Volume XI Issue 3 Year 2026 | Page 717-729 ISSN: 2527-9866
Received: ***11-05-2026*** | *Revised:* ***14-06-2026*** | *Accepted:* ***23-07-2026***

Design and Evaluation of a REST API Backend with Payment Integration for AR Self-Service Retail

Aji Sapta Pramulen[1], Jauari Akhmad Nur Hasim[2], Irma Wulandari[3], Fardani Annisa Damastuti[4], Ibrohim Yofid Fananda[5]
[1,2,3,4,5]Department of Creative Multimedia Technology, Politeknik Elektronika Negeri Surabaya, Indonesia
e-mail: aji@pens.ac.id[1], jauari@pens.ac.id[2], irma@pens.ac.id[3], fardani@pens.ac.id[4], ibrohim@pens.ac.id[5]
**Correspondence:* aji@pens.ac.id

Abstract: *Augmented reality (AR) virtual try-on has become common in self-service retail, yet most research still concentrates on the frontend, and the server side and its measured performance receive little attention. This study designed, implemented, and evaluated a REST API backend for AR retail transactions, including end-to-end payment, in a smart-mirror deployment. Built on Laravel, the backend exposes a single JSON contract to both the web-based AR try-on client and an administrative dashboard, protects every privileged endpoint with JWT, and integrates a Midtrans Snap gateway through an asynchronous, SHA-512-verified webhook that reconciles each transaction server-to-server, independently of client connectivity. The backend was evaluated using four methods: black-box testing of twelve CRUD modules, five formal integration scenarios, a load test from one to one hundred concurrent users (about four thousand requests), and a structured expert review. Under the intended single-station workload, login averaged about 0.4 s at a 0.83% error rate and catalog reads about 0.15 s with no errors; the read endpoints remained error-free up to fifty concurrent users, whereas the password-hashing path saturated first. Three experts rated the system 120 of 135 ("very good"). These results apply to the single-station scenario tested, not to multi-store or high-concurrency deployment.*

Keywords: *REST API, Backend Architecture, Payment Gateway Integration, Webhook Reconciliation, Augmented Reality, Self-Service Retail, Performance Evaluation.*

1. Introduction

Interactive multimedia has reshaped retail, and augmented reality (AR) with virtual try-on lies at the centre of this shift: customers can manipulate a product on screen, examine its fit and colour, and decide with greater confidence before purchasing [1], [2], [27]. The most familiar form is the smart mirror, a reflective display with depth sensing and AR rendering that allows shoppers to try on eyewear, clothing, or accessories without handling the item [3], [10]. The mirror, however, represents only half of the product. Behind the glass, the installation must still authenticate the shopper, present a live catalog, host the fitting, and settle a payment, all without staff intervention.

For store operations, the greater value of virtual try-on is that it underpins self-service: the customer proceeds from browsing to checkout independently, which raises operational efficiency [4], [5], [26], [28], [29]. This does not depend on rendering alone. The AR client must also retrieve a current catalog, reflect administrative changes to price and stock, persist order records beyond the session, and delegate payment to a gateway whose response can be trusted before any goods are released [5], [17]. The smart-mirror and AR-retail literature, however, remains focused on the frontend. A typical study details the interface, reports a human-centred design iteration [6], [20], and presents usability scores, while the backend is left as an unspecified data source with no architecture and no performance figures [6], [9]. Browser-based AR renderers such as Three.js widen this gap, since they push every asset, catalog update, and order event through the HTTP

interface in real time [19]; when such prototypes are moved toward production, they scale poorly, their data integration is brittle, and real-time interaction degrades [9], [13]. Yet the backend is precisely where data communication, service integration, and business-process orchestration reside. REST over HTTP remains the default integration style because it is flexible, interoperable, and stateless [7], [12], and a framework such as Laravel adds a mature MVC, an ORM, routing, and token authentication that shorten the path from data model to a working REST service [8], [16]. On that basis, a Three.js smart-mirror client and the admin console can share a single JSON contract, with the business logic kept in one testable location [8], [14].

Most REST API research, however, has been situated within conventional data-management applications [13], [15]. It is rarely paired with interactive AR multimedia, and it generally stops at functional correctness, without reporting response time, throughput, error rate, or behaviour under concurrent load [9], [33]. Fewer studies still follow the loop into a real payment gateway and examine how an asynchronous webhook behaves when it must be reconciled against an order created within an AR session [38], [40]. The link between backend design and retail outcomes has likewise received limited attention. A digital installation is expected not only to provide a satisfactory experience but also to raise operational efficiency and yield measurable value [10], [25], [31]; yet most work stops at the technical layer and does not trace how a backend choice such as webhook-driven payment reconciliation translates into a smoother checkout at the point of sale [17], [25].

Considered together, the pattern is consistent: smart mirrors, AR virtual try-on, and REST API backends have largely advanced on separate tracks. This makes it difficult to construct a system that is simultaneously interactive, robust under realistic load, commercially complete, and validated in the field. This study integrates these strands into a single system: a Laravel backend behind a JWT-secured HTTP interface that manages twelve storefront modules under MVC and incorporates a Midtrans payment integration whose asynchronous webhook closes the commercial loop initiated by the AR mirror.

Building on this motivation, the paper offers four main contributions:

- A concrete three-tier backend architecture that serves a Three.js web-based AR client and a web admin dashboard from a single JWT-secured REST contract, removing the dual-stack split typical of AR retail prototypes.
- A specified and verified asynchronous payment-reconciliation pipeline in which an SHA-512-signed Midtrans webhook applies the authoritative transaction outcome idempotently and server-to-server, closing the AR-to-checkout loop even when the client disconnects.
- A four-method, server-side evaluation (black-box functional testing, formally specified integration scenarios, a multi-user load test with percentile and error-rate reporting across a 1–100 concurrent-user sweep, and a structured expert review), rather than frontend usability scores alone.
- An industrially deployed system at PT. Alton Teknologi Indonesia, validated through stakeholder acceptance and on-site integration testing, together with an explicit, evidence-bounded statement of the conditions (single smart-mirror station) under which the performance claims hold.

Table 1 makes the gap concrete: as far as the authors are aware, no existing work brings together, in one system, an AR client, a fully specified REST contract, a payment-gateway integration with verified asynchronous reconciliation, a quantitative server-side evaluation, and a field deployment. This study addresses all of these dimensions in one integrated, industrially deployed system.

2. Literature Review

The relevant literature falls into four strands, each addressing part of the problem but leaving the integration targeted here unresolved. (1) AR retail and smart-mirror systems. Rauschnabel et al. [1], Hilken et al. [2], and Bonetti et al. [3] establish the consumer-acceptance case for AR in retail, and Huang and Liu [4], Schultz and Kumar [27], and Recalde et al. [28] extend it to virtual try-on. These studies concentrate on the frontend artefact and on user perception, and the server is treated as a generic data source, even though try-on responsiveness is ultimately a backend property. (2) REST and microservice engineering. Fielding [7], Richardson and Amundsen [13], Masse [15], and Wang et al. [9] address endpoint design and contract evolution; Newman [14] and Stauffer [8] document the modular and Laravel idioms adopted here; and recent backend performance-evaluation work [32]–[35] provides measurement methods. None of these, however, addresses the latency and synchronisation constraints imposed by a browser-based AR client [19]. (3) Digital payments and webhook-based reconciliation. The Midtrans reference [17] specifies the protocol and signature scheme; studies of payment security and privacy [36], [37] and of JWT-secured APIs using HMAC-SHA512 [39] establish the security primitives; and recent work on scalable webhook handling [38] and webhook-driven POS integration [40] indicates that the pattern is emerging. None of these, however, measures a complete AR-to-checkout loop end to end. In this context, webhook reconciliation is not merely a convenience feature; it is the mechanism that preserves consistency, reliability, and transaction integrity when the authoritative payment outcome must be applied even though the client may disconnect mid-transaction. (4) Self-service technology and retail operations. Bulander [5], Norman [6], and recent studies of autonomous and unattended stores [29]–[31] document the shift toward staff-free checkout but do not connect it to a measured backend. These strands are interdependent: a unified REST contract is valuable only if it performs under load, and a payment integration is trustworthy only if its reconciliation is verified. Table 1 cross-tabulates these references against six dimensions and identifies the missing intersection.

The four contributions listed above are not individually unprecedented; their novelty lies in being delivered and evaluated together within a single deployed system. As Table 1 indicates, prior AR-retail studies cover at most a subset of these dimensions, and, to the best of the authors' knowledge, none both verify the asynchronous payment-reconciliation loop and report quantitative server-side performance. The contribution is therefore primarily one of engineering and system integration; its scientific value lies in the joint, quantified evaluation of a feature combination that the literature has so far examined only in isolation.

Table 1. Comparison of related work across six capability dimensions

Reference / System	AR client	REST contract	Payment gateway	Webhook recon.	Quant. server eval.	Industrial deployment
Rauschnabel et al. [1]	Yes (conceptual)	No	No	No	No	No
Hilken et al. [2]	Yes	No	No	No	No	No
Bonetti et al. [3]	Yes (review)	No	No	No	No	No
Huang & Liu [4]	Yes	No	No	No	No (UX only)	No
Bulander [5]	No (review)	No	No	No	No	No
Wang et al. [9]	No	Yes	No	No	Partial	No
Newman [14]	No	Yes (microservice)	No	Generic	No	No
Gimpel et al. [25]	No	No	Strategy only	No	No	Yes (case)

This work	Yes (Three.js)	Yes (12 modules, JWT)	Yes (Midtrans Snap)	Yes (SHA-512 verified)	Yes (JMeter + 4 methods)	Yes (PT. Alton Teknologi Indonesia)

3. Methods

The backend followed a five-stage waterfall process [24], shown in Fig. 1: eliciting requirements, designing the system, implementing the code, testing the build, and deploying the result. This sequential model was preferred over an iterative one such as the spiral [23] because the backend requirements were stable from the outset and the industrial timeline favoured a sequential plan-then-build approach rather than repeated cycles. Implementation used PHP 8 on Laravel with MySQL, in an MVC structure in which each storefront entity (product, category, colour, banner, slider, sponsor, team, coupon, background, setting, order, and user) has its own model, resource controller, and Eloquent migration [7], [8]. Every write executes within a database transaction, and every authenticated route is protected by a JWT bearer token issued at login [11], [39]. Requirement analysis. Requirements were elicited through structured interviews with three stakeholders at PT. Alton Teknologi Indonesia, selected to cover the technical, engineering, and operational perspectives of the deployment (Table 2). The interviews produced the functional and non-functional requirements in Table 3, which guided the design and subsequently served as the reference for the expert-review questionnaire.

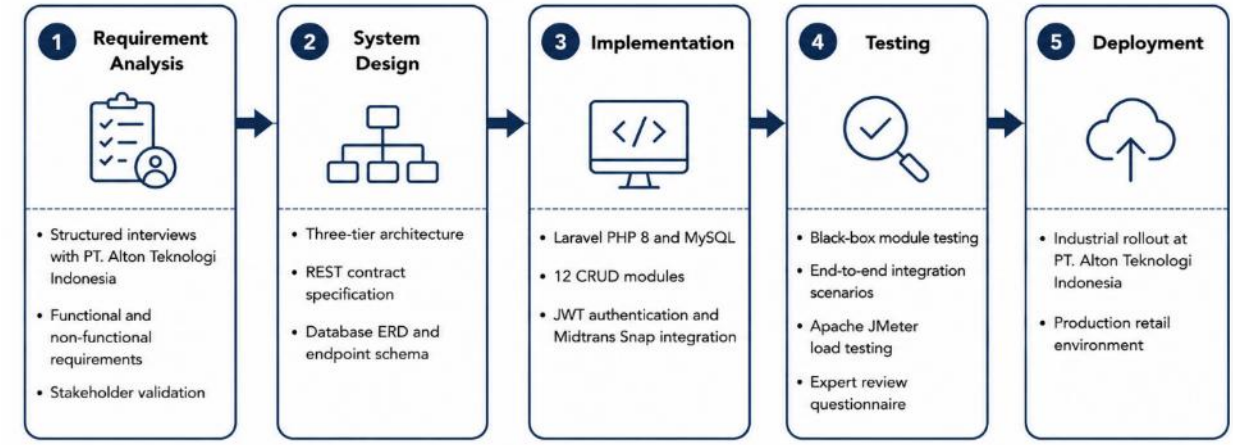


Figure 1. Waterfall methodology: five phases from requirement analysis to deployment.

Table 2. Interviewees and their roles in requirement elicitation

Role	Responsibility in the project	Contribution to requirements
Technical lead (CTO)	Owns the smart-mirror retail deployment	Business goals, security, acceptance criteria
Backend engineer	Builds and maintains the REST API	Endpoint contract, data model, payment flow
Retail operations supervisor	Runs the store and the checkout process	Catalog management, order handling, usability in-store

The requirements are summarised below; the non-functional requirements in particular define the targets against which the evaluation in Section 4 is interpreted.

Table 3. Functional and non-functional requirements

Type	ID	Requirement
Functional	FR-1	JWT-based registration and login for AR and admin clients
Functional	FR-2	CRUD management of twelve storefront entities via the admin dashboard
Functional	FR-3	Single JSON catalog contract consumed by the Three.js AR client
Functional	FR-4	Order creation returning a Midtrans Snap payment token
Functional	FR-5	Asynchronous webhook reconciling payment status to the order record
Functional	FR-6	Role-based access control separating customer and admin operations

Non-functional	NFR-1	Interactive response time within the 1 s usability window at single-station load
Non-functional	NFR-2	0% error rate under the expected single-mirror workload
Non-functional	NFR-3	Signature-verified, idempotent payment reconciliation (transaction integrity)
Non-functional	NFR-4	Stateless REST design enabling later horizontal scaling
Non-functional	NFR-5	Maintainable MVC separation of concerns

The system is organised in three tiers (Fig. 2). The presentation tier hosts two clients side by side: a Three.js smart-mirror web application [19] that renders the AR try-on from glTF/GLB 3D assets [18], and a browser-based admin dashboard used by store operators to manage the catalog. The application tier is the Laravel REST API, which exposes JSON, validates the JWT on every protected route, and delegates persistence to MySQL. The integration tier handles external services: images and 3D assets are served from object storage, and payment is delegated to Midtrans in two steps, a Snap tokenisation call at checkout followed by an asynchronous HTTP notification to a dedicated webhook endpoint once the customer has paid. This arrangement keeps the AR frontend and the admin dashboard on a single request contract, maintains consistent authentication for both clients, and leaves payment responsibility with the certified gateway rather than the application server [14], [36].

The REST contract follows resource-oriented conventions [13], [15]. The registration and login routes accept a JSON credential payload and return a JWT access token on success. Catalog routes return arrays of objects carrying id, title, slug, price, stock, photo, discount, is_featured, and a category reference, the same shape consumed by the Three.js AR client, which therefore renders the catalog without further transformation. Admin mutations require both a valid JWT and an admin role claim. Order creation validates the cart, writes the record with status pending, and returns a Midtrans Snap token that the client passes to the Midtrans widget.

Payment reconciliation is the critical path of the commercial loop and runs through a dedicated webhook endpoint. When the customer pays in the Midtrans Snap widget, Midtrans issues an asynchronous HTTP notification carrying the order id, gross amount, transaction status, and a SHA-512 signature key. The handler verifies the signature against the order's server secret [39], maps the incoming status onto the local order lifecycle (pending, settlement, capture, expire, cancel, or deny) [17], and updates the order atomically. Only a successful settlement or capture marks the order as paid and triggers the stock adjustment. Consequently, a purchase initiated within an AR session completes correctly even if the client disconnects immediately after paying, because the authoritative status is received server-to-server rather than through the client [38], [40].

The backend was evaluated in four ways, functional, integrative, performance, and expert, each with a defined instrument. (i) Black-box functional testing exercised every CRUD module from the admin dashboard and from Postman (the test instrument), checking each endpoint against its contract under both valid and invalid input partitions. (ii) Integration testing defined five formally specified end-to-end scenarios, each with explicit preconditions, steps, and a priori acceptance criteria (Table 6), exercising the handoffs among clients, backend, and external services, including the full Midtrans payment loop. (iii) Performance testing used a multi-threaded HTTP load generator that issues concurrent requests and records per-request latency, methodologically equivalent to Apache JMeter [22] and consistent with recent API load-testing practice [33], [34], [35]. The live deployment was swept from one to one hundred virtual users, with 120–300 samples per level, capturing mean, percentile (p50/p90/p95/p99), and dispersion of response time, together with error rate, throughput, and network volume. (iv) The expert review engaged three independent experts who accessed the running system and completed a nine-item questionnaire spanning login,

backend implementation, and API implementation; the items were derived from the elicited requirements to establish content validity, each item was rated on a five-point Likert scale, and the aggregate was computed using the percentage-index formula in Eq. (2) [21].

Table 4 summarises the principal REST endpoints, indicating the client that consumes each route, the authentication required, and its role in the AR-to-checkout loop.

Table 4. Principal REST endpoints of the backend

Endpoint	Method	Consumer	Auth	Function
/api/register	POST	Web / AR	None	Create a customer account
/api/login	POST	Web / AR	None	Issue a JWT bearer token
/api/products	GET	Web / AR	JWT	Retrieve catalog for rendering
/api/categories	GET	Web / AR	JWT	Retrieve category list
/api/colors	GET	AR	JWT	Retrieve colour variants for try-on
/api/admin/products	POST / PUT / DELETE	Web	JWT + admin	Manage product records
/api/orders	POST	Web / AR	JWT	Create order, return Snap token
/api/payments/notification	POST	Midtrans	Signature	Reconcile payment asynchronously

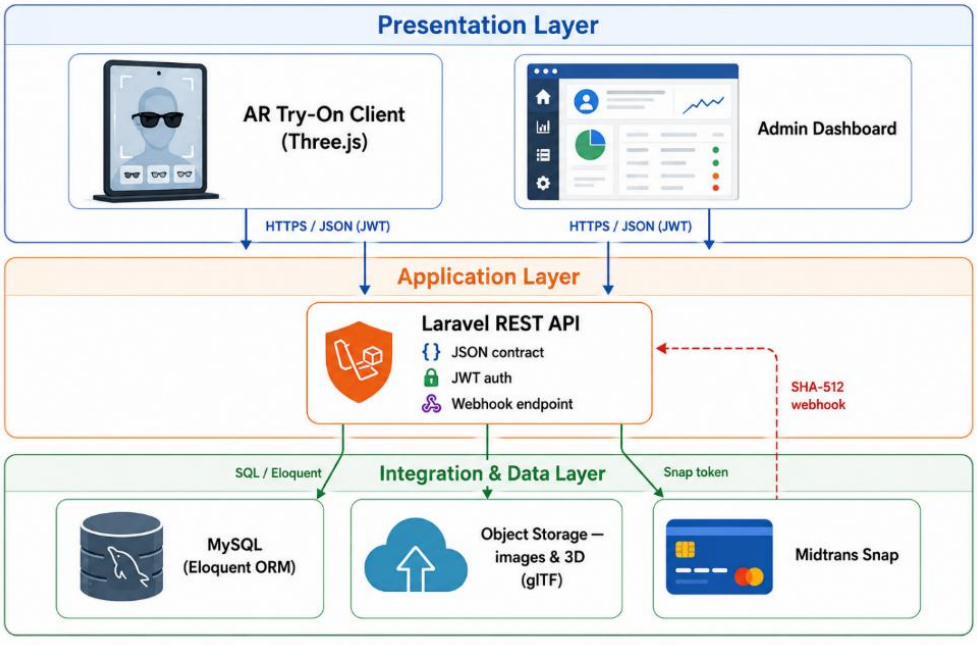


Figure 2. Three-tier architecture of the REST API backend (AR client and admin dashboard to MySQL and Midtrans).

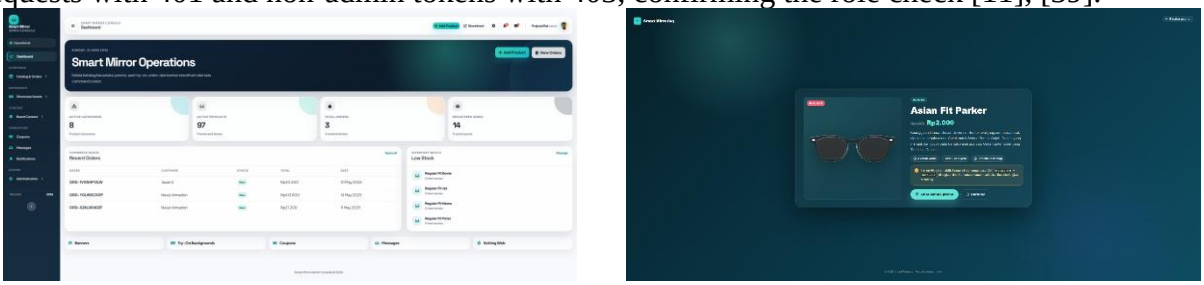
4. Results and Discussion

This section presents the evaluation in the sequence in which the experiments were carried out. Section A describes the implemented architecture and endpoint surface, Section B the Midtrans webhook pipeline, Section C the multi-user load test, Section D the integration test covering the AR-to-payment loop, Section E the black-box functional test, and Section F the expert review.

A. Implemented Architecture and Endpoint Surface

The backend was implemented as described in Section 3, and it served both clients through a single REST contract without friction; Fig. 3(a) and (b) show the deployed admin console and the AR

storefront. The system is in production and publicly accessible at https://sm.multimediaimaging.id (accessed June 2026). Twelve Eloquent-backed modules were exposed through resource controllers, and JWT authentication was applied uniformly on every protected route. Postman verification confirmed the contract: registration returned 201 Created, login returned 200 OK with a JWT from the Laravel auth guard, and the catalog route returned 200 OK with an array shaped exactly as the Three.js AR client consumes [13]. Admin endpoints rejected unauthenticated requests with 401 and non-admin tokens with 403, confirming the role check [11], [39].



(a) (b)

Figure 3. Deployed system: (a) admin dashboard; (b) AR try-on client.

B. Midtrans Webhook and Order Reconciliation

The payment-notification webhook behaved correctly across all six Midtrans transaction statuses. On the main settlement path, the customer initiates checkout from the AR client, the order endpoint returns a Snap token, the customer pays through the Midtrans widget, and the order moves from pending to paid as soon as Midtrans posts its server-to-server notification. Synthetic requests carrying a SHA-512 key that did not match the order's server secret were rejected at the signature check, so that only genuine Midtrans notifications could alter the order lifecycle [17], [39]. The design objective, never leaving an order inconsistent when the client disconnects immediately after payment, was met in practice: the authoritative status arrives over the webhook and is persisted by a database transaction independent of the client session, and because reconciliation is idempotent, a duplicate notification re-applies the same final state rather than processing the order twice [38], [40].

C. Multi-User Load Test of the Authentication Endpoint

The authentication endpoint was selected as the primary load-test target for two reasons: it gates every session, and it is the most demanding interactive path, being write- and CPU-bound owing to password hashing and session creation. The revised evaluation sweeps concurrency from one to one hundred virtual users against the live production deployment, with 120–300 samples per level (about four thousand requests) and full percentile reporting. The load was generated from an external client over the public internet, so the figures reflect realistic wide-area conditions rather than idealised loopback latency. The statistics in Table 5 follow standard definitions [22], [33]: arithmetic mean, sample standard deviation, median (p50), and the 90th/95th/99th percentiles, since tail latency, rather than the mean alone, governs the worst-case experience.

Table 5 first establishes the single-station operating point: under a sequential workload (one user, 120 samples), authentication averaged 395 ms (p95 443 ms, p99 697 ms) at a 0.83% error rate, while catalog reads averaged 167 ms (p95 180 ms) with no errors. Table 5 also reports the authentication endpoint across the concurrency sweep, visualised in Fig. 4(a).

At the single-station operating point, the 395 ms mean for authentication and the 136–167 ms catalog reads fall well within the one-second window that Nielsen [21] identifies for uninterrupted user flow, with the read paths approaching the 0.1 s “instant” threshold. Relative dispersion is reported as the coefficient of variation ($CV = \sigma/\bar{t}$, expressed as a percentage).

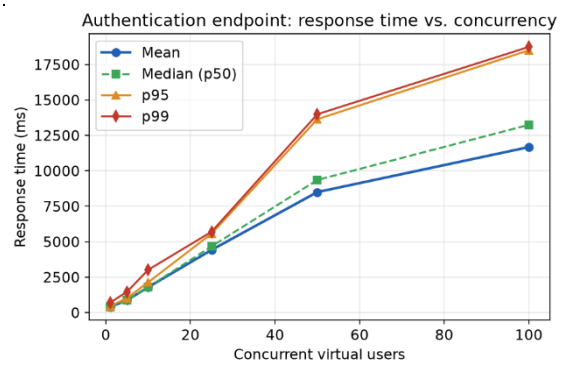
At this operating point (σ = 65 ms, $\bar{t}$ = 395 ms), CV is approximately 16.6%, remaining near 17% up to five concurrent users and widening to 39–42% beyond twenty-five as queueing develops. Sustained throughput λ is obtained by dividing the completed-request count N by the total elapsed time T.

Throughput peaks at about 5.6 req/s for authentication near five concurrent users and then declines as contention increases, whereas the read endpoints sustain 10–15 req/s. The two endpoint classes degrade differently (Fig. 4(b) and 4(c)). The read endpoints that carry most AR-client traffic, catalog, storefront, and product detail, maintain a 0% error rate up to fifty concurrent users and degrade gracefully. The authentication endpoint, constrained by bcrypt hashing and session writes, saturates first: its error rate rises from below 2% at five users to 8.3% at twenty-five, 16.7% at fifty, and 33.3% at one hundred. This is a concrete limitation with a clear capacity-planning implication: a single backend instance is sufficient for a sequential single-mirror station, but a fleet-wide or peak-concurrency rollout would require horizontal scaling, connection pooling, or hardware-accelerated hashing.

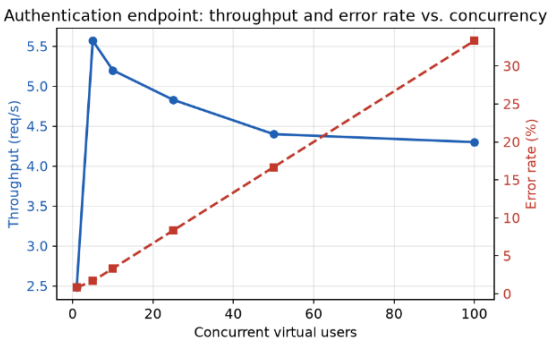
These findings are bounded by the experimental design and context. The sweep locates the first saturation point, but the single-instance production host (rather than a benchmarking cluster) and the wide-area path render the absolute latencies conservative upper bounds rather than best-case figures. By design, transaction risk is concentrated at the reconciliation path: because that step is server-to-server, signature-verified, and idempotent, a client disconnection or a duplicate notification does not corrupt order state, although a prolonged gateway outage would delay reconciliation rather than lose it, and clock-skew-induced delays remain untested. The figures should therefore be interpreted as evidence that the backend is appropriately sized for the intended single-station scenario, not as a general scalability claim.

Table 5. Backend performance: single-station baseline and authentication concurrency sweep (1–100 users).

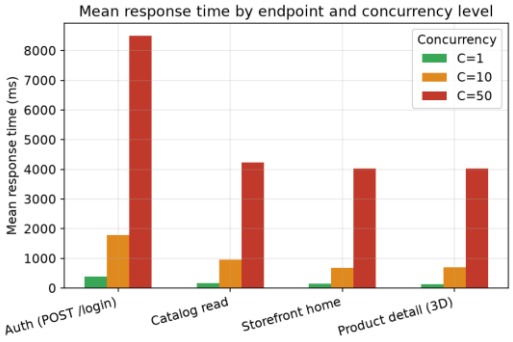
Scenario	Samples	Mean (ms)	p95 (ms)	p99 (ms)	CV (%)	Error (%)	Thr. (req/s)
Authentication (login), 1 user	120	395	443	697	16.6	0.83	2.48
Catalog read, 1 user	120	167	180	188	25.7	0.00	6.00
Storefront home, 1 user	120	140	158	166	27.1	0.00	7.12
Product detail (3D), 1 user	120	136	155	221	29.4	0.00	7.34
Authentication, 5 users	300	865	1021	1461	16.7	1.67	5.57
Authentication, 10 users	300	1787	2132	3012	18.5	3.33	5.20
Authentication, 25 users	300	4419	5566	5702	23.1	8.33	4.83
Authentication, 50 users	300	8498	13640	13988	39.0	16.67	4.40
Authentication, 100 users	300	11671	18499	18739	42.2	33.33	4.30



(a)



(b)



(c)

Figure 4. Authentication under concurrency: (a) response time; (b) throughput and error rate; (c) mean response time by endpoint.

D. End-to-End Integration Test

The five end-to-end integration scenarios defined in Section 3 were executed against the deployed backend; their preconditions, steps, and acceptance criteria are specified in Table 6. For each scenario, the acceptance criterion was defined a priori as the conjunction of (i) correct HTTP status codes at every step, (ii) database state consistent with the expected post-condition, and (iii) data observed by the consuming client identical to that produced by the producing client. All five scenarios satisfied these criteria.

Table 6. End-to-end integration test scenarios and outcomes

#	Scenario / precondition	Steps	Acceptance criterion	Result
1	Registered customer; valid credentials	POST credentials → receive JWT → fetch dashboard payload	200 OK at each step; JWT accepted by auth guard	Pass
2	Catalog populated	AR client GET catalog → render returned objects	Array matches AR schema; no transformation needed	Pass
3	Authenticated admin token	Admin creates then updates a product; non-admin retries	Records persist; non-admin rejected 403	Pass
4	Pending order; Snap token issued	Customer pays via Snap → webhook reconciles order	Order reaches paid; signature verified; stock adjusted	Pass
5	Existing colour/category records	Admin updates colour and category → AR client re-reads	Updated values visible to AR client on next read	Pass

The steps and outcomes for each scenario are listed in Table 6. Scenario four, the complete commercial loop, is the most significant: the customer opens the Midtrans Snap widget with the token from POST /api/orders, pays, and the webhook reconciles the order to paid within the same session, with the three-party handoff among client, backend, and Midtrans completing without intervention. Scenarios two and five confirmed that admin changes to the catalog, colour, and category records propagate to the AR client on its next read without schema transformation, and scenario three confirmed that the admin role check rejects non-admin tokens with 403 Forbidden. Taken together, the five scenarios demonstrate that the AR-to-payment loop is closed at the system level, not merely per endpoint. Scenario four supports the central claim, namely that webhook-driven reconciliation maintains order-state consistency even when the client terminates the session immediately after payment, while scenarios two and five show that a single REST contract serves both clients without forking the data model.

E. Black-Box Functional Test

Black-box testing followed equivalence-partitioning principles [24]. For each of the twelve storefront entities (authentication, dashboard, banner, category, colour, product, sponsor, team, slider, user, coupon, and order), the four CRUD operations were exercised against both valid and

invalid input partitions, yielding 48 nominal test cases and 24 negative cases covering missing required fields, malformed payloads, and unauthorised access. The outcome was identical across all twelve modules and is summarised in Table 7. Every nominal test produced the expected output, and every negative test was rejected with the correct HTTP error code (400 Bad Request, 401 Unauthorized, or 403 Forbidden, as appropriate). After each mutating operation, the database state was verified and matched the expected post-condition in every case. With the functional pass rate defined as

$$P = \left(\frac{N_{pass}}{N_{total}}\right) \cdot 100\% \qquad (1)$$

where N_{total} is the total number of executed test cases and N_{pass} the number that satisfied the expected post-condition, the observed result is P = (72/72) × 100 = 100%, indicating complete functional coverage of the CRUD surface under the test partitions used.

Table 7. Black-box functional test results per module (equivalence partitioning)

Module	Nominal cases	Negative cases	Passed	Pass rate (%)
All twelve modules (authentication, dashboard, banner, category, colour, product, sponsor, team, slider, user, coupon, order)	4 each	2 each	6 each	100
Total	48	24	72	100

Two conclusions follow. The CRUD surface is functionally complete, and input validation is enforced uniformly, which is a precondition for safe customer-facing operation. A regression run after the Midtrans integration reopened no defects in the storefront modules, indicating that the payment pipeline and the catalog operations remain decoupled at the data layer. This is corroborated by the integration tests in Section D, which exercised the same modules from the consuming clients' perspective.

F. Expert Review

A structured review engaged three independent experts who accessed the running system directly (profiles in Table 8): Expert 1, an industry software developer with ten years' experience; Expert 2, in software development at PT. Promedia Citra Informatika with seven years; and Expert 3, the technical lead at PT. Alton Teknologi Indonesia, who holds operational responsibility for the deployment, thereby combining independent technical judgement with operational validity. Each completed the same nine-item questionnaire across three aspects (login, backend, and API implementation), covering data accuracy, creation, edit/delete, media management, filtering and sorting, token-based authentication, error clarity, and response speed (per-item ratings in Table 9). Responses used a five-point Likert scale (1 = strongly disagree, 5 = strongly agree), aggregated into a percentage index following Nielsen [21]:

$$S = \left(\frac{\sum_{i=1}^{k} R_i}{k\ R_{max}}\right).100\% \quad (2)$$

where, in Eq. (2), ΣR_i is the sum of obtained ratings and $R_{max} = 5$ the maximum. The three experts each rated nine items (27 ratings: thirteen 5s, thirteen 4s, one 3), so $\Sigma R_i = 120$ against $27 \times R_{max} = 135$ and S ≈ 89%, in the 80–100% (“very good”) band. Given the small panel, this index is a qualitative acceptance indicator rather than a precise measurement. All three rated token-based authentication and the fast, error-free responses highly; their notes (mobile sub-menu behaviour, language consistency, a notification bug, a page error after switching to a non-admin role) concern the presentation layer rather than the backend contract, and their call for higher-concurrency testing is addressed by the expanded load test in Section C.

Table 8. Profiles of the three independent expert reviewers

Expert	Field / institution	Experience
Expert 1	Software developer (industry)	10 years

Expert 2	Software development, PT. Promedia Citra Informatika	7 years
Expert 3	Technical lead, PT. Alton Teknologi Indonesia (deployment owner)	8 years

Table 9. Expert-review ratings per questionnaire item (five-point Likert scale)

Aspect	#	Questionnaire item	E1	E2	E3
Login	1	Login page accessible; credentials verified correctly	5	5	5
Login	2	Dashboard shows accurate product/order/user statistics	3	5	4
Backend	3	Backend handles data creation with relevant attributes	4	5	5
Backend	4	Edit and delete operations work correctly without error	4	5	5
Backend	5	Media upload and management work consistently	4	5	4
Backend	6	Filtering and sorting work smoothly	4	5	4
API	7	API authentication uses tokens for access security	4	5	5
API	8	System returns clear error responses	4	5	4
API	9	System responds quickly without errors	4	4	4
Total		$\Sigma = 120$ of 135 $\rightarrow$ ≈89% (“very good”)	36	44	40

Although three independent experts provide stronger validity than a single in-house reviewer, the panel remains small: the experts' independence and industry experience lend the score good validity for this deployment, but generalisation to other retail settings would require a larger panel and a complementary end-user study. The score is therefore reported as fitness-for-purpose evidence for this deployment rather than as a generalisable quality measure.

5. Conclusions

This study designed, implemented, and evaluated a REST API backend that integrates a payment gateway into an AR virtual try-on smart mirror for self-service retail. The Laravel backend exposes a single JSON contract to both the Three.js AR client and the web admin dashboard, protects twelve CRUD modules with JWT authentication, and reconciles Midtrans Snap payments through an asynchronous, SHA-512-verified webhook, so that the order lifecycle completes correctly even when the client disconnects immediately after payment.

Within a clearly bounded scope, the evaluation supports the design. At the single-station operating point, authentication averaged 395 ms (p95 443 ms) at a 0.83% error rate and catalog reads 136–167 ms with no errors; across the sweep to one hundred users, the read endpoints remained error-free up to fifty concurrent users, while the password-hashing path saturated first. Every storefront module passed black-box testing, all five integration scenarios succeeded, and three independent experts scored the system 120 of 135 (“very good”). These figures apply to a single smart-mirror station, the current form of the PT. Alton Teknologi Indonesia deployment.

The limitations are explicit. The load test, although large-sample and multi-level, was run on a single-instance host over a wide-area path and does not characterise multi-tenant or sustained peak concurrency; the authentication saturation is a genuine ceiling for that instance. The three-expert panel is small and does not replace a large-scale end-user study.

The contribution is primarily one of engineering and system integration: it introduces backend architecture and quantitative, multi-level server-side evaluation into an AR-retail literature dominated by frontend studies. Future work will extend the load envelope with endurance testing and server-side profiling to address the authentication saturation (through horizontal scaling, connection pooling, or offloaded password hashing), replicate the deployment across multiple smart-mirror units, enlarge the expert panel with a complementary end-user study, and test resilience against adversarial scenarios including signature replay, gateway disruption, and clock-skew-induced webhook delays.

References

[1] P. A. Rauschnabel, B. J. Babin, M. C. tom Dieck, N. Krey, and T. Jung, "What is augmented reality marketing? Its definition, complexity, and future," Journal of Business Research, vol. 142, pp. 1140–1150, 2022. https://doi.org/10.1016/j.jbusres.2021.12.084

[2] T. Hilken, J. Heller, M. Chylinski, D. Keeling, D. Mahr, and K. de Ruyter, "Making omnichannel an augmented reality: the current and future state of the art," Journal of Research in Interactive Marketing, vol. 12, no. 4, pp. 509–523, 2018. https://doi.org/10.1108/JRIM-01-2018-0023

[3] M. Bonetti, G. Warnaby, and L. Quinn, "Augmented reality and virtual reality in physical and online retailing: a review, synthesis and research agenda," in Augmented Reality and Virtual Reality, Cham: Springer, 2018, pp. 119–132. https://doi.org/10.1007/978-3-319-64027-3_9

[4] Y.-C. Huang and S.-Y. Liu, "Factors influencing consumer acceptance of virtual try-on technology in online fashion retailing," Journal of Retailing and Consumer Services, vol. 58, 102283, 2021. https://doi.org/10.1016/j.jretconser.2020.102283

[5] R. E. Bulander, "Self-service technology in retailing: a review and research agenda," International Journal of Retail & Distribution Management, vol. 48, no. 8, pp. 859–878, 2020. https://doi.org/10.1108/IJRDM-10-2019-0325

[6] D. Norman, The Design of Everyday Things, Revised and Expanded Edition. New York: Basic Books, 2013.

[7] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[8] M. Stauffer, Laravel: Up and Running, 3rd ed. Sebastopol, CA: O'Reilly Media, 2023.

[9] S. Wang, I. Keivanloo, and Y. Zou, "How do developers react to RESTful API evolution? An exploratory study," in Proc. Int. Conf. Service-Oriented Computing (ICSOC), 2014, pp. 245–259. https://doi.org/10.1007/978-3-662-45391-9_17

[10] C. Flavián, S. Ibáñez-Sánchez, and C. Orús, "The impact of virtual, augmented and mixed reality technologies on the customer experience," Journal of Business Research, vol. 100, pp. 547–560, 2019. https://doi.org/10.1016/j.jbusres.2018.10.050

[11] Internet Engineering Task Force, "JSON Web Token (JWT)," RFC 7519, May 2015. https://doi.org/10.17487/RFC7519

[12] R. Fielding et al., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, June 2014. https://doi.org/10.17487/RFC7231

[13] L. Richardson and M. Amundsen, RESTful Web APIs. Sebastopol, CA: O'Reilly Media, 2013.

[14] S. Newman, Building Microservices, 2nd ed. Sebastopol, CA: O'Reilly Media, 2021.

[15] M. Masse, REST API Design Rulebook. Sebastopol, CA: O'Reilly Media, 2011.

[16] K. Indrasiri and D. Kuruppu, gRPC: Up and Running. Sebastopol, CA: O'Reilly Media, 2020.

[17] PT. Midtrans, "Midtrans Core API and Snap Integration Reference," [Online]. Available: https://docs.midtrans.com. Accessed: Jan. 2025.

[18] Khronos Group, "glTF 2.0 Specification," [Online]. Available: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html. Accessed: Jan. 2025.

[19] Three.js Authors, Three.js Documentation, r158 ed. [Online]. Available: https://threejs.org/docs/. Accessed: Jan. 2025.

[20] T. Brown, "Design thinking," Harvard Business Review, vol. 86, no. 6, pp. 84–92, 2008.

[21] J. Nielsen, Usability Engineering. San Diego, CA: Morgan Kaufmann, 1993.

[22] Apache Software Foundation, "Apache JMeter User Manual," [Online]. Available: https://jmeter.apache.org/usermanual/. Accessed: Jan. 2025.

[23] B. W. Boehm, "A spiral model of software development and enhancement," IEEE Computer, vol. 21, no. 5, pp. 61–72, 1988. https://doi.org/10.1109/2.59

[24] I. Sommerville, Software Engineering, 10th ed. Boston, MA: Pearson, 2016.

[25] H. Gimpel et al., "Structuring digital transformation: A framework of action fields and its application at ZEISS," Journal of Information Technology Teaching Cases, vol. 8, no. 1, pp. 54–61, 2018. https://doi.org/10.1057/s41266-018-0041-y

[26] R. Baber, P. Baber, and S. Narula, "Driving AR experience to purchase intention: examining the role of users' immersiveness, perceived innovativeness and engagement in virtual try-on," International Journal of Consumer Studies, vol. 50, no. 2, art. e70185, 2026. https://doi.org/10.1111/ijcs.70185

[27] C. D. Schultz and H. Kumar, "ARvolution: decoding consumer motivation and value dimensions in augmented reality," Journal of Retailing and Consumer Services, vol. 78, art. 103701, 2024. https://doi.org/10.1016/j.jretconser.2023.103701
[28] D. Recalde, T. C. Jai, and R. P. Jones, "I can find the right product with AR! The mediation effects of shopper engagement on intent to purchase beauty products," Journal of Retailing and Consumer Services, vol. 78, art. 103764, 2024. https://doi.org/10.1016/j.jretconser.2024.103764
[29] S. Benoit, B. Altrichter, D. Grewal, and C.-P. Ahlbom, "Autonomous stores: how levels of in-store automation affect store patronage," Journal of Retailing, vol. 100, no. 2, pp. 217–238, 2024. https://doi.org/10.1016/j.jretai.2023.12.003
[30] S. Hazée, A. De Keyser, B. Larivière, Y. Kim, A. Talebi, and I. Gordeliy, "Just walk out stores: the future of shopping? Examining configurations of reasons for and against consumer adoption," Psychology & Marketing, vol. 42, no. 4, pp. 987–1017, 2025. https://doi.org/10.1002/mar.22161
[31] T. P. Kohli, S. Malhotra, and V. Pingali, "Revolutionising retail: a review of intelligent unmanned convenience stores and their market implications," Technological Forecasting and Social Change, vol. 219, art. 124273, 2025. https://doi.org/10.1016/j.techfore.2025.124273
[32] M. Bolanowski, M. Ćmil, and A. Starzec, "New model for defining and implementing performance tests," Future Internet, vol. 16, no. 10, art. 366, 2024. https://doi.org/10.3390/fi16100366
[33] H. Alnuhait, W. Alzyadat, A. Althunibat, H. Kahtan, B. Zaqaibeh, and H. A. Al-Khawaja, "Web application performance assessment: a study of responsiveness, throughput, and scalability," International Journal of Advanced and Applied Sciences, vol. 11, no. 9, pp. 214–226, 2024. https://doi.org/10.21833/ijaas.2024.09.023
[34] A. Annisa, M. Ardiana, P. N. Rahayu, T. Brian, and M. A. Jami'in, "Performance evaluation of web applications using JMeter load testing for server capacity and response efficiency," Journal of Applied Informatics and Computing, vol. 10, no. 1, pp. 585–590, 2026. https://doi.org/10.30871/jaic.v10i1.11840
[35] K. X. Carmo, F. Ferreira, and E. Figueiredo, "Performance evaluation of back-end frameworks: a comparative study," in Proc. 20th Brazilian Symp. on Information Systems (SBSI), 2024, pp. 1–9. https://doi.org/10.1145/3658271.3658314
[36] A. M. Sahi, H. Khalid, A. F. Abbas, K. Zedan, S. F. A. Khatib, and H. Al Amosh, "The research trend of security and privacy in digital payment," Informatics, vol. 9, no. 2, art. 32, 2022. https://doi.org/10.3390/informatics9020032
[37] Afridon, M., et al. "Optimizing Data Security in Computer-Assisted Test Applications Through the Advanced Encryption Standard 256-Bit Cipher Block Chaining." *International Journal of Advanced Computer Science & Applications* 15.8 (2024): 163.
[38] P. Jagadeesh, M. Sathishkumar, E. Deepankumar, and C. Prathipa, "Real-time scalable and secure webhook notification handling using AWS: a serverless architecture approach," in Proc. 2025 2nd Int. Conf. on Computing and Data Science (ICCDS), 2025, pp. 1–4. https://doi.org/10.1109/ICCDS64403.2025.11209092
[39] S. Dalimunthe, E. H. Putra, and M. A. F. Ridha, "RESTful API security using JSON Web Token (JWT) with HMAC-SHA512 algorithm in session management," IT Journal Research and Development, vol. 8, no. 1, pp. 81–94, 2023. https://doi.org/10.25299/itjrd.2023.12029
[40] R. A. Setiadi, B. Purnomosidi, W. Andriyani, and S. R. C. Nursari, "Microservices architecture in point-of-sales application based on RESTful API and webhook," Journal of Intelligent Software Systems, vol. 3, no. 1, art. 31, 2024. https://doi.org/10.26798/jiss.v3i1.1336

Acknowledgements

The authors thank the Department of Creative Multimedia Technology at Politeknik Elektronika Negeri Surabaya for supervising this research, and PT. Alton Teknologi Indonesia for providing the industrial setting and the live deployment used to evaluate the backend on a realistic retail workload.