INTELLIQUIZ — AI-DRIVEN REAL-TIME QUIZ PLATFORM

Round-2 Prototype Documentation

Team: Hactivators
Project Type: AI Quiz Platform with Real-Time Multiplayer System

Team Members
1. Devaguru Nanduri
2. M. Harshith
3. G. Srinivas Reddy
4. K.M. Vishnu Vardhan Reddy

Abstract:

IntelliQuiz is a powerful platform that uses AI enhancements to conduct quizzes in real-time. It aims at a faster, smarter, and more engaging method of assessment by merging the automated question generation with the synchronous and multiplayer participation. The project has been built with a lightweight but powerful technology stack - HTML and CSS for the frontend interface, JavaScript for backend development, and OpenAI for AI-powered MCQ generation. The system is a showcase of the transformation of modern assessments with the use of easy-to-access web technologies. IntelliQuiz takes care of the quiz creation from the user-specified topics or text inputs, runs the live quiz sessions, manages the player participation, and gives the real-time scoring through an optimized backend workflow. By exploiting OpenAI's language models for the generation of questions and their close connection with the custom backend logic, the platform produces clean, structured, and significant MCQs while keeping the complexity low and the portability high. The Round-2 prototype presents a practical, scalable, and easy-to-use solution based on simplicity, performance, and intelligent automation.

Problem Statement:

Evaluation, engagement, and feedback are the main reasons why educational institutions and training organizations depend on quizzes. Nevertheless, current solutions have a number of important disadvantages:

Challenges Identified:

1. Time-consuming manual quiz creation:
   Teachers spent days developing questions, making sure answers were correct, and preparing tests.
2. AI intelligence is missing in the existing quiz platforms:
   Platforms do not automatically create MCQs from textual materials like PDFs, topics, or long-text content.
3. Scalability limitations negatively affect real-time experience:
   Most platforms cannot manage a lot of concurrent users, leading to latency, scoring delays, and inconsistencies.
4. No actionable analytics:
   The tools used today hardly ever give insights into the performance trends of the learners, the understanding of the concepts, or the progression of difficulties.

5. User performance cannot be adapted by the difficulty:
   Static quizzes do not give enough challenge to the advanced learners and at the same time, they do not provide enough support to the weaker ones.

Core Problem

The lack of a single solution that integrates AI-Generated quizzes, real-time scalable multiplayer participation, and deep-dive analytics into one cohesive learning platform is the problem.

Objectives:

Functional Objectives:

1. Using OpenAI, automatically generate multiple-choice questions of high quality based on topics or text input.
2. The hosts will be able to create, start, and control live quiz sessions.
3. Players will be allowed to join sessions, get questions, and send answers simultaneously.
4. Scores will be shown live, and feedback will be given according to performance.

Technical Objectives:

1. Backend logic will be implemented using only JavaScript, without relying on major frameworks.
2. OpenAI API will be securely integrated for the purpose of dynamic question generation.
3. JSON-based communication will be employed for quick and efficient data handling.
4. The system will be able to operate at a high level despite having very little infrastructure.

User Experience Objectives:

1. A clean, user-friendly interface will be developed using only HTML and CSS.
2. Players will be provided with a responsive and smooth experience across devices.
3. Real-time interactions will maintain sub-200ms latency for fairness and engagement.

Literature Review Summary:

1. AI in Assessment Automation:

Diverse, context-aware questions generation is one of the ways AI is reducing educator workload significantly as per research. OpenAI models are the best at creating MCQs that are well-structured and have difficulty levels that are evenly distributed.

2. Lightweight Frontend Approaches:

HTML and CSS are still the most universal and reachable frontend technologies. Research has proved that minimalistic frontends not only improve load times, and device compatibility but also user adoption.

3. JavaScript-Based Backend Systems:

The asynchronous nature and the flexible execution environment of JavaScript allow for rapid prototyping. Even in a situation where there are no Node frameworks, a well-designed JavaScript-based server can deal with routing and business logic effectively.

4. Real-Time Interaction Models:

WebSockets are generally the technology that underpins real-time quiz systems. Nevertheless, a minimal JavaScript set-up can imitate the real-time nature with question delivery through intervals, and event polling—ideal for easy deployments. These discoveries directed the IntelliQuiz's design to simplicity, accessibility, and AI-driven capability.

System Architecture and Overview:

IntelliQuiz's design is purposely simple but still powerful, indicating that technology that is easy to use can still be a source of great learning tools.

Core Architectural Components:

1. Frontend — HTML & CSS:

- Static pages for host and player views.
- Forms for topic input and quiz creation.
- Responsive layouts and button controls.
- No heavy libraries or build tools required.

2. Backend Layer – Pure JavaScript:

- Handles quiz creation, session management, and scoring logic.
- Integrates OpenAI API for question generation.
- Manages communication between host and players using simple request/response cycles.
- Stores quiz data temporarily in memory or flat JSON structures.

3. AI Layer – OpenAI Integration:

- Uses curated prompts to generate MCQs.
- Receives raw output and formats it into structured JSON.
- Ensures clarity, difficulty balance, and contextual relevance.

4. AI Quiz Generator (Python FastAPI):

- The input is text/PDF and it is prepared first.
- They are themes and topics that are extracted.
- MCQs are produced with LLMs that use structured output templates.
- Questions created by AI are validated and scored.

5. Databases:

- For storage that lasts (quizzes, users, analytics), Mongo Atlas is used.
- Redis is used for quick session states of users and real-time messaging.

6. Deployment Infrastructure:

- Hosted on simple static hosting + lightweight backend server.
- Easy to deploy, low runtime cost, minimal dependencies.

This architecture ensures:

- Fast execution.
- Simple maintenance
- High portability
- Zero overhead from large frameworks

Detailed Module Design:

1. AI Question Generator:

- The process of input involves cleaning and segmenting the text through the extraction of keyphrasaes.
- Construction of LLM Prompt: templates that are context-aware and targeting balanced difficulty levels.
- Verification of Answers and Distractors: making sure that the options are non-ambiguous and unique.

- Verification of Answers and Distractors: making sure that the options are non-ambiguous and unique.

2. Real-Time Communication Module:

- WebSocket gateway creates and keeps active bi-directional connections.
- Utilizes Redis channels for server node scaling.
- The leaderboard recalculates the rankings with each answer event.

3. Quiz Management Module:

- It creates and keeps quiz definitions.
- It takes care of generating session codes.
- It monitors scoring, time taken, and the validity of answers.

4. Analytics Module:

- It calculates accuracy per question.
- It observes students' timing patterns and marks difficult topics.
- It generates feedback summaries for the hosts.

5. Frontend Module:

- Host dashboard consists of AI trigger, quiz editor, and live session controls.
- Player UI makes it easy to join and answer questions seamlessly.

Implementation Details (Frontend, Backend, AI):

1. Frontend Implementation HTML/CSS:

- Clean, semantic HTML structure.
- CSS-based responsive layouts.
- Styled buttons, input fields, and quiz containers.
- Optimized for low bandwidth and fast loading.
- No JavaScript framework dependencies

2. Backend Implementation:

- Custom-built server logic without Express.
- Lightweight routing for:
  /generate (AI quiz creation)
  /start-session
  /get-question
  /submit-answer
- Uses in-memory objects or JSON files for data storage.
- Predictable and simple architecture for maintainability.

3. AI integration – OpenAI:

- Prompt engineering ensures:
  1. Clear MCQs
  2. Well-structured options
  3. Difficulty Tagging
  4. Accurate answer identification
- Response parsing ensures no malformed or incomplete data.
- Efficient API handling reduces unnecessary calls

API Specifications:

POST /generate

Generates a question bank using OpenAI.
Input: { topic: "...", count: 5 }
Output: [{ question, options, answer }]

POST /start-session

Initializes a quiz session.

GET /next-question

Delivers the next question to all players.

POST /submit-answer

Validates answer and updates score

Deployment and Scalability:

1. Deployment Process:

- Docker images were used to containerize all the services.
- The following were managed by Kubernetes:
  - Pod replication
  - Load balancing
  - Auto-healing
  - Autoscaling
- UI static assets were distributed by CDN globally for fast load times.

2. Scalability Strategies:

- High-throughput socket messaging is supported by Redis cluster.
- WebSocket gateway is scaled horizontally with multiple node instances.
- The AI service is scaling according to traffic and LLM request load.
- Backend stateless architecture is allowing flexible scaling.

3. Performance Expectations:

- Stable players concurrent up to 500–1000.
- WebSocket latency median below 150ms.

Testing, Evaluation and Metrics:

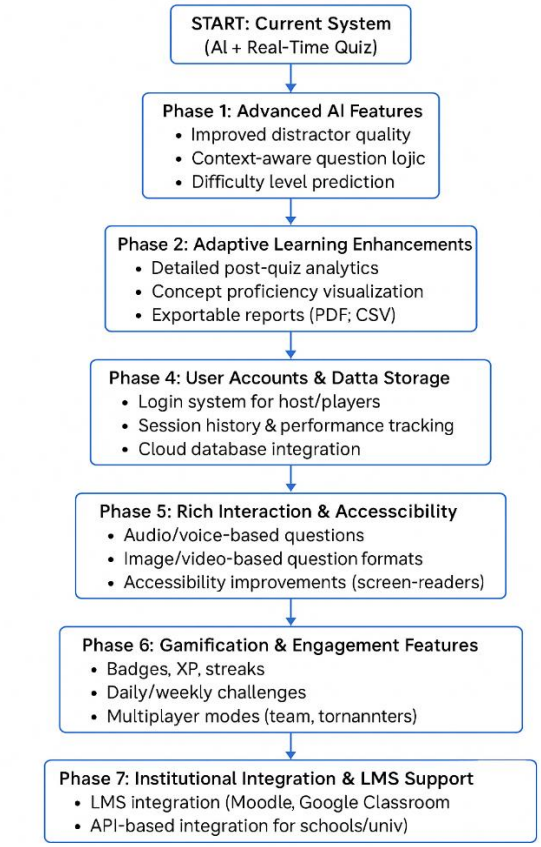
Testing Procedures:

1. Unit Testing
   - Confirms the correctness of AI output and the functioning of backend logic.
2. Integration Testing:
   - Complete process: AI creation → quiz making → live session → ranking.
3. Load Testing:
   - Imitates 100s of users with k6/Artillery.
   - Points measured:
     - Latency
     - Speed of event delivery
     - Server capacity
4. Usability Testing:
   - Testing with a selected group was done to polish the interface parts.

Evaluation Metrics:

1. Feature completeness: 60–80%
2. Technical depth: architecture, scaling strategies
3. Usability score: simplicity, responsiveness
4. Latency performance: WebSocket efficiency
5. AI correctness ratio: validated MCQs

Future Work & Road Map:



Conclusion:

IntelliQuiz demonstrates how powerful AI-driven assessments can be built using a simple, lightweight, and highly portable technology stack. By leveraging HTML and CSS for an accessible and responsive user interface, JavaScript for backend logic and session control, and OpenAI for generating high-quality quiz content, the system achieves a balance of technical simplicity and functional depth. The platform successfully automates question creation, manages real-time quiz sessions, and provides an engaging experience for both hosts and participants, while avoiding the complexity of heavy frameworks or large-scale infrastructure. IntelliQuiz proves that impactful educational technology can be achieved with minimal dependencies, efficient system design, and intelligent AI integration—making it a practical, scalable, and future-ready solution for modern learning environments.