Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Integration of Chatbot and Complaint Website Using Agile Scrum with Load Testing and UAT

Galih Adi Winasis[1)*], Erba Lutfina[2)] , Galuh Wilujeng Saraswati[3)]
[1,2,3)]Universitas Dian Nuswantoro, Indonesia
[1)]galih.adi.w05@gmail.com, [2)]erba.lutfina@dsn.dinus.ac.id, [3)]galuhwilujengs@dsn.dinus.ac.id

Submitted :January 30, 2026 | **Accepted** : Mar 2, 2026| **Published** : April 2, 2026

Abstract: This study investigates an integrated public complaint service that combines a non-AI, rule-based WhatsApp chatbot, a web-based administrative dashboard, and a RESTful API to improve early response, status traceability, and ticket-based two-way communication. The system was developed using an Agile Scrum approach, implementing the chatbot in Node.js, the backend services and dashboard in Laravel, and PostgreSQL as the centralized database, while real-time dashboard updates were delivered via WebSocket. Evaluation was conducted through User Acceptance Testing (UAT) for core functional flows and RESTful API load testing using Apache JMeter under gradual-load conditions (Typical Busy, Peak, Stress) and an extreme surge condition (Spike/Burst). The UAT results indicate that all core scenarios passed, covering ticket-based complaint submission, duplicate prevention via a one active ticket per WhatsApp number rule, administrator validation and routing, and real-time conversation synchronization within the ticket context. Under gradual-load conditions, all evaluated endpoints maintained a 0% error rate with sub-second average latency in the range of a few hundred milliseconds, indicating stable baseline behavior as workload increased progressively. Under Spike/Burst, the system remained error-free but latency increased, with average response times of 6,593 ms for create complaint, 18,010 ms for status tracking, 18,321 ms for chat message, and 14,308 ms for mixed load, with throughputs of 7.06 req/s, 2.62 req/s, 2.05 req/s, and 5.90 req/s, respectively. Overall, the results demonstrate end-to-end functional feasibility, stable baseline performance under gradual load, and a resilience boundary under extreme surge, motivating targeted optimization of synchronous processing, history retrieval, and payload serialization to improve Spike/Burst time responsiveness.

Keywords: Public Complaint Service, WhatsApp Chatbot, RESTful API, Agile Scrum, User Acceptance Testing, Load Testing

INTRODUCTION

Citizen complaint services constitute a core public governance mechanism that enables residents to submit grievances, aspirations, and incident reports in a traceable manner. In practice, persistent limitations remain in initial responsiveness and status traceability. Fragmented reporting through basic web forms, short messages, or manual communication shifts verification and classification into administrative routines performed by officers. This condition increases response time, delays confirmation, and raises the risk of unattended cases. A village-level web-based complaint platform study highlights centralized recording and status monitoring as prerequisites for transparency and follow-up control (Parisca et al., 2025). Therefore, improving responsiveness requires more than additional reporting channels; it requires an interaction mechanism that provides prompt feedback while maintaining data consistency and handling continuity.

A non-AI rule-based public-service chatbot is suitable for standardized procedures and recurrent inquiries, because responses can be curated by the agency and delivered consistently. A sub-district government chatbot implementation reports an average response time of approximately 1.7 seconds in specific testing scenarios, indicating potential improvements in early-stage responsiveness (Huda et al., 2024). Nonetheless, service quality depends on backend integration resilience. A responsive interface does not guarantee stability under increasing access loads. A JMeter-based load testing study demonstrates that response time increases as load rises, supporting

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

the need for performance measurement to ensure compliance with acceptable service thresholds (Raweyai & Widiasari, 2024).

This study proposes a ticket-based integrated complaint service that connects a non-AI WhatsApp chatbot and a complaint website through a RESTful API backbone and a centralized database. The chatbot supports structured complaint intake, early confirmation, and guidance toward the appropriate reporting flow. The complaint website functions as the administrative hub for ticket management, status tracking, and follow-up actions by government staff. The proposed solution is inherently complex because it integrates multiple components Node.js for the chatbot, Laravel for the RESTful API and complaint website, and PostgreSQL as the data backbone while serving two primary actors with distinct operational needs: citizens and administrators. Such complexity requires an iterative development approach, because public-service workflows often evolve through operational evaluation and field feedback. An SLR in the public sector identifies dominant agile-adoption challenges in the "development approach & lifecycle" and "project work" domains, notably requirement volatility, cross-stakeholder coordination, and governance constraints (Mekkawati et al., 2024). Accordingly, this study adopts Agile Scrum. Scrum is grounded in empiricism through transparency, inspection, and adaptation within sprint cycles, which aligns with complex development settings requiring frequent feedback incorporation (Schwaber & Sutherland, 2020). Scrum-based application development has also been reported to facilitate structured iterations through sprint cycles and periodic evaluation (Rachmawati et al., 2023).

To date, limited empirical evidence has examined the end-to-end integration of a non-AI, rule-based WhatsApp chatbot with a complaint website through a RESTful API layer as the core integration backbone in a ticket-driven workflow. Most existing studies either focus on public-service chatbots as a fast-response interface or concentrate on web-based complaint systems as recording and monitoring platforms, leaving limited evidence on cross-component synchronization (chatbot, RESTful API, website) for ticket-state governance, cross-department routing, and two-way ticket-based communication (Huda et al., 2024; Safitri & Rosadi, 2021; Agustian & Yuliana, 2024; Parisca et al., 2025; Rahman et al., 2025). Moreover, evaluation in prior work is typically partial, as acceptance testing often targets only one component and does not validate user acceptance across the chatbot, RESTful API, and administrative dashboard within a unified multi-component UAT protocol (Asrin, 2024; Budisaputro et al., 2024). In addition, RESTful API performance is frequently treated as an implicit assumption, even though the API layer governs real-time status tracking and conversation synchronization under concurrent load; load testing studies and comparative analyses of API performance tools indicate that response time tends to increase as workload grows and should therefore be measured explicitly at the API level (Raweyai & Widiasari, 2024; Khlamov et al., 2025; Di Meglio et al., 2023). Hence, a clear research gap remains in providing end-to-end evidence that combines non-AI chatbot–website integration, Scrum-based iterative development, and a combined evaluation framework using multi-component UAT and RESTful API load testing (Schwaber & Sutherland, 2020; Mekkawati et al., 2024; Rachmawati et al., 2023).

This study aims to integrate a chatbot and a complaint website using Agile Scrum to accelerate initial responses, reduce unattended cases through centralized recording, and enhance transparency through real-time status tracking. The contributions include an integration architecture for chatbot–RESTful API–website with PostgreSQL as the central datastore, an implementation using Node.js and Laravel, and a quality evaluation using UAT and load testing. UAT validates functional fit and user acceptance for the chatbot, RESTful API, and website, consistent with acceptance-testing practice in government applications and service systems reporting high acceptance when functions meet operational needs (Asrin, 2024; Budisaputro et al., 2024). Load testing quantifies RESTful API performance under increasing load to assess service readiness in measurable terms (Raweyai & Widiasari, 2024).

LITERATURE REVIEW

Web-based complaint studies emphasize centralized recording and status tracking for service traceability, yet early-stage interaction tends to remain passive (Parisca et al., 2025). To improve initial responsiveness, rule-based chatbots report fast response time in testing scenarios, while user acceptance can be validated through high UAT results in public-service chatbot deployments (Huda et al., 2024; Agustian & Yuliana, 2024). From a readiness perspective, JMeter-based load testing indicates that response time increases as load rises, supporting the need for performance measurement prior to real-world use under higher access demand (Raweyai & Widiasari, 2024). For development governance, Scrum provides an empirical framework based on transparency, inspection, and adaptation, which fits complex public-service projects (Schwaber & Sutherland, 2020).

Table 1. Comparative Table

Researcher(s)	Research Object	Key Findings	Limitations

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Parisca et al. (2025)	Web-based complaint system	Traceability improves through structured recording and administrator status tracking.	Initial feedback remains passive; instant response and two-way interaction are not emphasized.
Rahman et al. (2025)	Web-based complaint information system	Digital reporting improves documentation and handling through a centralized web channel.	Multi-channel integration and API-layer performance evaluation are not addressed.
Huda et al. (2024)	Rule-based public-service chatbot	Fast response is reported (≈1.7 seconds) in reported test scenarios for government information services.	End-to-end integration with ticket-based complaint management and API load testing is not evaluated.
Agustian & Yuliana (2024)	Website-based public-service chatbot	High acceptance is reported via UAT (93.12%).	API performance and full chatbot complaint dashboard integration are not assessed.
Raweyai & Widiasari (2024)	Apache JMeter load testing (JUTIF, SINTA 2)	Response time increases as load grows; load testing supports service readiness assessment using response time and failure metrics.	Focuses on a website context, not RESTful API as an integration backbone for chatbot website synchronization.
Khlamov et al. (2025)	API-based performance testing tools (JMeter vs Postman)	JMeter is more suitable for heavier workload profiles and complex load-testing scenarios.	Tool comparison focus; not an end-to-end public service complaint workflow study.
Mekkawati et al. (2024)	Agile project management in the public sector	Identifies dominant challenges and solutions for agile adoption in public-sector projects (e.g., requirement volatility, coordination, governance).	Literature synthesis; does not provide end-to-end empirical evidence for a specific integrated complaint system.

This mapping indicates a research gap in end-to-end integration of a non-AI rule-based chatbot with a complaint website via RESTful APIs, combined with cross-component acceptance validation and API-layer performance measurement. This study addresses the gap through Agile Scrum development and a combined evaluation using UAT and RESTful API load testing as the integration backbone.

METHOD

Research Design and Development Approach

This study applies a software engineering, system development approach focusing on integrating a public complaint service through a non-AI WhatsApp chatbot, a complaint website, and a RESTful API as the main integration layer. The development process adopts Agile Scrum to support rapid iterations, stakeholder involvement, and continuous adaptation to dynamic public-service requirements (Schwaber & Sutherland, 2020). The system emphasizes faster service response, ticket-based tracking, bidirectional communication, and quality evaluation through User Acceptance Testing (UAT) as well as performance evaluation through RESTful API load testing.

Integrated System Architecture

Fig. 1 Design Architecture System

This section provides the primary description of the system architecture; subsequent sections refer to Figure 1 to avoid repeating the same technical overview. The high-level architecture is shown in Figure 1, consisting of three core components: (1) a rule-based Node.js WhatsApp chatbot as the reporter-facing channel; (2) a Laravel RESTful API implementing complaint management, routing, conversation handling, tracking/status, rating, and

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

notification triggers; and (3) a Laravel-based complaint website as operational dashboards for call-center admins and agency admins. All data are centralized in PostgreSQL, while real-time updates of complaint lists and chat threads are delivered via WebSocket. When a new complaint is recorded, the backend triggers WhatsApp notifications to admins, reducing monitoring latency and supporting faster response.

Integration follows a single source of truth principle across the database and API. The chatbot is responsible for rule-based input/output only, while validation, persistence, workflow status transitions, and conversation history are handled by the Laravel backend. This improves ticket-status consistency, supports auditing, and simplifies performance evaluation because the main workload is concentrated on RESTful API endpoints.

Requirements and Process Modeling

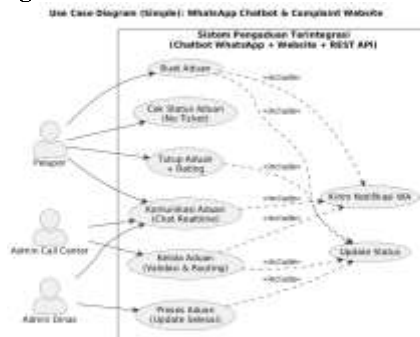

Fig. 2 Use Case Diagram

Functional requirements are modeled using the Use Case Diagram in Figure 2, which defines three main actors Reporter, Call Center Admin, and Agency Admin within the "Integrated Complaint System" boundary. Core functions include complaint submission, ticket-based tracking, real-time complaint communication, call-center validation and routing, agency processing, and reporter-driven closure with service rating. The include relationships for WhatsApp notification and status update indicate that each operational event produces mandatory status transitions and notification triggers to maintain service transparency.

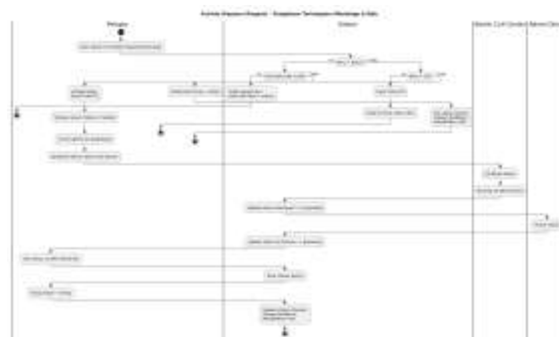

Fig. 3 Activity Diagram Models

The activity diagram in Figure 3, summarizes the end-to-end workflow of the integrated complaint service using a WhatsApp chatbot, a RESTful API, and a web dashboard. The process starts when the reporter selects one of three options: submit a complaint, check status, or close a complaint. For complaint submission, the system checks whether an active ticket already exists for the reporter's WhatsApp number to prevent duplicate reports; if an active ticket is found, the system rejects the new submission and returns the active ticket information and current status. If no active ticket exists, the system validates the input, generates a ticket ID, stores the complaint with the created status, pushes real-time updates to the dashboard, and sends a WhatsApp notification to the call center admin. The call center admin then verifies and routes the complaint to the relevant department, while the system updates the workflow status to received and then forwarded. The department admin processes the complaint until it is marked resolved. The reporter can later check the status using the ticket ID and close the complaint by providing a rating; the system records the feedback, sets the status to closed, and disables the chat channel for that ticket. This workflow strengthens traceability and response speed in public services, consistent with web-based complaint systems and public-sector chatbot implementations (Parisca et al., 2025; Rahman et al., 2025; Huda et al., 2024; Agustian & Yuliana, 2024), and it can be iteratively managed through Agile Scrum to support continuous inspection and adaptation (Schwaber & Sutherland, 2020).

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Workflow Status Definitions

Table 2 standardizes ticket workflow status terminology to ensure consistent usage across the manuscript, including process modeling, UAT scenarios, and load-testing interpretation.

Table 2 Standard Ticket Workflow Status Definitions

Status	Definition	Primary Actor/Trigger
Created	Ticket ID is generated and the complaint is stored as a new case.	System (after reporter submission)
Received	Complaint is acknowledged and verified by the call-center admin.	Call-center admin
Forwarded	Complaint is routed/assigned to the responsible department/agency.	Call-center admin
Resolved	Department marks the complaint as completed.	Department admin
Closed	Reporter closes the ticket and submits rating/feedback; chat is disabled for the ticket.	Reporter

User Interface Design

Fig. 4 User Interface Design Chatbot and Dashboard Admin

The user interface design is shown in Figure 4, combining the WhatsApp chatbot interface as the reporter channel and the website/dashboard interface as the admin operational channel. The chatbot uses a numeric menu to maintain consistent input for a rule-based (non-AI) chatbot, enabling complaint submission, ticket-based tracking, and closure with rating/comments through a concise flow. On the admin side, the dashboard supports ticket list monitoring, complaint details, workflow status management, and ticket-based chat modules for cross-role coordination. This separation optimizes ease of access for reporters while preserving process control for administrators, and the overall interface functionality is verified through UAT scenarios with measurable acceptance criteria (Budisaputro et al., 2024; Asrin, 2024).

Agile Scrum Development Method

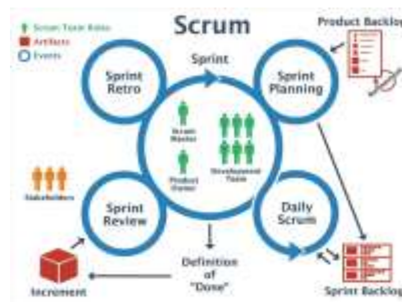

Fig. 5 Scrum Model Design

Development follows Agile Scrum as visualized in Figure 5. Scrum supports transparency, periodic inspection, and adaptation so that a public-service system can be refined using stakeholder feedback without long release cycles (Schwaber & Sutherland, 2020). In the public sector, Agile adoption is also supported by systematic evidence emphasizing iterative delivery and multi-stakeholder collaboration, while noting governance and maturity constraints (Mekkawati et al., 2024).

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Product Backlog

Table 3 outlines the product backlog as prioritized user stories with defined acceptance criteria, aligned with UAT scenarios and process diagrams to ensure measurable outcomes and consistency with the designed service workflow.

Table 3 Product Backlog of the Integrated Complaint System (Concise)

ID	User Story	Priority	Acceptance Criteria (concise)
PB-01	As a reporter, I can submit a complaint via WhatsApp by sending complaint text, incident address, and contact number.	High	Input validated; Ticket ID generated; initial status stored; ticket reply delivered to reporter.
PB-02	As the system, I prevent duplicate complaints so one WhatsApp number can have only one active ticket.	High	If an active ticket exists, new complaint is rejected; system returns active ticket + current status.
PB-04	As a call center admin, I can validate and route complaints to the responsible agency.	High	Status changes “Received” → “Forwarded”; assignment stored; agency notification delivered.
PB-05	As a reporter and admin, we can communicate bidirectionally on an active ticket in real time.	High	Admin messages delivered to reporter; reporter replies shown in web chat; real-time updates work.
PB-06	As a reporter, I can check ticket status via WhatsApp using “Cek” + Ticket ID.	Medium	System returns the latest workflow status; response format is consistent.
PB-07	As a researcher, I can conduct RESTful API performance testing using JMeter load testing.	Medium	Endpoints are test-ready; response time, throughput, error rate are recorded.

Sprint Planning / Sprint Plan

Sprint planning defines sprint objectives and prioritizes backlog items as development commitments, ensuring each sprint delivers verifiable system enhancements validated through UAT and load testing.

Table 4 Sprint Plan for System Development (Concise)

Sprint	Sprint Goal	Key Backlog Items	Deliverable/Increment
Sprint 1	Complete end-to-end “Create Complaint” flow	PB-01, PB-02	WA complaint → ticket → database → admin notification
Sprint 2	Enable admin routing and workflow status for operations	PB-03	Admin list; validation & routing; admin-phase statuses; agency resolution
Sprint 3	Stabilize real-time chat and status tracking	PB-05, PB-06	Web chat ↔ WhatsApp; auto binding by WA number; “Cek” + ticket
Sprint 4	Performance-test readiness	PB-07	JMeter mixed-load ready

Implementation Overview

The implementation follows the integrated architecture in Figure 1. The WhatsApp channel is delivered through a non-AI, rule-based Node.js chatbot, while the complaint website and RESTful API services are implemented in Laravel and backed by PostgreSQL as the single source of truth. Real-time updates for ticket lists and conversation threads are delivered via WebSocket, and WhatsApp notifications are triggered on key workflow events. This implementation scope supports multi-component UAT and RESTful API load testing as the primary evaluation focus.

System Environment and Test Configuration

All tests were conducted on Ubuntu 24.04.2 LTS under KVM virtualization with 16 vCPU, 16 GB RAM, and 500 GB storage. The stack comprised PHP 8.3.17 and Laravel 11.46.2 for the website and RESTful API, Node.js 18.16.0 for the WhatsApp chatbot, and PostgreSQL 16.4 as the centralized database. API performance testing used Apache JMeter, measuring response time, throughput, and error rate under workloads configured through virtual users, ramp-up period, duration, and think time to approximate user pacing and to contrast gradual-load versus Spike/Burst behavior, consistent with common API performance-testing practice (Raweyai & Widiasari, 2024; Khlamov et al., 2025).

JMeter Configuration

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

RESTful API performance testing was configured using Apache JMeter and implemented in a single test plan file (.jmx). The workload parameters for each scenario (virtual users/threads, ramp-up, duration, and think time) are taken directly from the load-testing scenario table, avoiding duplication in this section. The test plan contains separate Thread Groups for gradual-load scenarios and for the Spike/Burst scenario, and it includes HTTP Request Samplers for the integration-critical endpoints defined in the load-testing design, namely Create Complaint, Status Tracking, Chat Message, and Mixed Load. All requests are executed as HTTP/JSON with standard headers (Content-Type: application/json). Test inputs such as WhatsApp number, ticket id, complaint id, and message text are supplied in a controlled manner through CSV Data Set Config to keep input variation consistent and repeatable across runs. Think time is applied via Timers to emulate user pacing in gradual-load scenarios, whereas in Spike/Burst the delay is minimized to represent abrupt concurrency surges. Performance metrics are collected using Summary Report/Aggregate Report to obtain average response time, throughput, and error rate, and outputs are exported in CSV/HTML formats to populate the reported result tables (Raweyai & Widiasari, 2024; Khlamov et al., 2025).

Testing Strategy: UAT and Load Testing

UAT involved 25 respondents representing the operational roles of reporters and administrators. Each respondent executed the defined UAT scenarios, and outcomes were recorded using a pass/fail criterion based on the specified success criteria for each scenario. For quantitative reporting, scenario completion is expressed as a completion rate, defined as the number of respondents who successfully completed a scenario over the total respondents (n/25). In this study, all core scenarios achieved full completion across respondents (25/25), confirming that the functional workflow can be executed consistently in the operational context.

System evaluation uses two testing streams. First, User Acceptance Testing (UAT) validates the chatbot, website, and RESTful API using the scenarios in Table 5 to confirm functional compliance and operational acceptance (Budisaputro et al., 2024). UAT practices are also reflected in government-related application testing using scenario-based acceptance criteria (Asrin, 2024).

Table 5 UAT Design (Summary)

Code	Actor	Component	Test Scenario	Success Criteria
UAT-01	Reporter	WhatsApp Chatbot	Submit complaint (message, address, contact)	System generates a Ticket Number, initial status is saved, confirmation reply is sent
UAT-02	Reporter	WhatsApp Chatbot	Prevent duplicate complaints (1 WA number = 1 active complaint)	New complaint is rejected, system displays active ticket + status
UAT-03	Call Center Admin	Website Dashboard	Validate and route complaint to department	Status changes: Received → Forwarded to department and assigns task to department
UAT-04	Reporter & Admin	Chatbot + Website	Two-way real-time conversation (WA session)	Chat appears on website, messages sent via WA, admin notifications function properly

Second, performance evaluation employs Apache JMeter to load-test the RESTful API endpoints that constitute the integration backbone of the proposed system. The evaluation targets the operationally critical endpoints aligned with the design in Table 6, namely complaint creation, status tracking, chat messaging, and a mixed-load scenario that combines multiple interaction types. Each endpoint is assessed using standard performance indicators, including response time, throughput, and error rate, which are commonly reported in JMeter-based web/API performance testing (Raweyai & Widiasari, 2024). The use of JMeter is also supported by comparative studies on API performance testing tools that position JMeter as suitable for workload-based endpoint evaluation (Khlamov et al., 2025).

Table 6 RESTful API Load Testing Design (Summary)

Code	Endpoint Focus	Scenario	Key Metrics	Output
LT-01	Create Complaint	Multiple users submit complaints simultaneously	Response time, throughput, error rate	JMeter Summary/Aggregate
LT-02	Tracking Status	Multiple users perform status checks (“Check”)	Response time, throughput, error rate	JMeter Summary/Aggregate
LT-03	Chat Message	Multiple users send messages	Response time,	JMeter

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

e-ISSN : 2541-2019
p-ISSN : 2541-044X

		to active tickets	throughput, error rate	Summary/Aggregate
LT-04	Mixed Load	Combination of create + tracking + chat	Response time per endpoint, error rate	JMeter Summary/Aggregate

The performance evaluation focuses on the RESTful API because it governs ticket-state consistency, conversation synchronization, and status-tracking availability across the chatbot and the administrative dashboard. In this study, the endpoint focus defined in Table 6 is operationalized into workload scenarios that reflect both routine service demand and Spike/Burst access conditions. Table 7 therefore summarizes the load profiles used for execution, specifying virtual users (threads), ramp-up period, test duration, and think time to approximate user pacing. Scenarios Typical Busy, Peak, and Stress apply gradual ramp-up to represent traffic growth under operational conditions with approximately 30–50 active complaints, whereas the Spike/Burst scenario uses an extremely short ramp-up to model Spike/Burst concurrency and to observe response-time degradation under high contention (Raweyai & Widiasari, 2024; Khlamov et al., 2025).

Table 7 REST-API Load Testing Scenarios

Scenario	Purpose	Iteration	Virtual Users	Ramp-up	Duration	Think Time
Typical Busy	Realistic baseline busy period	1	50	300 s	15 min	1–3 s
Peak	High busy period (peak hour)	1	100	600 s	20 min	0.5–2 s
Stress	Above-normal load to observe limits	1	150	900 s	25 min	0–1 s
Spike/Burst	Sudden access surge (Spike/Burst concurrency)	1	150	1 s	3 min	0 s

Each scenario was executed once (single run); the reported values were taken from the JMeter Summary/Aggregate Report outputs. Throughput in JMeter reflects completed samples per unit time and is influenced not only by the number of virtual users but also by workload timing parameters. In the gradual-load design, scenarios with higher threads are paired with longer ramp-up and non-zero think time to emulate progressive traffic growth and user pacing. This configuration can keep the effective request arrival rate comparable across gradual-load scenarios, supporting a fair interpretation of performance trends as concurrency potential increases.

Code Availability Statement

The source code, application configuration, and implementation artifacts of the proposed system are not publicly released because they include operational complaint-service components and are subject to institutional security and confidentiality constraints. To support reproducibility of the evaluation, the relevant testing artifacts including workload scenarios, load parameters, and summarized outcomes of UAT and load testing are documented in the manuscript. If required for scholarly assessment, the authors can provide a sanitized replication package (e.g., JMeter configuration files, endpoint structure, and synthetic test data) on a limited basis to the editor/reviewers upon formal request and under appropriate use conditions.

RESULT

The results are presented in two dimensions: functional acceptance through User Acceptance Testing (UAT) and service performance through RESTful API load testing. The evaluation follows the integrated workflow described in the Method section and focuses on end-to-end complaint handling, including ticket submission, duplicate-prevention based on an active-ticket rule, administrative validation and routing, and ticket-based conversation synchronization.

User Acceptance Testing (UAT)

According to Table 8, all UAT scenarios achieved full completion across respondents (25/25), consistent with the Pass status reported in the UAT result table, indicating that the system satisfies the required functional behavior for the core features. In the complaint submission scenario, the system generated a ticket number, stored the initial status, and delivered a confirmation reply to the reporter. In the duplicate-prevention scenario, the system rejected new submissions when an active ticket existed and returned the active ticket identifier and status. From the operational perspective, the validation and routing scenario updated the workflow status from Received to Forwarded and recorded the assignment. In addition, the real-time conversation scenario confirmed message synchronization on the website interface, successful WhatsApp delivery, and administrator notifications. These

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

e-ISSN : 2541-2019
p-ISSN : 2541-044X

outcomes indicate that the integrated channels support ticket-based transparency and operational two-way communication.

Table 8 UAT Result (Execution Summary)

Code	Test Scenario	Success Criteria	Result (Pass/Fail)	Evidence	Notes/ Issue
UAT-01	Submit complaint (message, address, contact)	Ticket number generated; initial status saved; confirmation reply sent	Pass	Ticket ID created; status = "Created"	-
UAT-02	Prevent duplicate complaints (1 WA = 1 active ticket)	New complaint rejected; system returns active ticket + status	Pass	Rejection message + active ticket shown	-
UAT-03	Validate & route complaint to department	Status changes: Received → Forwarded; assignment saved	Pass	Status history updated; assignment recorded	-
UAT-04	Two-way real-time conversation (WA session)	Messages appear on website; WA delivery OK; admin notification works	Pass	Chat thread synced; notif received	-

Load Testing

The RESTful API load-testing results are reported in Tables 9–12 across four endpoint focuses: Create Complaint, Status Tracking, Chat Message, and Mixed Load. The first three scenarios, Typical Busy, Peak, and Stress, are treated as gradual-load conditions to represent baseline operational performance. The Spike/Burst scenario is positioned as a sudden surge test to observe performance degradation under extreme concurrency.

Table 9 Load Testing Typical Busy Result (Summary)

Code	Endpoint Focus	Samples	Avg Response Time (ms)	Std. Dev.	Throughput (req/s)	Error Rate (%)
LT-01	Create Complaint	50	244	69.99	0.171	0.000
LT-02	Status Tracking	50	323	522.72	0.169	0.000
LT-03	Chat Message	50	262	110.86	0.170	0.000
LT-04	Mixed Load	150	276	313.00	0.125	0.000

The Typical Busy scenario in Table 9, the average response time ranges from 244 to 323 ms per endpoint. Create Complaint records an average of 244 ms, Chat Message 262 ms, and Status Tracking is the highest at 323 ms. Throughput is approximately 0.17 req/s for each endpoint, while Mixed Load shows an aggregate throughput of 0.13 req/s with an average response time of 276 ms. All requests in this scenario yield an error rate of 0.000%, indicating no request failures under gradual load.

Table 10 Load Testing Peak Result (Summary)

Code	Endpoint Focus	Samples	Avg Response Time (ms)	Std. Dev.	Throughput (req/s)	Error Rate (%)
LT-01	Create Complaint	100	220	34.85	0.168	0.000
LT-02	Status Tracking	100	277	39.22	0.168	0.000
LT-03	Chat Message	100	242	38.61	0.167	0.000
LT-04	Mixed Load	300	246	44.34	0.126	0.000

The Peak scenario in Table 10, average response time ranges from 220 to 277 ms. Create Complaint records 220 ms, Status Tracking 277 ms, and Chat Message 242 ms. Throughput remains around 0.17 req/s per endpoint, while Mixed Load reports 0.13 req/s with an average response time of 246 ms. The error rate remains 0.000% across all endpoints, indicating consistent request success as the sample size increases.

Table 11 Load Testing Stress Result (Summary)

Code	Endpoint Focus	Samples	Avg Response Time (ms)	Std. Dev.	Throughput (req/s)	Error Rate (%)

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

LT-01	Create Complaint	150	237	38.40	0.168	0.000
LT-02	Status Tracking	150	280	37.90	0.168	0.000
LT-03	Chat Message	150	241	32.93	0.168	0.000
LT-04	Mixed Load	450	253	41.35	0.151	0.000

The Stress scenario in Table 11, average response time ranges from 237 to 280 ms, with Create Complaint at 237 ms, Chat Message at 241 ms, and Status Tracking again the highest at 280 ms. Mixed Load records an average response time of 253 ms with an aggregate throughput of approximately 0.15 req/s. The error rate remains 0.000% for all endpoints, showing that the system continues to serve requests without failures under gradually increased load.

Table 12 Load Testing Spike/Burst Result (Summary)

Code	Endpoint Focus	Samples	Avg Response Time (ms)	Std. Dev.	Throughput (req/s)	Error Rate (%)
LT-01	Create Complaint	150	6593	3843.84	7.06	0.000
LT-02	Status Tracking	150	18010	4338.86	2.62	0.000
LT-03	Chat Message	150	18321	3383.84	2.05	0.000
LT-04	Mixed Load	450	14308	6692.79	5.90	0.000

In contrast to the gradual-load scenarios, the Spike/Burst scenario in Table 12 produces a substantial latency increase. The average response time reaches 6,593 ms for Create Complaint, 18,010 ms for Status Tracking, and 18,321 ms for Chat Message, while Mixed Load records an average of 14,308 ms. Throughput increases to 7.06 req/s for Create Complaint, 2.62 req/s for Status Tracking, 2.05 req/s for Chat Message, and 5.90 req/s for Mixed Load. Despite the marked latency degradation under sudden surge conditions, all endpoints still show an error rate of 0.00%, indicating that requests did not fail even though responsiveness decreased under extreme concurrency.

DISCUSSIONS

The UAT results show that all core scenarios achieved Pass, covering ticket generation with initial status persistence, duplicate-prevention (one WhatsApp number equals one active ticket), administrator validation and routing, and synchronized two-way conversation within the active ticket context. These outcomes support functional acceptance of the end-to-end workflow, consistent with the role of UAT in confirming that system behavior matches operational needs for real use (Asrin, 2024; Budisaputro et al., 2024). The rule-based conversational flow also aligns with public-service chatbot studies emphasizing structured interactions and standardized responses when the chatbot is integrated into government information systems (Huda et al., 2024; Safitri & Rosadi, 2021; Agustian & Yuliana, 2024). In web-based complaint services, traceability and status monitoring are recurring requirements; thus, WhatsApp integration can be interpreted as an access channel that accelerates early interaction while preserving administrative process control via dashboards and status-history records (Parisca et al., 2025; Rahman et al., 2025).

The following interpretation refers to the integrated workflow described in the Method section and focuses on endpoint behavior rather than re-describing the system architecture. Under operational conditions (Typical Busy, Peak, and Stress), the RESTful API maintains sub-second average latency with a 0% error rate, representing a baseline service state under gradual load growth. Across Typical Busy and Peak, endpoint throughput remains in a comparable range, which is methodologically plausible because throughput reflects completed samples per unit time and is sensitive to ramp-up, duration, and think time rather than virtual-user count alone. Accordingly, the gradual-load throughput should be interpreted under the configured pacing (ramp-up, duration, think time), while baseline stability is inferred primarily from consistent sub-second latency and zero errors. The gradual-load label is supported by the scenario configuration, where ramp-up and test duration increase across Typical Busy, Peak, and Stress, spreading user activation over time and reducing instantaneous arrival pressure compared to surge testing. This pacing enables a baseline comparison under progressively introduced load before contrasting it with Spike/Burst. The Spike/Burst scenario is used as an extreme surge test to assess resilience under abrupt concurrency escalation. In this scenario, the system preserves availability (0% error rate), yet latency increases substantially for Status Tracking and Chat Message up to 18–19 seconds. This pattern remains consistent with JMeter-based performance evaluation that relies on response time, throughput, and error-rate metrics to identify

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

SinkrOn
Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

bottlenecks in service endpoints, and it also supports that API-level testing is more appropriate for backend performance assessment than UI-level testing (Raweyai & Widiasari, 2024; Khlamov et al., 2025).

The Spike/Burst latency increase can be explained by the implementation's compute and I/O characteristics. The throughput surge in Spike/Burst is expected because the extremely short ramp-up with minimized delay creates near-simultaneous arrivals, increasing request completions per unit time. Within Spike/Burst, Create Complaint achieves the highest throughput (7.06 req/s) because its execution path is relatively lightweight and dominated by persistence operations (ticket generation and complaint storage). In contrast, Status Tracking (2.62 req/s) and Chat Message (2.05 req/s) execute heavier read–write workloads, including history retrieval and response serialization, which increases service time per request and lowers throughput under the same burst pressure. In addition, the chat endpoint executes chained write operations, evidence-file persistence, large history retrieval, and iterative admin notifications, whereas the tracking endpoint incurs heavier layered resource serialization that expands payload size and processing cost. This interpretation is consistent with RESTful API performance research highlighting how endpoint workload characteristics, query costs, and I/O operations substantially influence latency (Di Meglio et al., 2023). Therefore, Spike/Burst indicates non-failing behavior under extreme surge, but responsiveness degrades and requires optimization to improve burst-time responsiveness on tracking and conversation paths.

Accordingly, the system's operational feasibility is supported by two primary forms of evidence: the successful completion of all core UAT scenarios and performance stability under gradual-load conditions, while the Spike/Burst scenario is treated as a resilience boundary and an optimization driver. Optimization should focus on reducing synchronous work in the response path, improving data-access efficiency, and applying caching mechanisms to reduce data-access latency in web systems; Redis-based caching is referenced as an optimization direction for history-heavy endpoints rather than an evaluated component in this study (Mellati et al., 2026). This study does not define a formal public-service SLA; performance is interpreted comparatively between gradual-load baseline and Spike/Burst surge behavior.

CONCLUSION

This study developed an integrated public complaint service that connects a non-AI, rule-based WhatsApp chatbot and a complaint website through a RESTful API backbone and a centralized ticket datastore. The UAT results confirm end-to-end functional acceptance, as all core scenarios achieved Pass, covering ticket generation with initial status persistence, duplicate-prevention for active tickets, administrator validation and routing, and synchronized two-way conversation within the ticket context (Asrin, 2024; Budisaputro et al., 2024). The findings also support that a rule-based conversational mechanism can deliver standardized public-service interactions when integrated into government service workflows (Huda et al., 2024; Safitri & Rosadi, 2021; Agustian & Yuliana, 2024).

From the RESTful API load-testing perspective, the system demonstrates stable baseline performance under gradual-load scenarios, maintaining sub-second average latency with a 0% error rate across the evaluated endpoints. Under the Spike/Burst surge scenario, the system remains available with 0% errors but exhibits substantial latency degradation on Status Tracking and Chat Message, reaching approximately 18–19 seconds. Rather than indicating a target service level, the 18–19 second Spike/Burst latency marks a resilience boundary under extreme surge conditions. These results indicate that requests can still be completed under extreme concurrency, while responsiveness becomes constrained by endpoint workload characteristics and I/O-intensive processing, consistent with evidence from API performance studies emphasizing the impact of chained operations, payload growth, and data-access costs on latency (Raweyai & Widiasari, 2024; Khlamov et al., 2025; Di Meglio et al., 2023).

Overall, the proposed integration architecture demonstrates end-to-end functional feasibility and stable baseline behavior under gradual-load conditions, as evidenced by the UAT outcomes and the 0% error rate with sub-second average latency in the progressive scenarios. In the gradual-load design, throughput reflects the configured pacing rather than maximum capacity, while baseline stability is supported by sub-second latency and zero errors. No formal public-service SLA is defined; results are interpreted comparatively between baseline and surge behavior. The Spike/Burst results further indicate that the service remains available under extreme concurrency, while Spike/Burst time responsiveness degrades on the tracking and conversation paths, marking the current resilience boundary and motivating targeted optimization. Future work should prioritize reducing synchronous work in the response path, improving data-access efficiency, and applying caching-based mechanisms to mitigate data-access latency, as illustrated by Redis-based optimization for efficient data handling (Asrin, 2024; Budisaputro et al., 2024; Raweyai & Widiasari, 2024; Khlamov et al., 2025; Mellati et al., 2026).

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026
DOI : https://doi.org/10.33395/sinkron.v10i2.15884
e-ISSN : 2541-2019
p-ISSN : 2541-044X

REFERENCES

Abdullah, P. P., Raharjo, T., Hardian, B., & Simanungkalit, T. (2023). Challenges and best practices solution of agile project management in public sector: A systematic literature review. *International Journal on Informatics Visualization, 7*(2), 606-614. https://doi.org/10.30630/joiv.7.2.1098.

Agustian, F., & Yuliana, A. (2024). Aplikasi chatbot pelayanan publik berbasis website (Studi kasus: Sekretariat DPRD Kota Cimahi). *Jurnal Informatika dan Teknik Elektro Terapan (JITET), 12*(3). https://doi.org/10.23960/jitet.v12i3S1.5202.

Asrin, F. (2024). Analisis pengujian menggunakan user acceptance test (UAT) pada aplikasi manajemen notulensi rapat BAPPEDA Kota Pontianak. *Jurnal Jaringan Sistem Informasi Robotik (JSR), 8*(1), 34–41. https://doi.org/10.58486/jsr.v8i1.340.

Budisaputro, C., Anardani, S., Riyanto, S., & Kusdwiadji, A. (2024). Medical record information system testing using user acceptance testing to determine system quality. *Brilliance: Research of Artificial Intelligence, 4*(1). https://doi.org/10.47709/brilliance.v4i1.4451.

Di Meglio, S., Starace, L. L. L., & Di Martino, S. (2023). Starting a new REST API project? A performance benchmark of frameworks and execution environments. *In International Workshop on Software Measurement (IWSM) and the 17th International Conference on Software Process and Product Measurement (MENSURA)*. https://ceur-ws.org/Vol-3543/paper19.pdf.

Huda, M., Kusumaningrum, A. M., & Aninditiyah, G. (2024). Pemanfaatan chatbot untuk meningkatkan layanan informasi publik di instansi pemerintah tingkat kecamatan. *Jurnal Ilmiah Teknologi Informasi Dan Komunikasi (JTIK), 15*(2), 413-424. https://doi.org/10.51903/jtikp.v15i2.1170.

Khlamov, S., Mendielieva, M., Vovk, O., & Deineko, Z. (2025). Comparative analysis of JMeter and Postman for API-based performance testing. *In 2025 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. https://ceur-ws.org/Vol-4048/paper34.pdf.

Mekkawati, H., Raharjo, T., & Yuniarti, R. (2024). Systematic literature review: Challenges and solutions on agile project management in public sector. *JITK (Jurnal Ilmu Pengetahuan Dan Teknologi Komputer), 10*(2), 449-460. https://doi.org/10.33480/jitk.v10i2.5654.

Parisca, Khairul, & Syahputra, Z. (2025). Platform layanan pengaduan masyarakat berbasis web pada Kantor Desa Helvetia. *Jurnal Minfo Polgan, 14*(2), 1550-1557. https://doi.org/10.33395/jmp.v14i2.15065.

Pinto, J. de S., & Leme, R. da S. (2024). Analysis of project management principles with the Scrum framework in systems development: A case study in a public organization. *Brazilian Journal of Operations & Production Management, 21*(2), e20241878. https://doi.org/10.14488/BJOPM.1878.2024.

Rachmawati, O. C. R., Wardani, D. K., Fatihia, W. M., Fariza, A., & Rante, H. (2023). Implementing agile Scrum methodology in the development of SICITRA mobile application. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), 7*(1), 41–50. https://doi.org/10.29207/resti.v7i1.4688.

Rahman, F. N., Fatah, M. F., & Fajriyanto. (2025). Sistem informasi pengaduan masyarakat berbasis website di Desa Curah Jeru. *JuTI (Jurnal Teknologi Informasi), 4*(1), 43–53. https://doi.org/10.26798/juti.v4i1.2029.

Raweyai, S. S., & Widiasari, I. R. (2024). Performance testing of academic website using load testing method supported by Apache JMeter™ at XYZ University. *Jurnal Teknik Informatika (JUTIF), 5*(3), 721–730. https://doi.org/10.52436/1.jutif.2024.5.3.1796.

Safitri, D. N., & Rosadi, M. I. (2021). Rancang bangun penyedia layanan informasi pelayanan masyarakat Kantor Kecamatan Pandaan menggunakan chatbot. *Jurnal Ilmu Komputer dan Desain Komunikasi Visual (JIKDISKOMVIS), 6*(2), 74-83. https://doi.org/10.55732/jikdiskomvis.v6i2.427.

Schwaber, K., & Sutherland, J. (2020). The Scrum Guide: The definitive guide to Scrum (Indonesian version).

Soulani, A. A., Nofiyati, & Ekowati, N. A. (2024). Implementation of low-code programming technology with agile method in developing a petty cash transaction management application (Case study: PT Bank Central Asia Tbk). *Jurnal Teknik Informatika (JUTIF), 5*(3), 941–951. https://doi.org/10.52436/1.jutif.2024.5.3.2303.

Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2021). RESTest: Automated black-box testing of RESTful web APIs. *In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21). Association for Computing Machinery*. https://doi.org/10.1145/3460319.3469082.

Viglianisi, E., Dallago, M., & Ceccato, M. (2020). RESTTESTGEN: Automated black-box testing of RESTful APIs. *In 2020 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. https://doi.org/10.1109/ICST46399.2020.00024.

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 2, April 2026 e-ISSN : 2541-2019
DOI : https://doi.org/10.33395/sinkron.v10i2.15884 p-ISSN : 2541-044X

Mellati, P., Saraswati, G. W., Mahmud, W., Lutfina, E., & Caturkusuma, R. M. (2026). Optimization of web-based printing order management system using Redis database for efficient data handling. *Sinkron: Jurnal dan Penelitian Teknik Informatika, 10*(1), 220-231. https://doi.org/10.33395/sinkron.v10i1.15502.

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.