iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85041

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

Design and Evaluation of TechLedger for Software Service Discovery and Comparison

Vedika Shinde

Abstract: ***TechLedger is a web application for discovering, researching, saving, and comparing software services. It addresses a practical problem in technology selection: information about capabilities, releases, and pricing is spread across vendor sites, and users must keep their own shortlist while reviewing it. The application combines a React and TypeScript interface with a bundled catalog, Gemini-assisted research, and account workspaces stored in Neon PostgreSQL. This paper describes the implementation and examines the public repository at commit 5425d313. Static checks identify 81 services across 12 categories, of which 70 use template defaults for versions, pricing, and capabilities. Seven diagnostic queries illustrate the limits of substring search, and source inspection identifies a mismatch between verification labels and the evidence stored in a profile. These findings support a design that separates missing values from known facts and records evidence for individual fields. The paper also explains how saved services and comparison support a personal software stack. No user study or live research accuracy test is reported.***
Keywords: software service discovery; developer tools; technology comparison; provenance; generative AI.

I. INTRODUCTION

Choosing a database, cloud platform, or AI service often involves several rounds of research. A developer may begin with a familiar product, discover alternatives, and then compare documentation, release notes, and pricing pages. The comparison becomes difficult when vendors describe capabilities differently or charge by different units. Keeping track of the candidates adds another task: notes, bookmarks, and spreadsheets can become separated from the evidence that produced them.

I built TechLedger to keep this work in one place. Users can browse a catalog, request a researched profile, save services, and compare up to three services or releases [1]. A guest workspace supports an initial visit without registration. An authenticated workspace stores the user's selections and research so they can return to them from another browser or device.

This paper examines how the application connects those views to its data and research routes. It asks two questions: how can one workspace support service discovery and comparison, and what limits the reliability of the information it presents? The contribution is an implementation account supported by static checks, with concrete changes proposed for data provenance and search. Benefits such as reduced research time remain questions for evaluation.

II. BACKGROUND AND DESIGN RATIONALE

Category browsing and keyword search support different stages of investigation. Hearst's work on faceted search explains how users can narrow a collection through categories while retaining the ability to search [2]. TechLedger uses twelve fixed technology categories and a text filter. Cards provide an overview, and a profile drawer supplies details without requiring users to leave the catalog.

AI-assisted research introduces a different problem: the response can be well structured while its claims remain weakly supported. Gemini's Search grounding connects a request to web material, while structured output constrains the response format [3], [4]. In TechLedger, these mechanisms make a profile easier to process and inspect; they do not independently validate its pricing or release information.

III. SYSTEM ARCHITECTURE

TechLedger uses React 19 and TypeScript with Next.js App Router conventions. vinext and Vite provide the development and build integration, Tailwind CSS supports styling. The client imports the baseline catalog directly. Server routes handle research, enrichment, account persistence, and exchange-rate requests. Figure 1 separates these paths from the guest workspace stored in the browser.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

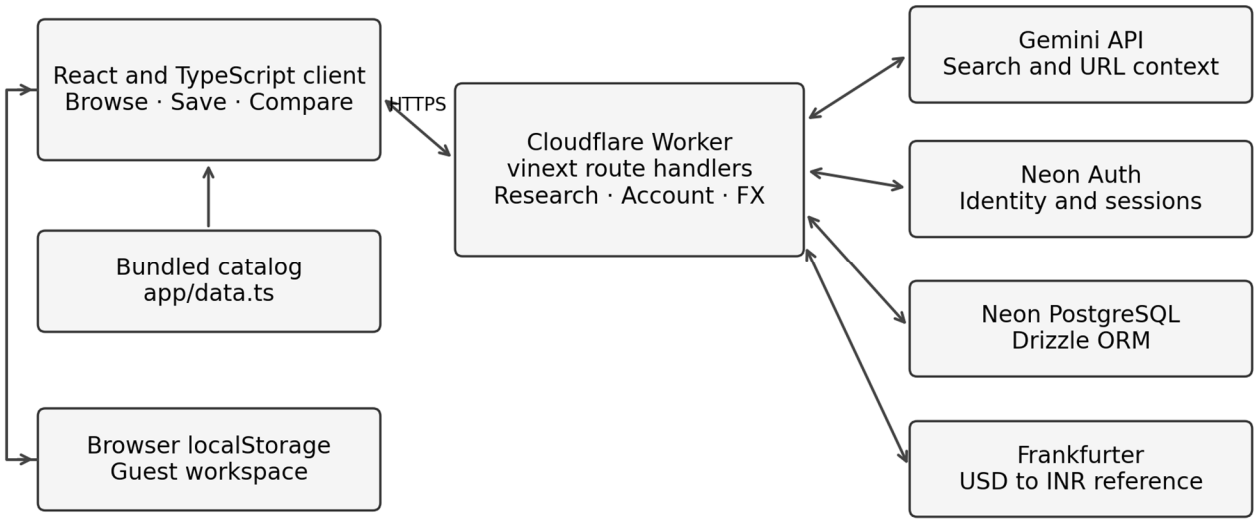


Figure 1 Implemented architecture of TechLedger. Arrows indicate data exchange between components.

A. Shared Service Records

Bundled and researched profiles use the same Service type. Each record contains an identifier, name, provider, category, description, version, release date, pricing text, capabilities, release history, and links. Optional fields hold plan prices, use cases, integrations, source URLs, and change metadata. The client gives a researched record precedence when its identifier matches a bundled record. This common representation lets the catalog, profile drawer, saved view, and comparison table display the same candidate.

A shared type simplifies the interface but does not make all records equally complete. Some catalog entries contain detailed attributes; others contain defaults. The current type also stores provenance mainly at the profile level. It does not associate each displayed capability, version, or price with a specific evidence passage.

B. Workspaces and Authentication

Guest mode keeps saved identifiers, comparison selections, researched profiles, alert state, and preferences in localStorage. This storage belongs to the browser and origin that wrote it [6]. With a Neon Auth session, the client reads the workspace through GET /api/account and sends debounced updates through PUT /api/account. The server scopes records to the authenticated user and checks the size and shape of incoming arrays. Drizzle ORM maps the workspace to Neon PostgreSQL [7]. The database stores workspace state and snapshots of saved profiles, allowing users to retain more than a list of product names. Account storage supports access from another device when the services are configured. Guest state is not automatically merged into a new account.

C. Engineering approach

The implementation follows the data flow from a service record to a reusable view, then adds server routes for research and persistence. Configuration keeps the Gemini key on the server and separates the database URL from the Neon Auth endpoint. The repository records an initial implementation followed by branding and setup documentation changes on 29 September 2026 [1]. It does not establish a longer development timeline.

IV. RESEARCH AND COMPARISON

A. Research Requests

The research route accepts a service name of 2 to 100 characters at POST /api/services/search. It checks for a server-side Gemini key, includes a matching catalog entry as context when available, and requests a structured response using Google Search and URL context. The prompt asks for official documentation, pricing pages, and changelogs, and instructs the model to return unknown values when evidence is insufficient.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The route parses the response, normalizes lists, gathers grounding and pricing URLs, and returns a Service record. POST /api/services/enrich applies a similar process to an existing catalog entry. Returned profiles can be retained in the research library and user workspace. If research is unavailable, the static catalog remains usable. The pipeline runs on demand; the inspected repository contains no scheduled ingestion process.

The stored timestamp records when the request produced a profile. It does not prove that every source was current or every field was checked. This distinction matters when a recent response contains an unknown version or an old price. Source links help the user investigate those fields, but their presence alone cannot resolve the uncertainty.

B. Service and Release Comparison

The comparison view has three selection slots and four sections: Overview, Pricing, Capabilities, and Releases. Corresponding attributes appear in the same row. Users can compare different services or choose entries from a service's release history. A release selection copies the current profile and overrides the version and release date. Other attributes, including pricing and capabilities, remain those of the current record. Historical selections therefore require care: they are not complete snapshots of past releases.

For comparable numeric prices, the server computes percentage movement as 100 × (current − previous) / previous, provided the previous amount is nonzero. The prompt requests the same plan and billing unit, but the application does not independently verify that condition. Cross-service pricing remains descriptive and does not estimate total cost for a workload.

The /api/fx route obtains a USD-to-INR reference rate from Frankfurter [8]. The interface displays a converted estimate when it can parse a dollar amount. Taxes, regional prices, card rates, and provider billing rules are outside that calculation.

C. Saved Services and Updates

Saving a service creates a shortlist that can represent tools already used in a project or alternatives under consideration. The saved view offers grid and list layouts, removal and comparison actions, and CSV export. Research history and preferences provide context when the user returns. The dashboard derives release and pricing items from available profiles. Users can mark items read or mute a service. These controls help organize a review, but the feed does not detect changes continuously. Its items describe stored records rather than changes observed between successive provider checks.

V. INTERFACE AND PRACTICAL USE

The Explore view gives users a direct research entry point and quick-search suggestions. Category navigation then supports browsing alternatives. Cards expose the main attributes and provide save and compare controls, while the profile drawer contains further details and source links.

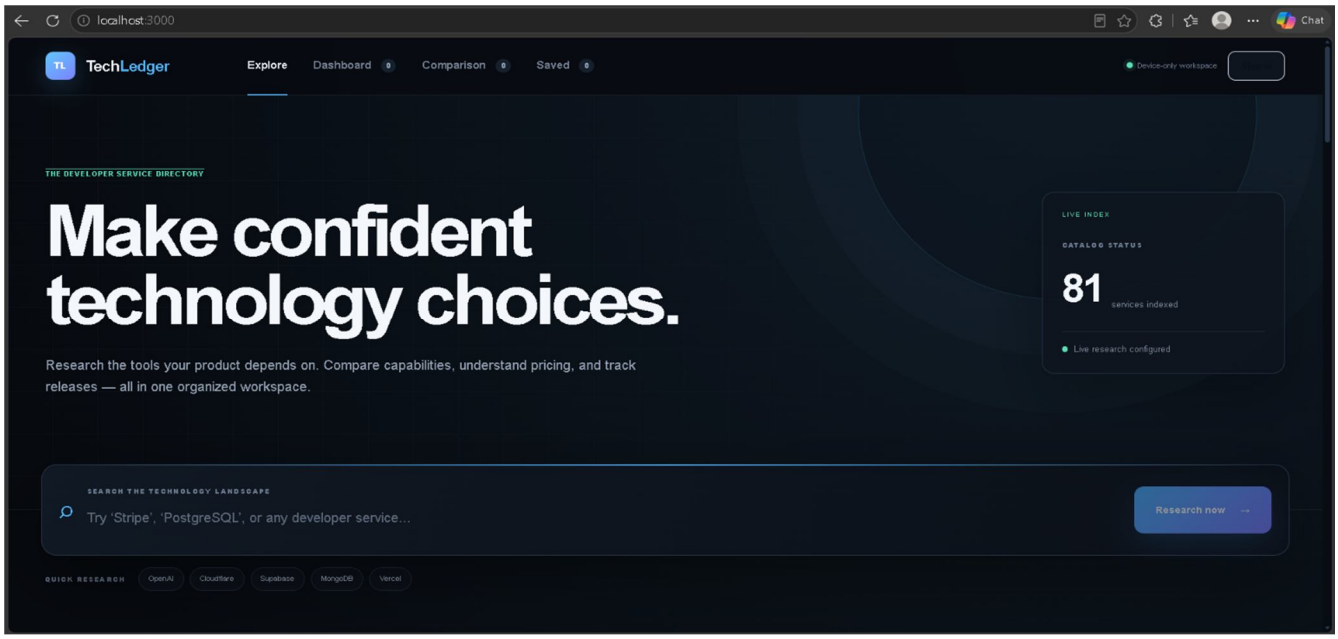


Figure 2 Explore view with the research input and quick-search suggestions.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

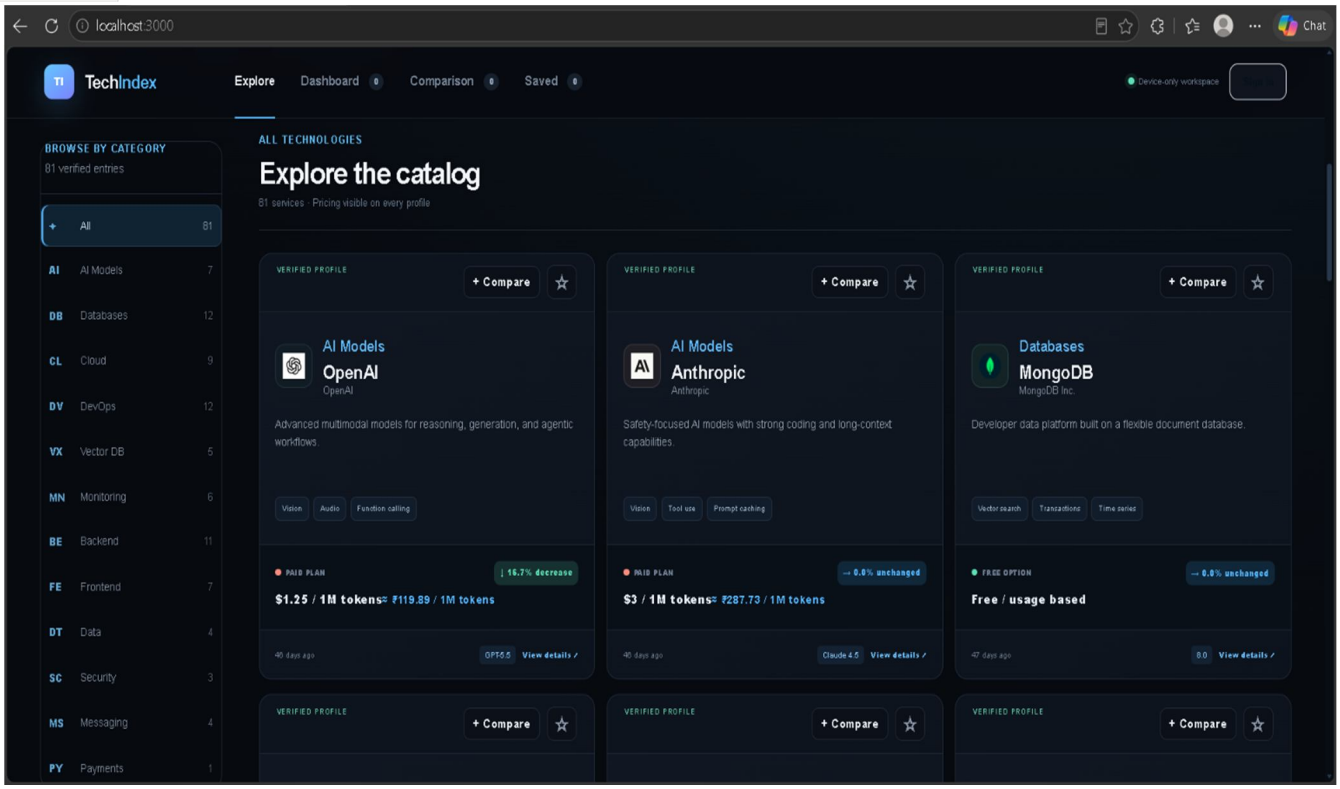


Figure 3 Category navigation and service cards with save and comparison controls.

The comparison view aligns attributes across three selections. The saved view keeps a reusable shortlist and exports its current contents. A developer planning a backend could save PostgreSQL, MySQL, and Redis, inspect the relevant profiles, compare requirements, and export a summary for discussion. This is an intended workflow, not a measured study of decision quality.

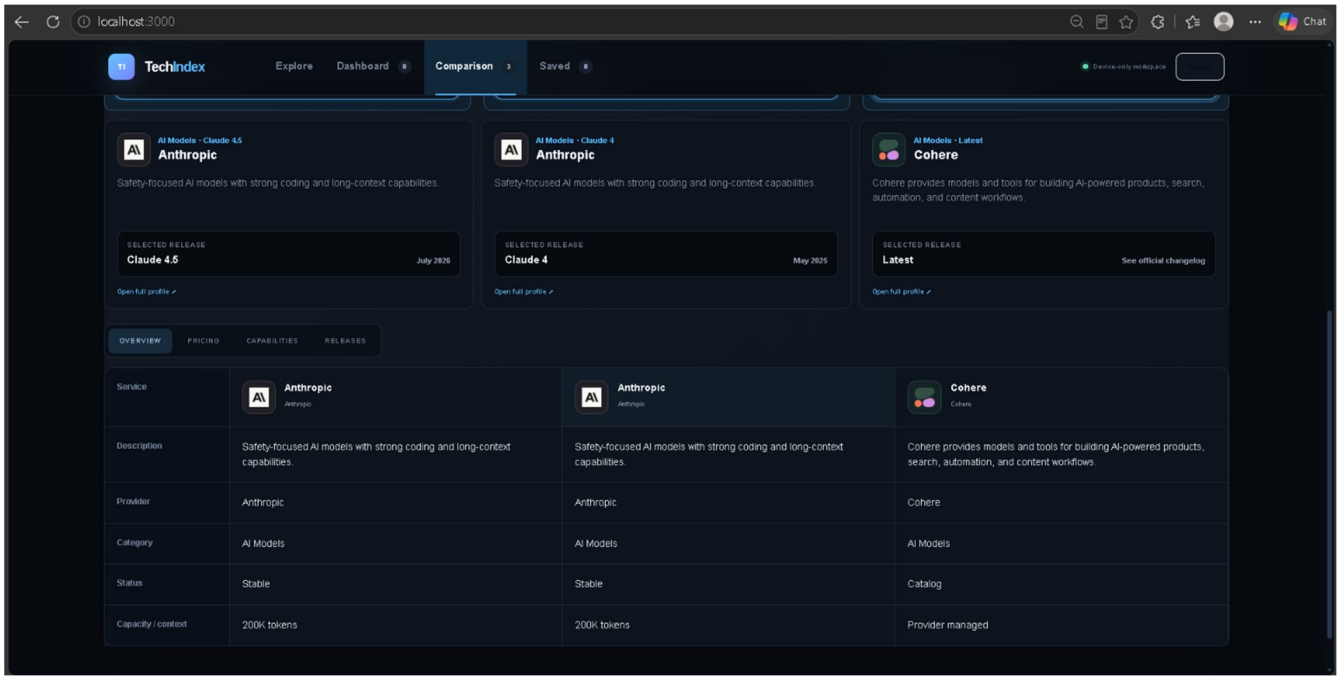


Figure 4 Comparison of two Claude release selections and Cohere.

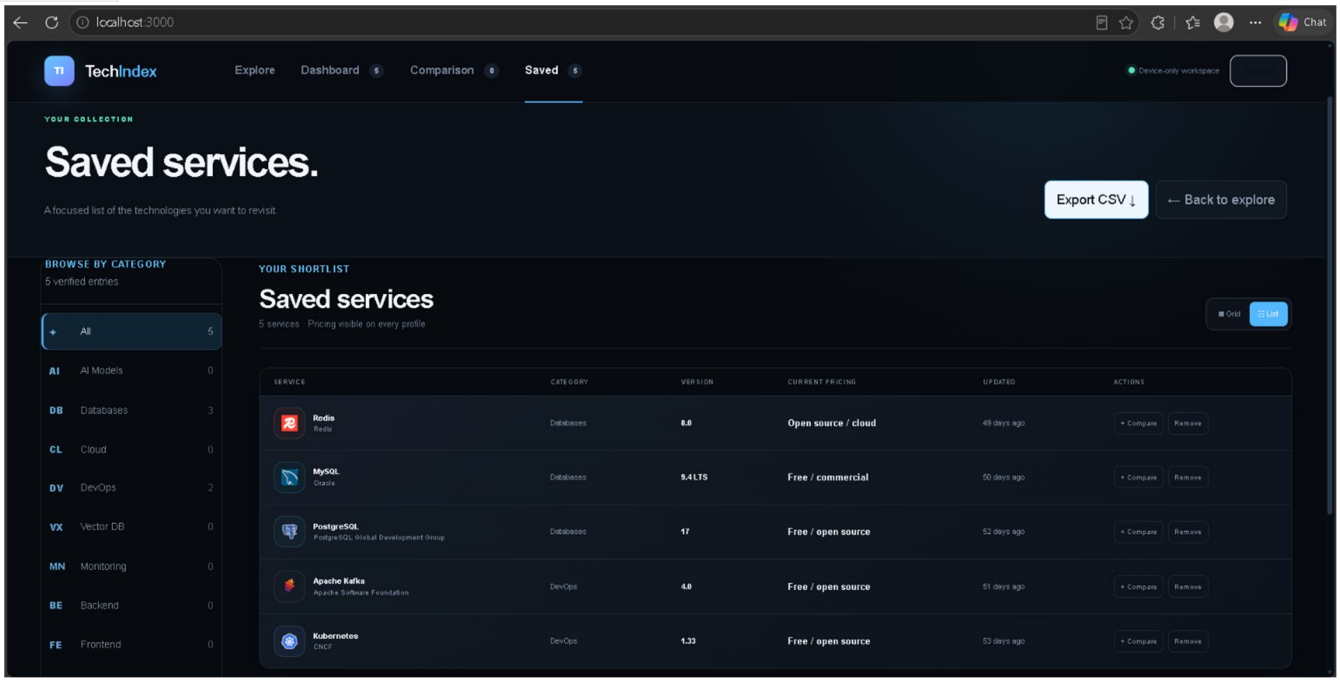


Figure 5 Saved services in list view with CSV export and a guest workspace.

TechLedger tracks information about software tools and services. The “device-only workspace” label refers to browser storage. Hardware discovery, serial numbers, ownership, telemetry, and maintenance records are outside the current implementation.

VI. STATIC EVALUATION

A. *Method and Scope*

The evaluation covers repository commit 5425d313e193cc6d1bf2914924d129b366366503 [1]. The catalog module was loaded in Node.js with TypeScript stripping enabled. Counts were computed over the resulting records, and seven fixed diagnostic queries were evaluated using the same case-insensitive substring expression as the Explore view. The expression searches name, provider, category, description, and capabilities, retaining catalog order. Table 2 reports every query and returned identifier.

Source inspection also examined the client’s labels, comparison selection, storage, export, and alert rules. These checks do not exercise Gemini, Neon, authentication, or the hosted runtime. The diagnostic queries are examples selected to expose behavior, not a representative benchmark; no retrieval accuracy rate or statistical comparison is inferred from them.

B. *Catalog Completeness*

Measure	Count	Interpretation
Bundled services	81	Entries across 12 technology categories
Records with template defaults	70 of 81	86.4 percent use the generated baseline template
Records with detailed attributes	11 of 81	Manually defined records; factual correctness not audited
Records with plan-level pricing	0 of 81	The optional pricingDetails field is absent
Derived release and price items	11 and 1	Items computed from stored metadata

Table 1 Counts reproduced from the bundled catalog at the inspected commit.

The 70 template records use “Latest” for version, “See provider pricing” for pricing, and “See official changelog” for release date. Their capabilities are the category name, “Official documentation”, and “Release tracking”. These entries provide directory coverage but little evidence for a detailed comparison. The eleven manually defined records contain more specific values, which were not compared with current vendor pages in this evaluation.

C. Search Behavior

Query	Returned service identifiers	Observation
postgresql	postgresql	Exact name matches
kuber	kubernetes	Prefix matches
k8s	None	Alias is not indexed
postgress	None	Misspelling is not corrected
claude	None	Explore does not index release names
vector database	pinecone	Literal phrase occurs in one profile
message queue	None	No matching phrase in the indexed fields

Table 2 Complete results for seven diagnostic queries against the bundled catalog.

These results follow directly from substring matching. A common abbreviation or misspelling requires an explicit matching string. A capability query depends on the wording of the description and tags. The comparison picker uses a different expression that includes release names and notes, so a query can behave differently between the two views.

VII. DISCUSSION AND IMPROVEMENTS

A. Evidence and Freshness

Bundled cards display “Verified profile”, and the category panel describes entries as verified. The label is selected from the profile’s dataSource value rather than a recorded verification process. The data contains no field-level verification state or timestamp. The label therefore communicates more certainty than the stored evidence supports.

Freshness has a similar limitation. Catalog timestamps are fixed values, while researched profiles receive the request time. Neither establishes when an individual price or release claim was checked. A clearer model would store a source URL, retrieval time, and verification state for each material field. Pricing evidence should also record the plan, unit, currency, and region. The interface could distinguish unverified, source-linked, and manually checked values using explicit definitions.

B. Missing Data and Release History

Missing values should be stored as null and displayed as unavailable. A placeholder such as “Latest” can otherwise look like a valid release, and a filled comparison cell can conceal the absence of evidence. Historical releases need their own attribute snapshots, or inherited fields should be marked clearly. This would let a reader tell whether a comparison describes two releases or two labels attached to the same current profile.

C. Search and Interface Consistency

An alias table could connect common terms such as k8s to Kubernetes. Capability tags and release-name indexing would make matching less dependent on prose. Typo tolerance could address misspellings, with a held-out query set used to evaluate the change. The same search expectations should apply across Explore and the comparison picker.

The supplied README lists a Ctrl/Cmd + K shortcut, but the inspected client contains no handler for it. The global key listener closes the profile drawer on Escape. Documentation should describe implemented behavior, and a shortcut should be tested before it is presented as a feature.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

D. Evaluation Still Required

A live data-quality study should compare researched versions, release dates, prices, and billing units with dated official sources. It should report correctness, missingness, and whether each citation supports the associated claim. Repeating the checks would measure how quickly profiles become stale. A usability study should ask developers to find a service, save alternatives, compare them, and locate the source of a price. Completion time, errors, and task success could be compared with ordinary web research using a counterbalanced task order.

Accessibility remains to be evaluated against WCAG 2.2, including keyboard reachability, focus order, contrast, and target size [9]. Security review should examine generated links, CSV formula handling, account retention, and the treatment of retrieved web content as untrusted input. The present evaluation provides no security or accessibility certification.

VIII. LIMITATIONS

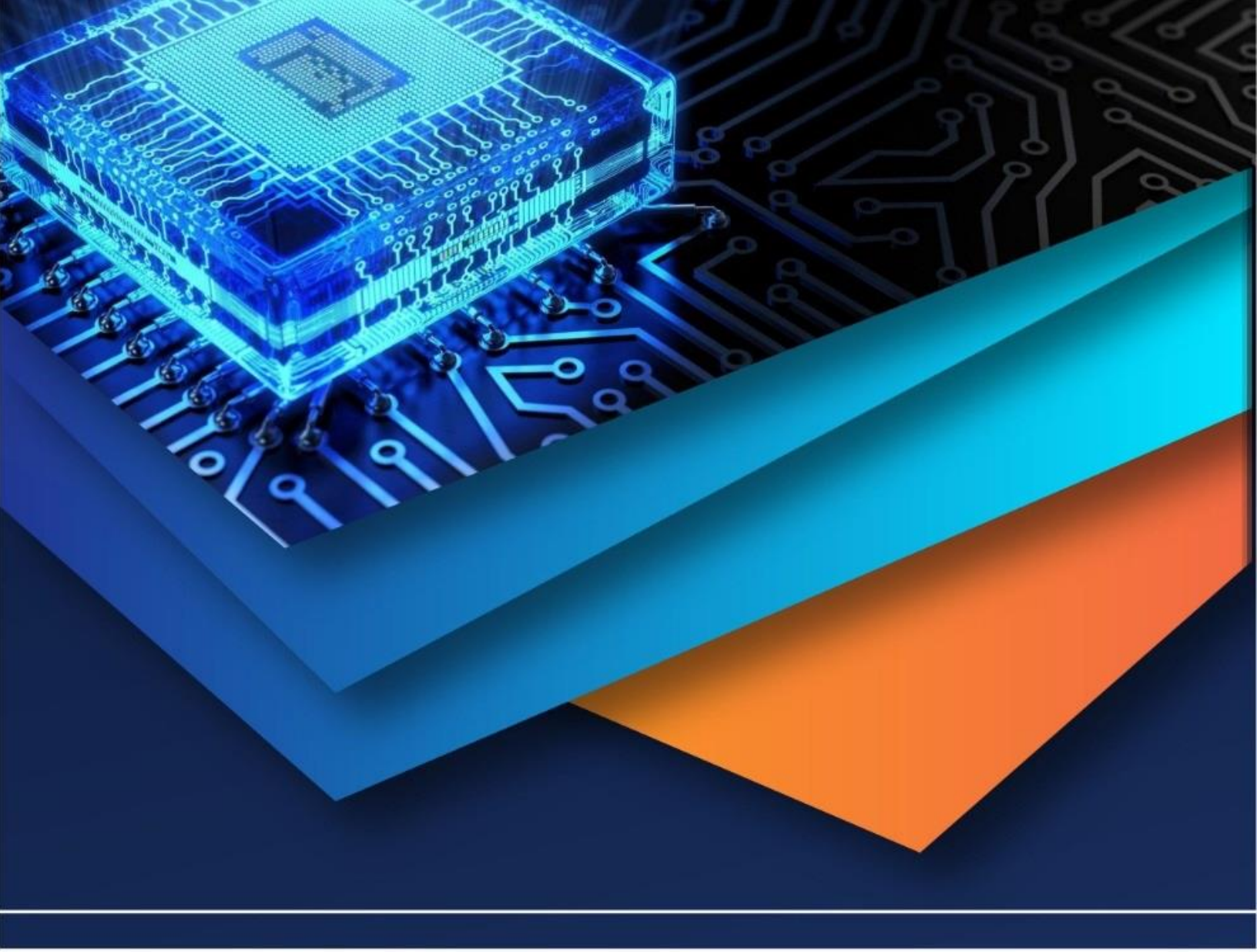
The static checks cover one commit and its bundled data. The author also built the application, so an independent review would strengthen the analysis. Screenshots establish the existence of views, not successful production operation. No live research accuracy, latency, scalability, or user satisfaction measurements are available.Scheduled refresh, push notifications, requirements-based scoring, and hardware inventory are possible extensions. They would require additional implementation and evaluation. Claims that TechLedger reduces research time or improves technology decisions should remain unmeasured objectives until those studies are completed.

IX. CONCLUSION

TechLedger connects service discovery, AI-assisted research, saved workspaces, and comparison in one application. Its shared service model allows the same candidate to move from browsing to a retained shortlist and a side-by-side review. The static evaluation also identifies limits that affect interpretation: most catalog entries contain template defaults, verification labels exceed the evidence recorded, and substring search misses common query forms. The next development step is to make the quality and age of each field visible, then evaluate researched profiles and user tasks. That would give users a firmer basis for deciding which tools belong in their software stack.

REFERENCES

[1] V. Shinde, TechLedger, GitHub repository, commit 5425d313e193cc6d1bf2914924d129b366366503, 29 September 2026. https://github.com/gitvedhub/TechLedger/tree/5425d313e193cc6d1bf2914924d129b366366503
[2] M. A. Hearst, “Design Recommendations for Hierarchical Faceted Search Interfaces,” ACM SIGIR Workshop on Faceted Search, 2006. https://flamenco.berkeley.edu/papers/faceted-workshop06.pdf
[3] N. F. Liu, T. Zhang, and P. Liang, “Evaluating Verifiability in Generative Search Engines,” Findings of EMNLP 2023, pp. 7001–7025, 2023. doi: 10.18653/v1/2023.findings-emnlp.467.
[4] Google AI for Developers, “Grounding with Google Search,” Gemini API documentation. https://ai.google.dev/gemini-api/docs/google-search
[5] Google AI for Developers, “Structured outputs,” Gemini API documentation. https://ai.google.dev/gemini-api/docs/structured-output
[6] Mozilla, “Window localStorage property,” MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
[7] Neon, “Serverless driver for JavaScript and TypeScript,” release guide. https://neon.com/blog/serverless-driver-ga
[8] Frankfurter, API documentation. https://frankfurter.dev/
[9] W3C, Web Content Accessibility Guidelines 2.2, W3C Recommendation, 2023. https://www.w3.org/TR/WCAG22/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)