https://doi.org/10.47641/JRS.2026-13-1.09
Palestine Technical College Journal for Research and Studies **ISSN 2518-5721**
Vol. 13, **No.** 1, 2026, **pp.** 286-311

Received: (11-08-2026), **Accepted:** (22-10-2026)

KnowVerse: An Academic Discussion Platform Based on an Auditable Reputation Ledger and Role-Based Access Control

Mohammed Osama AbuSaada[1]	**Amna Salama Abu Kheshipa**[2*]

Palestine Technical College – Deir El-Balah, Palestine
[1] mohammedabusaada126@gmail.com ORCID: 0009-0006-6171-843X
[2] azidan@ptcdb.edu.ps ORCID: 0009-0003-9092-2497
* Corresponding author

Abstract:
Online academic discussion environments need transparent and verifiable mechanisms for evaluating user contributions, yet existing platforms may emphasize broad social engagement or lack structured academic moderation and reputation governance. This paper presents KnowVerse, an academic discussion platform, implemented with Laravel, MariaDB, and WebSockets, built around an append-only, auditable reputation ledger and a Role-Based Access Control (RBAC) model. Every reputation-related action is recorded as an independent ledger entry, and every retraction as a compensating entry referencing the canceled action, so each user's score can be traced, verified, and recomputed. Accumulated reputation determines a platform-defined academic standing and unlocks a few participation privileges, whereas administrative authority is assigned manually through a five-tier RBAC model and never derived from reputation. An automated suite of 126 tests assessed access control, injection resistance, anti-farming safeguards, and ledger consistency. Concurrency experiments with up to 16 parallel processes on MariaDB confirmed that no ledger entry was reversed twice and no ledger invariant was violated, and showed that double reversals occur when both row locks and the unique constraint are removed. An ablation against a counter-only design showed that the ledger increases mean vote latency by 4.9 ms (27%) and 95th-percentile latency by 18 ms (55%), while enabling every injected alteration of the stored score to be detected and accurately restored. Performance was evaluated on a single workstation across repeated trials. The results show a transparent, auditable mechanism for reputation computation and role-based access control; its effect on user trust and interaction quality is left to future work.
Keywords: Academic Discussion Platforms, Auditable Reputation Ledger, Append-Only Ledger, Role-Based Access Control, Reputation Management, Concurrency Control, Laravel.

License: CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/). © Copyright held by the authors.

Introduction

Digital learning environments have significantly transformed academic knowledge sharing, making online discussion platforms important for collaboration among students, researchers, and educators. Despite their widespread adoption, many platforms provide limited transparency and accountability when user contributions are evaluated. General-purpose platforms such as Reddit (Reddit Inc., n.d.) and Quora (Quora Inc., n.d.) support broad user participation but are not designed specifically around scholarly governance, while academic networks such as ResearchGate (ResearchGate GmbH, n.d.) focus primarily on publication sharing rather than structured scholarly discussion. In addition, many traditional reputation systems represent reputation as a simple cumulative score, which limits visibility into how individual reputation changes arise (Jøsang et al., 2007; Resnick et al., 2000).

To address these limitations, this paper presents KnowVerse, an academic discussion platform based on an Auditable Reputation Ledger and Role-Based Access Control (RBAC). The platform records every reputation-related action as a traceable ledger transaction and maps rule-compliant contributions to a transparent platform-defined standing and a graduated ladder of participation privileges, while administrative roles are assigned separately by administrators. The proposed approach provides transparent and auditable mechanisms for reputation management and access control, while supporting real-time academic interaction and moderation. The platform is designed specifically for academic knowledge exchange, where structured discussion, peer interaction, and recognition of participation provide the educational context for the proposed technical mechanisms.

The individual mechanisms used by KnowVerse are established: append-only logs, compensating entries, and RBAC are well-known techniques. The contribution of this work lies in their integration and deliberate decoupling within an academic discussion platform: an append-only reputation ledger whose consistency is enforced by the database and verified under concurrent load, combined with administrative roles that are assigned explicitly and can never be obtained through reputation, alongside an open, documented, and repeatable evaluation of the platform's correctness, cost, and performance. The paper is therefore a design and technical-verification study: it evaluates whether these mechanisms behave as specified, under concurrent load, and at what cost, and it does not measure their effect on users' trust or on perceived fairness. KnowVerse is an academic project and is not affiliated with any commercial service of a similar name.

Literature Review

The landscape of online collaboration includes general-purpose social platforms, community question-answering (CQA) systems, and academic research networks. General-purpose platforms such as Reddit and Quora support large-scale user participation and knowledge exchange, but their interaction and ranking mechanisms are primarily designed for broad online communities rather than structured academic discussion. Research on Reddit has highlighted the challenge of sustaining sufficient contributions to support the production and maintenance of community content (Gilbert, 2013). In contrast, focused question-answering platforms such as Stack Overflow demonstrate how community activity and contribution mechanisms can generate value through questions, answers, and user interactions (Anderson et al., 2012). Stack Overflow also operates the most widely recognized reputation system in community question answering: users earn points when others vote on their posts or accept their answers, and higher

scores unlock moderation abilities such as editing or closing other users' questions (Movshovitz-Attias et al., 2013; Stack Overflow, n.d.). Bosu et al. (2013) examined which activities allow Stack Overflow users to build reputation, and a study of an online mathematics question-answering community found that contributors' reputation predicts how others judge the quality of their contributions (Tausczik & Pennebaker, 2011); Srba and Bielikova (2016) survey and classify the wider body of CQA research.

Community question-answering research has also investigated the identification of knowledgeable users and experts. Pal et al. (2012) examined the identification of experts and potential experts in community question-answering environments, highlighting the importance of analyzing user activity and question-selection behavior. Similarly, Wang et al. (2018) reviewed expert recommendation approaches in community question-answering systems and emphasized the role of expertise identification in facilitating effective knowledge exchange. These studies demonstrate the value of mechanisms that recognize knowledgeable contributors; however, they primarily focus on expertise identification and recommendation rather than on maintaining an auditable record of how reputation is accumulated, modified, or reversed.

More broadly, reputation systems aggregate feedback about past behavior to guide future interactions (Resnick et al., 2000), and surveys of trust and reputation systems stress that their usefulness depends on robustness against manipulation such as self-promotion and collusion (Jøsang et al., 2007). Hoffman et al. (2009) classify attacks on reputation systems and their defenses; among them is the Sybil attack, in which one person controls many identities, and Douceur (2002) showed that such attacks cannot be prevented without a trusted authority that certifies identities. This literature concerns how reputation is computed and whether it can be trusted; it gives less attention to how an individual score can be audited after the fact.

For the scientific community, academic social networking platforms such as ResearchGate have expanded opportunities for researchers to disseminate publications, communicate scholarly work, and measure aspects of scholarly activity (Thelwall & Kousha, 2015). However, academic networking and publication-sharing environments do not necessarily provide the structured, discussion-oriented interaction and governance mechanisms required for transparent academic communities. Van Noorden (2014) similarly described the growing role of online collaboration and social networking in scientific communication, indicating the broader transformation of scholarly interaction through digital platforms.

From a security and governance perspective, RBAC provides an established mechanism for managing permissions according to predefined roles and responsibilities. Sandhu et al. (1996) formalized RBAC models in which access permissions are associated with roles and users obtain permissions through role assignments, and these models were developed further through NIST standardization work (Ferraiolo et al., 2001). A central principle of RBAC is that permissions follow from roles that are assigned deliberately, rather than from a user's activity. This separation is particularly relevant to academic platforms where participation, moderation, and administrative responsibilities need to be distinguished. More recent work has also examined the application of RBAC in university-based digital environments, demonstrating its relevance to access management in academic systems (Ratna et al., 2025).

The reputation ledger in KnowVerse is inspired by the event-sourcing pattern, in which the state of an entity is derived from an append-only sequence of recorded events rather than stored only as its latest value (Fowler, 2005), and in which corrections are made by appending new facts

instead of overwriting old ones (Helland, 2016). Undoing an effect by recording a compensating action is likewise an established technique in transaction processing (Garcia-Molina & Salem, 1987). Industrial experience shows that event-sourced systems also carry engineering costs, notably the evolution of the schema of stored events (Overeem et al., 2021). Append-only logs can additionally be made tamper-evident with cryptographic structures such as hash chains (Crosby & Wallach, 2009; Schneier & Kelsey, 1999); KnowVerse does not use such structures, so its guarantee is auditability and detectable inconsistency rather than tamper-proofing. Append-only designs also sit in tension with the right to erasure under the General Data Protection Regulation (GDPR; European Parliament and Council of the European Union, 2016), a conflict examined by Politou et al. (2018); **the "Append-Only Ledger and the Right to Erasure" section** describes how KnowVerse resolves it.

KnowVerse differs from Stack Overflow in three primary respects:

1. **Governance Separation:** Stack Overflow uses reputation itself to grant administrative and moderation powers over other users' content (such as editing or closing questions), whereas KnowVerse limits reputation strictly to participation privileges and assigns administrative authority through explicit RBAC roles.
2. **Auditability and Traceability:** KnowVerse explicitly documents the ledger structure and its consistency invariant, and uses compensating entries for all retractions to ensure that reputation history remains fully auditable and recomputable.
3. **Empirical Verification:** KnowVerse provides a fully documented, repeatable evaluation verifying ledger consistency and performance under concurrent database loads.

In summary, the existing literature addresses the components of an academic discussion platform in isolation: CQA studies analyze reputation and expertise (Anderson et al., 2012; Bosu et al., 2013; Movshovitz-Attias et al., 2013; Pal et al., 2012; Srba & Bielikova, 2016; Tausczik & Pennebaker, 2011; Wang et al., 2018); reputation research focuses on score computation and anti-manipulation (Douceur, 2002; Hoffman et al., 2009; Jøsang et al., 2007; Resnick et al., 2000); event sourcing provides append-only auditability (Fowler, 2005; Garcia-Molina & Salem, 1987; Overeem et al., 2021); and RBAC formalizes access governance (Ferraiolo et al., 2001; Sandhu et al., 1996). However, prior work does not combine an auditable, append-only reputation ledger with a strict separation between earned privileges and assigned roles, nor does it verify such an integration empirically. KnowVerse addresses this gap by combining auditable reputation management with role-based administrative governance in a unified, transparent academic environment.

Table (1): Comparison of KnowVerse with Existing Online Discussion and Scholarly Platforms

Platform	Strengths	Main Limitations	KnowVerse design approach
Quora (Quora Inc., n.d.)	Broad access to user-generated questions and answers across diverse topics.	No formal scholarly governance model or verified academic role structure.	Separates earned participation privileges from administrative roles, which are assigned explicitly.
Reddit (Gilbert, 2013; Reddit Inc., n.d.)	Large-scale community discussion and diverse user-generated content.	Relies on pseudonymity and popularity-based ranking (karma) rather than expertise.	Records every reputation change in an append-only, auditable ledger.

Platform	Strengths	Main Limitations	KnowVerse design approach
Stack Overflow (Anderson et al., 2012)	Accurate answers and an effective community evaluation mechanism.	Q&A format with limited room for open-ended scholarly dialogue; reputation also confers moderation powers.	Discussion threads with Markdown and LaTeX; reputation grants privileges over one's own participation only.
ResearchGate (ResearchGate GmbH, n.d.; Thelwall & Kousha, 2015)	High credibility through verified researcher profiles and publication tracking.	Oriented toward publication sharing, with limited real-time discussion.	Real-time discussion with notifications and moderation.

Table (1) is a conceptual comparison of design features: proprietary platforms could not be examined or benchmarked directly, so no comparison of performance or outcomes is implied. Based on the identified gap, the following section presents the methodology and system architecture adopted for the development of KnowVerse.

Methodology

KnowVerse was developed using the Agile Scrum methodology (Schwaber & Sutherland, 2020) to support iterative development and continuous refinement. The platform is designed as an academic discussion environment that supports transparency, accountability, and contribution-based participation through the integration of an Auditable Reputation Ledger and RBAC. The system follows a layered architecture implemented using Laravel and a MySQL-compatible database (MariaDB). A core component of the platform is the Auditable Reputation Ledger, where every reputation-related action is recorded as a separate entry containing the action type, the signed point value, the item it relates to, and a timestamp. The ledger is append-only: entries are never updated or deleted, and a retraction is recorded as a new compensating entry referencing the original action. Database triggers enforce this rule by rejecting any direct update or deletion of a ledger entry, including one that bypasses the application. The single exception is the permanent deletion of an account at the account holder's request, described in the "Append-Only Ledger and the Right to Erasure" section. This approach enables transparent tracking, verification, and auditing of user contributions. Reputation points are awarded for defined contribution events, such as creating discussions, receiving community votes, and having a reply selected as the Author's Pick. Four key terms are defined and used consistently throughout this paper:

- **Reputation:** A user's point total, which always equals the sum of the user's ledger entries.
- **Academic Standing:** The title derived from that point total (e.g., Novice Scholar, Junior Researcher, Active Contributor).
- **Privilege:** A capability over the user's own participation that is unlocked automatically when reputation reaches a predefined threshold.
- **Role:** One of five RBAC levels (Guest, Unverified User, Verified User, Moderator, Administrator) that determines what a user may do, including any authority over other users' content or accounts.

Role changes never depend on reputation: a registered user becomes a Verified User upon email confirmation, while Moderator and Administrator roles are granted exclusively through manual administrative assignment. This separation prevents reputation farming from yielding moderation or administrative authority. The resulting framework is transparent and

accountable: participation privileges are earned through scholarly contributions, while administrative authority is assigned separately, as depicted in Figure (1).

Note on Terminology: In this study, a "validated contribution" refers to a contribution that satisfies the platform's defined interaction and governance rules; it does not imply independent expert verification of its scholarly correctness.

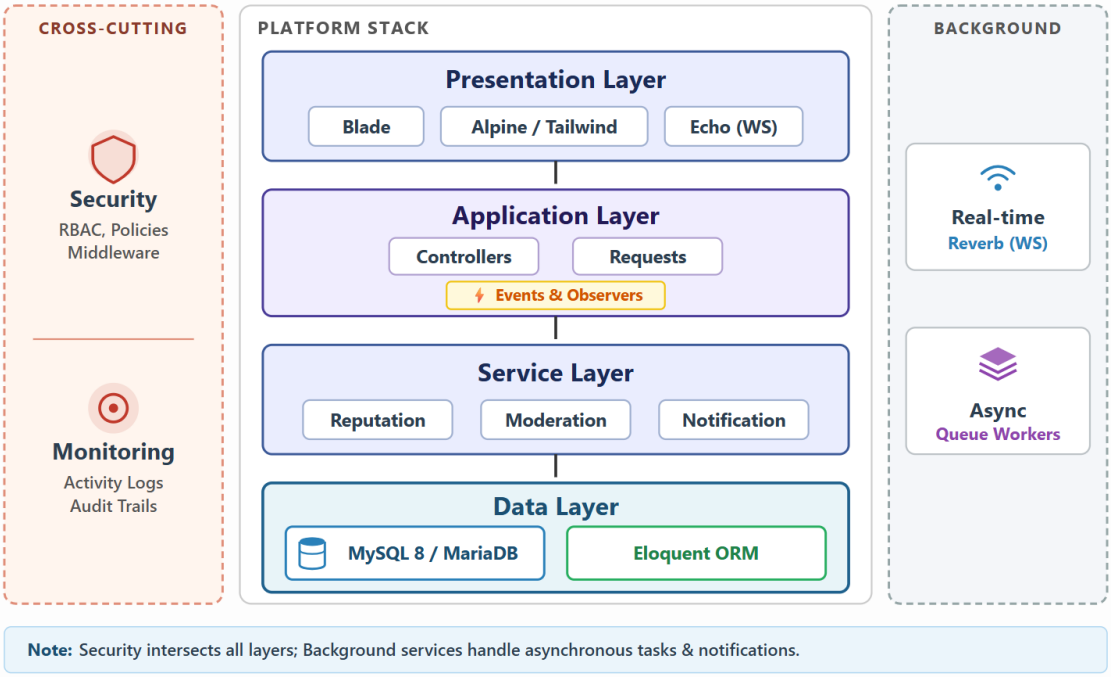


Figure (1): Overall System Architecture

Auditable Reputation Ledger Model

To reward contributions consistently, a rule-based scoring algorithm was developed. Points are awarded according to predefined contribution and interaction events representing defined platform activities. The system assigns each action a weight, recorded as a signed ledger entry (Table (2)).

Table (2): Reputation Action Scoring Policy

Action (recipient)	Points
Create a discussion (author)	+5
Post a comment (author)	+2
Receive an upvote (discussion / comment)	+5 / +2
Receive a downvote (discussion / comment)	−2 / −1
Reply selected as Author's Pick (reply author)	+15
Select an Author's Pick (discussion author)	+2
Content removed by moderation (content author)	−10
Report upheld by a moderator (reporter)	+2

The cumulative reputation score is mapped to predefined platform-standing levels that reflect participation within the platform and determine the participation privileges available to users. Table (3) summarizes the reputation thresholds and their corresponding standing labels. The point values were determined by design rather than calibrated from pilot data, because KnowVerse has not yet been used by a real community. Their relative sizes follow the reputation model of Stack Overflow (n.d.), whose accepted answers likewise earn +15 for the answer's author and +2 for the person who accepts it. Three principles apply:

1. Contributions and endorsements on discussions weigh more than those on comments (+5 versus +2), reflecting the greater effort involved.
2. Negative feedback is smaller than positive feedback (−2 versus +5), ensuring a single dissenting vote cannot cancel a valid contribution.
3. The strongest endorsement—a reply selected by the discussion author—earns the largest award (+15), while a moderation penalty (−10) outweighs any single vote.

The standing thresholds roughly double from one level to the next (50, 100, 250, 500, 1,000), requiring exponentially greater participation for higher titles. All point weights and thresholds are configuration settings; the ledger's core guarantees—traceability, single-reversal enforcement, and mathematical consistency between scores and ledger entries—do not depend on these specific values. Calibrating these weights based on empirical usage data is left to future work.

Author's Pick Mechanism: Author's Pick serves as the platform's best-answer mechanism: the author of a discussion may select the single reply that best resolves the topic. The reply's author receives the largest award (+15) because the selection is an endorsement by the question originator, who is best suited to evaluate its resolution. The discussion author receives +2 points as an incentive to actively select a satisfactory response. Each discussion can feature at most one Author's Pick. If the author changes their selection to a different reply, the reputation effects of the prior selection are automatically reversed via compensating entries before the new awards are logged. Consequently, switching selections cannot be exploited to farm reputation through repeated toggling (Table (6), A7).

The privileges unlocked at each level are listed in Table (3). These titles are platform-defined participation labels and do not represent formal academic qualifications, institutional ranks, or professional credentials. The 500 and 1,000 levels are honorific: they change the displayed title but unlock no further privilege. Table (4) contrasts these reputation-based privileges with administrative roles.

Table (3): Academic Standing Levels Based on Reputation Points

Academic Standing	Reputation Points	Privilege unlocked
Novice Scholar	< 50	—
Junior Researcher	50–99	Downvoting
Active Contributor	100–249	Exemption from the external-link limit
Associate Researcher	250–499	Pinning one's own discussions
Senior Scholar	500–999	— (honorific)
Distinguished Fellow	≥ 1,000	— (honorific)

Table (4): Reputation-based privileges versus administrative roles

Aspect	Reputation-based privilege	Administrative role (RBAC)
How obtained	Automatically, when reputation reaches a threshold	Assigned by an administrator (Verified User: by email confirmation)
Scope	The holder's own participation: downvoting, link-limit exemption, pinning own discussions	Authority over the platform and other users: moderation, user and role management
How lost	Automatically, if reputation falls below the threshold	Only when an administrator changes the role
Relation to the other	Never grants or changes a role	Independent of reputation (Table (7), P5)
Enforcement	Privilege checks in controllers and policies	Authentication middleware, role guards, and policies

To ensure transparent and auditable reputation management, KnowVerse employs two complementary algorithms. Algorithm 1 records every reputation-related action as a new append-only ledger entry, while Algorithm 2 handles reputation reversals by creating compensating ledger transactions instead of modifying or deleting existing records. Together, these algorithms maintain ledger integrity, traceability, and consistency throughout the reputation management process.

```
Algorithm 1 AwardReputation(u, a, d0, s)
-----------------------------------------
Input:  u  - user whose standing changes
        a  - action key, e.g. post_upvoted
        d0 - optional explicit delta
        s  - optional source item
Output: L  - the persisted ledger entry
rep(u) denotes reputation_points(u)
-----------------------------------------
1  begin transaction
2    lock row of u
3    d <- d0 if present, else points[a]
4    L <- APPEND <u, a, d, s, now>
5    if d != 0 then
6      rep(u) <- rep(u) + d
7      LogActivity(u, CHANGED, d, s)
8    end if
9  commit
10 return L
-----------------------------------------
Line 2 serializes ledger writes for u.
Postcondition: rep(u) = SUM(ledger(u));
|ledger(u)| grows by 1.
```

```
Algorithm 2 ReverseReputation(u, a, s)
---------------------------------------
Input:  u - user, a - original action key
        s - source item
Output: none (state changes persisted)
---------------------------------------
1  begin transaction
2    lock row of u
3    L <- LATEST entry <u, a, s> such that
          no R has R.reverses_id = id(L)
4    if L = absent then
5      commit; return        # no-op
6    end if
7    d <- delta(L)
8    APPEND <u, a + "_reverted", -d, s,
            reverses_id: id(L), now>
9    if d != 0 then
10     rep(u) <- rep(u) - d
11     LogActivity(u, CHANGED, -d, s)
12   end if
13 commit
14 on UNIQUE(reverses_id) violation:
     roll back, retry (at most 3 times)
---------------------------------------
Line 2 serializes ledger writes for u;
UNIQUE(reverses_id) rejects any second
reversal of the same entry.
Postcondition: rep(u) = SUM(ledger(u));
L is retained and reversed at most once.
```

As formalized in Algorithms 1 and 2, every change to a user's score is made within the same database transaction as the ledger entry that explains it, ensuring the stored score and the ledger cannot diverge through a partial update. Algorithm 2 enforces an at-most-once reversal policy for each ledger entry. Each compensating entry stores the identifier of the entry it cancels

(reverses_id), and a UNIQUE constraint on that column causes the database to reject any second compensating entry for the same original action. Line 3 therefore selects only entries that have not yet been reversed; when none remain, the call produces no effect, preventing a repeated retraction—such as withdrawing the same vote twice—from deducting points multiple times. Concurrent requests are managed at two levels: every award and reversal first locks the affected user's row (Line 2 in both algorithms) to serialize writes per user. If two concurrent requests nevertheless attempt to reverse the same entry, the UNIQUE constraint rejects the second attempt, which is rolled back and retried against the next outstanding entry (Line 14 of Algorithm 2). **The "Consistency under Concurrent Writes" section** reports experiments exercising these guarantees under parallel loads of up to 16 processes, including the effect of each level on its own.

Auditable Ledger Implementation and Academic Standing

From an implementation perspective, the system utilizes Laravel Observers and database transactions to maintain an append-only record of reputation. Instead of merely updating a single integer field, every adjustment is recorded as an independent entry in a Reputation Ledger table, and retractions (e.g., a withdrawn vote or removed content) are applied as compensating ledger entries rather than hard deletions. This guarantees:

1. **Traceability:** Every reputation change contributing to a user's standing is auditable. Ledger entries are never overwritten or removed, and database triggers reject any attempt to do so, with permanent account deletion as the single documented exception (see "Append-Only Ledger and the Right to Erasure").
2. **Integrity:** The denormalized reputation score is strictly maintained as equal to the sum of the ledger entries. Each vote is executed alongside all its downstream effects within a single transaction that locks affected rows and retries automatically if a deadlock occurs, preventing concurrent requests from causing data inconsistencies.
3. **Academic Standing & Participation Privileges:** As a user's cumulative score crosses predefined thresholds, their displayed academic title advances and the participation privileges summarized in Table (3) are automatically unlocked.

Append-Only Ledger and the Right to Erasure

Append-only designs must also accommodate requests to delete personal data, such as the right to erasure under Article 17 of the GDPR (European Parliament and Council of the European Union, 2016; Politou et al., 2018). KnowVerse therefore defines what it retains and what it erases. During normal operation, a user's data is kept for as long as the account exists. Deactivating an account is reversible: the account is hidden, but its data, including its ledger entries, is kept and restored when the user signs in again. The application applies no automatic retention period; data is removed only when an account is permanently deleted, either by its holder after confirming their password or by an administrator.

Permanent deletion is the single exception to append-only operation; it is carried out by the database's cascading deletion of the account, the only path by which ledger entries can be removed. It removes the account profile together with the user's votes, the user's own reputation ledger entries, notifications, follow relationships, saved discussions, activity log, notification preferences, and the reports the user filed. Discussions and replies the user wrote are kept so that conversations remain readable, but they are detached from the account and displayed as written by a "Deleted Scholar"; their text is not altered, so personal information that a user

wrote into a post remains unless that post is deleted first. Reputation that other users earned from the deleted user's votes is retained, because those ledger entries belong to the recipients and record only the recipient and the item that was voted on, and reversing them would penalize users for another person's decision. Vote counters affected by the deletion are recalculated. Automated tests verify each of these behaviors (see "Administrative Compliance").

This section describes the technical retention and erasure behavior of the platform; it is not a legal assessment, and this paper does not claim that KnowVerse complies with the GDPR or any other regulation. Compliance would also depend on matters outside the application, such as the lawful basis for processing, privacy notices, the retention of database backups, server logs, and session records, and organizational procedures, none of which were evaluated in this study.

System Implementation

Database Schema Design

To support the dynamic nature of KnowVerse, a normalized relational database schema was designed. The schema prioritizes data integrity and efficient retrieval of reputation-based metrics. The following key entities were implemented:

1. **Users Table:** Stores core identity information and the cumulative reputation score, which determines each scholar's academic standing and gates their participation privileges.
2. **Reputation Ledger:** An append-only table that logs every reputation change, including the user_id, action type, signed point value, related item, the entry being reversed (for compensating entries), and a timestamp. Database triggers reject any update or deletion of its rows. This ledger makes each reputation change auditable and allows any inconsistency with the stored score to be detected.
3. **Topics and Interactions:** Tables for discussions, comments, and votes (upvotes/downvotes), linked via foreign keys to enforce strict referential integrity.

System Workflows

As illustrated in Figure (2) (Scenario A), the system handles discussion creation through a multi-layered sequence. Once a Verified User submits content, the Backend initiates a validation and lexical filtering process. Upon successful sanitization, the discussion is persisted in the Database, and the Reputation System records the corresponding entry in the append-only ledger and updates the user's score within the same database transaction.

To ensure high performance, the system decouples follower notifications from the primary request–response cycle. When a discussion is created, a NotifyFollowersOnNewPost job is dispatched to a background queue and processed asynchronously by a dedicated queue worker, allowing the author to receive an immediate HTTP response. As the background job executes, each follower receives a RealTimeNotification broadcast over WebSockets via **Laravel Reverb**. This architecture keeps the main request path highly responsive while delivering timely, real-time updates across connected clients.

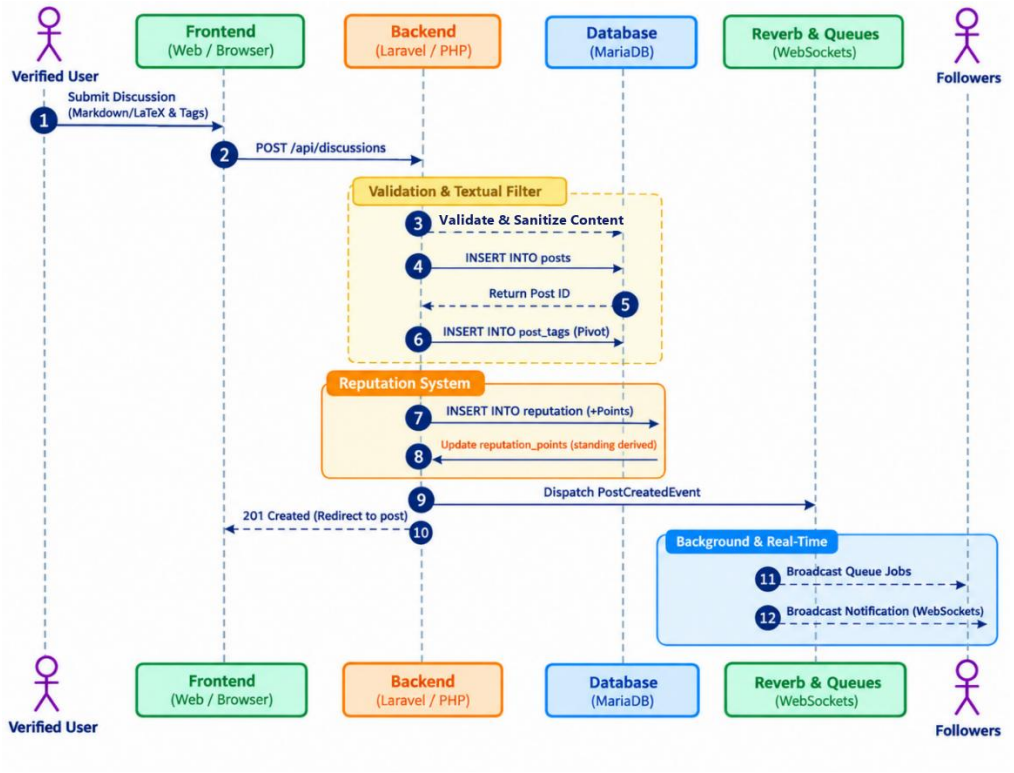


Figure (2): Scenario A – Create Discussion and Real-Time Broadcasting

To ensure data integrity and real-time feedback, a synchronized voting workflow is implemented as depicted in Figure (3) (Scenario B). Upon receiving a vote, the backend validates the request to prevent duplicate votes and self-voting. Once verified, the vote and all of its associated effects are executed within a single database transaction that locks the target item row: the system records the vote, updates the aggregate vote counters, logs the compensating or new entry in the Reputation Ledger to adjust the author's academic standing according to the action weight, and returns the updated tally to the voter's interface via an asynchronous response—updating the score dynamically without a page reload. Furthermore, once the transaction has successfully committed, the content's author receives a real-time notification over WebSockets via **Laravel Reverb**.

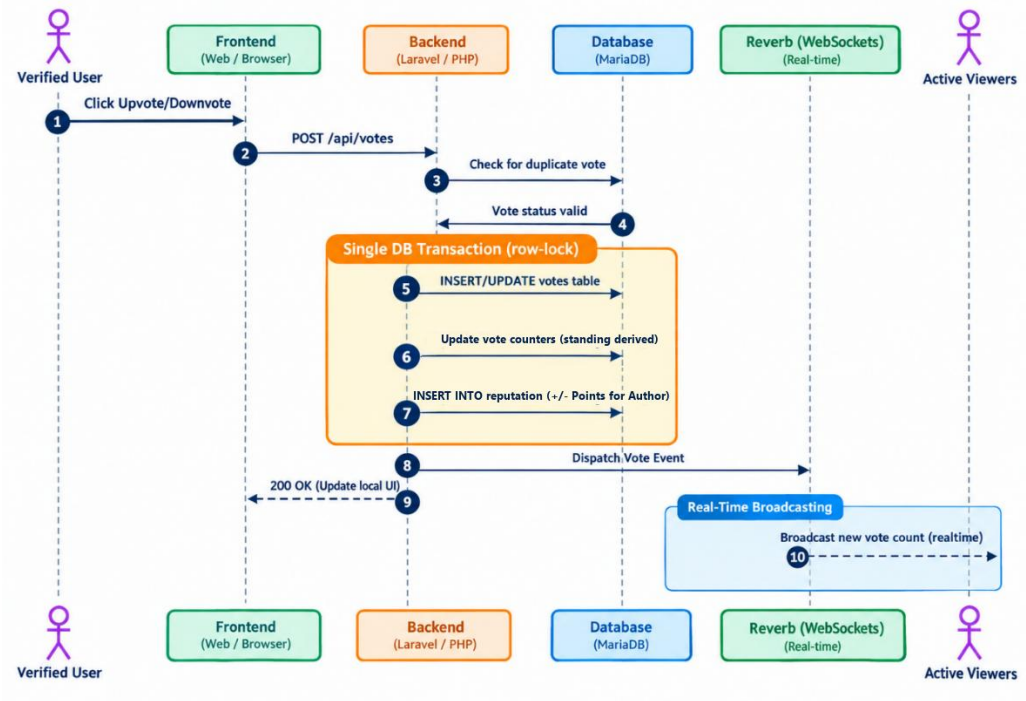


Figure (3): Scenario B – Vote and Reputation Update.

To support community standards, a structured reporting and review process is implemented as detailed in Figure (4). When a user flags violating content, the system registers a 'Pending' status within the database. Authorized moderators, validated via RBAC, can subsequently audit these flags through a dedicated moderation dashboard. Upon report approval, the system triggers an automated penalty execution flow that hides or soft-deletes the non-compliant content and logs a corresponding penalty entry in the violator's reputation ledger.

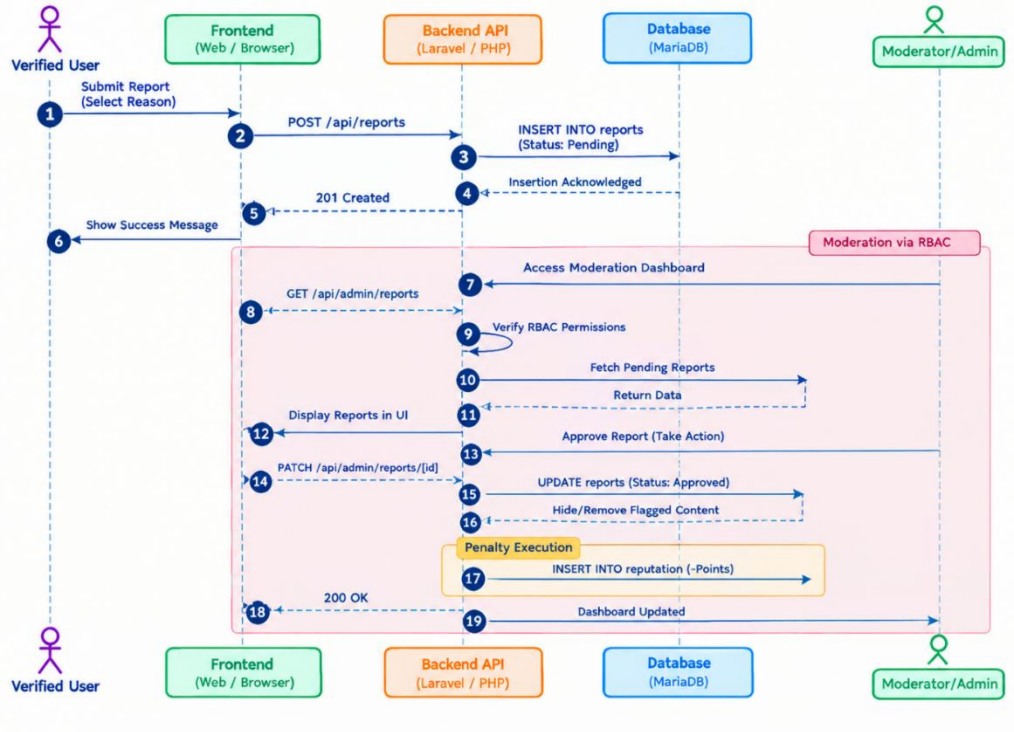


Figure (4): Scenario C – Report and Review

User Interface and Reputation Visualization

The reputation model and backend logic are presented to the user through the reputation dashboard, as shown in Figure (5). This interface presents each scholar's ledger and current academic standing in real time. Key features illustrated in the dashboard include:

- **Academic Standing Display:** A visual indicator of the user's current academic standing (e.g., "Novice Scholar"), which dynamically scales based on their cumulative reputation score.
- **Live Contribution Ledger:** A detailed, chronological history of merits and retractions (e.g., +2 for comments, +5 for discussion upvotes), providing full transparency for every reputation adjustment.
- **Gamification Elements:** Visual cues and progress indicators designed to encourage constructive academic engagement, with every displayed point remaining fully traceable to an entry in the append-only audit trail.

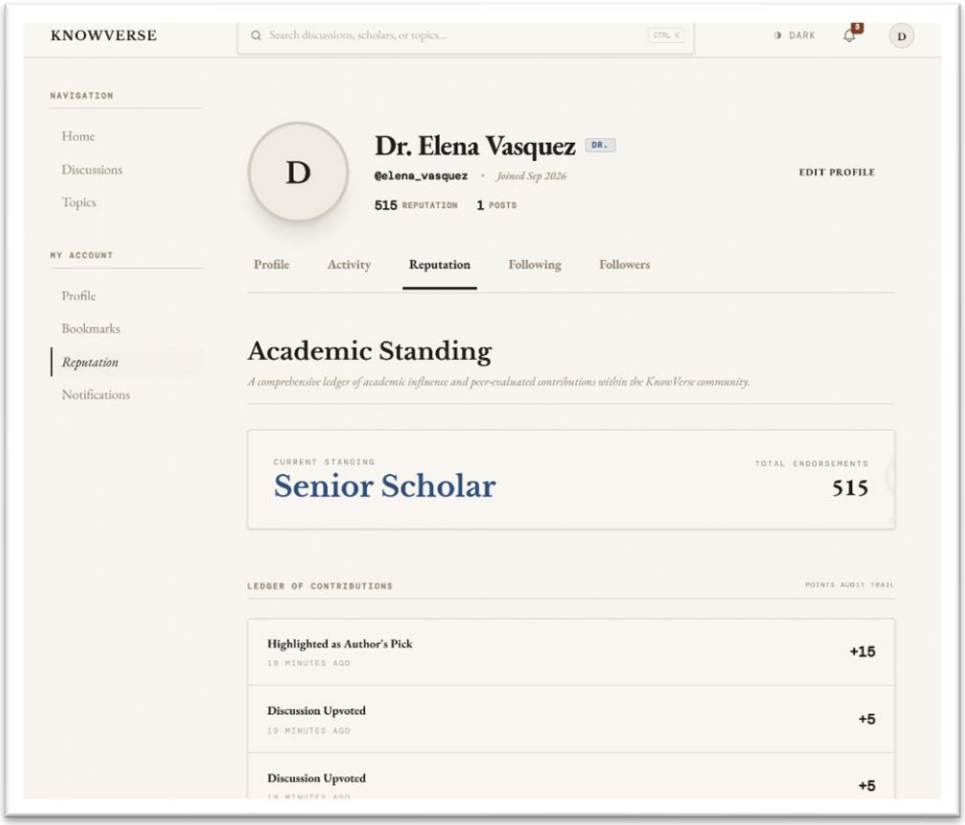


Figure (5): User Reputation Dashboard and Points History

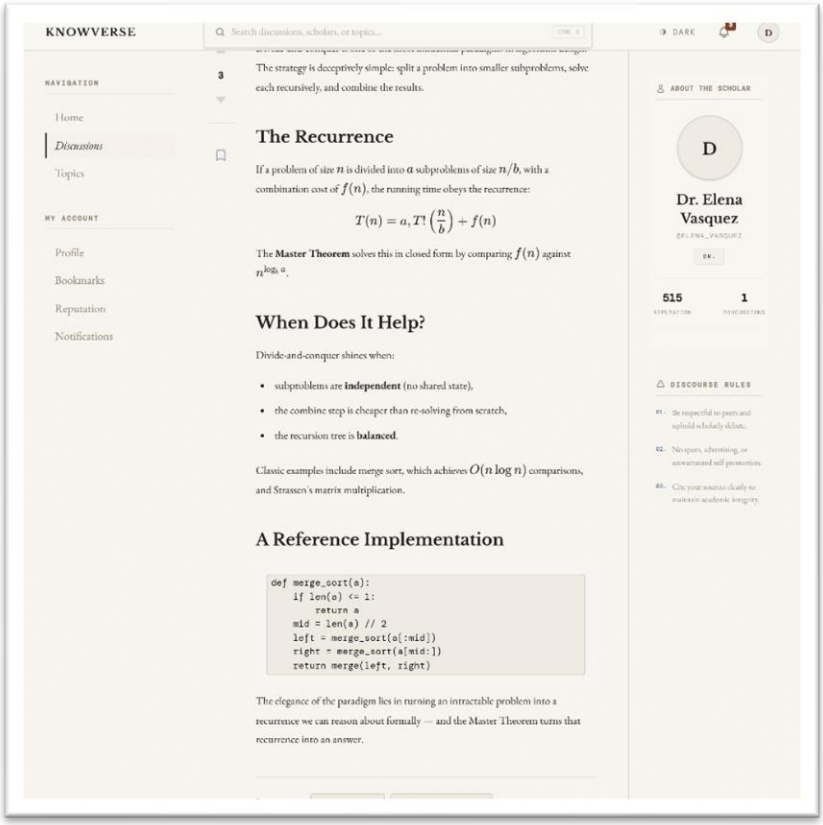


Figure (6): A rich academic discussion rendered with Markdown, LaTeX, and code.

Security Design

Security is a foundational pillar of the KnowVerse platform, governing both academic content integrity and user data protection. The system utilizes a multi-layered security architecture:

Authentication and Session Integrity

KnowVerse uses the authentication scaffolding of Laravel Breeze, and passwords are stored as Bcrypt hashes; both are provided by the framework. Laravel's cross-site request forgery (CSRF) protection, also framework-provided, issues a secret token for each session and verifies it on every state-changing request, so a request forged by another website, which cannot read that token, is rejected. The session identifier is regenerated at login to prevent session fixation.

Access Control and Vulnerability Mitigation

The system enforces strict RBAC to restrict sensitive operations—such as moderation and system monitoring—to authorized personnel only. To mitigate common web vulnerabilities, the implementation includes:

Input Validation and Output Sanitization: User-generated Markdown/LaTeX content is validated and sanitized on the server side to mitigate Cross-Site Scripting (XSS), while database queries use parameter binding through the ORM to mitigate SQL injection. Markdown sanitization is provided by the CommonMark library and parameter binding by Laravel's Eloquent ORM; KnowVerse contributes their configuration and the tests that verify them.

Automated Activity Logs: Significant administrative actions and reputation adjustments are recorded in activity logs, supporting accountability and system auditing.

Project-Specific Controls: The security logic specific to KnowVerse comprises the anti-farming rules, the ledger consistency checks, the reputation-gated privileges, and the content-

governance rules (language filtering, link limits, and reserved usernames). Table (5) marks which controls are provided by the framework.

Testing and Functional Validation

To evaluate the reliability of KnowVerse and its security controls, a structured testing phase was conducted, addressing both functional integrity and system performance.

Evaluation Methodology. We evaluated KnowVerse along two complementary tracks, since correctness and performance demand different conditions. For the first track, we developed an automated suite of 126 tests (295 assertions) using the Pest/PHPUnit framework to assess functional correctness, security, and data integrity. These tests execute against an isolated in-memory SQLite database with a fresh schema per test, which makes them deterministic but cannot reproduce concurrent writes; the concurrency guarantees were therefore tested separately on MariaDB (**see "Consistency under Concurrent Writes"**). The suite exercises access-control scenarios aligned with the Open Worldwide Application Security Project (OWASP) Top 10, injection resistance (stored XSS and SQL injection), anti-farming safeguards, account erasure, and—most importantly—the reputation ledger's integrity invariant, which is checked after each stress scenario by a consistency oracle that confirms every user's stored score equals the exact sum of their ledger entries. The suite is reproducible with a single command (php artisan test), and all 126 tests pass on the current build, including in continuous integration on PHP 8.2 and 8.3.

For the second track, we measured performance on a single workstation equipped with an Intel Core i7-10510U (4 cores) and 16 GB of RAM, running Windows 11 with Apache 2.4 and mod_php (PHP 8.2) and a MariaDB 10.4 database, with the load generator on the same machine. The server was configured as for production: PHP's OPcache enabled, Laravel's configuration, route, and view caches built, and debug mode disabled. To approximate realistic load, we generated a synthetic dataset of 1,000 users, 10,000 discussions, 20,000 comments, and approximately 100,000 votes using a reproducible bulk-seeding routine. We then applied three measurement tools: (i) a concurrent HTTP load generator that sent 400 requests to each principal read endpoint at each of 10, 50, 100, and 200 simultaneous clients, recording response-time percentiles, throughput, and error rate; (ii) a server-side instrument that timed the complete vote-processing cascade (vote → reputation-ledger entry → activity log → notification) over 1,000 operations and re-verified the ledger invariant; and (iii) a WebSocket probe that measured end-to-end delivery latency from the application server to a connected client over Laravel Reverb. Each measurement was repeated five times, and we report the mean and standard deviation across runs. The results are reported in Tables (9), (10), and (11). Two further experiments, reported in Tables (12) and (13), test the ledger under concurrent writes and compare it with a counter-only design.

Code and Data Availability. The source code, the automated test suite, the benchmark commands that produced Tables (5)–(13), and the raw measurement files are available in the public repository (https://github.com/mohammedabusaada/knowverse), together with instructions for reproducing each table.

The functional-correctness track (Track 1) was organized into four validation domains, each exercised by dedicated automated tests:

Real-Time Integrity and Rendering

The system's ability to parse complex academic syntax (Markdown and LaTeX) was verified, ensuring accurate rendering across all user interfaces. Furthermore, real-time synchronization was tested using Laravel Reverb. Automated tests confirmed that notifications for upvotes, comments, and moderation actions are dispatched as real-time broadcasts, and the delivery latency measured in Table (11) stays below 20 ms for 95% of events, so users see updates without refreshing the page.

Reputation Accuracy and Anti-Farming

Testing of the Reputation Ledger focused on transaction accuracy. The system successfully validated point increments for "Author's Picks" and interactions while the anti-farming rules blocked self-votes, duplicate votes, repeated vote flips, repeated reversals, and switching of the Author's Pick (Table (6)). This ensures that each user's academic standing remains exactly consistent with their recorded reputation history.

RBAC Enforcement and Security Boundaries

Access control tests confirmed that middleware strictly enforces user roles. Guests and Unverified Users were successfully restricted to "Read-Only" mode, while administrative dashboards remained accessible only to authorized personnel. Moderation workflows, including the flagging and removal of content, were validated to ensure they follow the defined hierarchical permissions.

Administrative Compliance

Finally, automated tests confirmed the administrative tools for role management and account deactivation, and the account-erasure behavior described in **the "Append-Only Ledger and the Right to Erasure" section**: the erased user's own data and ledger entries are removed, authored content is retained without attribution, and reputation earned by other users is preserved with the ledger invariant intact. The search and tagging features allow users to filter discussions by academic discipline.

Consistency under Concurrent Writes

Because SQLite serializes all writes, the ledger's concurrency guarantees were tested separately on MariaDB 10.4 (InnoDB, repeatable-read isolation). A test harness launched 8 or 16 separate PHP processes, each with its own database connection, which waited for a common start time so that their writes overlapped. Each of the five runs per configuration combined two workloads. In the reversal race, one user first received 20 identical awards for the same discussion, and every process then attempted to reverse those awards five times: 40 attempts with 8 processes and 80 with 16, against only 20 reversible entries. In the mixed vote workload, every process cast, changed, and retracted votes at random on a shared set of discussions (250 operations per process with 8 processes and 150 with 16), drawing on a small pool of voters so that the same voter often acted on the same discussion at the same moment.

After each run, an automated checker evaluated four conditions: (i) the ledger invariant, that every affected user's stored score equals the sum of their ledger entries; (ii) that no ledger entry had been reversed more than once; (iii) that each discussion's vote counters equal the votes actually stored; and (iv) that each author's reputation from votes equals what the remaining votes imply (+5 per upvote and −2 per downvote). Condition (iv) is stricter than the invariant: the invariant still holds when the ledger itself records a wrong effect, because the stored score then equals the sum of the wrong entries. To isolate the vote-processing path, the mixed

workload was run both through a non-atomic vote path, in which a vote and its reputation update are written in separate transactions, and through the atomic path of KnowVerse (see "Auditable Ledger Implementation and Academic Standing"), in both cases with the same ledger service. Table (12) reports the totals over five runs. In every configuration, the reversal attempts beyond the 20 available entries per run had no effect, so exactly 100 reversals were made; no entry was reversed twice, the invariant held after every run, and no process reported an error. The configurations differ only in the last column. With the non-atomic path, 1 discussion (8 processes) and 12 discussions (16 processes) ended with an author's reputation that did not match the votes received, although the invariant held. Two causes were identified: a vote could be saved while its reputation update, run in a separate transaction, was aborted by a deadlock, and two simultaneous identical requests could both apply the same vote change. With the atomic path, which applies each vote and all of its effects in one locked transaction, no mismatch occurred in 22,000 operations.

To determine which mechanism prevents double reversals, the reversal race was repeated with each of the two guards of Algorithm 2 switched off in turn: the row locks (the per-user lock and the lock on the selected entry) and the UNIQUE constraint on reverses_id. Each of the four combinations was run five times with 8 and five times with 16 processes, with the same 20 awards and the same number of attempts. With both guards, and with the row locks alone, every run made exactly 20 reversals, none twice, and no process reported an error. With the UNIQUE constraint alone, no entry was reversed twice, but competing processes repeatedly selected the same entry: 154 attempts failed with a constraint violation or a deadlock after their retries were exhausted, and three of the ten runs made fewer than 20 reversals. With neither guard, every run reversed some entries more than once: 128 entries in total, while at least 18 awards were never reversed, so that 310 reversals were made against 200 awards. The ledger invariant held in all of these runs, because each wrong reversal was recorded faithfully; only check (ii) detected the error. The UNIQUE constraint therefore guarantees correctness on its own, while the row locks allow concurrent reversals to succeed without errors; KnowVerse uses both.

Ablation against a Counter-Only Design

To quantify the cost of auditability, we compared the ledger with the conventional design in which reputation is stored only as a numeric score (Resnick et al., 2000), keeping every other step of vote processing identical (Table (13)). Across five runs of 1,000 vote operations per design, the ledger added 4.9 ms (27%) to the mean latency (22.8 ± 0.6 ms versus 17.9 ± 0.5 ms) and 18.2 ms (55%) to the 95th-percentile latency (51.6 ± 3.7 ms versus 33.4 ± 2.0 ms), reduced single-process throughput by 21%, and used about 200 bytes of storage per entry. We then altered the stored reputation of 50 randomly selected users in each of five repetitions. A counter-only design has no reference against which such changes can be detected; it could be extended with a separate audit log, which would cost a comparable insert per operation but, unless the score were defined as the sum of that log, would provide traceability without the detection and exact recovery reported here. With the ledger, a consistency scan of all 1,000 users identified every altered account with no false positives in 124 ms on average, and recalculation restored all 250 accounts to their exact previous values. The ledger does not, however, detect an incorrect entry that was recorded faithfully, such as the duplicated vote effects described above; such an entry becomes traceable rather than self-evident. The latencies in Table (13) are lower than in

Table (10) because the workload differs, so only the two designs within Table (13) should be compared.

Table (5): Tested security controls (OWASP-aligned). (F) marks a mechanism provided by the Laravel framework or a library; all other mechanisms are specific to KnowVerse.

#	OWASP	Control / attack vector	Enforcement mechanism	Result	Test
S1	A01	Guest accesses admin area	auth middleware (F)	Redirected to login	AccessControlTest
S2	A01	Verified user accesses admin dashboard	IsAdmin middleware	403 Forbidden	AccessControlTest
S3	A01	Moderator accesses user management	Controller role guard	403 Forbidden	AccessControlTest
S4	A01	Edit another user's discussion	PostPolicy::update	403 Forbidden	AccessControlTest
S5	A01	Delete another user's comment	CommentPolicy::delete	403 Forbidden	AccessControlTest
S6	A01	Mark another user's notification read (IDOR)	NotificationPolicy	403 Forbidden	AccessControlTest
S7	A01	Unverified user posts content	verified middleware (F)	Redirected to verify	AccessControlTest
S8	A01	Banned user performs any request	CheckBanned + Gate::before	Logged out + denied	AccessControlTest
S9	A03	Stored XSS via discussion Markdown	CommonMark html_input=strip (F)	HTML/script stripped	InjectionTest
S10	A03	Stored XSS via comment Markdown	CommonMark html_input=strip (F)	HTML/script stripped	InjectionTest
S11	A03	SQL injection via search	Eloquent parameter binding (F)	No injection	InjectionTest
S12	A04	Profanity / non-academic language	ContentModerationService	Submission rejected	ContentGovernanceTest
S13	A04	Link spam (> 3 external URLs)	Link-count heuristic	Submission rejected	ContentGovernanceTest
S14	A01	Username hijacks a system route	ReservedUsername rule	Registration rejected	ContentGovernanceTest
S15	A07	Weak password at registration	Password::defaults() (F)	Registration rejected	ContentGovernanceTest
S16	A07	Brute-force login	Rate limiter (5 attempts) (F)	Throttled	ContentGovernanceTest

Table (6): Anti-farming validation.

#	Manipulation attempt	Defense	Result	Test
A1	Vote on own content (HTTP)	Controller self-vote guard	403, no change	AntiFarmingTest
A2	Vote on own content (data layer)	VoteObserver self-vote guard	No reputation awarded	AntiFarmingTest
A3	Cast two votes on one target	UNIQUE(user, target) + locked update	Single vote, single award	AntiFarmingTest
A4	Farm by repeatedly flipping a vote	Reverse-then-apply in observer	No accumulation	AntiFarmingTest
A5	Self-award Author's Pick	CommentPolicy::markBest	403 Forbidden	AntiFarmingTest
A6	Banned user votes	CheckBanned	Session terminated	AntiFarmingTest
A7	Switch Author's Pick back and forth	Previous pick reversed before new award	No accumulation	ReputationReversalIntegrityTest
A8	Reverse the same ledger entry twice	UNIQUE(reverses_id) + row lock	Second reversal has no effect	ReputationReversalIntegrityTest

These tests demonstrate that reputation-based participation privileges and administrative RBAC roles operate as separate authorization mechanisms.

Table (7): Reputation-driven privileges and RBAC decoupling.

#	Capability / property	Mechanism	Result	Test
P1	Downvoting (≥ 50 rep)	Privilege gate in VoteController	Low-rep blocked, high-rep allowed	ReputationPrivilegesTest
P2	Anti-spam friction bypass (≥100 rep)	Trusted-user exemption	Trusted users bypass the link limit	ReputationPrivilegesTest
P3	Pin own discussion (≥250 rep)	PostPolicy::pin	Gated to owner + threshold	ReputationPrivilegesTest
P4	RBAC role assignment	Admin-only, manual	Admins only; moderators/users 403	RoleManagementTest
P5	Reputation ↔ role decoupling	Deliberate (no auto-escalation)	5,000 rep does not change a user's role	RoleManagementTest

Table (8): Ledger integrity under stress (SQLite test suite).

Scenario	Operations	Violations	Result	Test
Interleaved voting + retraction	8 users, 8 posts, ~150 votes, 30 retractions	0	Consistent	ReputationConsistencyTest
Moderation penalty + reporter reward	report → resolve (-10 / +2)	0	Consistent	ReputationConsistencyTest
Author's Pick award + retraction	mark → unmark (+15/+2 reversed)	0	Consistent	ReputationConsistencyTest
Single-vote retraction (regression)	multi-upvoter retraction	0	Consistent	ReputationLedgerTest
Append-only guarantee	retraction adds compensating entry	original rows retained	Retained	ReputationLedgerTest
Direct modification of the ledger	raw UPDATE and DELETE of an entry	0	Rejected by trigger	ReputationLedgerTest

Table (9): HTTP response time, throughput, and error rate versus concurrency (Apache 2.4 + mod_php 8.2 with OPcache, MariaDB 10.4; 400 requests per cell; mean ± SD over five runs).

Endpoint	Metric	10 clients	50 clients	100 clients	200 clients
GET / (home feed)	p50 (ms)	995 ± 320	4,498 ± 1,221	8,463 ± 489	11,837 ± 1,169
	p95 (ms)	3,052 ± 1,382	6,869 ± 1,926	12,804 ± 3,096	22,899 ± 4,315
	throughput (req/s)	8.0 ± 2.5	10.8 ± 2.8	10.4 ± 1.4	13.5 ± 0.9
	errors (%)	0	0	0	16.2
GET /posts (list)	p50 (ms)	307 ± 69	1,687 ± 573	3,243 ± 836	5,473 ± 767
	p95 (ms)	956 ± 467	3,160 ± 1,196	4,104 ± 1,156	19,598 ± 1,797
	throughput (req/s)	24.6 ± 3.1	26.0 ± 5.1	31.2 ± 8.3	20.4 ± 2.1
	errors (%)	0	0	0	0.05
GET /posts/{id} (thread)	p50 (ms)	202 ± 68	818 ± 184	1,505 ± 364	2,196 ± 704
	p95 (ms)	735 ± 398	1,544 ± 630	2,123 ± 554	13,135 ± 2,006
	throughput (req/s)	39.7 ± 11.5	56.6 ± 14.7	62.3 ± 15.8	30.1 ± 4.2
	errors (%)	0	0	0	0
GET /search?q=	p50 (ms)	931 ± 196	4,397 ± 694	9,572 ± 3,158	16,554 ± 8,711
	p95 (ms)	2,507 ± 958	5,863 ± 2,212	12,295 ± 5,290	30,102 ± 194*
	throughput (req/s)	8.8 ± 1.6	11.2 ± 2.2	10.9 ± 3.6	10.8 ± 2.8
	errors (%)	0	0	0	24.1

* Equal to the 30 s client timeout. Errors at 200 clients were database connection refusals (limit 151) and client timeouts.

Table (10): Vote-processing latency and ledger consistency (mean ± SD over five runs).

Metric	Value (mean ± SD)
Vote operations	5 runs × 1,000
Mean latency (ms)	34.76 ± 1.50
p50 latency (ms)	31.32 ± 0.61
p95 latency (ms)	56.12 ± 3.60

Metric	Value (mean ± SD)
p99 latency (ms)	89.72 ± 5.31
Throughput (votes/s, single process)	28.7 ± 1.2
Ledger invariant violations	0 (checked after each run)

Table (11): Real-time (WebSocket / Reverb) delivery (mean ± SD of per-run percentiles over five runs of 100 events; application, Reverb, and client on the same host).

Metric	Value (mean ± SD)
End-to-end delivery latency, event → client, p50 (ms)	2.59 ± 1.17
End-to-end delivery latency p95 / p99 (ms)	16.62 ± 1.66 / 19.41 ± 5.00
Delivery reliability	500 / 500 events delivered
App → Reverb publish call (HTTP round trip) p50 / p95 / p99 (ms)	4.41 ± 1.30 / 18.59 ± 1.74 / 21.25 ± 5.37

Both latencies start when the application timestamps the event. The publish call ends when Reverb's HTTP response reaches the application; Reverb pushes the event to subscribed clients before it sends that response, so a client normally receives the event before the publish call returns.

Table (12): Consistency under concurrent writes on MariaDB 10.4 (totals over five runs per row; the non-atomic path writes a vote and its reputation update in separate transactions, and the atomic path is that of KnowVerse, described in "Auditable Ledger Implementation and Academic Standing").

Vote path	Parallel processes	Vote operations	Reversal attempts → reversals made	Double reversals / invariant violations	Reputation mismatches
Non-atomic	8	10,000	200 → 100	0 / 0	1
Atomic	8	10,000	200 → 100	0 / 0	0
Non-atomic	16	12,000	400 → 100	0 / 0	12
Atomic	16	12,000	400 → 100	0 / 0	0

Reversal attempts → reversals made: attempts against the 20 reversible entries available in each run. Reputation mismatches: discussions whose author's vote-derived reputation did not match the stored votes, summed over the five runs.

Table (13): Ablation: append-only ledger versus a counter-only design (mean ± SD over five runs; 134,250 ledger entries).

Metric	Ledger	Counter only
Mean latency per vote operation (ms)	22.79 ± 0.61	17.94 ± 0.47
p95 latency (ms)	51.61 ± 3.70	33.37 ± 2.02
Throughput (operations/s, single process)	43.9 ± 1.2	55.8 ± 1.4
Storage per ledger entry	≈ 204 bytes	—
Altered scores detected (5 × 50 users)	250 / 250, no false positives	Not detectable
Altered scores restored exactly	250 / 250	Not possible
Consistency scan of 1,000 users (ms)	124 ± 45	—

Results and Discussion

Overall, the functional validation confirms that KnowVerse implements the mechanisms proposed to address the limitations identified in existing academic discussion platforms. The OWASP-aligned tests in Table (5) confirm that the access-control rules and input handling reject the tested attacks, including insecure direct object references, stored cross-site scripting, SQL injection, and unauthorized route access; several of these protections are provided by the Laravel framework, as marked in the table. These tests cover a defined set of attack vectors: they show that the tested controls work as intended, not that the system as a whole is secure. The anti-farming tests in Table (6) show that self-votes, duplicate votes, repeated vote flips,

repeated reversals, and switching the Author's Pick do not accumulate reputation. Table (7) confirms that reputation-driven privileges are enforced without ever changing a user's administrative role, and the stress scenarios in Table (8) recorded no ledger invariant violations. The concurrency experiments in Table (12) extend these guarantees to genuinely parallel writes on MariaDB, and the ablation in Table (13) indicates that maintaining the auditable ledger increases the mean latency of a vote operation by 4.9 ms (27%) and its 95th-percentile latency by 18 ms (55%). In the conducted experiment, the ledger detected every injected stored-score alteration and restored the affected scores exactly.

The performance results describe a single-workstation deployment and depend strongly on its configuration. A one-factor-at-a-time analysis (Table (14)) showed that the framework's development defaults dominate latency: with PHP's OPcache disabled and without Laravel's configuration and route caches, the median latency at 10 concurrent clients was 0.98–1.35 s, depending on the page; enabling OPcache alone reduced it by 46–83% and raised throughput up to 5.6-fold, and without a cached configuration 1.7–5.0% of requests failed under concurrency because loading the environment configuration is not thread-safe on the test platform. All measurements in Tables (9)–(11) therefore use the production configuration described in the "Testing and Functional Validation" section. Up to 100 concurrent clients, every request succeeded. Throughput was 8–11 requests per second for the home feed and search, 25–31 for the discussion list, and 40–62 for a discussion thread (Table (9)); beyond that point, additional clients increased latency through queueing rather than raising throughput. The home feed and search remain the slowest pages because they run aggregate and substring-search queries over the whole dataset. At 200 concurrent clients the deployment exceeded its connection limits of 150 web-server threads and 151 database connections, and 16% of home-feed and 24% of search requests failed or reached the 30-second client timeout. The run-to-run variation shown by the standard deviations reflects a client and server sharing one machine. These results are a worst-case baseline rather than an estimate of production capacity, and no claim of scalability is made beyond what they show.

Table (14): One-factor-at-a-time diagnosis of the server configuration at 10 concurrent clients (median latency in ms / throughput in requests per second; one run of 120 requests per page and configuration; each row adds one change to the row above).

Configuration	Home feed	Discussion list	Discussion thread	Search	Errors (%)
Development defaults	1,319 / 7.1	985 / 7.8	1,049 / 8.6	1,353 / 6.8	1.7–5.0
+ Configuration and route caches	1,245 / 7.3	943 / 9.6	878 / 10.8	1,153 / 8.4	0
+ Production mode (debug off)	1,146 / 8.4	849 / 10.8	835 / 11.7	1,101 / 8.6	0
+ OPcache	617 / 15.5	254 / 35.8	143 / 66.0	588 / 16.3	0

Vote processing took 34.8 ± 1.5 ms per operation, with no invariant violation in 5,000 operations (Table (10)). Real-time delivery added little latency: the median time from broadcast to receipt was 2.6 ms, the 95th percentile 16.6 ms, and all 500 events were delivered (Table (11)). The end-to-end latency is lower than the publish latency because the two intervals overlap: both start when the application timestamps the event, but the publish latency also includes the time for Reverb's HTTP acknowledgement to return to the application, whereas a client receives the event as soon as Reverb forwards it, which happens before that acknowledgement is sent. Because the client and server shared one host, these figures exclude network transit.

Collectively, these findings show that transparent reputation auditing, reputation-driven participation privileges, and role-based administrative control can be integrated into a single, verifiable platform. Whether this integration increases user trust or the quality of scholarly collaboration was not measured in this study and is left to future work.

Limitations

The evaluation of KnowVerse has six primary limitations:

1. **Single-Node Testbed:** All performance measurements were conducted on a single workstation with the load generator running on the same machine; thus, the benchmarks do not fully capture network latency or the characteristics of a distributed, multi-server production deployment.
2. **Synthetic Workloads:** The dataset used for testing is synthetic. While its volume approximates that of a departmental-scale community, the simulated content and interaction patterns may not fully reflect real-world user behavior.
3. **Lack of User/Behavioral Evaluation:** No empirical user study was conducted. While this evaluation validates the technical correctness, auditability, security controls, and governance mechanisms of KnowVerse, it does not empirically measure the platform's impact on user trust and motivation, nor does it prove that reputation incentives improve the intrinsic academic quality or scholarly value of discussions.
4. **Uncalibrated Parameters:** The assigned point values, privilege thresholds, and rate limits represent arbitrary design choices that have not yet been calibrated against empirical usage data.
5. **Scope of Auditability vs. Tamper-Proofing:** The reputation ledger provides internal auditability rather than cryptographic tamper-proofing. Database triggers reject direct changes to ledger entries, but a user with administrative privileges on the database can remove the triggers and alter both a score and its ledger entries consistently; the ledger alone therefore cannot detect tampering by such a privileged user.
6. **Sybil and Collusion Attacks:** The anti-farming rules prevent users from gaining reputation through their own actions (Table (6)), voting requires a verified email address, and downvoting requires a minimum reputation. These rules do not detect one person operating several accounts (a Sybil attack; Douceur, 2002) or a group of users who agree to vote for one another (collusion; Hoffman et al., 2009). Because each ledger entry identifies the item that produced it and each vote identifies its voter, such patterns can be investigated after the fact, but detecting them automatically is outside the scope of this study.

Research Contributions

The contribution of this work lies in the integration of established mechanisms rather than in new mechanisms, applied to an educational setting: a discussion space in which students, researchers, and educators exchange academic knowledge and receive transparent recognition for their participation. Its main contributions are:

1. **Auditable Reputation Ledger Framework**: An append-only reputation ledger, following the event-sourcing pattern, in which every reputation change is an independent entry and every retraction a compensating entry that references the entry it cancels. Database triggers reject any modification or deletion of an entry, a database

constraint prevents any entry from being reversed twice, and the consistency between each score and its ledger is verified by an automated oracle and under concurrent load.

2. **Reputation-Driven Privilege Progression with Decoupled RBAC:** An auditable reputation mechanism that maps validated academic contributions to a transparent hierarchy of scholarly titles and a graduated set of participation privileges, including downvoting, reduced anti-spam restrictions, and pinning one's own discussions. These reputation-based privileges are deliberately separated from administrative authority, which is governed independently through a five-tier RBAC framework with manual role assignment. This separation ensures that reputation accumulation cannot be converted into administrative authority or moderation privileges through reputation farming.
3. **Real-Time Academic Discussion and Moderation Infrastructure*:*** An event-driven architecture integrating WebSockets and background queues to provide immediate synchronization of discussions, voting activities, notifications, and moderation actions.
4. **Integrated Auditable Academic Platform:** A unified platform that combines reputation auditing, role-based access control, and real-time interaction mechanisms in a complete implementation of the proposed integration, providing a basis for future studies of trust and contribution quality in scholarly communities.
5. **Documented and Repeatable Evaluation:** An automated test suite and benchmark tools that support reproduction of the experiments reported in this paper, including the concurrency experiment and the ablation against a counter-only design.

Conclusion

This paper presented KnowVerse, an academic discussion platform that provides transparent and auditable mechanisms for reputation management and role-based access control in online scholarly communities. The proposed platform integrates an Auditable Reputation Ledger with RBAC so that participation privileges follow recorded contribution events, while administrative roles are assigned separately. The implementation demonstrated that recording reputation activities as auditable ledger transactions makes reputation changes traceable under normal operation, while the RBAC framework keeps privilege management clearly scoped. Experiments on MariaDB showed no double reversals or ledger-invariant violations under the tested concurrent workloads. An ablation study showed that maintaining the auditable ledger adds 4.9 ms (27%) to the mean latency of a vote operation and 18 ms (55%) to its 95th percentile, while every stored-score alteration injected in the experiment was detected and restored to its original value. Furthermore, the integration of real-time discussion, voting, and moderation mechanisms provides the technical capabilities required for real-time participation and timely content management.

KnowVerse provides a transparent and auditable mechanism for computing reputation and enforcing role-based access control. Measuring the effect of this mechanism on users' actual trust and on the quality of their interactions is outside the scope of this study and is left to future work. Overall, KnowVerse shows that transparent reputation auditing and role-based governance can be combined in a practical, verifiable implementation.

Future Work. Five directions follow from the limitations above. A user study is needed to evaluate the effect of the auditable reputation mechanism on users' trust and on the quality of scholarly interaction. A deployment on separate application and database servers, with connection pooling, caching of aggregate queries, and indexed full-text search, would allow

evaluation of the platform under more realistic multi-user load. The point values and thresholds should be calibrated from usage data once such a deployment exists. Automatic detection of Sybil and collusion patterns could build on the provenance that the ledger and the vote records already provide. Finally, the ledger could be made tamper-evident by chaining each entry to a cryptographic hash of its predecessor (Crosby & Wallach, 2009).

Acknowledgements

The authors thank the members of the graduation project team for their contributions to the KnowVerse implementation.

Use of AI-Assisted Tools

The authors used AI-assisted tools to support software development tasks, including test scaffolding, the benchmarking harness, and editorial refinement of the manuscript. All experimental results were generated by executing the system on the authors' infrastructure and were verified by the authors, who take full responsibility for the content of this work.

References

1. Anderson, A., Huttenlocher, D., Kleinberg, J., & Leskovec, J. (2012). Discovering value from community activity on focused question answering sites: A case study of Stack Overflow. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 850–858). ACM. https://doi.org/10.1145/2339530.2339665
2. Bosu, A., Corley, C. S., Heaton, D., Chatterji, D., Carver, J. C., & Kraft, N. A. (2013). Building reputation in StackOverflow: An empirical investigation. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)* (pp. 89–92). IEEE. https://doi.org/10.1109/MSR.2013.6624013
3. Crosby, S. A., & Wallach, D. S. (2009). Efficient data structures for tamper-evident logging. In *Proceedings of the 18th USENIX Security Symposium* (pp. 317–334). USENIX Association.
4. Douceur, J. R. (2002). The Sybil attack. In P. Druschel, F. Kaashoek, & A. Rowstron (Eds.), *Peer-to-peer systems (IPTPS 2002)* (Lecture Notes in Computer Science, Vol. 2429, pp. 251–260). Springer. https://doi.org/10.1007/3-540-45748-8_24
5. European Parliament and Council of the European Union. (2016). *Regulation (EU) 2016/679 (General Data Protection Regulation)*. Official Journal of the European Union. https://eur-lex.europa.eu/eli/reg/2016/679/oj
6. Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security, 4*(3), 224–274. https://doi.org/10.1145/501978.501980
7. Fowler, M. (2005). *Event sourcing*. https://martinfowler.com/eaaDev/EventSourcing.html
8. Garcia-Molina, H., & Salem, K. (1987). Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (pp. 249–259). ACM. https://doi.org/10.1145/38713.38742
9. Gilbert, E. (2013). Widespread underprovision on Reddit. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work* (pp. 803–808). ACM. https://doi.org/10.1145/2441776.2441866
10. Helland, P. (2016). Immutability changes everything. *Communications of the ACM, 59*(1), 64–70. https://doi.org/10.1145/2844112

11. Hoffman, K., Zage, D., & Nita-Rotaru, C. (2009). A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys, 42*(1), 1–31. https://doi.org/10.1145/1592451.1592452
12. Jøsang, A., Ismail, R., & Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems, 43*(2), 618–644. https://doi.org/10.1016/j.dss.2005.05.019
13. Movshovitz-Attias, D., Movshovitz-Attias, Y., Steenkiste, P., & Faloutsos, C. (2013). Analysis of the reputation system and user contributions on a question answering website: StackOverflow. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (pp. 886–893). ACM. https://doi.org/10.1145/2492517.2500242
14. Overeem, M., Spoor, M., Jansen, S., & Brinkkemper, S. (2021). An empirical characterization of event sourced systems and their schema evolution—Lessons from industry. *Journal of Systems and Software, 178*, Article 110970. https://doi.org/10.1016/j.jss.2021.110970
15. Pal, A., Harper, F. M., & Konstan, J. A. (2012). Exploring question selection bias to identify experts and potential experts in community question answering. *ACM Transactions on Information Systems, 30*(2), Article 10. https://doi.org/10.1145/2180868.2180872
16. Politou, E., Alepis, E., & Patsakis, C. (2018). Forgetting personal data and revoking consent under the GDPR: Challenges and proposed solutions. *Journal of Cybersecurity, 4*(1), Article tyy001. https://doi.org/10.1093/cybsec/tyy001
17. Quora Inc. (n.d.). *Quora*. Retrieved September 2026, from https://www.quora.com/
18. Ratna, V., Khairnar, P., Bhardwaj, S., & Khatri, R. (2025). Securing academic social platforms: Implementing role-based access control (RBAC) in university-based digital systems. *International Journal of Research Trends and Innovation, 10*(5), a433–a442. https://www.ijrti.org/papers/IJRTI2505044.pdf
19. Reddit Inc. (n.d.). *Reddit*. Retrieved September 2026, from https://www.reddit.com/
20. ResearchGate GmbH. (n.d.). *ResearchGate*. Retrieved September 2026, from https://www.researchgate.net/
21. Resnick, P., Kuwabara, K., Zeckhauser, R., & Friedman, E. (2000). Reputation systems. *Communications of the ACM, 43*(12), 45–48. https://doi.org/10.1145/355112.355122
22. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer, 29*(2), 38–47. https://doi.org/10.1109/2.485845
23. Schneier, B., & Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security, 2*(2), 159–176. https://doi.org/10.1145/317087.317089
24. Schwaber, K., & Sutherland, J. (2020). *The Scrum guide*. https://scrumguides.org/scrum-guide.html
25. Srba, I., & Bielikova, M. (2016). A comprehensive survey and classification of approaches for community question answering. *ACM Transactions on the Web, 10*(3), 1–63. https://doi.org/10.1145/2934687
26. Stack Overflow. (n.d.). *What is reputation? How do I earn (and lose) it?* Retrieved September 2026, from https://stackoverflow.com/help/whats-reputation
27. Tausczik, Y. R., & Pennebaker, J. W. (2011). Predicting the perceived quality of online mathematics contributions from users' reputations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1885–1888). ACM. https://doi.org/10.1145/1978942.1979215

28. Thelwall, M., & Kousha, K. (2015). ResearchGate: Disseminating, communicating, and measuring scholarship? *Journal of the Association for Information Science and Technology, 66*(5), 876–889. https://doi.org/10.1002/asi.23236
29. Van Noorden, R. (2014). Online collaboration: Scientists and the social network. *Nature, 512*(7513), 126–129. https://doi.org/10.1038/512126a
30. Wang, X., Huang, C., Yao, L., Benatallah, B., & Dong, M. (2018). A survey on expert recommendation in community question answering. *Journal of Computer Science and Technology, 33*(4), 625–653. https://doi.org/10.1007/s11390-018-1845-0