International Journal Software Engineering and Computer Science (IJSECS)
6 (3), 2026, 1009-1022
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.v6i3.8255.

RESEARCH ARTICLE Open Access

Development of a Web-Based Barter Platform Using Node.js

Jeffrey Herly [1*], **Pratyaksa Ocsa Nugraha Saian** [2]

[1*,2] Department of Informatics Engineering, Faculty of Information Technology, Universitas Kristen Satya Wacana, Salatiga City, Central Java Province, Indonesia.

Article History:

Received: July 30, 2026.
Revised: July 30, 2026.
Accepted: August 27, 2026.
Available online: September 20, 2026.
Published: December 1, 2026.

***Corresponding Author:**
jeffreyherly9@gmail.com

Abstract: Before the emergence of modern currency systems, human economic transactions were conducted through barter, the direct exchange of goods or services. However, contemporary barter faces persistent operational friction, including the difficulty of establishing a double coincidence of wants, the absence of standardized exchange values, and inefficient manual matching. To address these challenges, this study develops a responsive, token-free web barter platform prototype engineered using the Waterfall development model. The system utilizes Node.js and the Fastify backend framework for high-throughput, non-blocking asynchronous request processing, paired with an EJS and Webix presentation layer and a PostgreSQL relational database. To streamline partner discovery, the platform couples PostgreSQL full-text search and trigram similarity matching (pg_trgm) for typo tolerance with an automated recommendation pipeline driven by recent search history. To mitigate cold-start friction for new users, the recommendation mechanism incorporates an adaptive fallback strategy that prioritizes listings by visit frequency. User data security is enforced through HTTP-only JSON Web Token (JWT) session handling, bcrypt cryptographic password hashing, and strict AJV schema validation. Functional verification via Black-Box testing achieved a 100% success rate across 16 transactional scenarios, while an OWASP ZAP active scan detected zero high-risk vulnerabilities. Performance benchmarking with Grafana k6 confirmed robust backend efficiency, recording a 95th percentile response time of 205.58 ms under sustained multi-user load and maintaining operational resilience up to 200 concurrent virtual users. This throughput stability confirms the efficacy of persistent database connection pooling and asynchronous thread execution under concurrent traffic. Finally, usability evaluation using the System Usability Scale (SUS) with 17 respondents yielded a composite score of 71.91, classifying the platform as Good with an Acceptable level of user reception.

Keywords: Barter; Web Platform; Node.Js; Fastify; Recommendation System; System Usability Scale.

1. Introduction

Before the emergence of modern currency systems, human economic transactions were conducted through the direct exchange of goods or services known as barter. This mechanism allowed individuals and communities to fulfill reciprocal needs without monetary intervention. Kim *et al*. (2024) noted that while barter represents an early transactional foundation, conventional implementations face severe operational frictions, notably the difficulty of finding suitable trading partners, establishing equivalent valuations, and overcoming geographical constraints. As market participation expands, a barter system theoretically gains viability due to the increased probability of identifying matching counterparts. Molina-Jiménez *et al*. (2020) demonstrated that contemporary technology can revitalize currencyless economic models through decentralized platforms that enable direct exchange without intermediaries, promoting goods reuse and presenting practical alternatives to monetary trade. Nevertheless, modern barter systems still encounter the core obstacle of establishing a double coincidence of wants coupled with the absence of standardized exchange values. As Shuhaiber *et al*. (2023) highlighted, these limitations reduce transactional efficiency in inventory handling and market reach, underscoring the urgent necessity for dedicated digital platforms that can effectively reconcile reciprocal user demands. Several preliminary efforts have explored computerized barter systems, yet notable gaps persist in the literature. Some implementations are restricted to specific domains such as mobile-based service exchanges that rely entirely on manual searches without matchmaking assistance (Nawawi *et al*., 2025), while others introduce artificial intermediate tokens that complicate the direct nature of pure bartering (Firmansyah *et al*., 2023). Furthermore, conceptual frameworks frequently lack concrete technical architectures and empirical evaluations covering security, scalability, and usability (Shuhaiber *et al*., 2023). A difficult-to-use platform impairs user adoption, while the absence of intelligent discovery mechanisms reduces an exchange platform to a passive catalog. To address these collective limitations, a dedicated, direct, token-free barter platform equipped with automated discovery capabilities is required.

To support high concurrent transaction volumes across an expanding user base, Node.js was selected as the runtime environment due to its event-driven, non-blocking I/O architecture, which ensures responsive handling of simultaneous connections without performance degradation. Fastify was selected as the backend framework owing to its demonstrated high-throughput request execution and precise schema validation capabilities, ensuring processing speed and data consistency across transaction pipelines. Coupled with a PostgreSQL database utilizing full-text search and trigram matching, the system is designed to provide responsive querying and flexible partner discovery. Based on this problem context, the research addresses how to design an intuitive web-based barter platform, how to construct a search feature that enhances user experience, and how to formulate an automated recommendation mechanism to mitigate search friction. Specifically, this research aims to: 1) build a web-based barter prototype that facilitates partner discovery; 2) implement essential transactional features, including user registration, item catalog management, and search functionalities; 3) integrate an automated exchange recommendation mechanism based on user search patterns; and 4) evaluate platform functionality, baseline security, and system usability. To maintain methodological focus, several boundaries are established: the implementation is strictly web-based and does not extend to native mobile operating systems (Android/iOS); the exchange workflows accommodate goods-for-goods, service-for-service, and goods-for-service models without auctioning or monetary payment gateways; the recommendation mechanism relies on search keyword matching rather than complex machine learning models; and security measures center on JSON Web Token (JWT) session handling, bcrypt password hashing, and input validation to counteract common threats such as SQL injection and cross-site scripting (XSS). The remainder of this paper is organized as follows. Section 2 reviews related literature on digital barter architectures and supporting technologies. Section 3 delineates the proposed methodology and technical system design. Section 4 presents the experimental implementation, verification results, and performance benchmarks. Finally, Section 5 concludes the paper with principal findings and outlines avenues for future work.

2. Related Work

Several prior studies have explored computerized and digital barter implementations, highlighting practical opportunities while exposing functional constraints. Nawawi *et al*. (2025) developed an Android-based barter service platform using React Native and Laravel; while it successfully integrated core features such as registration, service cataloging, search, and direct messaging, it was restricted to mobile service exchanges

and lacked automated recommendations, leaving exchange matching reliant entirely on manual user discovery. Shuhaiber *et al.* (2023) introduced the conceptual framework of E-Barter as a sustainable community mechanism, yet the study lacked concrete technical documentation regarding software architecture, database management, and vulnerability hardening. Addressing implementation mechanics, Firmansyah *et al.* (2023) constructed a Laravel-based barter system utilizing intermediate exchange tokens; however, reliance on synthetic token economics deviates from pure reciprocal exchange and introduces unnecessary transactional friction. Studies on modern barter communities further reveal that while online platforms successfully cultivate social trust and facilitate circular resource utilization (Alam *et al.*, 2025; Wong, 2025), persistent hurdles in partner discovery and system scalability remain. Consequently, there is an evident need for a web-based platform that facilitates direct, token-free exchanges supported by automated partner discovery mechanisms. Theoretically, barter represents a reciprocal transaction model that exchanges goods or services without monetary mediation. While conventional monetary theory posits that money emerged to resolve barter inefficiencies, recent anthropological and economic analyses demonstrate that barter remains viable and naturally coexists with monetary systems, particularly in structured communal environments (Fauvelle, 2025; Taskinsoy, 2023). Kim *et al.* (2024) observed that transactional coordination within barter markets matures progressively, conditioned by mutual trust, social connectivity, and matching efficiency. Modern digital architectures can reinforce these transactional environments by reducing information asymmetry and mitigating fraud risks through standardized digital workflows (Alam *et al.*, 2025). To ensure high responsiveness and operational reliability under concurrent usage, the platform stack was selected based on proven empirical benchmarks. Node.js was chosen as the execution runtime due to its event-driven, non-blocking I/O architecture, which provides superior efficiency in handling simultaneous network connections (Libertino, 2020). As the backend framework, Fastify was selected for its minimal overhead, high request-processing throughput, and native schema validation capabilities using AJV, outperforming traditional Node.js frameworks under benchmark conditions (Kuffel & Walter, 2024; Pratama & Raharja, 2023). For the relational data layer, PostgreSQL was selected due to its robust support for high concurrency, transactional integrity, and Multi-Version Concurrency Control (MVCC). Multiple empirical evaluations have demonstrated PostgreSQL's performance advantages over alternative relational and non-relational database management systems across large-scale querying and standard transactional operations (Han & Choi, 2024; Klimek & Skublewska-Paszkowska, 2021; Putra *et al.*, 2022; Salunke & Ouda, 2024; Zapata, 2024). From an architectural standpoint, the system adopts a layered architecture (N-tier model) separating presentation, business logic, and data persistence, which enhances codebase modularity, testability, and long-term maintainability (Tu, 2023), as illustrated in Figure 1.

Figure 1. Illustration of Layered Architecture / N-Tier Model

In addition to architecture selection, the development lifecycle requires a structured engineering approach. The Waterfall model was adopted due to its systematic, sequential progression across analysis, design, implementation, verification, and maintenance phases. Saravanos and Curinga (2023) and Senarath (2021) emphasized that Waterfall is particularly suitable when functional specifications and security requirements are explicitly defined prior to implementation, ensuring rigorous documentation and phase validation throughout development. Furthermore, empirical findings by Sunarya *et al.* (2025) confirm that the disciplined execution of the Waterfall methodology provides high structural quality and verification traceability in web-based information systems.

3. Methodology

The application was engineered following the Waterfall development lifecycle, which comprises five sequential phases: Requirements Analysis, System Design, Implementation, Testing, and Maintenance. During Requirements Analysis, functional and non-functional requirements were identified and documented based on an evaluation of existing barter platforms and user needs, resulting in a formal Software Requirements Specification (SRS). In System Design, the system architecture, procedural flowcharts, Entity-Relationship Diagrams (ERD), the relational PostgreSQL database schema, and Fastify REST API specifications were structured. The Implementation phase realized the design blueprints through backend development using Node.js and Fastify, database provisioning in PostgreSQL, and the integration of JSON Web Token (JWT)

authentication. The Testing phase executed Black-Box functional verification across all functional endpoints, coupled with baseline security assessments, input validation checks, and usability testing using the System Usability Scale (SUS). Finally, the Maintenance phase encompassed performance monitoring, system documentation, and usability evaluations to inform future architectural enhancements. The Waterfall model was selected because its structured, sequential milestones facilitate rigorous documentation and phase-gate verification, as illustrated in Figure 2.

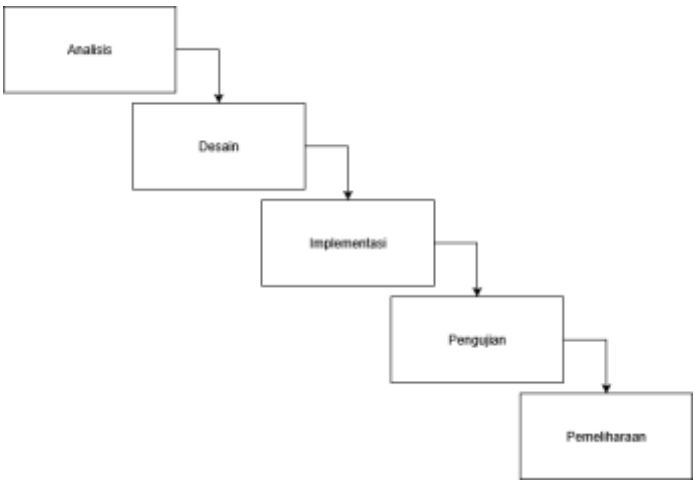

Figure 2. Illustration of Waterfall Model

From a architectural execution standpoint, the application enforces a strict separation of concerns across a four-tiered backend pipeline. Every inbound HTTP request is intercepted by Fastify Routes, which govern URI endpoints, route prefixes, and pre-validation middleware, including authentication guards. Validated requests are dispatched to the Controller layer, which orchestrates request handling, invokes the appropriate business services, and either renders an EJS server-side template or returns structured JSON payloads. The Service layer encapsulates core business logic, executes operational validation rules, formats data structures, and leverages system utilities. Data persistence is decoupled within the Repository layer, which serves as the exclusive interface interacting with PostgreSQL via fastify-postgres (fastify.pg) to execute parameterized SQL transactions. Query results flow back sequentially through the Service and Controller layers to the client, ensuring modularity, testability, and clear execution boundaries.

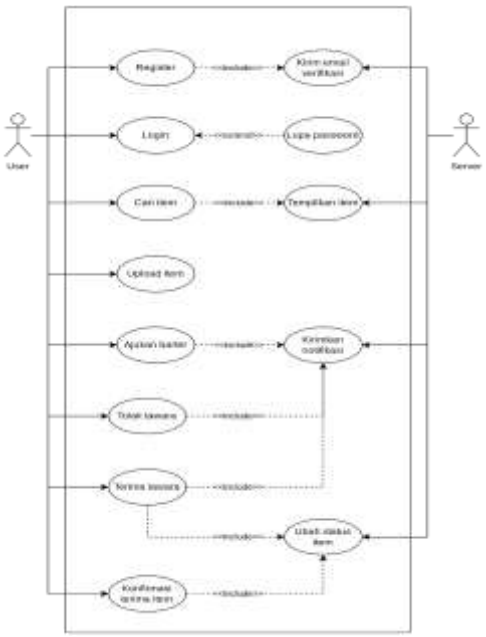

Figure 3. Use Case Diagram

System interactions and functional capabilities are formalized using Unified Modeling Language (UML) artifacts. As depicted in the Use Case Diagram in Figure 3, the platform delineates core user responsibilities across account registration and verification, item catalog management, search operations, barter proposal

negotiations, and peer-to-peer review submissions, ensuring all functional requirements are satisfied through structured actor-system interactions.

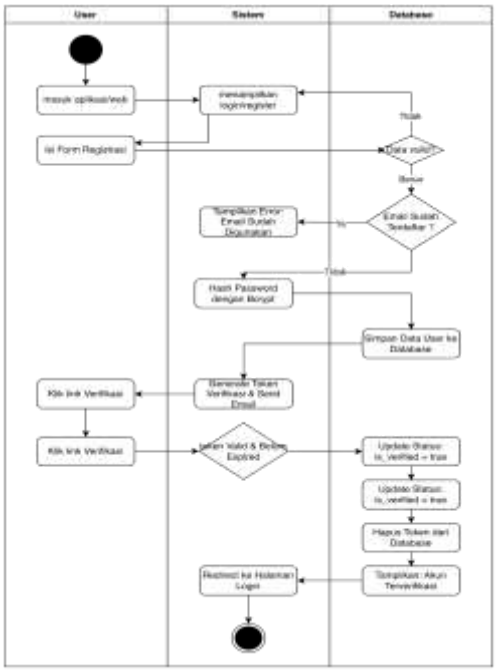

Figure 4. Activity Diagram of Registration & Login

Operational logic across critical workflows is delineated through activity diagrams. The user onboarding and authentication sequence begins with credential input followed by server-side format and unique-email validation, as detailed in the registration and login activity diagram in Figure 4. If invalid inputs or duplicate emails are encountered, the system returns descriptive error notifications; otherwise, account records are staged, and a secure verification link is dispatched via email. Upon email confirmation, the user account is elevated to verified status, granting access to authentication routes.

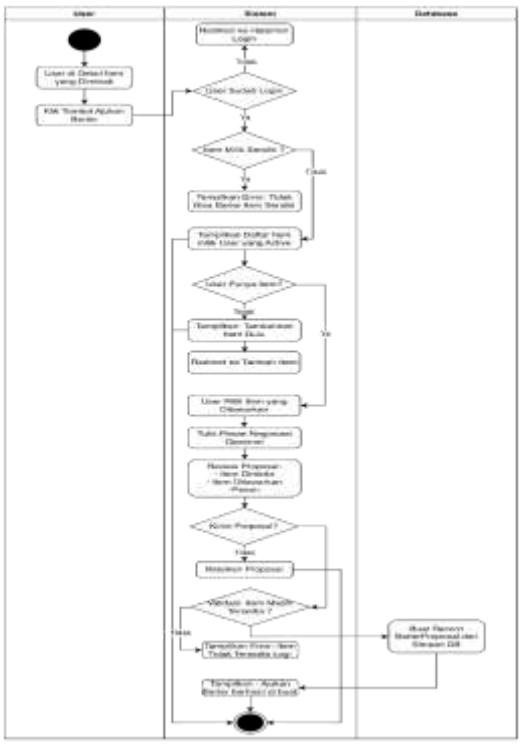

Figure 5. Activity Diagram of Barter Submission

The trade initiation sequence is governed by the barter submission workflow, as illustrated in Figure 5. A user identifies an item of interest and initiates a trade proposal. The system verifies active session authentication and asserts that the requested item does not belong to the initiating user. The user then selects an available item from their personal inventory to offer as an exchange asset and optionally appends a negotiation message. Once inputs are validated and asset availability is reconfirmed, the proposal is staged with a pending status, an expiration threshold is assigned, and an asynchronous notification is dispatched to the target asset owner.

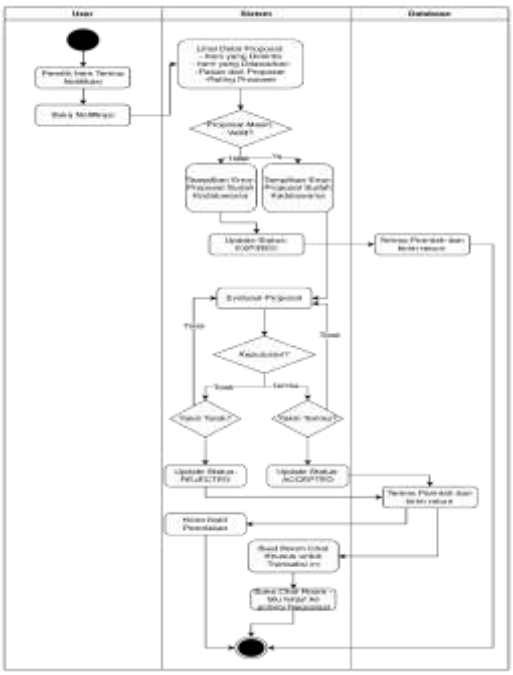

Figure 6. Activity Diagram of Accept or Reject Barter

Figure 6 illustrates the proposal evaluation workflow executed by the recipient asset owner upon receiving an exchange notification. The system first evaluates the proposal timestamp; if the duration exceeds the allowable window, the proposal is marked as expired. If active, the recipient can accept or reject the exchange. A rejection transitions the proposal to rejected status and returns the proposed items to available status. Conversely, an acceptance updates the proposal to accepted status, triggers reciprocal notifications, locks both exchange assets against competing proposals, and initializes a dedicated direct chat channel to facilitate trade fulfillment.

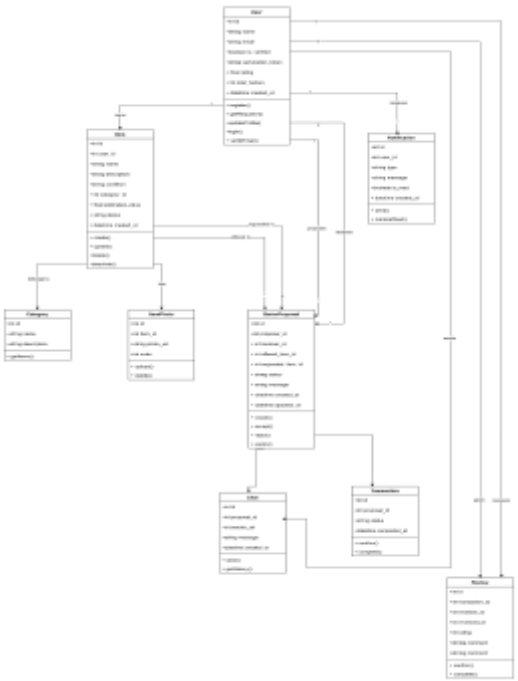

Figure 7. Class Diagram

The structural domain model and data associations supporting the transactional lifecycle are defined in the Class Diagram shown in Figure 7. The User entity serves as the principal actor, establishing one-to-many associations with Item, BarterProposal, Notification, Chat, Transaction, and Review entities. Each Item is assigned to an individual User, belongs to a defined Category, and can associate with multiple ItemPhotos and ongoing BarterProposals. The BarterProposal entity captures bilateral negotiations across pending, accepted, rejected, and expired lifecycle states. Proposal state changes generate corresponding Notification entities. Upon successful proposal acceptance, communication is established via the Chat entity, leading to a formal Transaction record that confirms asset transfer. Finally, completed transactions enable reciprocal Review submissions, establishing trust and reputation metrics across the platform.

4. Results and Discussion

4.1 Results

The web-based digital barter platform was constructed using Node.js and the Fastify framework, following the layered architecture delineated in Section 3. Data persistence is managed via PostgreSQL with environment-based configuration parameters. Security controls incorporate JSON Web Token (JWT) session tokens stored within HTTP-only cookies, bcrypt cryptographic password hashing, strict input validation, rate limiting, and mandatory email activation workflows. The server-rendered presentation layer combines the EJS template engine with the Webix UI library to provide responsive interface interactions (Daniswara & Susetyo, 2024).

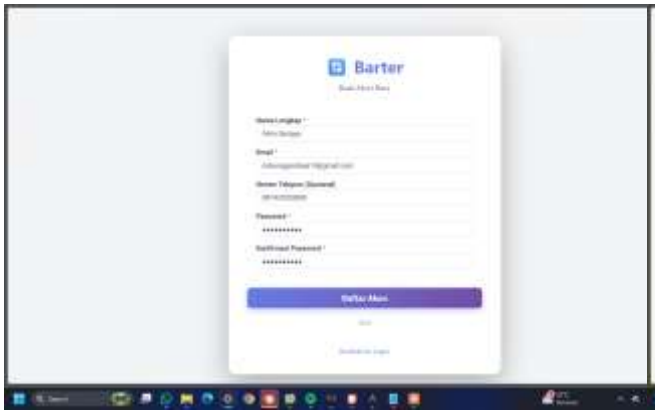

Figure 8. Registration Page Display

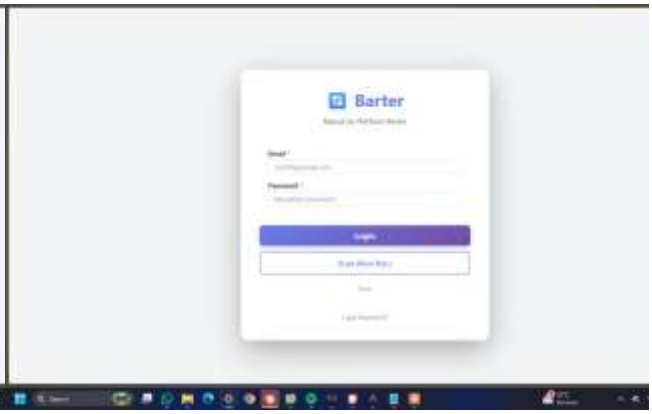

Figure 9. Login Page Display

The user onboarding sequence begins at the registration interface, as shown in Figure 8. Upon form submission, the backend asserts data validity, verifies email uniqueness, hashes the user password, and generates an email verification token. Following email confirmation, users access the platform through the authentication interface displayed in Figure 9, where the system validates credentials and issues an HTTP-only JWT session cookie.

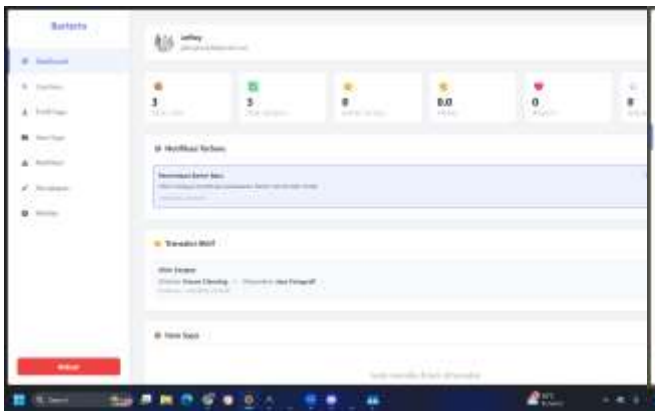

Figure 10. Dashboard Page Display

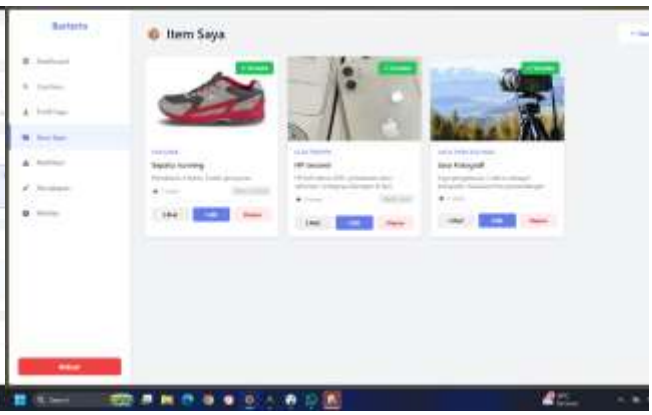

Figure 11. Item Management Page Display

Upon successful authentication, users are redirected to the main dashboard interface depicted in Figure 10, which provides navigational access to inventory controls, catalog discovery, active negotiations, and notifications. The inventory management interface displayed in Figure 11 enables users to create, update, and remove listings from their personal exchange catalog before entering negotiations.

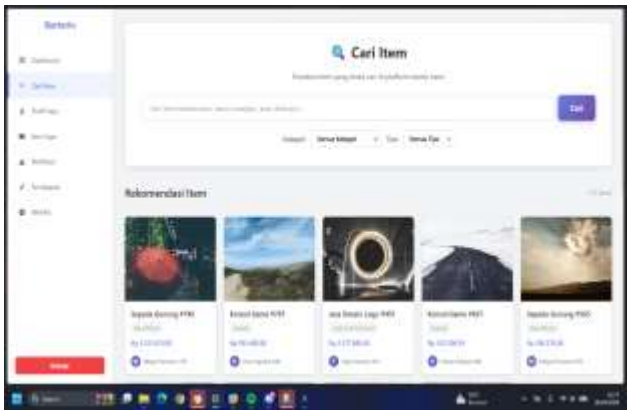

Figure 12. Item Search Page Display

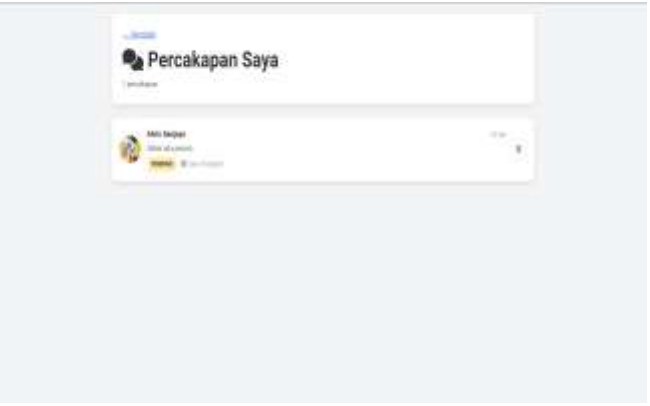

Figure 13. Conversation Page Display

Catalog exploration is handled via the item search interface shown in Figure 12. The search pipeline couples PostgreSQL Full-Text Search for keyword queries with the pg_trgm extension, providing trigram similarity matching to accommodate user typographical errors. The platform also integrates an automated recommendation pipeline driven by recent user activity. The system extracts up to five search keywords from the user_search_history relation and executes dynamically constructed SQL queries utilizing PostgreSQL ILIKE operators to identify matching items listed by other users. To prevent empty query states for newly registered users, an adaptive fallback mechanism activates popularity-based ordering, displaying listings ranked by total visit counts (view_count). Once negotiations are initiated, users coordinate trade details through the direct messaging interface shown in Figure 13. Functional correctness was evaluated using Black-Box Testing based on equivalence partitioning and boundary value analysis (Pratama & Dadaprawira, 2023; Santi *et al.*, 2022). A total of 16 comprehensive test scenarios were executed across five core application modules, as detailed in Table 1.

Table 1. Black-Box Testing Results

Tested Module / Feature	Test Scenario	Expected Result	Actual Result
User Authentication	User registration with complete and valid credentials.	System saves user record and dispatches email verification token.	Pass
	Registration attempt using an already registered email address.	System rejects registration and returns a duplicate-email error message.	Pass
	User login using valid email and correct password.	Credentials validated; user session initialized and redirected to dashboard.	Pass
	Login attempt using incorrect password or unregistered email.	System rejects access and displays an invalid-credentials error message.	Pass
Item Management	Uploading a new item with complete fields (Name, Description, Tag/Category, Photo).	Item successfully persisted and rendered in user inventory catalog.	Pass
	Uploading an item missing required fields (e.g., omitted photo or item name).	System halts submission and displays input validation errors on empty fields.	Pass
	Editing details of an owned item (e.g., updating description or category tags).	Modifications persisted and immediately reflected on item detail page.	Pass

	Deleting an owned item from the personal inventory.	Item record removed; listing no longer appears in search or dashboard views.	Pass
Search & Recommendation	Searching items using keywords (item name) or category filter tags.	System returns active listings from other users matching keywords or filters.	Pass
	Accessing the discovery interface to view item recommendations.	System renders relevant recommendations based on search history or fallback logic.	Pass
Barter Transaction (Core Flow)	Submitting an exchange proposal to another user by selecting an owned item as trade asset.	Proposal staged as *Pending*; asynchronous notification dispatched to target owner.	Pass
	Recipient accepts an incoming barter proposal.	Proposal status transitions to *Accepted*; both exchange items are locked against other trades.	Pass
	Recipient rejects an incoming barter proposal.	Proposal status transitions to *Rejected*; offered items revert to *Available* status.	Pass
	Completing a previously accepted exchange transaction.	Transaction marked as *Completed*; asset ownership records are updated/archived.	Pass
	Initiator cancels an exchange proposal prior to recipient review.	Proposal transitions to *Cancelled*; offered item returns to available status.	Pass
Communication (Messaging)	Dispatching direct chat messages to a prospective barter partner.	Messages transmitted and rendered in real-time or preserved in conversation inbox.	Pass

Across all 16 evaluated scenarios, expected outputs aligned completely with actual system behavior, achieving a 100% pass rate. Security vulnerability verification was executed using OWASP ZAP version 2.17.0 via Active Scan against authenticated and unauthenticated endpoints. The scan results are summarized in Figure 14.

Figure 14. Security Testing Results Using OWASP ZAP

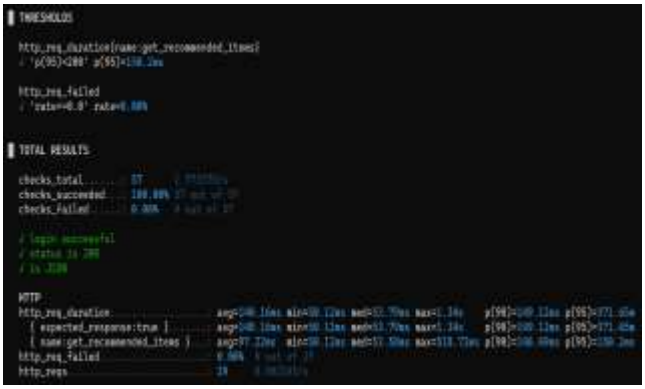

Figure 15. Response Time Test Results Using Grafana k6

The Active Scan identified zero (0) High-risk vulnerabilities. The analysis revealed 5 Medium-risk findings, 5 Low-risk findings, and 7 Informational notices. The identified Medium and Low findings were primarily related to security header configurations, such as tuning Content Security Policy (CSP) directives, missing strict transport security flags, and anti-CSRF token enforcement on specific forms, rather than exploitable code injection or broken authorization flaws. System throughput and latency were evaluated across three sequential workloads using Grafana k6: a baseline response test, a multi-user load test, and an extreme stress test. Under the baseline response test (1 Virtual User for 30 seconds), the platform executed 57 HTTP requests with a 100% success rate (0% error rate), as shown in Figure 15. The system recorded an average latency of 140.16 ms, a median of 53.79 ms, a minimum of 50.12 ms, and a 95th percentile (p95) latency of 371.65 ms across all endpoints. The recommendation endpoint (get_recommended_items) achieved a p95 latency of 150.2 ms, comfortably below the 200 ms service target.

Figure 16. Load Test Results Using k6

Figure 17. Stress Test Results Using k6

Under the multi-user load test simulating concurrent operational traffic, 3,034 requests were processed with a 100% success rate (0% error rate), as presented in Figure 16. The average response time was 95.4 ms, with a median of 78.88 ms and an overall p95 latency of 205.58 ms. Specifically, the get_recommended_items route recorded a p95 latency of 205.43 ms, satisfying the 500 ms operational threshold. Under the stress test workload ramping up to 200 concurrent Virtual Users over 3 minutes and 30 seconds (Figure 17), the system handled 41,011 total requests. The platform successfully fulfilled 30,201 requests (73.64%), while 10,810 requests (26.35%) timed out or failed as resource contention saturated backend thread queues, resulting in an overall p95 latency of 450.84 ms without server daemon crashes. System usability was evaluated through a System Usability Scale (SUS) study involving 17 undergraduate respondents who completed 10 standardized functional tasks covering account onboarding, catalog searches, proposal negotiations, and chat interactions. Individual item response distributions across the 10 SUS questionnaire statements are presented in Figures 18 through 27.

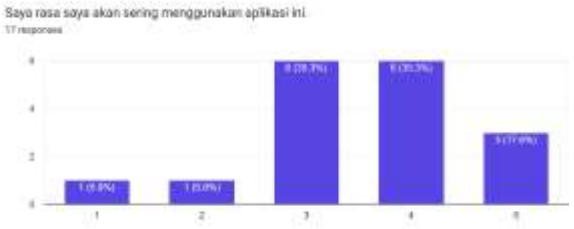

Figure 18. SUS Questionnaire Chart — Statement 1

Figure 19. SUS Questionnaire Chart — Statement 2

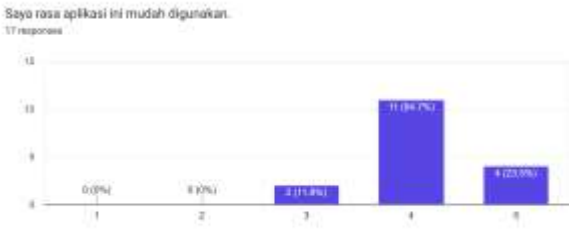

Figure 20. SUS Questionnaire Chart — Statement 3

Figure 21. SUS Questionnaire Chart — Statement 4

Figure 22. SUS Questionnaire Chart — Statement 5

Figure 23. SUS Questionnaire Chart — Statement 6

Figure 24. SUS Questionnaire Chart — Statement 7

Figure 25. SUS Questionnaire Chart — Statement 8

Figure 26. SUS Questionnaire Chart — Statement 9

Figure 27. SUS Questionnaire Chart — Statement 10

The calculated contribution scores and overall composite SUS score are presented in Table 2.

Table 2. System Usability Scale (SUS) Testing Results

No.	Usability Statement	Mean Item Score	Calculated Contribution Score
1	I think that I would like to use this system frequently.	3.529	2.529
2	I found the system unnecessarily complex.	2.176	2.824
3	I thought the system was easy to use.	4.118	3.118
4	I think that I would need the support of a technical person to be able to use this system.	2.118	2.882
5	I found the various functions in this system were well integrated.	4.294	3.294
6	I thought there was too much inconsistency in this system.	2.706	2.294
7	I would imagine that most people would learn to use this system very quickly.	4.176	3.176
8	I found the system very cumbersome to use.	1.706	3.294
9	I felt very confident using the system.	4.059	3.059
10	I needed to learn a lot of things before I could get going with this system.	2.706	2.294
Final SUS Composite Score (Sum of Contribution Scores × 2.5)			71.91

Based on the 17 respondents, the empirical composite SUS score was calculated at 71.91.

4.2 Discussion

The empirical findings demonstrate that the developed digital barter platform successfully resolves critical operational frictions identified in prior literature, particularly regarding architectural overhead, partner discovery, and user adoption barriers. A critical dimension of this study was verifying whether a token-free, direct exchange architecture could operate reliably without intermediate monetary abstractions. In contrast to the tokenized framework introduced by Firmansyah *et al.* (2023), which added administrative complexity by requiring artificial token accounting without actual payment integration, the direct bilateral mechanism implemented in this study proved functionally sound, achieving a 100% pass rate across all transactional scenarios in Table 1. Furthermore, while the mobile service-barter platform proposed by Nawawi *et al.* (2025) restricted trades to services and relied solely on manual user browsing, the current platform accommodates diverse reciprocal configurations (goods-for-goods, service-for-service, and goods-for-service). The integration of PostgreSQL Full-Text Search and pg_trgm fuzzy matching significantly mitigated partner discovery friction, allowing users to locate exchange counterparts even when search queries contained typographical errors. From an architectural standpoint, the combination of Node.js, Fastify, and a layered N-tier architecture delivered high processing efficiency. The load testing benchmarks (p95 latency of 205.58 ms at 3,034 requests

with 0% failure) corroborate prior comparative analyses indicating that Fastify's compiled schema parsing via AJV delivers lower request latency and higher request throughput than traditional frameworks (Kuffel & Walter, 2024; Pratama & Raharja, 2023). Notably, the average latency observed during the concurrent load test (95.4 ms) was lower than that of the single-user response test (140.16 ms). This behavior is attributable to database connection pool reuse and internal query caching mechanisms within PostgreSQL, which optimize execution paths under sustained connection loads. Under extreme stress testing (200 Virtual Users), the platform handled over 30,000 requests before reaching connection saturation at a 26.35% error rate without service crashes, validating the fault tolerance of Node.js non-blocking I/O routines under peak traffic. The automated recommendation engine proved effective in addressing the classic "double coincidence of wants" problem without incurring high computational overhead. Unlike heavy machine learning models that demand substantial training datasets and compute resources, the adaptive ILIKE keyword extraction algorithm operated well within performance limits, achieving a p95 latency of 205.43 ms under concurrent load. The dual-mode mechanism—shifting seamlessly between keyword-matched items and visit-count popularity fallback—ensures content discovery is maintained even during cold-start scenarios for new accounts. Security testing via OWASP ZAP revealed zero High-risk vulnerabilities, confirming that server-side JWT verification, bcrypt password hashing, and parameterized SQL queries in the repository layer successfully insulated the platform from critical threats such as SQL injection (SQLi) and direct Cross-Site Scripting (XSS). The recorded Medium-risk findings were primarily related to security hardening practices, including Content Security Policy (CSP) headers and Anti-CSRF token enforcement on certain form submissions. These represent straightforward deployment-level configurations rather than structural flaws in the core backend code. Regarding usability, the composite SUS score of 71.91 places the platform in the "Good" category with an "Acceptable" rating according to the benchmark criteria defined by Bangor *et al.* (2009). Items 5 (well-integrated functions, mean 4.294) and 8 (low cumbersomeness, mean 1.706) generated the highest contribution scores, indicating that participants found the core exchange flows coherent and accessible. Conversely, items 6 (perceived inconsistency, contribution 2.294) and 10 (learning curve, contribution 2.294) highlighted areas for user experience refinement. Qualitative feedback revealed that occasional inconsistencies stemmed from UI styling differences between standard EJS views and specialized Webix data components. Future iterations should harmonize frontend styling standards and provide interactive onboarding walkthroughs to minimize initial learning friction.

5. Conclusion and Recommendations

This research has successfully designed, implemented, and empirically evaluated a web-based, token-free digital barter platform engineered using Node.js, Fastify, and PostgreSQL within a layered N-tier architecture. The platform effectively mitigates the fundamental friction of establishing a double coincidence of wants by coupling PostgreSQL Full-Text Search and trigram similarity (pg_trgm) matching with an automated, adaptive recommendation pipeline driven by user search history. Black-Box functional verification confirmed a 100% success rate across all 16 operational test scenarios, validating complete lifecycle workflows from catalog management and direct bilateral proposal negotiations to real-time messaging. System security controls—comprising server-side JWT session management via HTTP-only cookies, bcrypt cryptographic hashing, and AJV schema validation—successfully eliminated high-risk vulnerabilities during OWASP ZAP active scanning. Furthermore, usability evaluation via the System Usability Scale (SUS) with 17 representative respondents yielded a composite score of 71.91, placing the platform in the "Good" category with an "Acceptable" rating and confirming high user reception and functional coherence. Despite these achievements, several technical and operational boundaries must be acknowledged. First, the application is strictly deployed as a responsive web system and currently lacks native mobile applications tailored for Android or iOS ecosystems. Second, the automated recommendation mechanism relies on deterministic keyword extraction via ILIKE pattern matching and popularity-based fallback sorting, rather than advanced machine learning algorithms capable of capturing latent contextual preferences. Third, while basic input validation and session security are fully operational, comprehensive security hardening—specifically rigorous Content Security Policy (CSP) headers and global anti-CSRF token enforcement—remains pending. Finally, empirical stress testing revealed connection queue saturation at 200 concurrent virtual users, establishing an operational boundary that warrants architectural scaling before large-scale commercial deployment. To address these limitations, several directions are outlined for future research and engineering. First, native mobile clients for Android and iOS should be developed to enhance ubiquitous mobile access and streamline peer-to-peer bartering. Second, the partner matchmaking engine should transition from rule-based keyword matching to personalized machine learning models, such as collaborative filtering or hybrid graph neural networks, to improve recommendation precision. Third, platform security should be hardened by implementing comprehensive CSP directives, subresource integrity verification, and explicit CSRF middleware across all state-altering requests. Finally,

backend infrastructure should be enhanced through Node.js cluster mode execution, Redis distributed caching, and horizontal containerized replication behind a reverse proxy load balancer to support high-throughput concurrent transaction volumes.

References

Alam, M. I., Sahoo, S., Sharma, S., & Verma, A. (2025). Barterchain: A blockchain-based barter system in smart cities. *Digital Finance*, *7*(4), 871–900. https://doi.org/10.1007/s42521-025-00133-8

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, *4*(3), 114–123.

Brooke, J. (1996). SUS: A quick and dirty usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland (Eds.), *Usability evaluation in industry* (pp. 189–194). Taylor & Francis.

Daniswara, B. A., & Susetyo, Y. A. (2024). Rancang bangun API management pada aplikasi SSC menggunakan framework Webix di PT XYZ. *JIKA (Jurnal Informatika)*, *8*(2), 238–245. https://doi.org/10.31000/jika.v8i2.10977

Fauvelle, M. (2025). The trade theory of money: External exchange and the origins of money. *Journal of Archaeological Method and Theory*, *32*(1), Article 20. https://doi.org/10.1007/s10816-025-09694-9

Firmansyah, D., Purwanto, H., Wiharko, T., & Gustian, R. (2023). Pemanfaatan token dalam pengembangan sistem informasi barter barang berbasis web menggunakan framework Laravel. *INTERNAL (Information System Journal)*, *6*(2), 101–114. https://doi.org/10.32627/internal.v6i2.850

Han, J., & Choi, Y. (2024). *Analyzing performance characteristics of PostgreSQL and MariaDB on NVMeVirt* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2411.10005

Kim, J., Conte, M., Oh, Y., & Park, J. (2024). From barter to market: An agent-based model of prehistoric market development. *Journal of Archaeological Method and Theory*, *31*(3), 1232–1271. https://doi.org/10.1007/s10816-023-09637-2

Klimek, B. M., & Skublewska-Paszkowska, M. (2021). Comparison of the performance of relational databases PostgreSQL and MySQL for desktop application. *Journal of Computer Sciences Institute*, *18*, 61–66. https://doi.org/10.35784/jcsi.2314

Kuffel, S., & Walter, B. (2024). Performance comparison of popular Node.js web frameworks. In *Proceedings of the 20th International Conference on Web Information Systems and Technologies (WEBIST 2024)* (pp. 112–123). SCITEPRESS.

Libertino, L. F. C. (2020). Performance analysis and monitoring of computational resources with a comparison between Node.js and PHP and .NET Core platforms. *International Journal of Development Research*, *10*(5), 35539–35550. https://doi.org/10.37118/ijdr.18695.05.2020

Molina-Jiménez, C., al Nakib, H. D., Song, L., Sfyrakis, I., & Crowcroft, J. (2020). *A case for a currencyless economy based on bartering with smart contracts* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2010.07013

Nawawi, A., & Komalasari, R. (2025). Perancangan aplikasi barter service mobile berbasis React Native dan Laravel. *Digital Transformation Technology (Digitech)*, *5*(1), 1–12. https://doi.org/10.47709/digitech.v5i1.6500

Pratama, I. P. A. E., & Raharja, I. M. S. (2023). Node.js performance benchmarking and analysis at Virtualbox, Docker, and Podman environment using Node-Bench method. *JOIV: International Journal on Informatics Visualization*, *7*(4), 2240–2246. https://doi.org/10.62527/joiv.7.4.1762

Pratama, S. D., & Dadaprawira, M. N. (2023). Pengujian black box testing pada aplikasi Edu Digital berbasis website menggunakan metode equivalence dan boundary value. *Jurnal Teknologi Sistem Informasi dan Sistem Komputer TGD*, *6*(2), 560–569. https://doi.org/10.53513/jsk.v6i2.8427

Putra, Y. Y., Purwaningrum, O., & Winata, R. H. (2022). Perbandingan performa respon waktu kueri MySQL, PostgreSQL, dan MongoDB. *Jurnal Sistem Informasi dan Bisnis Cerdas (SIBC)*, *15*(1), 39–48. https://doi.org/10.33005/sibc.v15i1.7

Salunke, S. V., & Ouda, A. (2024). A performance benchmark for the PostgreSQL and MySQL databases. *Future Internet*, *16*(10), Article 382. https://doi.org/10.3390/fi16100382

Santi, P. A. D. A., Afwani, R., Albar, M. A., Anjarwani, S. E., & Mardiansyah, A. Z. (2022). Black box testing with equivalence partitioning and boundary value analysis methods. In *Proceedings of the First Mandalika International Multi-Conference on Science and Engineering 2022 (MIMSE 2022)* (pp. 207–219). Atlantis Press. https://doi.org/10.2991/978-94-6463-084-8_19

Saravanos, A., & Curinga, M. X. (2023). Simulating the software development lifecycle: The waterfall model. *Applied System Innovation*, *6*(6), Article 108. https://doi.org/10.3390/asi6060108

Shuhaiber, A., Alfayadhi, H. S. A., AlAwlaqi, M. M., & Ali, M. A. A. (2023). 'E-Barter' exchanging system: Toward a smart and sustainable community. In *Lecture Notes in Networks and Systems* (Vol. 595, pp. 355–371). Springer. https://doi.org/10.1007/978-981-19-7660-5_31

Sunarya, P. A., Rahardja, U., Santoso, N. P. L., Mulyati, Mustofa, K. I., & Bennet, D. (2025). Pengaruh metode Waterfall dalam penyempurnaan proses pengembangan sistem informasi akademik secara sistematis. *Technomedia Journal*, *9*(3), 360–373. https://doi.org/10.33050/tmj.v9i3.2421

Syahrohim, I., & Parhusip, J. (2026). Pengukuran usability aplikasi web menggunakan SUS (System Usability Scale) dan pengujian black box pada website IconPlus-Warehouse. *Jurnal Informatika dan Teknik Elektro Terapan*, *14*(1), 1–10. https://doi.org/10.23960/jitet.v14i1.8193

Taskinsoy, J. (2023). *The reincarnation of barter trade & barter economy* [Preprint]. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.4456717

Tu, Z. (2023). Research on the application of layered architecture in computer software development. *Journal of Computing and Electronic Information Management*, *11*(3), 34–38. https://doi.org/10.54097/jceim.v11i3.08

Tullis, T. S., & Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Proceedings of the Usability Professionals' Association (UPA) 2004 Conference* (pp. 7–11). UPA.

Zapata, J., & Zapata, J. G. (2024). *MySQL vs PostgreSQL: A comparative analysis of relational database management systems (RDBMS) technologies response time in web-based e-commerce* [Research report]. ResearchGate. https://doi.org/10.13140/RG.2.2.24791.69288