

Mind Tussle: Design, Implementation, and Performance Evaluation of a Real-Time Gamified Quiz Platform for Collaborative Learning

Adyan

Department of Computer
Science & Engineering
Integral University, Lucknow,
India

adyan@student.iul.ac.in

Ayaz Ahmad Department
of Computer Science &
Engineering

Integral University, Lucknow,
India

ayaaz@student.iul.ac.in

Abdullah

Department of Computer Science
& Engineering
Integral University, Lucknow, India
abdulalam@student.iul.ac.in

ABSTRACT

Web-based learning platforms have created rising demand for assessment tools capable of supporting large concurrent user populations without compromising responsiveness or data integrity. Commercially available quiz platforms, though feature-rich, operate within proprietary ecosystems that restrict institutional customization, impede integration with existing infrastructure, and transfer ownership of user-generated data to third parties. This paper presents the design, engineering, and empirical evaluation of Mind Tussle, a real-time multi-participant gamified quiz system built on the MERN technology stack (MongoDB, Express.js, React.js, and Node.js). The platform incorporates stateless JSON Web Token (JWT) authentication, creator-controlled quiz authoring, live leaderboard computation, and a modular REST API architecture oriented toward horizontal scalability. Performance testing on a single-host deployment recorded an average API response latency of 57 ms, with endpoint measurements ranging from 6 ms for registration to 137 ms for login the latter attributable to bcrypt-based password hashing. Concurrent session trials confirmed reliable isolation of per-participant state, accurate score recalculation, and leaderboard consistency under simultaneous submission loads. These results establish Mind Tussle as a technically sound foundation for collaborative digital assessment, extensible toward WebSocket-driven synchronization, adaptive

difficulty adjustment, and AI-assisted learning analytics.

General Terms

Design, Algorithms, Performance, Security, Web Systems, Educational Technology

Keywords

gamification, MERN stack, real-time systems, collaborative learning, REST API, JWT authentication, educational technology, quiz platform

1. INTRODUCTION

Gamification the deliberate embedding of game-design mechanics into non-game context has emerged as a prominent instructional strategy within technology- enhanced learning environments. Point accumulation, ranked leaderboards, timed challenges, and reward pathways collectively transform routine assessment activities into participatory events that sustain learner attention and encourage repeated engagement [1]. A substantial body of educational- technology research corroborates these claims: appropriately designed gamified systems demonstrably strengthen intrinsic motivation, sharpen attentional focus, and improve knowledge retention when their structural affordances align with clearly articulated pedagogical objectives [2].

Despite these advantages, dominant commercial platforms such as Kahoot! and Quizizz impose architectural constraints limiting their suitability for institutional adoption. Backend logic is opaque and non-configurable; integration with campus learning management systems requires workarounds; and user-generated content is stored on third-party servers beyond institutional control [5]. A further gap in the literature concerns empirical documentation: prior work on gamified assessment tools emphasizes motivational and learning outcomes while offering little systematic evidence about backend performance, concurrency behavior, or architectural robustness under realistic load conditions [2,5].

The present study addresses these limitations through the design and evaluation of Mind Tussle, a full-stack, self-hosted quiz platform developed to support real-time multi-user participation. The system combines a decoupled client-server architecture with secure authentication, session-scoped multi-participant management, and automated leaderboard recalculation. Unlike most prior contributions in this space, this work moves beyond descriptive design to include quantitative latency measurement and concurrency validation, providing empirical

grounding for claims about feasibility and scalability in collaborative digital assessment settings.

2. THEORETICAL FOUNDATION

2.1 Gamification in Education

Gamification is broadly understood as the application of game-design elements and mechanics to non-game activities with the intent of driving engagement and behavioral change [1]. Within educational contexts, the mechanism of action is frequently explained through Self-Determination Theory (SDT), which identifies competence, autonomy, and relatedness as three fundamental psychological needs whose satisfaction underlies sustained intrinsic motivation [3]. Well-constructed gamified assessments address all three needs: scoring and leaderboard visibility signal competence, voluntary participation preserves autonomy, and synchronous multi-user competition fosters social relatedness. Experimental studies confirm that when these conditions hold, measurable gains in motivation and learning engagement follow [2]. Effective implementations embed game mechanics within core interaction design rather than layering them atop existing workflows, ensuring motivational affordances are experienced as integral rather than decorative [5].

2.2 Collaborative Learning and Social Interdependence

Collaborative learning theory holds that shared problem-solving contexts and peer interaction catalyze deeper cognitive processing and higher achievement than individual study alone [4]. Johnson and Johnson's social interdependence framework identifies positive interdependence the perception that individual success is linked to group outcomes as a driver of cooperative effort and knowledge co-construction [4]. Synchronous competitive quiz sessions operationalize this principle: participants simultaneously collaborate within a shared session context while competing on individual performance metrics, maintaining engagement while providing the social referencing cues that peer comparison affords.

1.1 2.3 Real-Time Web Architecture

Scalable web platforms increasingly rely on REST architectural principles, which prescribe stateless, cacheable, layered communication between clients and servers [6]. The stateless constraint is especially relevant for concurrent assessment environments, as no client context is retained between requests, enabling the server to handle large numbers of independent sessions efficiently. Node.js provides a non-blocking, event-loop-driven execution model well-suited to I/O-intensive workloads involving many simultaneous connections [7]. For applications requiring lower-latency bidirectional communication, the WebSocket protocol maintains persistent full-duplex connections and eliminates repeated HTTP handshake overhead [8]. Mind Tussle draws on all three of these foundations, with REST used for current data exchange and WebSocket integration planned for subsequent development phases.

3. SYSTEM ARCHITECTURE

Mind Tussle adopts a decoupled client-server architectural pattern in which the presentation, application, and data layers operate as independent subsystems communicating through well-defined programmatic interfaces. This separation allows components to be developed, tested, and deployed independently; bugs can be isolated to a single tier without cascading effects; and individual layers can be scaled horizontally without requiring modifications elsewhere in the stack. The three primary layers are: (1) the Presentation Layer, a React.js Single Page Application; (2) the Backend Layer, a Node.js/Express.js REST API server encapsulating all business logic; and (3) the Data Layer, a MongoDB document store accessed through the Mongoose ODM.

3.1 Presentation Layer

The client-side application is implemented as a React.js SPA that loads a single HTML document on initial request and subsequently updates the interface through DOM manipulation rather than full-page reloads, minimizing perceived latency during quiz participation. Tailwind CSS provides the

utility-class styling framework, enabling consistent responsive layouts across viewport sizes. Axios serves as the HTTP client library for request serialization, error interception, and response parsing. Following successful authentication, the issued JWT is stored in the browser's local storage and automatically attached to the Authorization header of all subsequent API requests, maintaining session continuity across page interactions.

3.2 Backend Layer

The application server is built on Node.js with the Express.js framework. Node.js's non-blocking, event-driven runtime is particularly suited to the concurrency demands of a multi-user quiz platform, where numerous participants may simultaneously submit answers within narrow time windows [7].

Rather than spawning a thread per connection a model that incurs significant memory overhead at scale Node.js manages all I/O operations through an asynchronous callback queue, allowing a single process to service many concurrent connections. Express.js structures the application into discrete route and controller modules covering authentication, quiz authoring, session lifecycle, answer submission and scoring, and leaderboard generation. JWT middleware intercepts incoming requests to protected endpoints, verifying the token signature before the request reaches business logic.

3.3 Data Layer

MongoDB serves as the persistence layer, accessed through the Mongoose ODM. MongoDB's document model naturally accommodates the nested data structures characteristic of quiz sessions: a single session document can embed the full list of participants, their answer histories, and their current scores, rather than distributing this information across multiple normalized tables. The embedded participant schema eliminates join operations during leaderboard retrieval, allowing a single document fetch to supply all data required for ranking computation. Score updates are performed as in-place document mutations, reducing write amplification relative to normalized relational approaches

and prioritizing read efficiency for leaderboard queries.

4. SYSTEM IMPLEMENTATION

4.1 Authentication Subsystem

User authentication is implemented using a stateless JWT scheme combined with bcrypt-based password hashing. When a new user submits registration credentials, the backend queries the users collection to confirm the supplied email address has not been previously registered; duplicate detection returns HTTP 400 with appropriate messaging (recorded response time: 7 ms). On successful registration, the password is hashed using bcrypt prior to storage, ensuring plaintext credentials are never persisted. At login, the submitted password is verified against the stored hash; if valid, the server issues a signed JWT encoding the user's identifier and role. Because token state is self-contained, the server maintains no session table, which simplifies horizontal scaling.

4.2 Core API Endpoints

The REST API exposes six primary endpoints organized around the main system resources, following REST constraints as formalized by Fielding [6], including uniform interface, statelessness, and layered architecture. Table 1 enumerates each endpoint, its HTTP method, and functional purpose.

Table 1. Core REST API Endpoints

Endpoint	Method	Description
/api/auth/register	POST	Register new user account
/api/auth/login	POST	Authenticate and issue JWT
/api/sessions	POST	Create a new quiz session

/api/sessions/:id/join	POST	Join active session via code
/api/sessions/:id/answer	POST	Submit answer for scoring
/api/sessions/:id/results	GET	Retrieve ranked leaderboard

5. EXPERIMENTAL EVALUATION

All evaluation trials were conducted under controlled local deployment conditions, with the Node.js application server and MongoDB instance running concurrently on the same physical host. This eliminates network-induced variability between the application and database tiers and provides a reproducible baseline for characterizing intrinsic backend latency. Results should be interpreted as lower bounds rather than representative production metrics.

5.1 API Response Latency

Response times were measured for three representative operations spanning authentication and session-retrieval domains, recorded at the server layer to exclude client-side rendering time. Table 2 presents the recorded values and their arithmetic mean.

Registration latency (6 ms) reflects a lightweight uniqueness check with a single document insert. Session lookup (27 ms) involves a single indexed document retrieval representative of leaderboard-refresh latency. Login latency (137 ms) is substantially higher due to bcrypt—a deliberately computationally expensive algorithm calibrated to resist brute-force attacks. For an infrequent operation such as session login, this latency is unlikely to produce perceptible degradation in user experience.

Table 2. API Response Time Measurements

Operation	Latency (ms)
User Registration	6
User Login	137
Session Lookup	27
Average	57

1.2 5.2 Concurrency and Session Isolation

Concurrent multi-user behavior was validated by constructing a two-participant simulation in which accounts joined the same active session and submitted answers independently within overlapping time windows. The backend was required to maintain separate answer histories, compute independent running scores, and produce a unified leaderboard without state collision or data leakage between participant records. Throughout the trial, each participant's answer submissions were correctly attributed to their respective session records. No instance of cross-participant data overwrite or session document corruption was observed. MongoDB's atomic document-level write semantics ensured sequential score updates did not produce inconsistent intermediate states. The extent to which these isolation guarantees hold under higher concurrency remains to be established through systematic load testing.

1.3 5.3 Functional Correctness Testing

Functional testing covered five critical system behaviors: (1) duplicate registration prevention, confirmed by consistent HTTP 400 responses to re-used email addresses; (2) session access control, validated by rejection of join requests referencing invalid session identifiers; (3) scoring accuracy, verified by submitting known answer sequences and confirming accumulated scores matched manual computations; (4) leaderboard ordering, checked against participant scores after each submission round; and (5) HTTP response code conformance, audited across all

endpoints to confirm RESTful interface conventions [6].

2. 6. DISCUSSION

The experimental results present a coherent picture of a backend that is functionally complete, architecturally sound, and performant within the bounds of its tested deployment context. Sub-100 ms average latency across non-cryptographic operations is consistent with responsiveness thresholds identified in prior work as necessary for maintaining user engagement; delays exceeding approximately 200 ms are associated with perceptible interface lag that disrupts flow states in time-sensitive tasks [7]. The stateless JWT authentication model is a particularly consequential design decision: by encoding identity information within the token itself rather than in server-side session tables, the system eliminates a major horizontal-scaling impediment, as any backend instance can validate any request without consulting shared session state. The most significant architectural gap identified is the absence of WebSocket-based push communication; integration of Socket.IO is identified as the highest-priority near-term development objective.

7. LIMITATIONS

Several limitations bound the scope of conclusions that can be drawn from this evaluation. The most significant is the single-host deployment context; latency figures obtained under these conditions are lower bounds rather than representative production metrics. The two-participant concurrency simulation provides no evidence about system behavior under loads of 50, 100, or more simultaneous users. Additionally, the absence of containerized deployment means architectural properties claimed for horizontal scalability have not been empirically tested.

The platform currently lacks WebSocket-based real-time push communication, relying on REST polling which introduces latency relative to a push-based alternative. Finally, the AI-assisted modules described in future work have not been implemented in the current system version.

8. FUTURE WORK

Planned development falls into four broad categories. The most immediate priority is integration of bidirectional real-time communication using Socket.IO, which would allow the server to push question-delivery events and leaderboard updates to all connected clients simultaneously. Second, the system will incorporate adaptive difficulty mechanisms that adjust quiz complexity in response to per-participant performance trajectories, drawing on item response theory. Third, AI-assisted recommendation features will surface relevant quizzes based on participation history, topic preferences, and knowledge gaps using collaborative filtering approaches. Fourth, production-scale deployment will be validated through containerized orchestration using Docker and Kubernetes, with Redis introduced as an in-memory caching layer for leaderboard state and session metadata. Systematic load testing with simulated user populations of 100 or more will establish production-grade throughput baselines.

9. CONCLUSION

This paper has presented the architectural design, implementation, and empirical evaluation of Mind Tussle, a real-time gamified quiz platform developed on the MERN stack to address the customization and transparency limitations of existing commercial assessment tools. The system integrates stateless JWT authentication, a modular REST API organized around clearly bounded functional domains, an embedded MongoDB document schema optimized for leaderboard queries, and a React.js SPA delivering a responsive participant experience. Performance testing recorded an average API latency of 57 ms across registration, login, and session-lookup operations. Concurrency trials confirmed that the event-driven Node.js backend correctly maintains per-participant session isolation, accurate score computation, and consistent leaderboard ranking under simultaneous submission loads. These results indicate that Mind Tussle constitutes a technically sound and functionally complete foundation for collaborative digital assessment

environments, contributing empirical evidence to a literature historically dominated by motivational outcome studies.

3. ACKNOWLEDGMENTS

The authors thank Mrs. Azra Iftikhar for her guidance and supervision throughout the development of this research. The authors also acknowledge colleagues who participated in testing and validation of the Mind Tussle platform.

4. REFERENCES

- [1] Deterding, S., Dixon, D., Khaled, R., and Nacke, L. 2011. From game design elements to gamefulness: Defining gamification. In Proc. 15th Int. Academic MindTrek Conf., Tampere, Finland, 9-15.
- [2] Sailer, M., Hense, J., Mayr, S., and Mandl, H. 2017. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. *Computers in Human Behavior*, 69, 371-380.
- [3] Ryan, R. M. and Deci, E. L. 2000. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1), 54-67.
- [4] Johnson, D. W. and Johnson, R. T. 2009. An educational psychology success story: Social interdependence theory and cooperative learning. *Educational Researcher*, 38(5), 365-379.
- [5] Domínguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fernández-Sanz, L., Pagés, C., and Martínez-Herráiz, J. 2013. Gamifying learning experiences: Practical implications and outcomes. *Computers & Education*, 63, 380-392.
- [6] Fielding, R. T. 2000. Architectural styles and the design of network-based software architectures. Ph.D. dissertation, Univ. of California, Irvine, CA, USA.
- [7] Tilkov, S. and Vinoski, S. 2010. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, 14(6), 80-83.

- [8] Pimentel, V. and Nickerson, B. 2012. Communicating and displaying real-time data with WebSocket. IEEE Internet Computing, 16(4), 45-53.
- [9] Brusilovsky, P. and Millan, E. 2007. The adaptive web. In The Adaptive Web, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Springer, Berlin, 3-51.
- [10] Zichermann, G. and Cunningham, C. 2011. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. O'Reilly Media, Sebastopol, CA.