domain-Driven Design Approach and Agile Development

**Zulfiandri[1], Sarip Hidayatuloh[2], Muhammad Asyraf Faiz Kamil[3]**
[1,2,3] Department of Information Systems, Faculty of Science and Technology, UIN Syarif Hidayatullah Jakarta, Indonesia

**ABSTRACT**

The creative economy assessment program involves high domain complexity and multi-stakeholder workflows that conventional content management systems fail to accommodate adequately. This paper presents the development of a Domain-Driven Design (DDD)-based Content Management System (CMS) for the creative economy assessment program, integrated with Agile (Scrum) development methodology. The study employed strategic DDD design including bounded context identification, ubiquitous language definition, and context mapping, followed by tactical design comprising entity, value object, aggregate, domain service, domain event, and repository pattern specifications. A Headless CMS architecture using Laravel 10 as the backend API and Next.js 15 as the frontend was implemented across four sprint cycles, resulting in a modular, maintainable system with clear domain separation. The system successfully modeled four bounded contexts (Borang Management, Assessment, User Management, Notification) and implemented Clean Architecture layering validated through Deptrac. Results demonstrate that the DDD approach effectively addresses creative economy assessment domain complexity, enabling independent development and testing of each bounded context while maintaining business rule integrity through aggregate design and domain events.

**Keyword: Agile Development; Bounded Context; Content Management System; Domain-Driven Design; Scrum**.



***Corresponding Author:***
Zulfiandri,
Department of Information Systems
UIN Syarif Hidayatullah Jakarta
Jl. Ir H. Juanda No.95, Kec. Ciputat Timur, Kota Tangerang Selatan, Banten 15412, Indonesia.
Email : zulfiandri@uinjkt.ac.id

***Article history:***
Received Feb 26, 2026
Revised Mar 10, 2026
Accepted Mar 21, 2026

## 1. INTRODUCTION

Enterprise-level content management systems operating in high-complexity domains face significant architectural challenges, particularly when the domain involves multiple stakeholders, dynamic business rules, and evolving regulatory requirements (Hassan & Eassa, 2024). Traditional monolithic CMS architectures with tightly-coupled Model-View-Controller (MVC) patterns struggle to accommodate rapid change while maintaining codebase maintainability (Ramalingam, 2016).

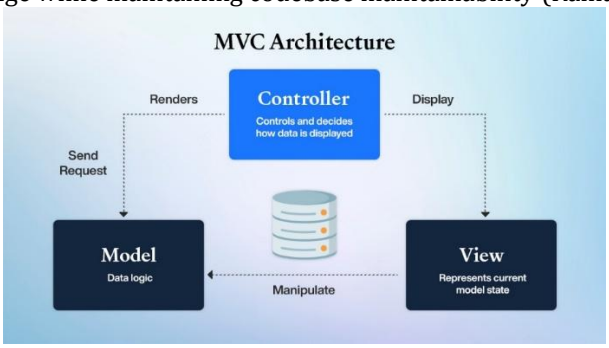


Figure 1. Monolithic MVC Architecture

The creative economy assessment program managed by the Creative Economy Institute is a prime example of this challenge. The program involves multiple actors such as local government representatives, creative economy practitioners, and national reviewers each interacting through

complex assessment workflows with multi-level approval mechanisms, audit trail requirements, and dynamic form structures per creative economy sub-sector.

Domain-Driven Design (DDD) offers a structured approach to managing such complexity by centering software design around the core business domain (Vernon, 2013). DDD's strategic and tactical design patterns including bounded contexts, ubiquitous language, aggregates, and domain events provide architectural guidance that aligns code structure with business processes (Jaiswal & Agrawal, 2024). When combined with Agile (Scrum) methodology, DDD enables iterative domain refinement through sprint cycles, ensuring the model evolves alongside stakeholder feedback (Ömerand et al., 2018).

Prior research has explored DDD in enterprise contexts, but applications to government creative economy assessment systems remain limited. Aziz et al. demonstrated DDD's benefits for maintainability in fintech applications but noted performance concerns under high load (Rama Sakti Hafidz Fadhilah Aziz et al., 2024). Hassan and Eassa confirmed Headless CMS architectural advantages over traditional CMS in throughput and flexibility (Hassan & Eassa, 2024). Rahadinata et al. implemented GitLab CI/CD with Docker for deployment automation without DDD-based domain modeling (Kevin Julian Rahadinata et al., 2025). None of these studies addressed the integration of DDD with Agile methodology in a government creative economy assessment context.

This paper addresses the gap by presenting the complete DDD implementation for the creative economy assessment CMS, covering: (1) strategic design including bounded context identification and context mapping, (2) tactical design including entity, aggregate, and domain event specifications, (3) Agile sprint execution integrating DDD artifacts, and (4) the resulting system architecture and application demonstration. The novelty lies in the application of comprehensive DDD to a multi-stakeholder creative economy assessment domain, validated through working software across four sprint cycles.

## 2. RESEARCH METHOD

This research employs a design science methodology, combining participatory observation within the development environment with semi-structured interviews and systematic literature study.

### 2.1 System Analysis

The existing system was analyzed using the PIECES framework (Performance, Information, Economics, Control, Efficiency, Service). The analysis revealed that the legacy system used a monolithic Laravel MVC architecture where business logic was distributed across controllers, models, and service classes without clear domain separation (Chakiri & El Houda Touti, 2025). The assessment revealed four primary deficiencies: (1) tight coupling between presentation, application, and domain layers (Jain, 2021); (2) absence of explicit domain boundaries (Özkan et al., 2023); (3) manual deployment with no CI/CD integration (Laukkanen et al., 2016); and (4) difficulty scaling due to monolithic structure (Evans, 2004).

### 2.2 DDD Modeling Process

Domain modeling followed the DDD methodology in two phases: strategic design and tactical design. Strategic design employed Event Storming workshops and User Story Mapping to identify domain events, aggregate boundaries, and bounded context definitions. The creative economy assessment domain was analyzed through domain expert interviews, identifying the core business processes: form (borang) management, assessment review, user authentication, and notification delivery.

Context mapping was conducted to define integration patterns between bounded contexts. The resulting context map identified four bounded contexts with distinct integration strategies: Shared Kernel for User Management, Customer-Supplier pattern between Borang Management and Assessment, and Publisher-Subscriber for event-driven notification.

Tactical design specified building blocks within each bounded context, including entities with lifecycle invariants, value objects defining domain constraints, aggregate boundaries enforcing consistency rules, domain services encapsulating cross-entity logic, domain events representing state transitions, and repository interfaces abstracting data access.

### 2.3 Agile Development Process

System development followed the Scrum framework across four two-to-four-week sprints. Sprint planning integrated DDD artifacts directly: domain model designs were the Definition of Ready for sprint

items, and the Definition of Done required domain model validation alongside functional testing. The sprint structure was:

Sprint 1 (Domain & Architecture Foundations) focused on establishing the strategic DDD foundation. The team conducted Event Storming sessions with domain experts to identify bounded context boundaries, defined the Ubiquitous Language dictionary, and produced a high-level Context Map. The output was a validated domain model blueprint and a system landscape architecture diagram agreed upon by all stakeholders before any code was written.

Sprint 2 (Architecture Design Detail & Database Design) translated the strategic model into concrete technical specifications. Tactical DDD building blocks such as entities, value objects, aggregates, domain services, and repositories were specified per bounded context. The Clean Architecture layering structure was defined and enforced via Deptrac configuration. Database schema was designed per bounded context using PostgreSQL, with API contracts documented in OpenAPI format.

Sprint 3 (Implementation of Core Flows) was the primary development sprint, where the designed domain model was realized in code. Domain entities and value objects were implemented as first-class PHP classes in the Domain Layer, use cases were built in the Application Layer orchestrating domain operations, and the event-driven integration between bounded contexts was wired through Laravel's event and queue system. Core user stories such as borang creation, draft saving, submission, and review workflow were completed and validated through sprint review with the product owner.

Sprint 4 (Deployment & Evaluation) automated the delivery process and assessed the resulting system. The GitLab CI/CD pipeline was configured with Docker containerization and lightweight Kubernetes (K3s) orchestration, enabling automated build and rolling deployment on every merge to the main branch. An end-to-end demo scenario was executed against the live deployment, and DORA Metrics alongside DevEx dimensions were collected to evaluate the engineering quality of the full development process.

### 2.4 Technology Stack

The system was implemented using Headless CMS architecture: Laravel 10 (PHP 8.1) as the backend API with PostgreSQL 14 as the relational database, Redis 7 for caching and message queuing, and Next.js 15 with TypeScript as the frontend application. Docker containerization and Kubernetes (K3s) orchestration were used for deployment. Architecture validation was performed using Deptrac for enforcing Clean Architecture layering rules.

Table 1. Technology Stack

| Component | Technology | Responsibility | Communication Protocol |
|---|---|---|---|
| Backend API | Laravel 10 + PHP 8.1 | Business logic, domain model, data persistence | RESTful API |
| Frontend Application | Next.js 15 + React 18 + TypeScript | User interface, presentation logic, client-side state | HTTP/HTTPS |
| Database | PostgreSQL 14+ | Data persistence, relational storage | SQL Protocol |
| Cache Layer | Redis 7+ | Session storage, query caching, rate limiting | Redis Protocol |
| Message Queue | Redis Queue | Asynchronous job processing, event handling | Redis Protocol |
| Object Storage | MinIO (S3-compatible) | File uploads, attachments, documents | S3 API |

## 3. RESULTS AND DISCUSSION

### A. Strategic Design Results

Event Storming identified 47 domain events across the creative economy assessment workflow, which were subsequently grouped into four bounded contexts. The Borang Management Context serves as the core domain, handling the complete lifecycle of assessment forms from template creation through submission, revision, and archival. The Assessment Context manages reviewer assignment, scoring, and decision recording. User Management Context provides shared authentication and role-based authorization. Notification Context delivers event-driven communications through email channels.

The context map established integration patterns: User Management acts as a Shared Kernel referenced by all contexts for identity and authorization. Borang Management serves as Supplier to Assessment as Customer, providing submitted form data through well-defined BorangDTO contracts. Both contexts publish domain events consumed by Notification through a Publisher-Subscriber pattern. Anti-Corruption Layers were defined at external service boundaries, particularly between Notification Context and the email gateway.

The Ubiquitous Language Dictionary defined 23 domain terms with precise definitions, usage examples, and bounded context scope, ensuring consistent vocabulary between developers and domain experts across all development artifacts, documentation, and code.

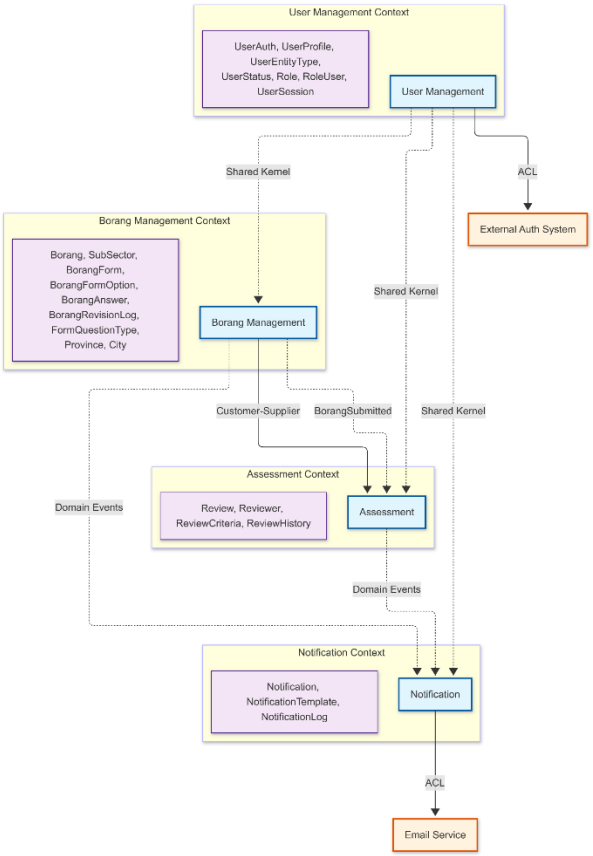


Figure 2. Bounded Context

### B. Tactical Design Results

Tactical design specified building blocks for all four bounded contexts. The Borang Aggregate, centered on the Borang entity as aggregate root, enforces a strict state machine with six valid statuses: DRAFT → SUBMITTED → UNDER_REVIEW → {APPROVED, NEEDS_REVISION, REJECTED} → ARCHIVED. The aggregate invariant ensures data can only be modified in DRAFT or NEEDS_REVISION states, and each revision request must generate a BorangRevisionLog entry with a documented reason.

Value objects enforce domain constraints as first-class types: StatusBorang validates state transition legality, Wilayah validates Indonesian province and city code formats, Score calculates assessment categories from numerical values, and ContactInfo validates email and phone formats. These

value objects are immutable, defined entirely by their attribute values, and contain domain logic preventing invalid states.

Five domain events were specified: BorangCreated, BorangSubmitted, ReviewCompleted, RevisionRequested, and BorangStatusChanged. Each event carries typed payloads and has defined subscriber contexts, enabling loose coupling between bounded contexts through asynchronous event-driven communication via Laravel's queue system with Redis.

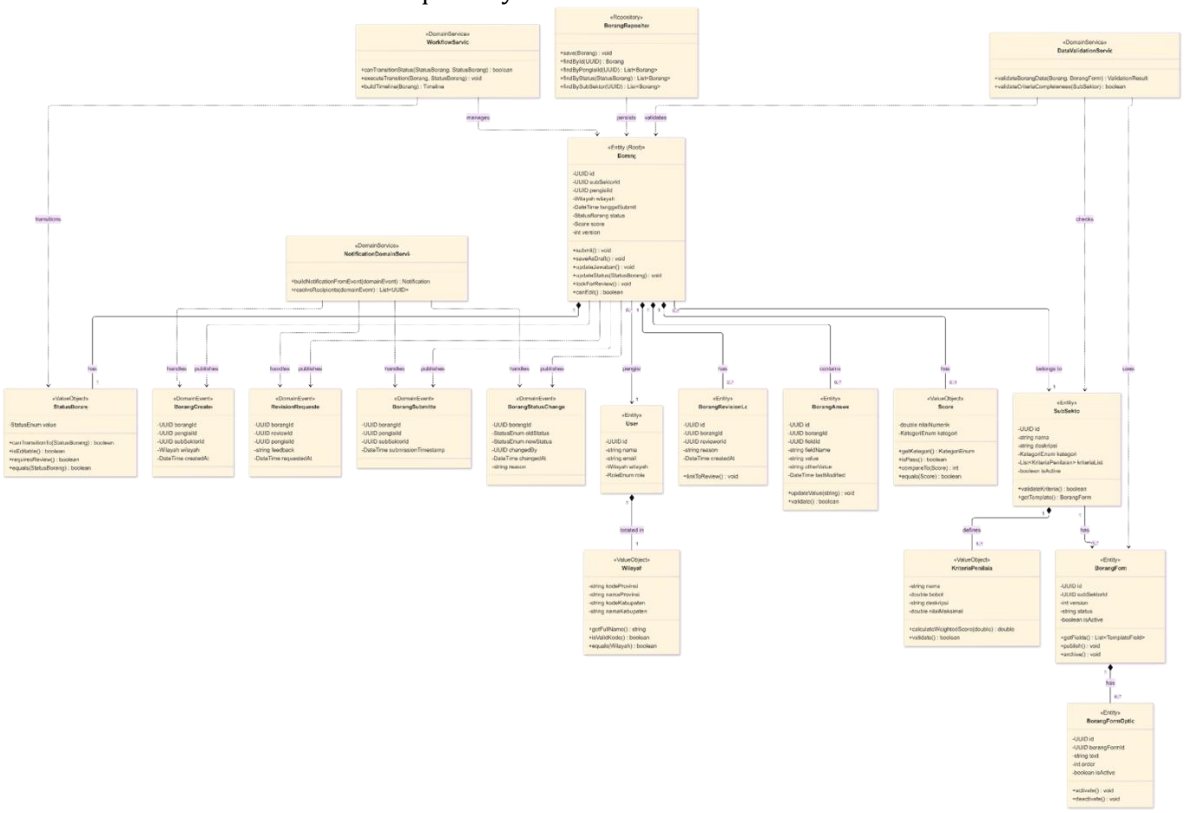

Figure 3. Domain Model Class Diagram

Repository interfaces were defined for each aggregate root (BorangRepository, ReviewRepository, BorangTemplateRepository, SubSektorRepository, UserRepository, NotificationRepository), providing domain-focused query methods while abstracting PostgreSQL implementation details from the domain layer.

**C. Architecture and Implementation**

Clean Architecture was implemented with four strict layers: Presentation Layer (Controllers, DTOs, API routes), Application Layer (Use Cases, Command/Query handlers), Domain Layer (Entities, Value Objects, Aggregates, Domain Services, Domain Events, Repository Interfaces), and Infrastructure Layer (Repository Implementations, Database Access, External API Clients, Event Publishers). Dependency rules were enforced: Domain Layer has no external dependencies; Infrastructure Layer implements Domain interfaces via Dependency Inversion; Application Layer orchestrates domain operations; Presentation Layer only communicates with Application Layer.

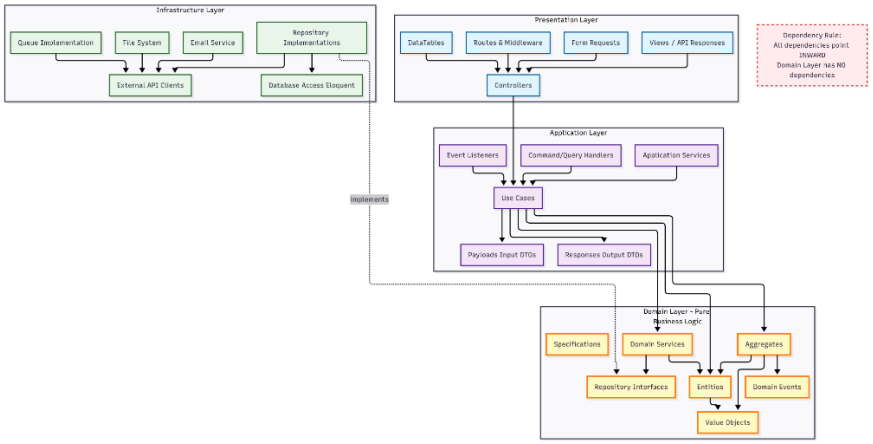


Figure 4. Clean Architecture Layering

Deptrac configuration validated these dependency rules as part of the CI pipeline, preventing architectural violations from entering the main branch. The module structure maps directly to bounded contexts: src/Core/BorangManagement/, src/Core/Assessment/, src/Core/UserManagement/, src/Core/Notification/, and src/Core/Reporting/.

The database schema was designed per bounded context with PostgreSQL. Each bounded context uses separate table namespaces, soft deletion (deletedAt), public identifiers (xid field), JSONB columns for flexible metadata and form snapshots, and JSON columns for audit fields (createdBy, modifiedBy). The formSnapshot field in the borang table captures an immutable snapshot of the form template at submission time, preserving data integrity even as template versions evolve.

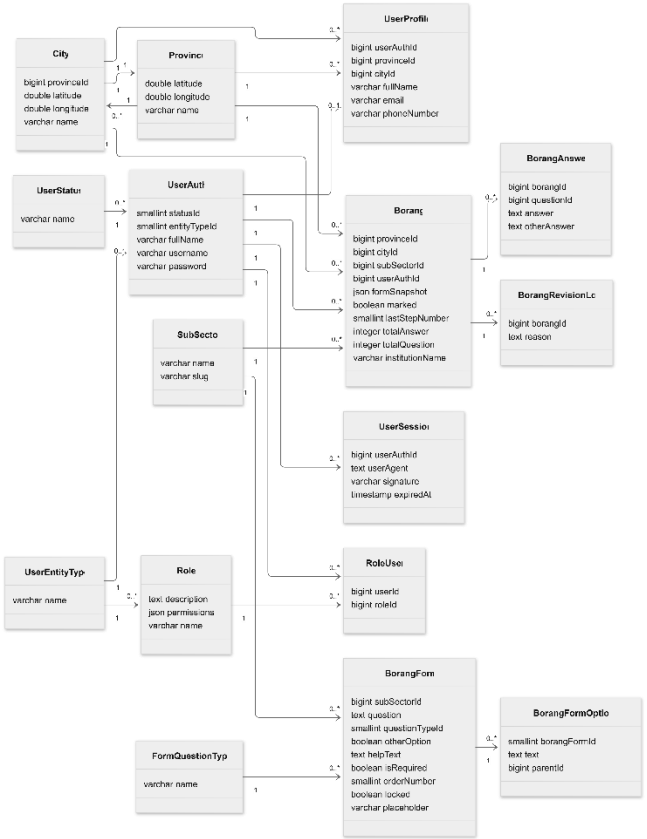


Figure 5. Mapping Class Diagram

**D. Development Results**

The Borang Management module enables creative economy practitioners to select their sub-sector (17 sub-sectors including culinary, fashion, digital applications), dynamically loads the corresponding question bank, supports draft saving with progress tracking, and implements submit workflow triggering the BorangSubmitted domain event.

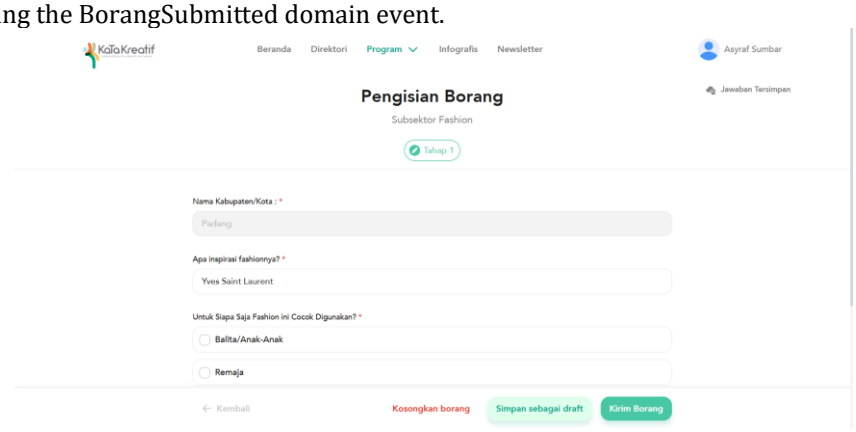


Figure 6. Borang Assessment Form

The Assessment module enables reviewers to view submitted forms with full audit history, provide structured feedback, and update assessment status following the defined workflow. Role-Based Access Control (RBAC) implemented through Spatie Laravel Permission ensures reviewers cannot assess their own submitted forms and cannot access forms outside their assigned scope.

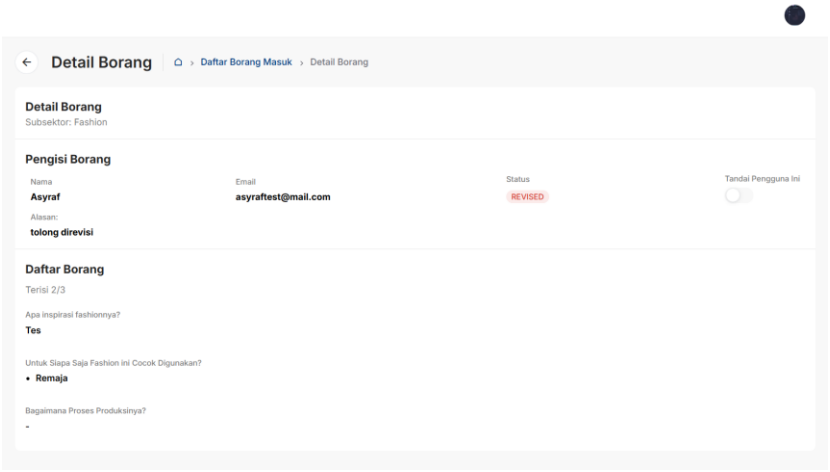


Figure 7. Review Submitted Borang

The User Management module supports four user entity types with distinct permission sets, managing the complex multi-stakeholder structure where local government representatives, creative economy practitioners, national reviewers, and system administrators each have different system access profiles.

## 4. CONCLUSION

This paper demonstrated the successful application of Domain-Driven Design integrated with Agile (Scrum) methodology for developing a Content Management System for the creative economy assessment program. The strategic DDD design identified four bounded contexts with appropriate integration patterns, while tactical design specified building blocks that accurately model the domain complexity including multi-level form workflows, role-based assessment processes, and audit trail requirements.

The resulting Headless CMS architecture using Laravel 10 and Next.js 15 with Clean Architecture layering provides a modular, maintainable foundation for continued system evolution. The Deptrac-validated architecture enforcement ensures that domain layer purity is maintained as the

system grows. The working application demonstration confirms the system's functional completeness across the core creative economy assessment workflow.

Future work should focus on completing the planned Assessment and Notification bounded contexts to full production readiness, implementing comprehensive unit and integration test coverage for all domain aggregates, and extending the RBAC model to accommodate provincial-level administrative hierarchies. The DDD approach demonstrated here provides a replicable blueprint for modernizing other government domain-complex applications in Indonesia's digital government transformation program (SPBE).

**REFERENCES**

Chakiri, H., & El Houda Touti, N. (2025). From Monolith to Microservices: Modernizing a National E-Government Platform: A Case Study of the Moroccan Civil Registry System. *2025 IEEE 8th Congress on Information Science and Technology (CiSt)*, 171–177. https://doi.org/10.1109/CiSt65886.2025.11224226

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Hassan, S. A. Z., & Eassa, A. (2024). Comparing Performance between Headless CMS and Traditional CMS. *International Journal of Business Information Systems*, *1*(1). https://doi.org/10.1504/IJBIS.2024.10066810

Jain, V. (2021). Headless CMS and the Decoupled Frontend Architecture. *International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences*, *9*, 1–5. https://doi.org/10.5281/zenodo.14752509

Jaiswal, S. K., & Agrawal, R. (2024). Domain-Driven Design (DDD)- Bridging the Gap between Business Requirements and Object-Oriented Modeling. *International Journal of Innovative Research in Engineering and Management*, *11*(2), 79–83. https://doi.org/10.55524/ijirem.2024.11.2.16

Kevin Julian Rahadinata, Fedelis Brian Putra Prakasa, & Theresia Devi Indriasari. (2025). Pembangunan Website Manajemen Hunian Sewa Dengan Implementasi Continuous Integration dan Continuous Delivery Berbasis Docker Container. *Jurnal Informatika Atma Jogja*, *6*(2), 13–24. https://doi.org/10.24002/jiaj.v6i2.12315

Laukkanen, E., Lehtinen, T. O. A., Itkonen, J., Paasivaara, M., & Lassenius, C. (2016). Bottom-up adoption of continuous delivery in a stage-gate managed software organization. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10.

Ömerand, U., Hauder, M., Kleehaus, M., Schimpfle, C., & Matthes, F. (2018). Supporting Large-Scale Agile Development with Domain-Driven Design. In X. and A. A. Garbajosa Juan and Wang (Ed.), *Agile Processes in Software Engineering and Extreme Programming* (pp. 232–247). Springer International Publishing. https://doi.org/10.1007/978-3-319-91602-6_16

Özkan, O., Babur, Ö., & Brand, M. van den. (2023). *Domain-Driven Design in Software Development: A Systematic Literature Review on Implementation, Challenges, and Effectiveness*.

Rama Sakti Hafidz Fadhilah Aziz, Irwan A. Kautsar, & Sumarno. (2024). Implementasi Domain Driven Design dan Clean Architecture dalam Pengembangan Web Service Aplikasi Alifarm Digital. *Journal of Internet and Software Engineering*, *1*(3), 15. https://doi.org/10.47134/pjise.v1i3.2511

Ramalingam, E. (2016). *Research Paper on Content Management Systems (CMS): Problems in the Traditional Model and Advantages of CMS in Managing Corporate Websites*. Harrisburg University of Science and Technology.

Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.