Innovative Journal of Computing Research (IJCR)
Vol. 2, No. 1, September 2026, pp. 27~47
ISSN: 3164-239X, DOI: -

Web-Based Nusantara Tourism Information System Using a REST API: Design, Implementation, and Functional Verification

Refin Jioh Natan[1], M. Aditia Pratama[2]

[1,2]Universitas Sains dan Teknologi Indonesia, Pekanbaru, Indonesia

[1]2310031802114@sar.ac.id; [2]2310031802097@sar.ac.id

Article Info

Article history:
Received:
Revised:
Accepted:

Keywords:
Tourism information system
REST API
Waterfall
Keyword search
Content moderation
Functional testing

ABSTRACT

Web-based tourism information systems require searchable destination records and controls over user-submitted content. This study develops the Tourism Nusantara application using a Research and Development approach with Waterfall stages comprising requirements analysis, design, implementation, functional verification, and deployment preparation. The application provides a destination catalogue, keyword search, category filtering, a submission form with a photo field, and an administrator dashboard for approving or rejecting submissions. To strengthen functional evidence, a verification prototype was developed to reproduce the core workflow through an HTTP/JSON API and SQLite database. Testing used 24 synthetic records and 56 deterministic black-box scenarios covering search, filtering, authentication, authorization, validation, submission, and moderation. All scenarios passed through 60 HTTP requests. Pending and rejected records remained inaccessible to public users, while approved submissions became searchable. These results demonstrate functional conformity within the tested environment, but do not establish real-world data accuracy, usability, production security, scalability, or comparative performance.

This is an open access article under the CC BY-SA license.

Corresponding Author:
[Konfirmasi penulis korespondensi dan email aktif sebelum pengajuan]
Universitas Sains dan Teknologi Indonesia, Pekanbaru, Indonesia

1. INTRODUCTION

Information technology has become an important medium for organising, distributing, and promoting tourism information. Its application in tourism villages can broaden public access to local attractions and support the digital promotion of regional potential [1]. Web-based tourism information systems have consequently been developed to present destination descriptions, facilities, photographs, and location data in an integrated form [2]. Similar development in Riau Province demonstrates the role of web applications in centralising tourism information that was previously distributed across different sources [3]. Recent studies have also implemented REST APIs to connect tourism databases with mobile applications [4] and geographic information systems [5]. These implementations improve data accessibility across platforms; however, technical integration alone does not ensure that users can locate and understand the required information. Interface structure, search facilities, and the organisation of destination content must therefore be considered as part of the overall system design [6].

This study develops the Tourism Nusantara application as a web-based directory that combines a destination catalogue, keyword search, category classification, a destination submission form with a photo field, and an administrator moderation dashboard. Data communication is organised through a REST API using resource-oriented URIs, HTTP methods, and JSON representations. This architecture separates data services from the user interface and facilitates structured exchanges between application components [7]. The development process follows the Waterfall model, consisting of requirements analysis, system and interface design, implementation, functional testing, and deployment preparation [8]. Unlike a conventional tourism catalogue that only publishes administrator-managed records, the proposed application allows users to submit destination information. Each submission is assigned a publication status and must be reviewed before

becoming publicly accessible. This mechanism connects information discovery with content governance and reduces the possibility that an unreviewed submission is immediately presented as public tourism information.

Previous tourism applications have been evaluated through black-box testing and user acceptance testing to determine whether their principal functions operate according to specified requirements [9]. Digital applications have also been used to strengthen the promotion of destinations whose information is not widely available [10]. Other studies extend tourism services through mobile applications [11], user-centred interface design [12], [13], conversational tourism assistants [14], [15], and recommendation systems that help users select destinations according to particular preferences [16]–[18]. Geographic information services further support the presentation of tourism-related locations through interactive maps [19], while REST API implementations in other domains demonstrate the value of separating data management from client applications [20]. Nevertheless, these studies generally emphasise information presentation, recommendations, interface usability, or cross-platform integration. The relationship among keyword search, user-generated destination submissions, role-based access control, and publication-status transitions has received less explicit functional verification. This gap forms the principal focus of the present study.

Accordingly, this study aims to design, implement, and functionally verify a REST API-based tourism information application. The evaluation addresses two questions: how tourism-information requirements can be translated into catalogue, search, category, submission, and moderation functions; and whether the implemented workflow consistently restricts public access to destinations with approved status while reserving moderation decisions for administrators. The study contributes an applied system artefact and a reproducible verification procedure in which expected identifiers, HTTP response codes, access restrictions, and publication-state transitions are checked through predefined test scenarios. The contribution is therefore not a new search or automatic-classification algorithm, but an explicit and testable workflow for managing the visibility of submitted tourism information. The research position and its differences from related studies are summarised in Table 1.

.

Table 1. Comparison of the scope of related studies

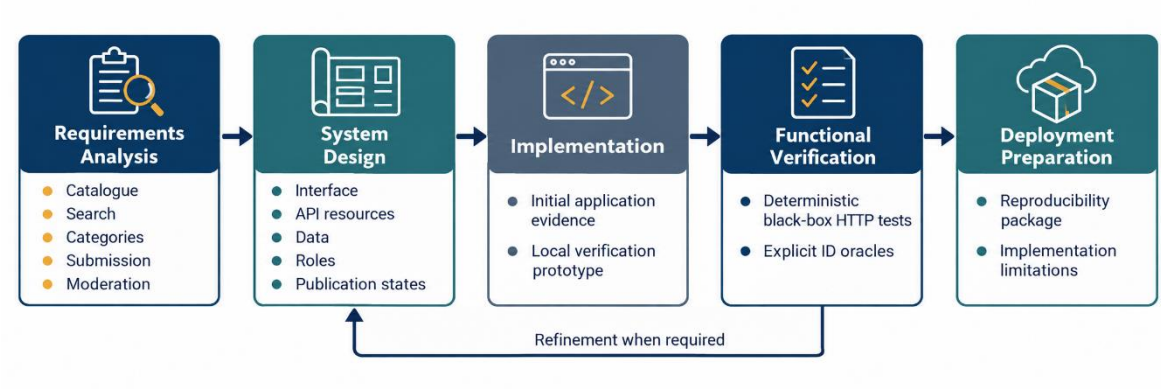
Study	Reported Focus and Evaluation	Position relative to this study
[2]	A web-based tourism information system for Nunukan that organises and presents destination information to the public.	Provides a comparable web-based tourism directory; it does not explicitly examine REST API access control or publication-status transitions..
[4]	Implementation of a REST API for an Android-based tourism information application in Yogyakarta, supported by functional testing.	Provides a comparison for API-based application integration; the present study uses a local web client and adds a moderated submission workflow.
[6]	UI/UX design for a tourism information application in Jepara using the Design Thinking method.	Provides an interface-design comparison; the present study evaluates API behaviour and publication control rather than usability through respondents.
[9]	An Android tourism application for Pekanbaru evaluated using black-box testing and user acceptance testing	Provides functional and user-acceptance comparisons; the present study uses deterministic HTTP scenarios and does not involve respondents
This Studi	A synthetic dataset, explicit identifier oracles, publication statuses, role-based restrictions, and recorded HTTP request-response traces	Provides reproducible local functional verification; real-world tourism-data accuracy, usability, and field benefits remain outside the evaluation scope

2. METHOD

2.1. Development approach

This study employed an applied research and development approach while retaining the Waterfall sequence reported in the original manuscript. The work was organised into requirements analysis, system and interface design, implementation, functional verification, and deployment preparation. Requirements analysis translated the intended tourism-information service into five core capabilities: a destination catalogue, keyword search, category filtering, user submission, and administrator moderation. The design stage specified the browser interface, addressable API resources, data entities, user roles, validation rules, and publication states. Implementation preserved the original interface evidence and added a separate local verification prototype for the executable workflow. Functional verification then exercised the observable HTTP behaviour, while deployment preparation assembled the specifications, fixtures, logs, and version identifiers needed to review the experiment. This artefact-oriented sequence is consistent with design-science guidance that connects problem definition, artefact construction, and evaluation . Figure 1 summarises the stages and the evidence produced at each stage.

Innovative Journal of Computing Research (IJCR)
Vol. 2, No. 1, September 2026, pp. 27~47
ISSN: 3164-239X, DOI: -



skemasFigure 1. Development and prototype-evaluation stages.

The evidence from the original application was deliberately separated from the evidence produced by the verification prototype. Three screenshots from the earlier work document the reported appearance of the catalogue, submission form, and administrator interface, but screenshots cannot establish server-side authorisation, filtering, or publication-state behaviour. Because the original source code and executable environment were unavailable, the original application could not be rerun or treated as the system under test. The local prototype was therefore implemented as a new and limited reproduction of the observable core workflow rather than as a reconstruction of the complete application. Consequently, the results of the 56 verification scenarios are reported only for this prototype and are not attributed retrospectively to the earlier application. The prototype also omits binary photo upload: a photo field remains visible in the original form evidence, whereas the executable verification focuses on textual destination data, search, submission, role checks, and moderation. No survey, interview, usability measurement, or field deployment was conducted.

2.2. Architecture and implementation environment

Figure 2 presents the local prototype architecture and the path through which experimental evidence was generated. A browser-based client implemented with HTML, CSS, and JavaScript communicates with an API server through loopback HTTP. The server ran on Python 3.12.14 using the standard-library http.server module , while persistent data were managed with SQLite 3.53.1 through Python's sqlite3 interface. Execution took place on a Linux x86_64 environment bound to 127.0.0.1. Each test run created a new database from the same deterministic fixture before the first request was issued, preventing records modified in an earlier run from affecting later results. The Python server was selected because it made the local workflow compact and reproducible; its use does not represent a recommended production configuration, as the module provides only basic server functionality and is not intended for production deployment.

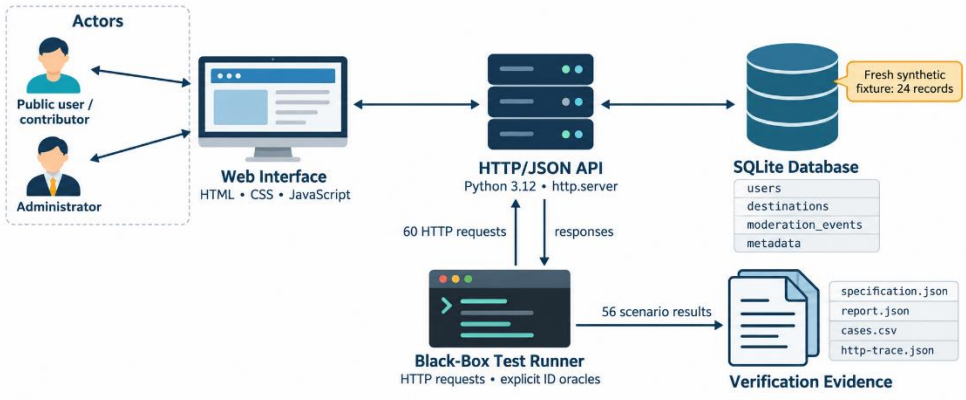


Figure 2. Prototype components and experimental evidence flow.

Request and response bodies use JavaScript Object Notation (JSON), following the data-interchange syntax standardised in RFC 8259. Resource paths, request methods, and response status codes were defined according to HTTP semantics . Successful authentication produces a locally generated token signed with HMAC-SHA256 and configured to expire after 3,600 seconds. The token is carried in the Authorization header using the Bearer syntax described in RFC 6750. This choice only describes how the prototype transmits its credential: the implementation is not an OAuth 2.0 authorisation server, and the token is not a JSON Web

Token. Because all requests were executed through unencrypted loopback HTTP, the experiment does not demonstrate transport confidentiality, certificate management, or secure token transmission over a public network. These limitations are stated explicitly to prevent functional authentication checks from being interpreted as a complete security evaluation.

The database schema contains four tables. users stores demonstration accounts and their roles; destinations stores submitted fields, ownership, and publication status; moderation_events records the administrator, decision, reason, and time of each moderation action; and metadata identifies the fixture version used in a run. A status update and its corresponding moderation event are committed within one SQLite transaction so that a successful decision cannot be stored without its audit record . Submitter and administrator relationships are represented by foreign keys, with enforcement enabled for every database connection . A valid submission is assigned pending status by the server, rather than accepting a client-supplied publication state. Administrator decisions are applied through PATCH because the operation partially modifies an existing submission resource.

2.3. Synthetic verification data and information boundaries

The initial fixture contains 24 fictional destination records, including deliberately artificial names such as Sample Lake One and Test Region A. The records are evenly distributed across the Nature, Culture, and History categories. Within each category, six records are approved, one is pending, and one is rejected, producing 18 publicly eligible records and six non-public records before testing begins. This arrangement was designed to support positive and negative assertions for every category and publication state; it is not a sample drawn from a tourism population. Each run starts from the composition shown in Table 2, after which the test sequence creates two additional submissions for approval and rejection paths. The fixed starting state enables the expected identifier sets to be written before execution and makes the results comparable across repeated local runs.

All names, locations, and descriptions in the fixture are synthetic and do not represent actual attractions, government records, businesses, or individuals. The dataset contains no respondent data, personal identifiers, behavioural measurements, or claims about destination quality. Its sole purpose is to exercise retrieval, validation, access control, and state-transition rules under controlled conditions. Accordingly, successful verification cannot be interpreted as evidence that real tourism information is accurate, complete, current, or useful to travellers. Those questions would require a separately governed data-collection process and an evaluation involving relevant users or domain stakeholders.

Table 2. Fixture composition before test executio

Category	Approved	Pending	Rejected	Total
Nature	6	1	1	8
Culture	6	1	1	8
History	6	1	1	8
Total	18	3	3	24

2.4. Functional contract and publication control

The API contract separates public retrieval, contributor submission, and administrator moderation, as summarised in Table 3. Public clients may list or retrieve only destinations whose status is approved. An authenticated contributor may create a submission but cannot choose its initial status or access administrator functions. An administrator may inspect the moderation queue, approve or reject pending submissions, and retrieve the recorded decision events. These restrictions are enforced by the server for every protected request rather than by merely hiding interface controls. Negative scenarios were selected with reference to the distinction between object-level and function-level authorisation failures in the OWASP API Security Top 10 . The resulting checks are targeted functional controls: they do not constitute penetration testing, source-code review, vulnerability scanning, or a complete security audit.

Table 3. Main prototype endpoints

Method and endpoint	Access	Principal input and output
GET /api/destinations	Public	Optional q and category; approved records and total count.
GET /api/destinations/{id}	Public	Approved record detail; hidden or absent object returns 404.

Method and endpoint	Access	Principal input and output
POST /api/auth/login	Public	Username and password; token, role, and expiry.
POST /api/submissions	Contributor	name, location, category, and description; 201 with pending status.
GET /api/admin/submissions	Admin	Optional status; submissions matching the requested state.
PATCH /api/admin/submissions/{id}	Admin	decision and reason; approved or rejected result.
GET /api/admin/events	Admin	Recorded moderation-decision events.
GET /api/categories; GET /api/health	Public	Category choices; service version and synthetic-data marker.

For public search, the q value is stripped of leading and trailing whitespace and normalised with Unicode-aware case folding. Substring matching is then applied to the destination name, location, or description. The category filter requires equality with one of the three permitted category values, and when q and category are supplied together their conditions are combined with the logical AND operator. Percent and underscore characters are escaped before the database comparison so that they are treated as literal input rather than SQL wildcard operators. Returned records are ordered deterministically and accompanied by a total count, allowing the test oracle to compare both the complete identifier set and the reported quantity. This mechanism is a manually specified keyword search and category filter; it is not automatic classification, semantic retrieval, recommendation, or relevance ranking.

Submission validation constrains name and location to 3-100 characters, description to 20-1,000 characters, and category to one of the permitted values. A request body may not exceed 16,384 bytes, and protected JSON routes reject unsupported media types or malformed JSON. Any submission containing a status field is rejected, preventing a contributor from bypassing moderation through mass assignment. A valid record begins in pending state and may transition only to approved or rejected after an administrator provides a reason of 5-500 characters. Approved and rejected are terminal states in this prototype; a second decision attempt returns HTTP 409 Conflict. These rules define the publication boundary tested in the experiment and ensure that public visibility changes only as a consequence of an authorised server-side transition.

2.5. Test design and acceptance criteria

Black-box verification was performed by a separate Python test script that launched the server process and sent actual HTTP requests without importing application functions or accessing internal handler state. Before each run, the script created the database from the fixed fixture and loaded explicitly written reference identifiers for the expected search results. A test oracle is necessary because successful request processing alone cannot determine whether the returned behaviour is correct . Accordingly, an HTTP 200 response was considered insufficient unless the returned identifiers, total count, publication status, field values, and ordering also satisfied the applicable specification. Negative scenarios similarly required the predefined status code and error condition rather than accepting any non-success response as evidence of correct rejection.

The suite contains 56 scenarios divided into five groups. Public scenarios P01-P20 cover matches in names, locations, and descriptions; empty results; the three categories; combined conditions; literal wildcard characters; hidden objects; and invalid parameters. Authentication scenarios A01-A07 cover valid login, invalid credentials, absent credentials, expired or malformed tokens, and role mismatches. Submission scenarios V01-V10 cover valid creation, required fields, field lengths, category values, JSON syntax, media type, body size, and attempts to define status directly. Moderation scenarios M01-M15 examine queue retrieval, approval and rejection transitions, public visibility before and after a decision, repeated decisions, role enforcement, validation, and audit-event consistency. Integration scenarios I01-I04 check unknown routes, unsupported methods, the health endpoint, and delivery of the browser asset. Table 4 maps these groups to the acceptance criteria used to interpret the recorded responses.

Several paired inputs were included to check stable relations between executions. For example, the keyword Lake and the same keyword with altered capitalisation and surrounding whitespace were required to produce the same identifier set after normalisation. Similar pairs verify that adding a valid category narrows rather than expands a result set. These relations are consistent with the principle of metamorphic testing, in which expected relationships between related inputs and outputs supplement direct oracles. However, the study does not claim a complete metamorphic-testing framework because the relations and source cases were manually selected. The suite uses no random generation, fuzzing, mutation testing, code-coverage measurement, statistical sampling, or estimate of failure probability.

Innovative Journal of Computing Research (IJCR)
Vol. 2, No. 1, September 2026, pp. 27~47
ISSN: 3164-239X, DOI: -

Table 4. Acceptance criteria and test-case mapping

Criterion	Oracle or acceptance rule	Related cases
Search and filtering	Returned identifiers equal the reference set, and every result is approved.	P01–P14
Public visibility	Only approved records are available; pending, rejected, and absent identifiers return 404.	P15–P17; M01–M02; M05–M06; M09
Request validation	Non-conforming input is rejected with the specified status code.	P18–P20; V02–V10; M12–M14
Identity and role	Valid login is accepted; invalid credentials or insufficient roles are rejected.	A01–A07; M11
Publication lifecycle	Only an administrator may decide pending records; repeated decisions are rejected.	V01; M03–M08; M10
Audit and protocol	Two correct moderation events are recorded; routes, methods, and assets follow the specification.	M15; I01–I04

A scenario passed only when every assertion associated with that scenario succeeded. The pass rate was calculated as the number of passed scenarios divided by the number executed and multiplied by 100%. Because some scenarios require more than one interaction, the 56 scenarios generated 60 HTTP requests. Moderation cases were intentionally executed in sequence: the identifier returned by a valid submission became the input for the subsequent approval or rejection request and for the later public-read assertion. Evidence was written to specification.json, report.json, cases.csv, and http-trace.json. The trace records time, method, path, role, request body, HTTP status, and response while masking passwords and tokens. SHA-256 hashes of the application code and fixture link the report to the tested versions. This package supports repeatable inspection of the stated local scenarios without treating screenshots as the sole source of evidence.

3. RESULTS AND DISCUSSION

3.1. Original application interface evidence

The original Tourism Nusantara report documents three principal interfaces that define the application concept: a public destination catalogue, a destination-submission form, and an administrator dashboard. These screenshots are retained because they establish the intended interaction flow and show that search, category selection, user contribution, and moderation were represented in the earlier design. They are interpreted as interface evidence rather than executable test evidence because the original source code and deployment environment were unavailable for re-execution. The labels visible in the screenshots remain in Indonesian, reflecting the language of the original application. No attempt was made to redraw the screens or replace their displayed records, since doing so would weaken the distinction between the previously reported artefact and the separate prototype used for reproducible verification.

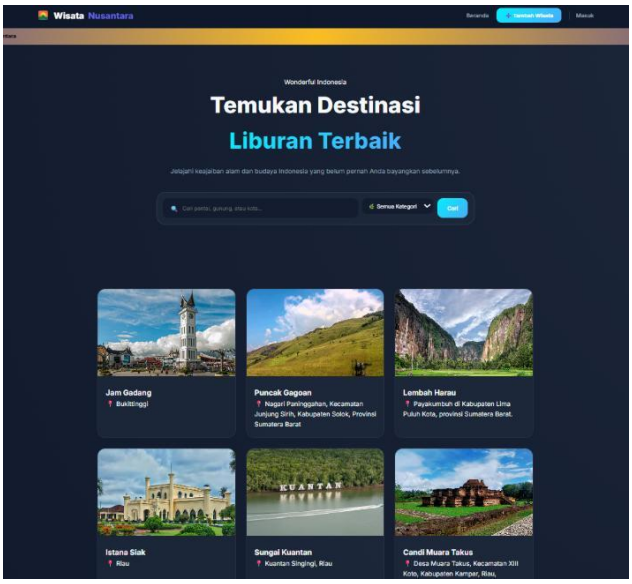


Figure 3. Main page of the original Tourism Nusantara application.

Figure 3 shows a public landing page that combines a keyword field, a category selector, and destination cards. Each visible card presents a destination name, location, and photograph, allowing a user to scan the

Innovative Journal of Computing Research (IJCR)
Vol. 2, No. 1, September 2026, pp. 27~47
ISSN: 3164-239X, DOI: -

catalogue before opening a detailed record. The arrangement supports the requirements identified in the Introduction by keeping discovery and classification within the same page. It also demonstrates that the initial application design anticipated multiple destination types rather than presenting a single undifferentiated list. However, the screenshot does not reveal the data source, the completeness of the catalogue, the ordering algorithm, or whether a displayed query was executed. Claims about search behaviour are therefore based on the verification prototype and its recorded HTTP responses, not on the visual appearance of this page.

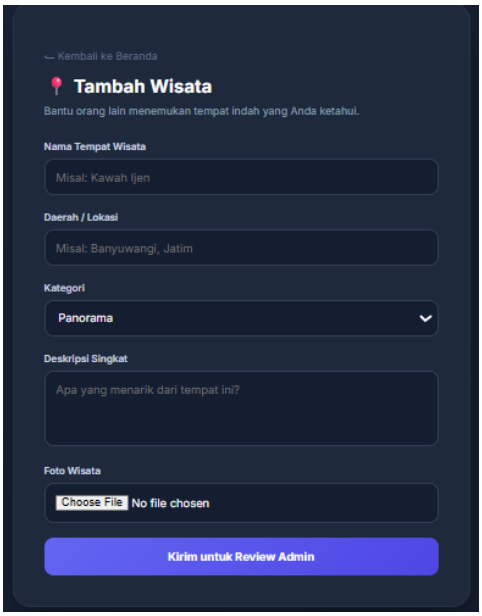


Figure 4. Destination-submission form in the original application.

The submission interface in Figure 4 requests a destination name, location, category, short description, and photograph. Its action label indicates that submitted content is sent for administrator review rather than published immediately. This is an important design feature because it makes the publication decision explicit to the contributor. The screenshot nevertheless provides no direct evidence that the server prevents a user from setting an approved status, rejects malformed input, or stores a submission as pending. Those behaviours were reproduced and tested through the local API prototype. The photo field was not included in the executable verification because the available evidence was insufficient to reconstruct the original upload, storage, and media-validation logic without introducing unsupported assumptions.

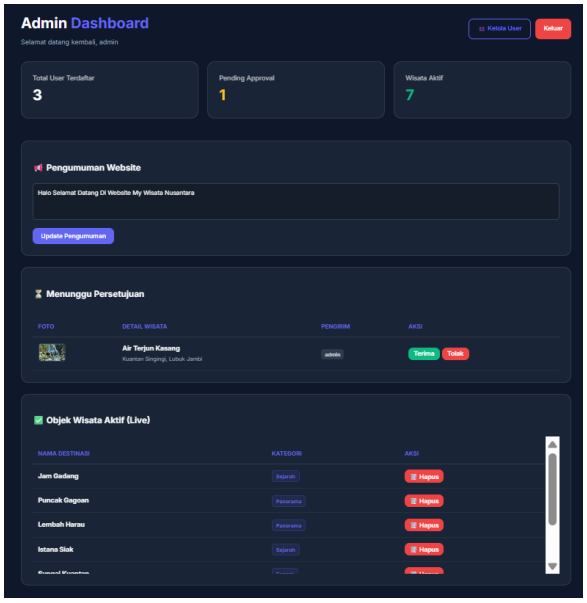


Figure 5. Administrator dashboard for destination moderation.

Figure 5 shows the intended administrative view, including summary counts, a pending-approval area, accept and reject controls, and a list of active destinations. Together, Figures 3-5 describe a coherent workflow in which public discovery, user contribution, and administrative publication are connected. The numeric values

visible in the dashboard are not treated as research measurements because they represent one captured interface state whose database cannot be independently inspected. Instead, the verification results below report only states and responses generated by the deterministic local fixture. This separation preserves the useful evidence from the original work while preventing the interface screenshots from being used to support claims that require executable server behaviour.

3.2. Overall functional verification outcome

The recorded verification run executed all 56 predefined black-box scenarios and generated 60 HTTP requests. Every scenario satisfied all of its assertions, producing a scenario-level pass rate of 100% for the tested local configuration. Table 5 separates the result by functional group. The public group contains the largest number of cases because it covers keyword search, category filtering, combined conditions, hidden objects, and invalid query parameters. Moderation is the second-largest group because a publication decision affects several observable states: the administrator queue, the public collection, individual-resource access, repeated decisions, and the audit trail. Authentication, submission validation, and integration checks complete the suite. No scenario was removed or reclassified after execution to improve the reported pass rate.

Table 5. Functional verification results by scenario group

Scenario group	Case range	Executed	Passed	Failed	Pass rate
Public search and retrieval	P01-P20	20	20	0	100%
Authentication and roles	A01-A07	7	7	0	100%
Submission and validation	V01-V10	10	10	0	100%
Moderation and publication	M01-M15	15	15	0	100%
Integration and protocol	I01-I04	4	4	0	100%
Total	P01-I04	56	56	0	100%

Figure 6 visualises the number of scenarios contributed by each group and confirms that the passed count equalled the executed count in every group. The difference between 56 scenarios and 60 requests is expected: a scenario may require an initial request that creates or authenticates a resource followed by another request that observes the resulting state. Scenario-level counting avoids overstating evidence by treating each request in a multi-step workflow as an independent test. The recorded outcome demonstrates conformance to the explicitly defined acceptance rules for this fixture and implementation version. It does not provide a confidence interval, defect-density estimate, or probability that an untested request will succeed, because the cases were deterministic and manually specified rather than statistically sampled.

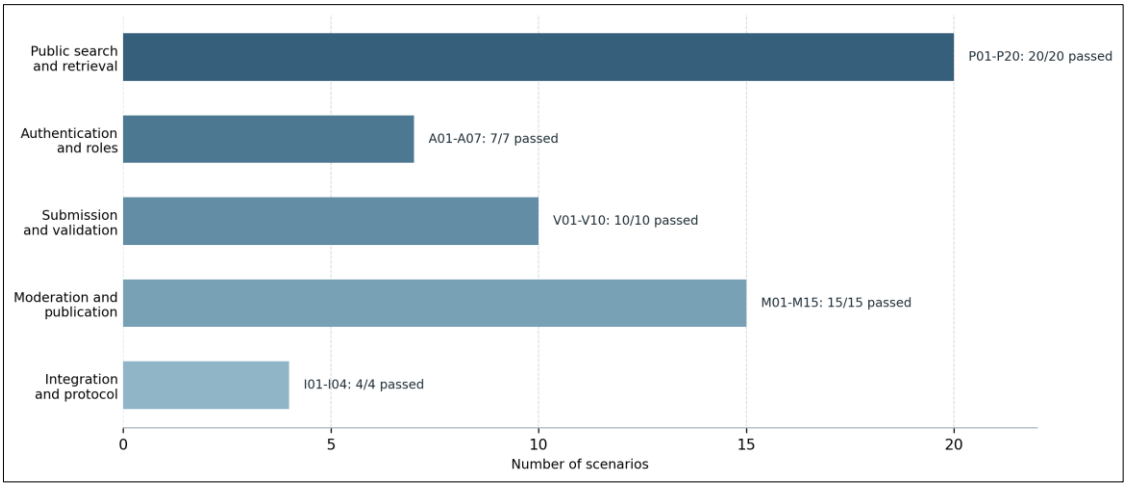


Figure 6. Executed and passed scenarios by functional group.

This evaluation differs from interface-oriented or respondent-based assessments reported in related tourism applications. The comparison study in , for example, combines black-box testing with user acceptance testing, whereas the present work deliberately limits its evidence to observable API behaviour and does not report user satisfaction. Likewise, the REST API implementation in provides a relevant architectural comparison, but the contribution examined here is the explicit link between role restrictions, submission status,

and public visibility. The complete pass result should therefore be read as reproducible functional verification, not as evidence that the system is more usable, secure, scalable, or effective than the compared applications.

3.3. Search, category, and public-visibility results

Public retrieval behaved consistently with the contract defined in the Method. For each keyword and category case, the response identifiers were compared with an explicit reference set rather than checked only for a successful status code. Searches targeting text in the name, location, or description returned the expected approved records. Leading and trailing whitespace and changes in letter case did not alter the identifier set after normalisation. Category filters for Nature, Culture, and History returned the corresponding approved subset, while a keyword combined with a category produced the intersection of the two conditions. Percent and underscore characters were processed as literal query content, preventing SQL wildcard semantics from silently broadening the result. The main observable outcomes are summarised in Table 6.

Table 6. Observed search and visibility behaviour

Tested behaviour	Acceptance rule	Recorded outcome
Unfiltered public catalogue	Every returned record has approved status.	Passed
Name, location, and description queries	Identifier sets equal the predefined oracle sets.	Passed
Case and surrounding-space variations	Equivalent inputs return identical identifier sets.	Passed
Nature, Culture, and History filters	Only approved records in the selected category are returned.	Passed
Combined keyword and category	Results satisfy both conditions through logical AND.	Passed
Literal % and _ characters	Characters are matched literally rather than as wildcards.	Passed
Pending, rejected, and absent identifiers	Individual public retrieval returns 404.	Passed
Invalid public parameters	Request is rejected with the specified error response.	Passed

The visibility assertions are particularly important because an endpoint can return syntactically correct JSON while still disclosing a pending or rejected record. In the initial fixture, only the 18 approved destinations were eligible for the public collection; the three pending and three rejected records remained excluded. Direct requests for non-public identifiers also returned the same not-found response used for an absent identifier, avoiding confirmation that a hidden record existed. After an administrator approved a newly submitted destination, that record became available through the public endpoints and could be found using its search term. A separately rejected submission remained unavailable. These results answer the second research question at the level of the tested implementation: public discovery was determined by publication status rather than by the mere presence of a database row.

3.4. Submission, role restriction, and moderation results

A valid contributor request created a new destination with HTTP 201 and pending status. The server ignored no hidden client-side convention; instead, it explicitly rejected a request that attempted to provide its own status field. Missing fields, unsupported categories, out-of-range text lengths, malformed JSON, unsupported media types, and oversized bodies produced the predefined rejection outcomes. Protected administrator endpoints were inaccessible without an accepted administrator credential, and a contributor credential did not grant moderation privileges. These cases show that role separation was implemented at the API boundary rather than being dependent on the visibility of buttons in the web interface. The result is narrower than a security certification, but it directly verifies the publication controls that are material to the proposed workflow.

Table 7. Publication-state trace during the moderation scenarios

Recorded stage	Approved	Pending	Rejected	Total	Publicly visible
Initial deterministic fixture	18	3	3	24	18
After two valid submissions	18	5	3	26	18

After approving one submission	19	4	3	26	19
After rejecting one submission	19	3	4	26	19

Table 7 reconstructs the state counts from the deterministic moderation sequence. Creating two valid submissions increased pending records from three to five without changing the public count. Approval moved one record from pending to approved and increased the publicly visible count from 18 to 19. Rejection moved the second new record from pending to rejected without increasing public visibility. The corresponding moderation events identified the administrator, decision, and reason, and both events were written consistently with the destination updates. A second decision attempt against a terminal record returned HTTP 409 Conflict, preserving the one-way transition from pending to either approved or rejected. The trace therefore connects the access-control result to an observable information-governance outcome: administrator action, rather than contributor input, determines whether a submitted destination becomes discoverable.

3.5. Interpretation, contribution, and result boundaries

Taken together, the interface evidence and the executable verification support two different conclusions. The original screenshots show that the application concept included a catalogue, search controls, a contribution form, and administrator moderation. The local prototype demonstrates that the corresponding core rules can be represented as testable API behaviour with deterministic inputs and explicit oracles. The principal contribution is therefore not a new retrieval algorithm or an empirical claim about tourism promotion. It is a traceable implementation pattern that links destination submission, server-side role enforcement, publication-state transitions, public search, and an audit event. This interpretation is consistent with the artefact-and-evaluation orientation adopted in the Method and with the need for an oracle that distinguishes correct outputs from requests that merely complete without an error.

Several limitations remain. The 24 starting records and two test submissions are synthetic, so the experiment cannot assess data accuracy, regional coverage, image quality, or maintenance of real destination information. The suite contains manually written cases and provides no code-coverage result, concurrency test, load measurement, long-duration reliability observation, or comparison with another implementation. Authentication was exercised through local tokens over loopback HTTP without TLS, and the tests do not establish resistance to attacks beyond the stated role and object-access scenarios. The original interface was not evaluated with respondents, making usability, accessibility, satisfaction, and task-completion time outside the evidence. Future work should rerun the same acceptance contract in a production-oriented framework, add media-upload controls, evaluate real data governance, and conduct an ethically reviewed user study before claiming practical benefits for tourists or destination managers.

4. CONCLUSION

This study preserves the original design and implementation evidence of the Tourism Nusantara web application, including its destination catalogue, keyword search, category filtering, photo-based submission form, and administrative moderation dashboard. The revised study clarifies the publication rules and introduces a separate, reproducible prototype to verify the core functional workflow. In a local evaluation using 24 synthetic records, all 56 predefined scenarios passed across 60 HTTP requests. Pending and rejected records remained inaccessible through public endpoints, approved submissions became searchable, and moderation functions were restricted to administrators. These results demonstrate functional conformance with the specified search, submission, publication, and access-control requirements under the tested local conditions. However, because the original application source code was unavailable, these findings apply only to the verification prototype and should be interpreted as supporting evidence for the design logic rather than direct validation of the original system. Future studies should evaluate usability, real-world tourism data accuracy, production security, scalability, and deployment performance.

REFERENCES

[1] I. Ananda, “Pemanfaatan Teknologi Informasi pada Desa Wisata di Indonesia: A Systematic Literature Review,” JATISI (Jurnal Teknik Informatika dan Sistem Informasi), vol. 8, no. 4, pp. 2291–2300, 2021, doi: 10.35957/jatisi.v8i4.1095.

[2] E. N. Annisa, N. H. Matondang, and S. Afrizal, “Sistem Informasi Pariwisata Berbasis Web pada Kabupaten Nunukan,” JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), vol. 7, no. 2, pp. 478–486, 2022, doi: 10.29100/jipi.v7i2.2822.

[3] Yulisman, Y. Irawan, and D. A. Ramadhini, “Sistem Informasi Pariwisata di Provinsi Riau Menggunakan Metode HMVC dengan Framework CodeIgniter,” Jurnal Ilmu Komputer, vol. 10, no. 1, pp. 48–54, 2021, doi: 10.33060/JIK/2021/Vol10.Iss1.193.

[4] A. A. Saputra and Sutarman, "Implementasi REST API pada Sistem Informasi Tempat Wisata di Yogyakarta Berbasis Android," Jurnal Indonesia: Manajemen Informatika dan Komunikasi, vol. 5, no. 1, pp. 435–444, 2024, doi: 10.35870/jimik.v5i1.496.

[5] M. Tinambunan and S. Sintaro, "Aplikasi RESTful pada Sistem Informasi Geografis Pariwisata Kota Bandar Lampung," Jurnal Informatika dan Rekayasa Perangkat Lunak, vol. 2, no. 3, pp. 312–323, 2021, doi: 10.33365/jatika.v2i3.1230.

[6] A. Rachman and J. Sutopo, "Perancangan UI/UX Aplikasi Sistem Informasi Pariwisata Kabupaten Jepara Menggunakan Metode Design Thinking," Jurnal Indonesia: Manajemen Informatika dan Komunikasi, vol. 5, no. 1, pp. 757–763, 2024, doi: 10.35870/jimik.v5i1.543.

[7] R. Choirudin and A. Adil, "Implementasi REST API Web Service dalam Membangun Aplikasi Multiplatform untuk Usaha Jasa," MATRIK: Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer, vol. 18, no. 2, pp. 284–293, 2019, doi: 10.30812/matrik.v18i2.407.

[8] S. Bin Tambak, A. Musrifah, A. Yulianto, and I. P. Wangi, "Analisa Metode PIECES untuk Perancangan Sistem Informasi Pariwisata Berbasis Web dengan Menggunakan Model Waterfall," REMIK, vol. 7, no. 4, pp. 1903–1916, 2023, doi: 10.33395/remik.v7i4.13090.

[9] M. T. Rahmawita and D. Iriadi, "Aplikasi Objek Wisata Berbasis Android di Kota Pekanbaru," Jurnal Ilmiah Rekayasa dan Manajemen Sistem Informasi, vol. 7, no. 1, pp. 1–10, 2021, doi: 10.24014/rmsi.v7i1.10264.

[10] U. Khaira, M. A. Qodri, A. F. Rosyadi, U. Sulistiyo, and D. Y. Pebrianto, "Rancang Bangun Aplikasi Elok Sekoja sebagai Media Promosi Wisata Seberang Kota Jambi," Jurnal Ilmiah Rekayasa dan Manajemen Sistem Informasi, vol. 9, no. 1, pp. 19–36, 2023, doi: 10.24014/rmsi.v9i1.22057.

[11] P. A. E. Putra, I. P. Satwika, and B. P. W. Nirmala, "Perancangan dan Implementasi Aplikasi Sistem Informasi Pariwisata Berbasis Mobile (TRIPTO)," Jurnal Teknologi Informasi dan Komputer, vol. 7, no. 4, 2021, doi: 10.36002/jutik.v7i4.1528.

[12] N. Handayani, Fandhilah, and H. Mayatopani, "Perancangan UI/UX Aplikasi Destinasi Wisata Berbasis Web Menggunakan Metode Human Centered Design," JOISIE, vol. 7, no. 1, pp. 35–43, 2023, doi: 10.35145/joisie.v7i1.2907.

[13] Y. Purbo, F. Utomo, and Y. Purwati, "Analisis dan Perancangan Antarmuka Aplikasi Wisata Menggunakan Metode User Centered Design," Jurnal Teknologi Terpadu, vol. 9, no. 2, pp. 123–132, 2023, doi: 10.54914/jtt.v9i2.977.

[14] A. A. Rosida and K. Hadiono, "Implementasi Chatbot Berbasis Framework RASA untuk Sistem Rekomendasi Wisata di Semarang," JIPI, vol. 9, no. 3, 2024, doi: 10.29100/jipi.v9i3.5380.

[15] A. D. Ferdian and S. N. Anwar, "Pengembangan Chatbot untuk Informasi Wisata Interaktif di Tangerang Selatan Menggunakan Framework RASA," Jurnal Teknologi dan Sistem Informasi Bisnis, vol. 5, no. 4, pp. 476–483, 2023, doi: 10.47233/jteksis.v5i4.953.

[16] S. R. Cholil, N. A. Rizki, and T. F. Hanifah, "Sistem Rekomendasi Tempat Wisata di Kota Semarang Menggunakan Metode Collaborative Filtering," JIKO (Jurnal Informatika dan Komputer), vol. 7, no. 1, 2023, doi: 10.26798/jiko.v7i1.727.

[17] D. Widiyawati, D. Dedih, and W. Wahyudi, "Implementasi Metode MAUT dan SAW dalam Pemilihan Tempat Wisata di Kabupaten Karawang," Jurnal Interkom, vol. 17, no. 2, pp. 71–80, 2022, doi: 10.35969/interkom.v17i2.231.

[18] A. K. Wardhani and A. Anindyaputri, "Sistem Informasi Pemilihan Tempat Wisata Menggunakan Metode Weighted Product," Indonesian Journal of Technology, Informatics and Science, vol. 2, no. 1, pp. 27–32, 2020, doi: 10.24176/ijtis.v2i1.5649.

[19] F. M. Sabir, Asrul, A. N. Puteri, Amiruddin, Mashud, and S. S. Patalangi, "Rancang Bangun Aplikasi GIS Pusat Oleh-Oleh Toraja Berbasis Web," Jurnal Minfo Polgan, vol. 12, no. 1, pp. 1361–1369, 2023, doi: 10.33395/jmp.v12i1.12742.

[20] R. Simbulan and J. Aryanto, "Implementasi REST API Web Services pada Aplikasi Sumber Daya Manusia," Jurnal Indonesia: Manajemen Informatika dan Komunikasi, vol. 5, no. 1, pp. 552–560, 2024, doi: 10.35870/jimik.v5i1.511.