IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

SkillSwap: A Secure Peer-to-Peer Platform for Non-Monetary Skill Exchange and Accountable Collaborative Learning in Higher Education

Kinjal Dnyaneshwar Mali[1], Prof. Manoj V. Nikum[2], Dr. D. M. Patel[3]

Student MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[1]

HOD MCA & Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[2]

Principal, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[3]

Abstract: University students face economic and pedagogical barriers when acquiring practical technical skills: commercial platforms and tutoring services require recurring fees, massive open online courses report low completion, and informal messaging groups lack structure, scheduling and accountability. This paper presents SkillSwap, a web-based peer-to-peer platform on which students exchange competencies they can teach for competencies they wish to learn, without monetary transactions. The system follows a three-tier architecture comprising a vanilla JavaScript client, a Python Flask 3.0 application tier with 21 REST routes, and a Supabase-hosted PostgreSQL 15 persistence tier that enforces Row Level Security (RLS), JSON Web Token (JWT) authentication and trigger-based profile provisioning. A bilateral workflow governs peer discovery through debounced live autocomplete, exchange negotiation, session scheduling, rescheduling and dual-confirmation completion, while every state change is written to an audit table. Validation comprised 28 functional test cases and benchmarks of six transactional endpoints under one to 200 concurrent simulated users. All 28 test cases passed. The live autocomplete endpoint recorded a median latency of 28 ms, every endpoint met its 95th-percentile service target, and the platform sustained 890 requests per second at 200 concurrent users with a 0.00% error rate. The results indicate that a structured, secure and auditable barter model for peer learning is technically feasible on commodity infrastructure.

Keywords: Collaborative learning, Flask, peer-to-peer learning, Row Level Security, skill exchange, Supabase.

I. INTRODUCTION

Continuous skill acquisition has become a prerequisite for engineering undergraduates, because domains such as cloud infrastructure, full-stack development and machine learning evolve faster than university curricula can be revised. Learners therefore turn to online alternatives, including course platforms [1], video marketplaces [2] and paid tutoring services [3]. These services generally depend on subscriptions, per-course fees or hourly rates, which limits access for financially constrained students, and recorded lectures offer little personalised feedback. Completion rates reported for massive open online courses (MOOCs) are correspondingly low [4].

Peer learning offers a pedagogically grounded alternative. Vygotsky's zone of proximal development holds that learners progress further with the help of a more knowledgeable other [5], and a slightly more advanced peer can often fulfil this role effectively [6]. Tutors also benefit: explaining material to others promotes knowledge building [7] and increases the effort invested in learning, a phenomenon known as the protégé effect [8]. A reciprocal arrangement in which each participant alternately teaches and learns therefore benefits both parties. In practice, however, such arrangements depend on matchmaking, scheduling and mutual commitment, and informal channels such as messaging communities [9] provide none of these in a structured form.

This paper presents SkillSwap, a web platform that operationalises reciprocal peer learning as a non-monetary skill exchange among university students. The objectives are to (i) let students catalogue skills they can teach and wish to learn, (ii) support low-latency peer discovery, (iii) govern exchange negotiation and session scheduling through explicit bilateral state machines, (iv) isolate user data at the database tier, and (v) validate functional correctness and performance quantitatively. The novelty lies in combining, within one open web stack, three features that the surveyed approaches do not offer together: a strictly barter-based exchange model, bilateral session governance with dual-confirmation completion, and database-enforced multi-tenant authorisation with a per-session audit trail. The main contributions are:

1) a three-tier architecture and relational schema for peer skill barter, comprising five tables and ten RLS policies;
2) two coupled state machines for exchange requests and learning sessions, including dual-confirmation completion and audit logging;
3) an implementation with 21 REST routes and a debounced live-autocomplete discovery subsystem; and

 This work is licensed under a Creative Commons Attribution 4.0 International License

4) a quantitative evaluation consisting of 28 functional test cases and latency and concurrency benchmarks for up to 200 virtual users.

The remainder of the paper is organised as follows. Section II reviews related work, Section III states the problem and describes the architecture, implementation and methodology, and Section IV presents the results, discussion, limitations and future scope. Section V concludes.

II. LITERATURE REVIEW

A. Peer Learning Foundations

Vygotsky's account of the zone of proximal development [5] underpins most peer-tutoring models. Topping's review of peer learning [6] discusses cognitive and metacognitive gains for both tutors and tutees, and Roscoe and Chi [7] examine how tutor learning arises from reflective knowledge building rather than from simple knowledge telling. Social interdependence theory further explains why cooperative structures promote achievement and positive relationships [10], while research on computer-supported collaborative learning studies how technology can mediate such interaction [11]. These studies motivate the dual-role design of SkillSwap, but they do not prescribe a software architecture for matching, scheduling or accountability.

B. Existing Platforms and Approaches

Four archetypes of existing solutions were compared. MOOC platforms such as Coursera [1] deliver structured video curricula but are broadcast-oriented, offer limited peer interaction and exhibit low completion [4]. Video marketplaces such as Udemy [2] sell individual courses and treat the learner as a consumer. Tutoring marketplaces such as Wyzant [3] provide synchronous one-to-one instruction at hourly rates. Informal messaging communities such as Discord [9] are free and synchronous but lack a skill taxonomy, scheduling and accountability mechanisms. Table I summarises the comparison, which is based on the characterisation of these approaches in the source project report.

TABLE I: Comparison of SkillSwap with Existing Approaches

Approach	**Cost model**	**Skill matching**	**Session governance and accountability**
Coursera (MOOC) [1]	Paid (subscription)	Course recommendation	Self-paced deadlines; automated quizzes
Udemy (marketplace) [2]	Paid (per course)	Keyword search	Self-paced; none
Wyzant (tutoring) [3]	Paid (hourly)	Instructor directory	Calendar scheduling; financial penalty
Discord / messaging groups [9]	Free (informal)	Manual message scanning	None
SkillSwap (proposed)	Free (strict barter)	Live autocomplete over skill catalogue	Bilateral scheduling; dual-confirmation completion; audit log

C. Identified Gaps & Our Contributions

The survey of existing approaches reveals four primary gaps:

- **G1:** No dedicated non-monetary barter architecture exists for collegiate skill exchange.
- **G2:** Peer matching is rarely interactive and low-latency.
- **G3:** Bilateral state machines for scheduling and mutual verification are absent.
- **G4:** No persistent audit trail records schedule revisions and their rationale.

SkillSwap is designed to address these four gaps by delivering a lightweight, end-to-end web application that integrates a barter-based exchange model, live autocomplete discovery, bilateral session governance and a persistent audit trail.

III. METHODOLOGY

A. Problem Statement and Requirements

The problem is to design and validate a web platform that enables university students to exchange skills without monetary transactions, such that peer discovery is fast, exchanges and sessions are mutually agreed, and personal data remain private to authorised participants. Each student is modelled as a user holding a set of skills to teach and a set of skills to learn; an exchange is a bilateral agreement between two users, and a session is a scheduled meeting that belongs to an accepted exchange. The following requirements were derived:

R1 Registration, authentication and automatic profile provisioning.

R2 Skill cataloguing by type (teach or learn) and by three proficiency levels.

R3 Live skill search with autocomplete latency suited to interactive typing.

 This work is licensed under a Creative Commons Attribution 4.0 International License

R4 Exchange negotiation (request, accept, reject, cancel) with self-requests and duplicate requests prohibited.

R5 Session scheduling, rescheduling, cancellation and completion with mutual consent.

R6 Row-level isolation, so that users access only their own exchanges, sessions and audit records.

R7 Responsiveness under concurrent load, evaluated for up to 200 virtual users.

The proposed methodology for SkillSwap is structured as a decoupled three-tier operational framework designed for secure peer discovery, bilateral exchange governance and auditable session management. Fig. 1 illustrates the architecture, consisting of three tiers: Presentation, Application and Persistence.

B. Three-Tier Architecture

The presentation tier runs in the browser and is implemented with HTML5, CSS3 and vanilla ES6+ JavaScript; it performs client-side validation, issues asynchronous Fetch requests and renders autocomplete suggestions. The application tier is a Python Flask 3.0 service [12] that handles WSGI request dispatching, encrypted session cookies, authentication gatekeeping, parameter validation and state transitions. The persistence tier is a Supabase-hosted PostgreSQL 15 database [13] that provides authentication, relational storage, constraints, triggers and Row Level Security [14]. The application tier reaches the database through the Supabase Python client and its PostgREST interface, following resource-oriented REST principles [15].

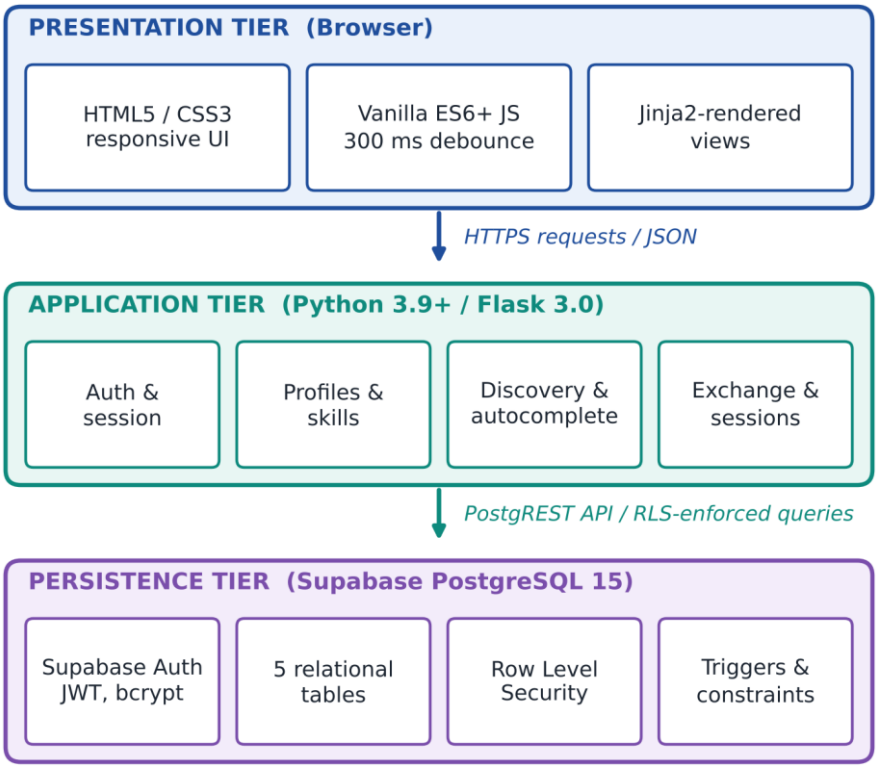


Fig. 1 Three-tier architecture of SkillSwap: browser client, Flask application tier and Supabase PostgreSQL persistence tier.

C. Data Model and Security

The schema comprises five tables (Fig. 2): profiles, skills, exchange_requests, learning_sessions and session_history. A database trigger creates a profiles row whenever a user is added to the authentication schema. Integrity is enforced through foreign keys with cascading deletes, CHECK constraints on skill type (teach or learn), proficiency level (beginner, intermediate, advanced), request status and session status, and a unique constraint on the (requester, receiver) pair that prevents duplicate requests. Ten RLS policies restrict access: profiles and skills are publicly readable to support discovery but writable only by their owner; exchange requests are visible only to their requester and receiver; and session and history rows are accessible only to participants of the corresponding accepted exchange.

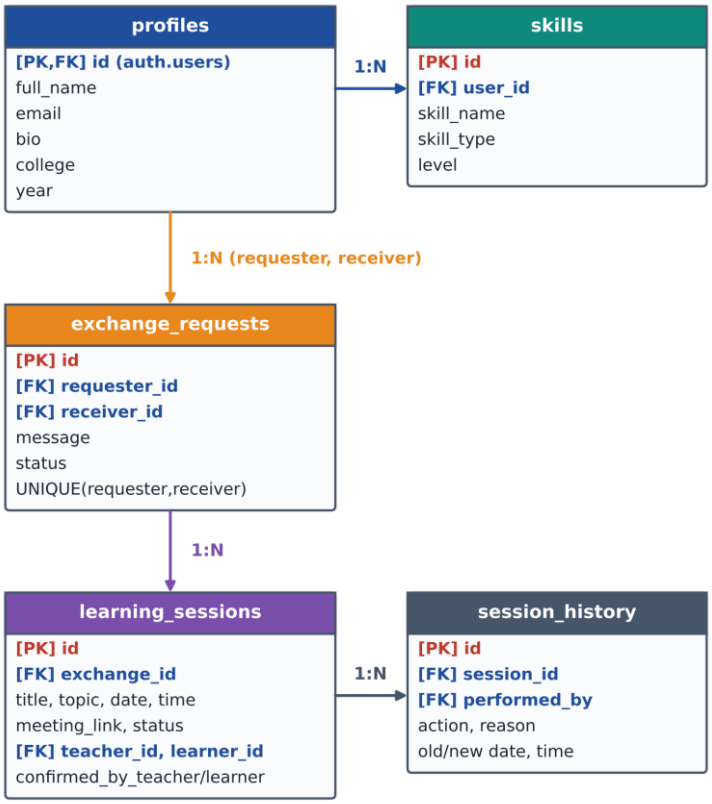


Fig. 2 Entity-relationship diagram of the SkillSwap database (PK: primary key, FK: foreign key).

D. Exchange and Session Workflow

The end-to-end workflow is shown in Fig. 3. After authentication, a student catalogues skills and searches the catalogue; a 300 ms client-side debounce limits the request rate, and each query is resolved by a case-insensitive pattern match (ilike) on the skills table. The student then sends an exchange request, which is created in the pending state; the receiver may accept or reject it, and the requester may cancel it. Acceptance unlocks session scheduling. A session is created in the Scheduled state, and either participant may propose a new date and time, which moves the session to Reschedule Requested until the partner accepts (restoring Scheduled with the new time) or declines (preserving the original schedule). A cancellation passes through Cancellation Requested and requires a justification. For completion, each participant sets an individual confirmation flag, and the session becomes Completed only when both flags are true. Every transition appends a row to session_history containing the action, the actor, the previous and new date and time, and the reason.

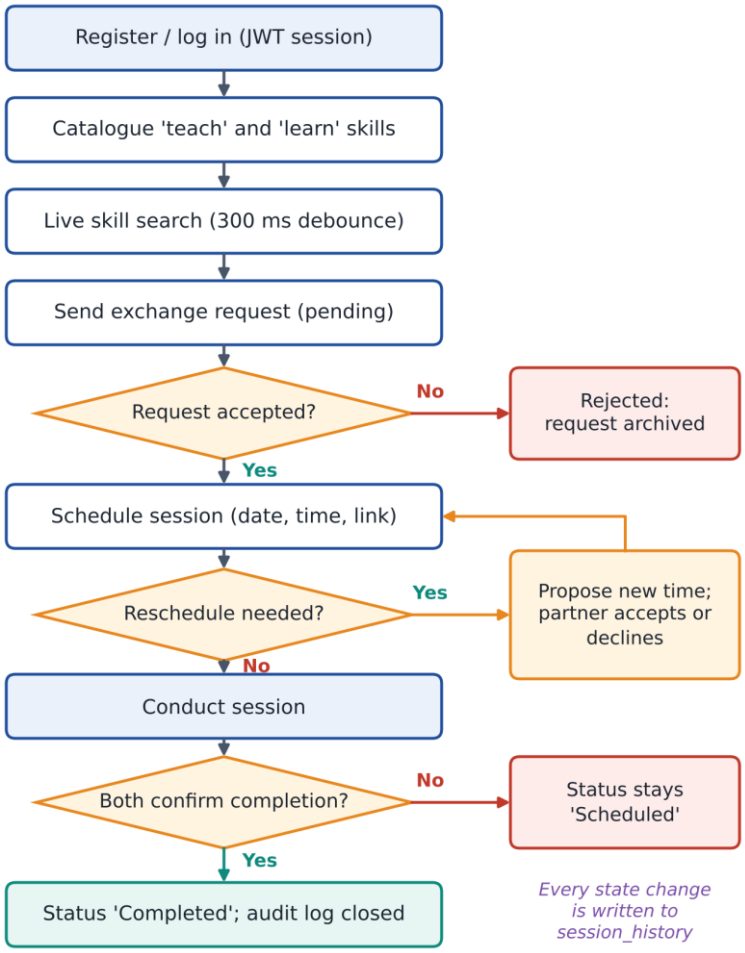


Fig. 3 Workflow of peer matching, exchange negotiation and session lifecycle in SkillSwap.

E. Development Methodology

Development followed six phases executed iteratively in an agile manner: requirement analysis and domain modelling, relational schema and security design, modular backend implementation, client-tier implementation, session lifecycle engine, and testing and profiling.

F. Environment and Technology Stack

Table II lists the minimum requirements and the environment used in the experiments. The application runs on commodity hardware and requires no accelerator. It was tested with Python 3.9.13 and 3.11.4, Flask 3.0.0 with Werkzeug 3.0 or later, supabase-py 2.3 or later and python-dotenv 1.0.0, which separates configuration from code.

TABLE II: Hardware and Software Requirements

Component	**Minimum / supported specification (tested environment)**
Processor	Quad-core, 2.0 GHz or higher (tested: Intel Core i7-12700H)
Memory	4 GB or more (tested: 16 GB DDR4)
Storage	500 MB free (tested: 512 GB NVMe SSD)
Network	1 Mbps or more (tested: 100+ Mbps)
Operating system	Windows 10/11, Ubuntu 20.04+ or macOS 12+
Python	3.9.0 or higher (tested: 3.9.13, 3.11.4)
Web framework	Flask 3.0.0 with Werkzeug 3.0+
Database	Supabase PostgreSQL 15.1 with RLS
Client libraries	supabase-py 2.3+, python-dotenv 1.0.0
Browsers	Chrome, Firefox, Edge 110+; Safari 16+

 This work is licensed under a Creative Commons Attribution 4.0 International License

G. Application Tier

The Flask application exposes 21 routes grouped into authentication, profile and skill management, discovery, exchange negotiation and session management. Protected routes are guarded by a @login_required decorator that redirects unauthenticated requests to the login page. Session cookies are configured with the HttpOnly flag and SameSite=Lax, and inputs are validated before database access. The single asynchronous endpoint, GET /api/skills/search, returns matching skill names and counts as JSON for the autocomplete component. Views are rendered with Jinja2 template inheritance.

H. Identity and Persistence

User identity is handled by Supabase Auth, which issues JSON Web Tokens [16] and stores passwords hashed with bcrypt [17]. Trigger-based profile provisioning removes the need for a separate profile-creation transaction in application code. Because the ten RLS policies are evaluated inside PostgreSQL [14], authorisation does not depend solely on the correctness of the application tier.

I. Client Tier

The interface uses a blue-and-white palette, Inter typography and responsive Flexbox and Grid layouts, and was exercised on mobile, tablet and desktop viewports. Dependency-free JavaScript implements the debounced autocomplete using the Fetch API and direct DOM manipulation.

IV. RESULT

A. Validation Methodology

Three strategies were applied. First, a functional suite of 28 test cases was executed against the Flask test client and the Supabase PostgREST endpoints. Second, latency and concurrency were benchmarked with synthetic traffic generators and profiling middleware, measuring six transactional endpoints over 1,000 requests each and load levels from 1 to 200 concurrent virtual users. Third, a pilot with 25 engineering students exercised the interface across viewports. Service-level agreement (SLA) targets were defined per endpoint as internal performance budgets.

B. Functional Verification

Table III summarises the 28 cases by subsystem, and Fig. 4 shows their distribution. All 28 cases passed, and no regression defects were reported. The suite covers positive and negative paths: short passwords are rejected (TC-02), unauthenticated dashboard access is redirected (TC-07), invalid skill types are rejected by the CHECK constraint (TC-10), deletion of another user's skill is blocked (TC-12), self-requests and duplicate requests are blocked (TC-17, TC-18), a unilateral completion leaves the session in the Scheduled state (TC-26), and a direct PostgREST query on another user's exchange requests returns an empty array (TC-28).

TABLE III: Functional Test Results by Subsystem

Subsystem	**Test cases**	**Verified behaviours**	**Passed**
Authentication	TC-01 to 04	Registration with profile trigger; short-password rejection; login with HttpOnly cookie; logout	4/4
Profiles	TC-05 to 07	Profile retrieval and update; redirect of unauthenticated access	3/3
Skills	TC-08 to 12	Add teach and learn skills; reject invalid type; delete own skill; block foreign delete	5/5
Discovery	TC-13 to 15	Student directory; keyword filter; autocomplete JSON API	3/3
Exchange	TC-16 to 20	Send request; block self and duplicate requests; accept; reject	5/5
Sessions	TC-21 to 27	Create session and audit entry; propose, accept and decline reschedule; unilateral and bilateral completion	7/7
Security (RLS)	TC-28	Direct query returns no foreign rows	1/1
Total	28		28/28

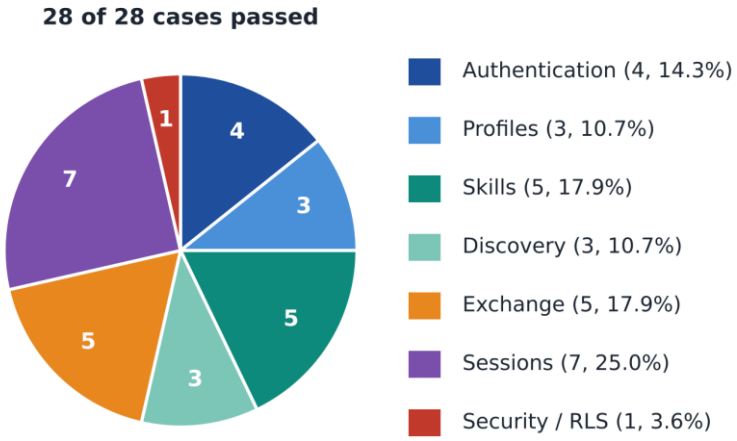


Fig. 4 Distribution of the 28 functional test cases by subsystem; all cases passed.

C. Endpoint Latency

Table IV and Figs. 5 and 6 present median, mean and 95th-percentile latency for six endpoints. The autocomplete endpoint was the fastest, with a median of 28 ms and a 95th percentile of 48 ms. The authentication endpoint was the slowest, with a median of 142 ms, which may reflect the round trip to the hosted identity service. The 95th percentile remained below the SLA target for every endpoint, consuming between 48.0% and 70.0% of the target (Fig. 6). The session-scheduling endpoint, which also writes an audit row, showed a median of 115 ms and a mean of 128 ms.

TABLE IV: Endpoint Latency over 1,000 Requests per Endpoint (ms)

Endpoint	p50	Mean	p95	SLA target	p95 / SLA (%)
Auth / JWT validation	142	158	185	< 300	61.7
Live skill autocomplete	28	34	48	< 100	48.0
Profile fetch and render	64	72	95	< 150	63.3
Send exchange request	88	98	125	< 200	62.5
Schedule session + audit	115	128	165	< 250	66.0
Reschedule transition	96	110	140	< 200	70.0

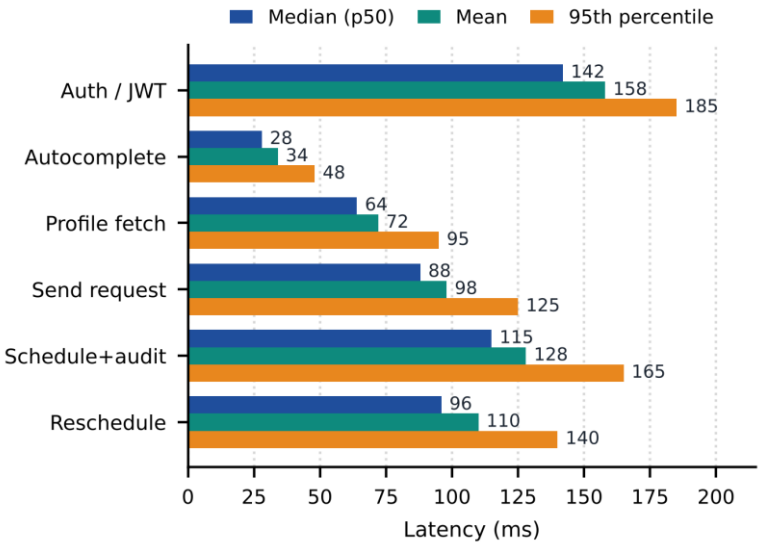


Fig. 5 Median, mean and 95th-percentile latency of six transactional endpoints.

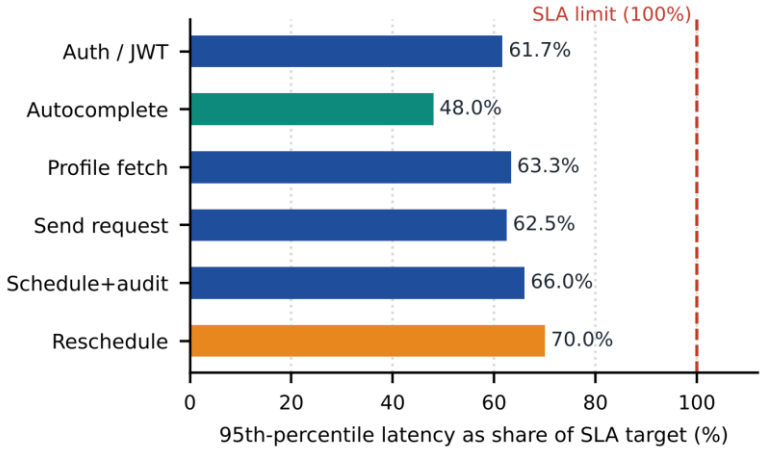


Fig. 6 95th-percentile latency expressed as a share of each endpoint's SLA target.

D. Scalability under Concurrent Load

Fig. 7 shows throughput and mean latency for 1 to 200 virtual users. Throughput rose from 12 to 890 requests/s while mean latency increased from 18 ms to 130 ms, and no errors were recorded at any load level (0.00%). Between 100 and 200 users, throughput increased by 23.6% (720 to 890 requests/s), whereas mean latency increased by 73.3% (75 to 130 ms), which indicates the onset of saturation on the test configuration. Memory consumption remained below 85 MB at 100 users.

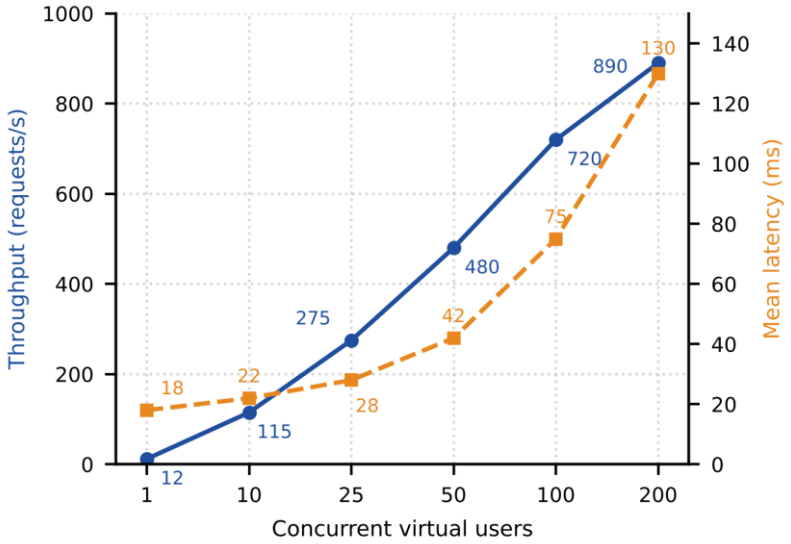


Fig. 7 Throughput and mean latency versus the number of concurrent virtual users.

E. Discussion

Fig. 8 summarises the principal findings. Relative to the approaches in Table I, SkillSwap is the only one that combines a barter model with explicit bilateral session governance and an audit trail. The low autocomplete latency supports gap G2 and requirement R3. The dual-confirmation tests (TC-26, TC-27) support G3 and R5, and the audit entry verified at session creation (TC-22) supports G4. The direct-query isolation test (TC-28) supports R6, although a single test case provides limited coverage of the ten policies. No head-to-head quantitative comparison with the platforms in Table I was performed because those platforms differ in purpose and access model; the comparison is therefore qualitative.

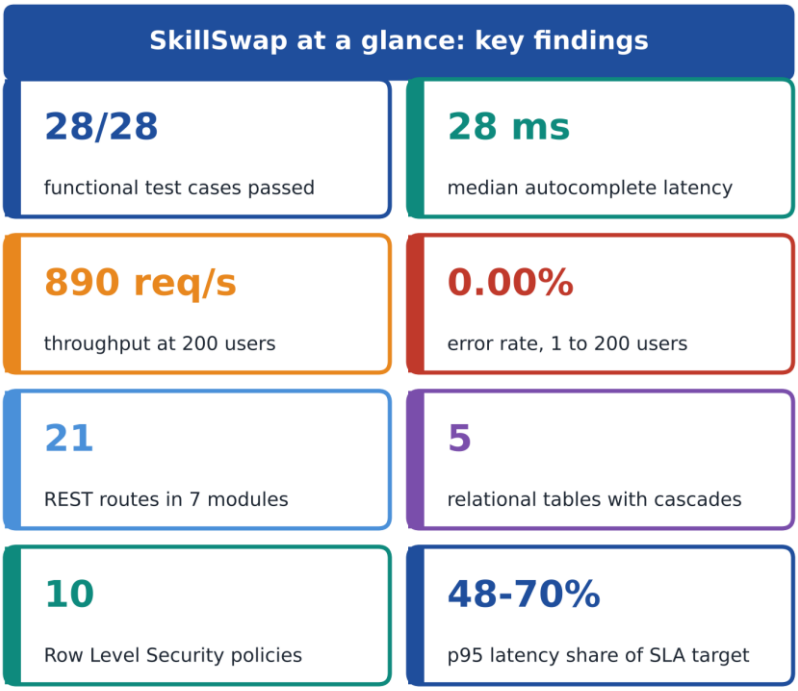


Fig. 8 Infographic summarising the key findings of the SkillSwap evaluation.

F. Limitations and Future Scope

Several limitations apply. First, benchmarks were run on a single development workstation against a hosted database with synthetic traffic, so absolute figures may not transfer to production deployments, and loads above 200 users were not tested. Second, the evaluation does not measure learning outcomes or user satisfaction; the 25-student pilot is described only qualitatively, and quantitative usability data are not available. Third, discovery relies on keyword matching rather than semantic similarity. Fourth, the immutability of the audit table is not independently verified by the reported tests, since TC-22 confirms only that an audit record is created. Fifth, sessions rely on externally supplied meeting links, and the platform provides no in-application messaging.

Future work includes semantic skill matching based on embeddings of profile text and learning goals, in-browser video and shared code editing through WebRTC, verifiable micro-badges for completed mentorship hours, WebSocket-based notifications and messaging, native mobile applications that reuse the existing REST endpoints, and automated content moderation with reputation scoring.

V. CONCLUSION

This paper presented SkillSwap, a peer-to-peer platform that organises student skill exchange as a non-monetary barter governed by bilateral state machines and an audit trail, and that isolates user data through database-level Row Level Security. The implementation combines a Flask application tier with a Supabase PostgreSQL persistence tier and a dependency-free JavaScript client. All 28 functional test cases passed, every benchmarked endpoint met its 95th-percentile target, the autocomplete endpoint reached a 28 ms median latency, and the system sustained 890 requests/s at 200 concurrent virtual users without errors. These results support the technical feasibility of structured and accountable peer skill exchange on commodity infrastructure. Evaluation with real cohorts, including learning outcomes and usability, remains the principal direction for further work.

ACKNOWLEDGMENT

The author thanks the project guide, Prof. Manoj V. Nikum, for guidance and constructive feedback throughout this work.

REFERENCES

[1] Coursera Inc., "Coursera." [Online]. Available: https://www.coursera.org
[2] Udemy Inc., "Udemy." [Online]. Available: https://www.udemy.com
[3] Wyzant Inc., "Wyzant: Find a tutor." [Online]. Available: https://www.wyzant.com
[4] K. Jordan, "Initial trends in enrolment and completion of massive open online courses," Int. Rev. Res. Open Distrib. Learn., vol. 15, no. 1, pp. 133–160, Jan. 2014.

[5] L. S. Vygotsky, Mind in Society: The Development of Higher Psychological Processes. Cambridge, MA, USA: Harvard Univ. Press, 1978.
[6] K. J. Topping, "Trends in peer learning," Educ. Psychol., vol. 25, no. 6, pp. 631–645, Dec. 2005.
[7] R. D. Roscoe and M. T. H. Chi, "Understanding tutor learning: Knowledge-building and knowledge-telling in peer tutors' explanations and questions," Rev. Educ. Res., vol. 77, no. 4, pp. 534–574, Dec. 2007.
[8] C. C. Chase, D. B. Chin, M. A. Oppezzo, and D. L. Schwartz, "Teachable agents and the protégé effect: Increasing the effort towards learning," J. Sci. Educ. Technol., vol. 18, no. 4, pp. 334–352, Aug. 2009.
[9] Discord Inc., "Discord." [Online]. Available: https://discord.com
[10] D. W. Johnson and R. T. Johnson, "An educational psychology success story: Social interdependence theory and cooperative learning," Educ. Researcher, vol. 38, no. 5, pp. 365–379, Jun. 2009.
[11] P. Dillenbourg, "What do you mean by 'collaborative learning'?" in Collaborative-Learning: Cognitive and Computational Approaches, P. Dillenbourg, Ed. Oxford, U.K.: Elsevier, 1999, pp. 1–19.
[12] Pallets, "Flask documentation." [Online]. Available: https://flask.palletsprojects.com
[13] Supabase Inc., "Supabase documentation." [Online]. Available: https://supabase.com/docs
[14] PostgreSQL Global Development Group, "PostgreSQL 15 documentation: Row security policies." [Online]. Available: https://www.postgresql.org/docs/15/ddl-rowsecurity.html
[15] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, USA, 2000.
[16] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Eng. Task Force, RFC 7519, May 2015.
[17] N. Provos and D. Mazières, "A future-adaptable password scheme," in Proc. USENIX Annu. Tech. Conf., FREENIX Track, Monterey, CA, USA, Jun. 1999, pp. 81–91.

 This work is licensed under a Creative Commons Attribution 4.0 International License