International Journal Software Engineering and Computer Science (IJSECS)
6 (3), 2026, 1131-1141
Published Online December 2026 in IJSECS (http://www.journal.lembagakita.com/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v6i3.8154.

RESEARCH ARTICLE Open Access

Analysis of the Paradigms and Effectiveness of Django Admin and Filament V4 in Rapid and Interactive Information System Development: A Case Study of the Borneo Link Material E-Commerce Application

Praneta Dwi Indarti [1], **Dinar Nugroho Pratomo** [2*]

[1,2*] Department of Electrical Engineering and Informatics, Universitas Gadjah Mada, Sleman Regency, Special Region of Yogyakarta, Indonesia.

Article History:

Received: July 23, 2026.
Revised: July 25, 2026.
Accepted: August 27, 2026.
Available online: September 20, 2026.
Published: December 1, 2026.

***Corresponding Author:**
dinar.nugroho.p@ugm.ac.id

Abstract: Administrative panels are critical for e-commerce data governance, yet their implementation is often hindered by tight deadlines. Filament v4 and Django Admin provide rapid, model-driven back-office solutions, but direct empirical evaluations remain scarce. This study analyzes the architectural paradigms and operational effectiveness of Filament v4 and Django Admin using Rapid Application Development (RAD) on the Borneo Link e-commerce system, managing 1,000 material records under a two-month constraint without prior interface design. Both systems were constructed with equivalent schemas and validated through 35 black-box test cases across administrator and partner roles. The architectural evaluation demonstrates that Django Admin delivers superior simplicity and faster setup (2 minutes 4 seconds vs. 4 minutes 38 seconds), whereas Filament v4 provides greater modularity via dedicated resource and relation managers. Usability evaluations across ten representative respondents indicated that Filament v4 outperformed Django Admin in learnability (85.0% vs. 69.0%) and user engagement (84.5% vs. 60.5%), facilitated by built-in detail pages and input masking. Conversely, Django Admin exhibited higher operability (81.3% vs. 67.3%) and significantly faster response times across pagination, search, and filtering (529.2–613.8 ms vs. 901.28–1119.4 ms). These performance divergences stem from architectural differences between Django's native server-side rendering and Filament's Livewire-driven asynchronous request pipeline. These findings delineate an empirical trade-off between execution speed and interface interactivity, establishing actionable criteria for framework selection in rapid enterprise systems development.

Keywords: Django Admin; Effectiveness; Filament v4; Paradigm; Rapid Application Development.

© The Author(s) 2026, corrected publication 2026. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution, and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third-party material in this article are included in the article's Creative Commons license unless stated otherwise in a credit line to the material. Suppose the material is not included in the article's Creative Commons license, and your intended use is prohibited by statutory regulation or exceeds the permitted use. In that case, you must obtain permission directly from the copyright holder. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

1. Introduction

Advances in information technology have fundamentally transformed how businesses conduct transactions and manage operations. Commercial enterprises increasingly leverage e-commerce platforms alongside conventional channels to expand market reach and improve transaction efficiency (Tafonao & Nadia Raslina, 2025). Within an e-commerce architecture, the administrative panel represents a critical component responsible for governing product catalogs, order workflows, and user records in a structured and accessible manner (Maulana *et al.*, 2022). Consequently, operational efficiency depends heavily on the speed, interactivity, and usability of this back-office interface. An unintuitive or sluggish admin panel increases human error, impedes administrative throughput, and elevates long-term maintenance overhead (Meleshkevich, 2025). Corroborating this perspective, Jiang *et al.* (2020) demonstrated that interface usability directly governs task performance and error rates, underscoring the necessity of intuitive administrative designs. Despite its operational importance, administrative panel development frequently receives less design and usability consideration than customer-facing front-end interfaces. Administrators consequently encounter friction when processing data, impairing overall organizational productivity. Establishing an efficient back-office interface during early development stages is substantially more beneficial than deploying iterative interface patches after system rollout (Maryniuk *et al.*, 2024). To mitigate these development bottlenecks, engineering teams increasingly adopt web frameworks integrated with administrative panel builders. This approach expedites delivery through pre-built layouts, automated CRUD scaffolding, and direct database ORM integration. However, selecting an appropriate framework and admin builder remains challenging due to varying architectural conventions, customization trade-offs, and extensibility limits across ecosystems (Kettering, 2026). Within modern web development, Laravel and Django provide two dominant ecosystems offering rapid administrative solutions. In the Laravel ecosystem, Filament v4 provides an add-on administrative architecture featuring automated CRUD scaffolding, resource management, and Tailwind CSS v4 integration to accelerate dashboard delivery (Christiantama, 2025; Ferreira, 2025). Conversely, Django provides built-in administrative capabilities via Django Admin, which inspects model metadata to automatically generate functional management interfaces without requiring third-party libraries (Django Software Foundation, 2026; Srivastava *et al.*, 2024). Both frameworks adhere to model-driven development paradigms, wherein interface layouts reflect underlying data structures. Django Admin reads model metadata directly (Django Software Foundation, 2026), whereas Filament v4 employs dedicated resource classes derived from Laravel Eloquent models (Ferreira, 2025). Prior empirical studies have separately examined Django and Filament in back-office implementations. Roshini *et al.* (2025) applied the Django framework to develop an e-commerce management panel, reporting that its architectural modularity and native security satisfied administrative operational criteria under User Acceptance Testing (UAT). Similarly, Apriandi *et al.* (2025) utilized Laravel Filament to construct a web-based reservation management panel, securing an acceptance rate of 86.91% and validating its capability to deliver responsive administrative interfaces rapidly. Although these studies confirm the individual feasibility of both frameworks, existing literature evaluates them in isolation across disparate domains. Direct comparative analyses examining architectural trade-offs, implementation overhead, interactivity, and operational performance between Django Admin and Filament v4 remain scarce, leaving developers without standardized empirical benchmarks for framework selection. The requirement for rapid yet effective administrative development emerged concretely during the engineering of the Borneo Link e-commerce platform by PT Lima Cahaya Kotim. The application manages approximately 1,000 building material items across Kotawaringin Timur Regency, imposing substantial data entry and logistical tracking demands. System construction was constrained by a rigorous two-month delivery schedule without prior user interface prototypes, necessitating a rapid, model-driven development approach that maintained administrative usability. Accordingly, this study analyzes the development paradigms and practical effectiveness of Django Admin and Filament v4 using the Rapid Application Development (RAD) methodology applied to the Borneo Link material management module. By evaluating development structure, setup overhead, system responsiveness, and usability dimensions, this research provides actionable technical criteria for selecting administrative panel frameworks in high-throughput enterprise systems.

2. Related Work

Advances in e-commerce technology have intensified the operational demand for administrative panels capable of orchestrating inventory, order fulfillment, and user data. Rinanti and Wahyuningsih (2025) developed a Laravel-based e-commerce platform for micro, small, and medium enterprises (MSMEs) integrating automated inventory tracking and financial reporting. Similarly, Maulana *et al.* (2022) utilized CodeIgniter 4 to implement core administrative workflows governing customer records and order validation. Further implementations across diverse commercial domains (Atmaja *et al.*, 2023; Probonegoro *et al.*, 2024;

Copyright © 2026 IJSECS

Raihan & Hidayat, 2024; Siki *et al.*, 2024; Voutama *et al.*, 2022) confirm that centralized administrative interfaces significantly increase operational efficiency compared to conventional, fragmented record-keeping processes. Alongside functional delivery, interface usability serves as a critical determinant of administrative productivity. Hijriani *et al.* (2022) evaluated the ePakan marketplace admin dashboard across transaction monitoring, merchant verification, and fund disbursements, obtaining a usability score of 83.0% and demonstrating that streamlined layout navigation directly minimizes operator friction. To accelerate the delivery of such administrative systems, development teams increasingly integrate modern web framework tooling with rapid methodologies. Roshini *et al.* (2025) established that Django delivers a secure, modular back-office architecture suitable for rapid enterprise deployment. In the PHP ecosystem, Apriandi *et al.* (2025) implemented Filament on Laravel for a service reservation platform, achieving an 86.91% User Acceptance Testing (UAT) score. Both studies employed Rapid Application Development (RAD) to shorten release cycles through iterative stakeholder feedback—a strategy consistently validated in broader e-commerce implementations (Atmaja *et al.*, 2023; Probonegoro *et al.*, 2024; Raihan & Hidayat, 2024; Siki *et al.*, 2024). Technical comparative studies have primarily evaluated web frameworks through network performance and security vulnerabilities. Amarulloh *et al.* (2023) benchmarked Laravel, Django, and Node.js web services, demonstrating that underlying framework engines generate substantial differences in response time, throughput capacity, and transmitted data overhead. In a security evaluation, Khaliqyar *et al.* (2025) mapped the native defense mechanisms of Django and Laravel against the OWASP Top 10 vulnerabilities, confirming distinct architectural paradigms in access control and request sanitization. Nevertheless, these technical evaluations primarily address front-facing services or protocol-level handling, omitting comparative evaluations of dedicated admin panel builders regarding development overhead, runtime rendering pipelines, and administrative usability. Taken together, existing literature substantiates the operational utility of Django and Laravel for rapid system delivery, yet comparative evaluations between their respective administrative tools remain scarce. Prior research rarely addresses the architectural trade-offs between integrated monolithic scaffolding, represented by Django Admin, and modular component-driven packages, represented by Filament v4 on Laravel. Consequently, empirical evidence is lacking regarding how these differing paradigms balance configuration overhead, structural modularity, runtime rendering speed, and end-user usability. To address this gap, this study conducts an empirical comparative evaluation of Django Admin and Filament v4 using the RAD framework within an identical e-commerce material management case study.

3. Methodology

3.1 Research Procedure

This study applies the Rapid Application Development (RAD) methodology, structured into four sequential phases: Requirements Planning, User Design, Construction, and Cutover. This method was selected because it supports rapid system delivery within a constrained two-month timeline through iterative prototyping and active stakeholder feedback. The Requirements Planning phase focused on identifying operational bottlenecks within the material administration workflow of the Borneo Link platform. A literature review encompassing back-office architectures, modern web frameworks, and RAD practices was conducted alongside stakeholder consultations at PT Lima Cahaya Kotim. These activities established the functional and non-functional specifications required to manage an estimated catalog of 1,000 building material items across Kotawaringin Timur Regency. During the User Design phase, system specifications were translated into functional module architectures. The back-office features planned for parallel implementation across both frameworks comprised product catalogs, categorized classifications, user and driver roles, delivery fleet assignments, and transactional order reporting. Identical schemas and business rules were enforced across both implementations to guarantee objective comparative validity. In the Construction phase, parallel administrative modules were constructed across both frameworks using a shared PostgreSQL database structure. The Django Admin implementation utilized Python, Django, and its built-in administrative engine, while the Filament v4 implementation utilized PHP, Laravel, and the Filament v4 administrative builder. Both systems were configured to support two role-based interfaces, designated for superadministrators and material partners. Architectural complexity, command overhead, and file modifications were systematically recorded throughout this phase. The Cutover phase verified functional conformance through black-box testing. A total of 35 test cases were formulated from operational use cases, covering session authentication, catalog CRUD operations, fleet management, driver records, and order fulfillment workflows. Identical test inputs and procedures were applied across both frameworks to evaluate functional parity. Table 1 presents representative black-box test cases utilized during system verification.

Copyright © 2026 IJSECS

Table 1. Examples of Test Cases

Code	Scenario	Steps	Test Data	Conclusion
LOG_01	Login Page	1. Access login URL 2. Enter email Enter password 3. Click login button	Email: admin@gmail.com Password: 1234	Successfully logged in to both systems
MMT_02	Material Management Page	1. Click Add Material button 2. Fill in the data 3. Click Save button	Name: Batu Kapur Wangi Type: Batu Category: Batu Kapur Description: Batu Kapur for minimalis stuff Kecamatan: Sentolo Unit: Truck Minimum order: 3 Distance Range: 50 Price: 180.000 Location: -2.5312, 112.9581	Successfully added material to both systems

3.2 Analysis Indicators

To evaluate the two frameworks systematically, specific analytical parameters were established. These indicators measure structural complexity, implementation effort, and operational characteristics throughout the development lifecycle: Setup time measures the total duration required to install and configure the framework until the development environment is fully operational. The number of setup commands records the total shell instructions executed during installation and configuration, reflecting the procedural complexity of the initial setup workflow. Basic folder structure evaluates how the framework organizes core components according to architectural patterns (MVC or MVT), illustrating structural modularity and project maintainability. Integration with the admin panel examines the architectural mechanisms connecting administrative models and interface components with the broader application ecosystem. The number of modified files tracks modifications made to existing project files during feature implementation, representing structural coupling and separation of concerns. Finally, the number of files added quantifies newly generated files required to support system functionality, reflecting the architectural footprint and modularity enforced by the framework.

3.3 Analysis Aspects

The comparative evaluation of Django Admin and Filament v4 was structured across two primary dimensions: Development Paradigm and Development Effectiveness. The development paradigm examines architectural approaches, configuration overhead, and coding workflows, whereas development effectiveness evaluates runtime system responsiveness and end-user usability. The development paradigm encompasses four distinct analytical aspects: initial configuration, project structure, model implementation, and feature interactivity. Initial installation and configuration evaluates the setup process required to prepare the development environment during the implementation phase. In this study, the configuration milestone was reached once the system successfully hosted two distinct role-based admin panels. The indicators for this aspect are presented in Table 2.

Table 2. Initial Installation and Configuration Indicators

Indicator	Description	Method
Setup Time	Time from installation to the admin panel being set up	Calculation
Number of Setup Commands	Number of commands executed during setup	Calculation

Project structure evaluates the organization of application components according to the architectural patterns of the framework, reflecting system maintainability and scalability. Django Admin operates as a built-in feature utilizing the Model-View-Template (MVT) pattern, whereas Filament operates as an administrative package within Laravel adhering to the Model-View-Controller (MVC) pattern. The indicators for this aspect are outlined in Table 3.

Table 3. Project Structure Indicators

Indicator	Description	Method
Basic Folder Structure	Folder structure based on architecture (MVC/MVT)	Observation
Integration with the Admin Panel	Management and integration of the admin panel	Observation

Copyright © 2026 IJSECS

Model implementation represents the data-layer modeling process during development. Both frameworks require predefined models defining entities and relationships. This aspect evaluates Object-Relational Mapping (ORM) principles across attribute mapping, relationship definitions, and administrative interface integration, measured by file modifications and file additions as detailed in Table 4.

Table 4. Model Implementation Indicators

Indicator	Description	Method
Number of Modified Files	The number of files modified for the model implementation	Calculation, Observation
Number of Files Added	The number of files added for the implementation of the model	Calculation, Observation

Feature interactivity examines the implementation effort required to deliver dynamic user controls. This aspect assesses data exploration features (searching, filtering, detail pages, change history, and bulk actions) and user input controls (validation rules, placeholders, input masking, and helper text). The evaluation measures the availability of default features and the complexity of their implementation based on the indicators shown in Table 5.

Table 5. Interactive Feature Indicators

Indicator	Description	Method
Number of Modified Files	The number of files modified due to feature implementation	Calculation
Number of Files Added	The number of files added as a result of feature implementation	Calculation

Development effectiveness assesses the operational quality of the resulting systems through two primary aspects: performance effectiveness and user usability. Performance effectiveness evaluates runtime responsiveness under operational loads, aligning with the time behavior sub-characteristic of the ISO/IEC 25010 performance efficiency standard. System response times were benchmarked across pagination traversal, multi-column searching, and data filtering on a dataset of 1,000 material records using the Chrome DevTools Network panel, as described in Table 6.

Table 6. Performance Effectiveness Indicators

Indicator	Description	Method
Load Time	System response time in milliseconds (ms)	Testing

User usability examines user interaction, task efficiency, and operational comfort through usability testing. The evaluation involved ten representative stakeholders comprising four administrators, three logistics staff, and three informatics students. Participants rated both systems using a 5-point Likert scale across the evaluation criteria presented in Table 7.

Table 7. Description of User Usability Evaluation Criteria

Aspect	Description
Ease of Understanding (Learnability)	The system is easy to understand when first used. Clear navigation.
Ease of Operation (Operability)	Tasks can be completed quickly. Doesn't require many steps. The system is responsive when in use.
Design and Satisfaction (User Engagement)	Attractive interface. Information is neatly organized. I feel comfortable using this system. I would be willing to use this system again.

The usability evaluation data collected from the questionnaire were subsequently aggregated and analyzed using descriptive statistics to calculate the average score and percentage for each usability aspect. The results of this usability testing, together with the performance benchmarks and architectural indicators evaluated during the construction and cutover phases, provide the empirical basis for comparing Django Admin and Filament v4 in the following section.

Copyright © 2026 IJSECS

4. Results and Discussion

4.1 Results

The following sections evaluate the empirical results obtained across the six analytical aspects during the implementation and cutover phases, contrasting Django Admin with Filament v4. Across five repeated installation trials under identical runtime environments, configuring Filament v4 required nine terminal commands, whereas Django Admin required eleven commands. However, Django Admin demonstrated a substantially faster initial configuration duration, averaging 124 seconds (2 minutes 4 seconds), compared to 278 seconds (4 minutes 38 seconds) for Filament v4. This variance is primarily attributable to the extensive Composer dependency tree resolved and compiled during Filament's installation, alongside front-end asset scaffolding. Consequently, the operational command count does not directly correlate with configuration latency, as setup duration remains heavily governed by package dependency resolution. Table 8 summarizes these configuration metrics.

Table 8. Summary of Initial Installation Configuration Analysis

System	Average Installation Time	Number of Commands
Filament v4	4 minutes 38 seconds	9 commands
Django Admin	2 minutes 4 seconds	11 commands

Structural analysis assessed directory organization and admin integration across both architectural conventions. Regarding the basic folder structure, Laravel natively scaffolds a decoupled directory layout conforming to the Model-View-Controller (MVC) pattern, segregating models, controllers, and Blade templates. Conversely, Django implements the Model-View-Template (MVT) pattern, but its default application generator adopts a centralized convention where template directories must be created manually, and data models reside within a single models.py file. While Django supports structural modularization, achieving logical separation into dedicated packages requires manual refactoring. In contrast, Laravel enforces individual model class files by convention, facilitating immediate readability and modular scaling. Regarding integration with the admin panel, Filament v4 utilizes explicit, command-driven resource generation. Each administrative entity automatically provisions a dedicated directory containing resource classes, forms, tables, and routing definitions. In contrast, Django Admin leverages native integration embedded within the framework core, activating administrative features through declaration within the localized admin.py file. Thus, Filament enforces component-driven separation of concerns via generated directory structures, whereas Django prioritizes declarative, in-place configuration. Table 9 contrasts these structural characteristics.

Table 9. Summary of Project Structural Analysis

System	Basic Folder Structure	Integration with the Admin Panel
Laravel (Filament v4)	Built-in MVC directory separation with convention-based entity files	Generated modular resource directory tree
Django (Django Admin)	Centralized MVT layout; template and split model structures require manual configuration	Built-in declarative registration via localized application files

The model implementation analysis evaluated data modeling across four structural dimensions: model schema definitions, relationship mapping, administrative interface registration, and relationship management.
In defining model schemas, Laravel adopts dynamic, class-based definitions utilizing properties such as $table and $fillable within Eloquent ORM. This design simplifies table mapping without requiring verbose attribute assertions. In contrast, Django enforces explicit schema definitions through Python class attributes, typed field classes, and nested Meta declarations (such as db_table and managed). Django's approach enforces rigid compile-time schema boundaries, as summarized in Table 10.

Table 10. Summary of Structural Model Analysis

System	Model Structure Description
Laravel	Flexible, configuration-driven definitions via $table and $fillable
Django	Explicit, strongly-typed attribute assertions with nested Meta classes

Relationship definitions reveal divergent structural footprints. In Laravel Eloquent, establishing one-to-one, one-to-many, and many-to-many associations requires modifications across two distinct files (both the parent and child models) because relationships are declared as reciprocal class methods. Conversely, Django centralizes relational logic within single model files using ForeignKey, OneToOneField, or ManyToManyField.

Copyright © 2026 IJSECS

Only explicit intermediate join tables require secondary file modifications in Django. Table 11 outlines the file modification impact of relationship modeling.

Table 11. Summary of Model Relationship File Changes

System	One-to-One Relation	One-to-Many Relation	Many-to-Many Relation
Laravel	2 model file modifications	2 model file modifications	2 model file modifications
Django	1 model file modification	1 model file modification	2 model file modifications

Integration with the administrative interface highlights differing levels of configuration overhead. Django Admin delivers high abstraction, requiring only a single registration call within admin.py to generate complete CRUD views. Conversely, Filament v4 executes dedicated terminal generator commands that construct a modular resource folder containing eight supporting component files per entity, trading setup minimalism for granular UI control, as detailed in Table 12.

Table 12. Summary of Model Abstraction Analysis

System	Model Abstraction Description
Filament v4	1 terminal command generating 1 resource directory containing 8 files
Django Admin	1 declarative file modification (admin.py)

Administrative relationship management exhibits a similar trade-off. Django Admin governs associations centrally within admin.py using inline classes (TabularInline or StackedInline) without generating additional disk files. In contrast, Filament v4 isolates relational interfaces through dedicated Relation Manager components, requiring terminal commands that scaffold specialized resource subdirectories. Table 13 presents the resulting administrative footprints.

Table 13. Summary of Administrative Relationship Management Analysis

System	Relationship Management Description
Filament v4	2 terminal commands generating 1 resource subdirectory and 1 relation manager file
Django Admin	1 declarative file modification (admin.py)

The evaluation of interactive capabilities examined data discovery and input control mechanics. Filament v4 provides out-of-the-box support for comprehensive detail views, input masking, modal confirmations, and rich reactive components driven by Tailwind CSS v4. Conversely, Django Admin natively provides basic data discovery (search bars, categorical list filters, date hierarchies, and bulk actions) alongside audit change history logs, but lacks native input masking and interactive detail modals without manual template extension or third-party packages. Table 14 summarizes feature availability and configuration effort across both systems.

Table 14. Summary of Interactive Feature Implementation

Feature	Filament v4	Django Admin
Search	1 file modification	1 file modification
Filter	1 file modification	1 file modification
Bulk Actions	1 file modification	1 file modification
Details Page	Built-in native feature	Not available natively
History / Audit Page	Not available natively	Built-in native feature
Placeholder	1 file modification	1 file modification
Input Masking	1 file modification	Not available natively
Length Validation	1 file modification	1 file modification
Size Validation	1 file modification	1 file modification
Email Validation	1 file modification	1 file modification
Required Validation	1 file modification	1 file modification
Helper Text	1 file modification	1 file modification

System responsiveness was benchmarked using Chrome DevTools across pagination traversal, multi-column search, and categorical filtering over 1,000 material records. Django Admin demonstrated superior time behavior across all operations, recording response latencies of 588.48 ms for pagination, 529.20 ms for search execution, and 613.80 ms for filtering. Filament v4 exhibited longer latencies, requiring 901.28 ms, 957.00 ms, and 1119.40 ms across the respective test scenarios, as detailed in Table 15.

Copyright © 2026 IJSECS

Table 15. Summary of Performance Latency Benchmarks (Dataset: 1,000 Records)

System	Pagination Latency	Searching Latency	Filtering Latency
Filament v4	901.28 ms	957.00 ms	1119.40 ms
Django Admin	588.48 ms	529.20 ms	613.80 ms

The performance advantage of Django Admin stems from its synchronized server-side rendering (SSR) pipeline. Django processes database queries and renders static HTML templates directly on the server with minimal serialized overhead. In contrast, Filament v4 relies on Livewire's component lifecycle, which transmits stateful JSON payloads and DOM-diff structures over asynchronous HTTP requests, introducing additional parsing and client-side rehydration overhead. The usability evaluation conducted with ten representative stakeholders revealed distinct operational strengths. Filament v4 secured superior ratings in learnability (85.0%) and user engagement (84.5%), outperforming Django Admin, which scored 69.0% and 60.5% in the respective categories. Conversely, Django Admin outperformed Filament v4 in operability, securing an 81.3% rating compared to Filament's 67.3%. Table 16 summarizes these usability dimensions.

Table 16. Summary of User Usability Evaluation Results

System	Learnability	Operability	User Engagement
Filament v4	85.0%	67.3%	84.5%
Django Admin	69.0%	81.3%	60.5%

These findings indicate that non-technical users and occasional administrators find Filament's modern visual hierarchy, intuitive modals, and clear input affordances substantially more accessible. Meanwhile, frequent operators favor Django Admin's utilitarian layout, which prioritizes rapid, low-friction keyboard navigation and immediate data density without visual distraction.

4.2 Discussion

The empirical findings delineate clear architectural divergences between Django Admin and Filament v4. In terms of initial environment readiness, Django Admin demonstrates superior setup efficiency. Because administrative capabilities are natively compiled into Django's core, environment initialization is completed in under half the time required by Filament v4, despite requiring slightly more terminal commands. Filament v4 functions as an external administrative package requiring Composer dependency resolution, asset compilation, and panel provider configuration. Consequently, initial developer velocity is higher in Django for teams establishing rapid administrative prototypes under tight deadlines. These architectural philosophies extend directly to project structure and maintainability. Laravel enforces explicit architectural boundaries from the outset, providing dedicated directories for models and separate component classes. When integrated with Filament v4, this convention scales cleanly: each data entity is isolated within a dedicated resource directory, promoting modularity and team-based collaboration in large codebases. In contrast, Django prioritizes declarative centralization within models.py and admin.py. While this centralization maximizes rapid implementation for small-to-medium datasets, managing complex enterprise models with extensive custom behaviors requires intentional architectural refactoring to avoid monolithic configuration files. In feature interactivity, Filament v4 offers a modern, component-driven administration interface out of the box. Native provisions for slide-over detail modals, input masking, and reactive form states align with modern UX expectations, reducing the need for custom JavaScript scaffolding. Conversely, Django Admin emphasizes operational pragmatism, delivering robust audit logs and straightforward tabular lists but requiring external packages or custom template overrides to match modern interactive standards.

Performance benchmarking validates that interface interactivity introduces measurable runtime latency trade-offs. Django Admin achieved an average 40% reduction in response latency across pagination, search, and filtering operations compared to Filament v4. This efficiency is governed by Django's lightweight SSR pipeline, which executes queries and returns static HTML without intermediate component tracking. In contrast, Filament v4 utilizes Livewire's reactive layer, which serializes component states and processes DOM diffs via asynchronous payloads. While this provides a reactive single-page feel, it incurs server CPU cycles and network payload overhead when processing extensive datasets. The usability testing outcomes reflect this trade-off between visual modernism and operational simplicity. Filament's higher learnability (85.0%) and engagement (84.5%) scores demonstrate that intuitive iconography, modern typography, and modal dialogs reduce cognitive load for novice or occasional users. However, Django Admin's superior operability score (81.3%) indicates that experienced back-office staff prioritize high-density data tables, fast standard browser navigation, and zero-latency page transitions when managing repetitive data workflows. These results corroborate prior web framework benchmarking literature while contributing specific back-office architectural insights. The performance divergence observed aligns with Amarulloh *et al.* (2023), who established that core runtime architectures significantly influence request turnaround times. Furthermore, the high usability ratings

Copyright © 2026 IJSECS

obtained by Filament v4 are consistent with the back-office usability benchmarks documented by Hijriani *et al*. (2022). This study extends these foundational works by demonstrating that back-office tool selection involves an explicit architectural compromise: Django Admin provides rapid setup, low runtime latency, and operational simplicity, whereas Filament v4 provides component modularity, reactive interactivity, and superior end-user engagement.

5. Conclusion and Recommendations

This study evaluated the development paradigms and operational effectiveness of Django Admin and Filament v4 using the Borneo Link e-commerce material management platform as an empirical case study. The findings demonstrate fundamental architectural trade-offs between the two administrative environments. Django Admin adheres to a declarative, centralized paradigm that achieves significantly faster initial environment provisioning (2 minutes 4 seconds versus 4 minutes 38 seconds for Filament v4) and minimizes file churn during data model and relational mapping. In contrast, Filament v4 enforces a decoupled, resource-driven architecture within the Laravel ecosystem that introduces higher initial setup overhead and file dispersion, but provides superior component modularity and separation of concerns. In terms of operational effectiveness, a clear divergence emerged between backend runtime efficiency and front-end user experience. Django Admin delivered superior response latency across pagination, searching, and filtering on 1,000 records due to its synchronized server-side rendering pipeline, achieving an operability usability rating of 81.3% from back-office operators who prioritize rapid, utilitarian task execution. Conversely, Filament v4 leveraged reactive asynchronous component lifecycles to achieve higher learnability (85.0%) and user engagement (84.5%), indicating that its modern visual affordances, interactive modals, and input masking significantly lower cognitive barriers for non-technical users. Therefore, framework selection must align with project priorities: Django Admin is recommended for data-intensive enterprise back-ends prioritizing rapid prototyping, execution latency, and simple configuration, whereas Filament v4 is optimal for applications demanding modern reactive interfaces, granular UI modularity, and high end-user satisfaction. Future research should expand upon these findings across several dimensions. First, subsequent studies should investigate the security profiles of both frameworks, specifically benchmarking role-based access control (RBAC), permission inheritance, and API authentication resilience against standard vulnerability taxonomies. Second, usability evaluations should be scaled to encompass larger, more heterogeneous user cohorts across multi-enterprise deployments to strengthen statistical generalizability. Finally, runtime performance benchmarks should be conducted within production containerized architectures under concurrent multi-user load testing, utilizing significantly larger datasets to evaluate scaling bottlenecks across both administrative paradigms.

References

Amarulloh, A., Kurniasih, & Muchlis. (2023). Analisis perbandingan performa web service REST menggunakan framework Laravel, Django, dan Node.js untuk akses data dengan aplikasi website. *Jurnal Teknologi Informasi*, 7(2), 115–124. https://doi.org/10.31294/jti.v7i2.515

Andriansyah, D. (2019). Performance dan stress testing dalam mengoptimasi website. *Computer Based Information System Journal*, 7(1), 23–28. https://doi.org/10.33884/cbis.v7i1.995

Apriandi, M. A., Irawan, A. S. Y., & Purwantoro. (2025). Implementasi framework Laravel pada aplikasi pemesanan barbershop berbasis web (Studi kasus: Maiden Barberrock). *Jurnal Informatika dan Teknik Elektro Terapan*, 13(3), 96–108. https://doi.org/10.23960/jitet.v13i3S1.7520

Ariyanto, Y., Farhan, M., Rachmad, F., & Puspitasari, D. (2024). Laravel framework and native PHP: Comparison in the creation of REST API. *Matrix: Jurnal Manajemen Teknologi dan Informatika*, 14(2), 66–73. https://doi.org/10.31940/matrix.v14i2.66-73

Atmaja, R. D., Faizah, N., & Kambry, M. A. (2023). Aplikasi e-commerce Toko Sinar Bella dengan metode Rapid Application Development (RAD) menggunakan framework CodeIgniter 4. *Design Journal*, 1(1), 26–37. https://doi.org/10.58477/dj.v1i1.26

Christiantama, S. C. (2025). *Implementasi framework Laravel Filament untuk membangun sistem inventori pada UMKM Top Roti Kacang* [Undergraduate thesis, Universitas Semarang]. Repositori e-Skripsi USM.

Copyright © 2026 IJSECS

Copperbelt University. (2026). *Software development (CS 130): Design of input and control*. Scribd.

Django Software Foundation. (2026). *The Django admin site*. Django Documentation.

Esaki, K. (2013). System quality requirement and evaluation. *Global Perspectives on Engineering Management*, 2(2), 52–59. https://doi.org/10.1007/978-3-642-35795-4_12

Ferreira, L. (2025). *What's new in Filament v4? Feature overview*. Filament.

Filament. (2025). *Getting started*. Filament Documentation.

Hijriani, A., Setiawan, T., & Ardiansyah. (2022). Pengembangan modul dashboard admin "ePakan" dengan implementasi usability testing. *Jurnal Pepadun*, 3(1), 148–157. https://doi.org/10.23960/pepadun.v3i1.110

Hossain, M. I. (2023). Software Development Life Cycle (SDLC) methodologies for information systems project management. *International Journal for Multidisciplinary Research*, 5(5), 1–36. https://doi.org/10.36948/ijfmr.2023.v05i05.6223

Jasrotia, M. S. (2025). Digital SMED: Revolutionizing setup time optimization using Industry 4.0. *International Journal of Engineering Research*, 14(1), 1–10. https://doi.org/10.17577/IJERTV14IS010125

Kettering, J. (2026, February 10). *Top 10 admin panel builder tools for 2026—Ranked*. WeWeb.

Khaliqyar, R. (2025). Comparative security analysis of Django and Laravel web development frameworks: A documented feature evaluation. *International Journal of Research and Innovation in Applied Science*, 10(2), 1467–1475. https://doi.org/10.51584/IJRIAS.2025.100201

Maryniuk, A., Khamula, O., & Sosnovska, O. (2024). The impact of key aspects on admin panel design for online media sites. *Journal of Applied Technical and Educational Sciences*, 14(3), 1–12. https://doi.org/10.53564/706ktv47

Maulana, M., Aditya, Z., Hayuhardhika, W., Putra, N., & Arwani, I. (2022). Pengembangan sistem informasi e-commerce dengan pemanfaatan API Midtrans menggunakan framework Laravel (Studi kasus: Byboot.id). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, 6(8), 3899–3906.

Meleshkevich, S. (2025). *Designing admin panels for a site or web application*. AVADA MEDIA.

Nivedhaa, N. (2024). Software architecture evolution: Patterns, trends, and best practices. *International Journal of Computer Sciences and Engineering*, 1(2), 1–14.

Philippi, S. (2004). Model driven generation and testing of object-relational mappings. *Journal of Systems and Software*, 77(3), 193–207. https://doi.org/10.1016/j.jss.2004.07.252

Probonegoro, W. A., Sari, L. I., & Wijaya, B. (2024). Pengembangan sistem informasi e-commerce untuk pangkalan gas LPG dengan metode Rapid Application Development (RAD). *JSAI: Journal Scientific and Applied Informatics*, 7(2), 358–368. https://doi.org/10.36085/jsai.v7i2.6466

Raihan, M., & Hidayat, A. T. (2024). Rapid Application Development (RAD) dalam pengembangan aplikasi e-commerce berbasis mobile. *MALCOM: Indonesian Journal of Machine Learning and Computer Science*, 5(1), 93–100. https://doi.org/10.57152/malcom.v5i1.1742

Rauf, A., Bibi, H., Zubair, S., & Arif, S. (2025). Impact of design patterns and architecture on quality attributes. *Contemporary Journal of Social Review*, 3(3), 1329–1345. https://doi.org/10.63878/cjssr.v3i3.1101

Rinanti, S. A., & Wahyuningsih, E. (2025). Implementasi e-commerce di Toko The Familly Collection Desa Panjangsari Kecamatan Gombong Kabupaten Kebumen. *INSOLOGI: Jurnal Sains dan Teknologi*, 4(4), 721–734. https://doi.org/10.55123/insologi.v4i4.5694

Copyright © 2026 IJSECS

Rochimah, S., Akbar, R. J., & Langsari, K. (2023). Investigating design patterns impact on application performance and complexity. *IPTEK The Journal of Technology and Science*, 35(1), 1–17. https://doi.org/10.12962/j20882033.v35i1.16585

Rojas, H., Renteria, R., & Martinez, V. (2025). Understanding performance efficiency in ISO/IEC 25000: A systematic literature review. *International Journal on Informatics Visualization*, 9(6), 2405–2417. https://doi.org/10.62527/joiv.9.6.3074

Roshini, Anushika, Dharshini, Hari, S., & Kabilan. (2025). E-commerce website with admin panel. *International Journal of Scientific Research in Engineering and Management*, 9(4), 1–8. https://doi.org/10.55041/IJSREM31254

Salman, F. A., Baluch, B., & Bakar, Z. A. (2025). A model for classification usability testing practically from the agile methodology aspect. *International Journal on Informatics Visualization*, 9(1), 81–89. https://doi.org/10.62527/joiv.9.1.2459

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages* (pp. 336–342). IEEE. https://doi.org/10.1109/VL.1996.545307

Siki, Y. C. H., Talo, M. C., & Mamulak, N. M. R. (2024). Implementasi model Rapid Application Development (RAD) dalam pengembangan website penjualan produk UMKM Bikomi Utara. *Jurnal Penelitian Inovatif*, 4(1), 177–184. https://doi.org/10.54082/jupin.264

Siregar, M. E. (2026). Evaluasi implementasi Progressive Web App terhadap performa website dengan Google Lighthouse dan Chrome DevTools. *ALGORITME: Jurnal Mahasiswa Teknik Informatika*, 6(2), 382–394. https://doi.org/10.35957/algoritme.v6i2.15712

Srivastava, A., Shinde, V., Walhekar, V., Patil, A., Ganeshpurkar, A., Karthikeyan, M., & Kulkarni, R. (2024). Django unleashed: A deep dive into the features and advantages of the Django framework. *RGUHS Journal of Pharmaceutical Sciences*, 14(3), 1–9. https://doi.org/10.26463/rjps.14_3_7

Tafonao, F., & Raslina, N. (2025). Perancangan website e-commerce untuk Fayucom dengan integrasi payment gateway Xendit menggunakan HTML, CSS, dan JavaScript. *Jurnal Sosial dan Teknologi*, 5(8), 1145–1156. https://doi.org/10.59188/jurnalsostech.v5i8.32382

Vedder, D., Ankenbrand, M., & Sarmento Cabral, J. (2021). Dealing with software complexity in individual-based models. *Methods in Ecology and Evolution*, 12(12), 2324–2333. https://doi.org/10.1111/2041-210X.13716

Voutama, A., Garno, G., Irawan, A. S. Y., & Novalia, E. (2022). Design of e-commerce distro using Rapid Application Development (RAD) model. *Jurnal Riset Informatika*, 4(4), 363–370. https://doi.org/10.34288/jri.v4i4.357

Walther, J. B., Pingree, S., Hawkins, R. P., & Buller, D. B. (2005). Attributes of interactive online health information systems. *Journal of Medical Internet Research*, 7(3), Article e33. https://doi.org/10.2196/jmir.7.3.e33

Yu, X., Li, X., Wu, C., & Xu, G. (2022). Design and deployment of Django-based housing information management system. *Journal of Physics: Conference Series*, 2425(1), Article 012018. https://doi.org/10.1088/1742-6596/2425/1/012018

Zerkani, Y. (2017). *Implementation phase during Software Development Life Cycle* Bachelor's thesis, Helsinki Metropolia University of Applied Sciences. Theseus Repositori.

Copyright © 2026 IJSECS