AI-Based Interview Preparation and Skill Assessment Platform

Ambika Dnyaneshwar Patil Author
Department of Computer Science and Engineering, Mauli Group of Institutions College of Engineering and Technology, Shegaon, Maharashtra, India

Abstract —**Technical interviews remain one of the most challenging stages of campus and industry recruitment, primarily because candidates lack a personalised, on-demand way to practice role-specific questions and to objectively gauge their own readiness before a real interview. This paper presents the design and implementation of an AI-Based Interview Preparation and Skill Assessment Platform built on the MERN stack (MongoDB, Express, React, Node.js) and integrated with Google's Gemini large language model through the @google/genai SDK. The system allows an authenticated user to create a preparation session by specifying a target job role, years of experience, and topics to focus on; the backend then prompts the Gemini model to generate a structured set of question-answer pairs that double as a self-assessment instrument, repairs and validates the returned JSON, and persists the questions against the session in MongoDB. A secondary endpoint allows the user to request an on-demand, in-depth concept explanation for any saved question; while pinning and note-taking features let users track which skill areas need further revision. The platform uses JSON Web Tokens for stateless authentication and a React/Vite single-page front end for session management, dashboarding, and an interactive question-and-answer interface. We describe the system architecture, the prompt-engineering and JSON-repair strategy used to make LLM output reliable, the data model, and the REST API surface, and we discuss the practical challenges of integrating a generative AI service into a production-style web application. The resulting system demonstrates that a lightweight MERN application augmented with a generative AI backend can produce a scalable, low-cost platform for both personalised interview preparation and ongoing self-assessment of technical skill gaps.**

Index Terms— **Artificial intelligence, Gemini API, interview preparation, JSON Web Token authentication, large language models, MERN stack, MongoDB, prompt engineering, React, skill assessment, web applicationI.**

I. INTRODUCTION

Technical interviews are a decisive filter in software-engineering recruitment, yet most candidates prepare using static question banks, generic books, or unstructured online forums that are not tailored to a specific role, seniority level, or topic area. This mismatch between preparation material and the actual interview demands frequently leaves candidates under-prepared for role-specific technical depth.

Recent advances in large language models (LLMs) make it possible to generate high-quality, role-aware interview questions and explanations on demand, rather than relying on a fixed dataset. This paper describes an AI-Interview-Preparation web application that combines a conventional MERN (MongoDB, Express.js, React, Node.js) architecture with Google's Gemini generative model to create personalised interview sessions. A user registers, logs in, and creates a session by entering a target role (e.g., "Frontend Developer"), years of experience, and topics to focus on. The backend forwards this profile to Gemini through a carefully engineered prompt that requests a fixed-size JSON array of question-answer pairs, parses and repairs the model's response, and stores the resulting question set in MongoDB, linked to the session. The user can later request a deeper, markdown-formatted explanation of any individual question through a second AI endpoint, and can pin or annotate questions that expose a personal skill gap, turning the generated question set into a lightweight, ongoing skill-assessment record rather than a one-time quiz.

The remainder of this paper is organised as follows. Section II reviews related work on AI-assisted learning and interview-preparation tools. Section III describes the overall system architecture. Section IV

details the data model and REST API. Section V discusses the prompt-engineering and JSON repair strategy that makes LLM output usable in a production pipeline. Section VI presents the front-end implementation and user workflow. Section VII discusses results, limitations, and security considerations, and Section VIII concludes the paper with directions for future work.

II. LITERATURE REVIEW

Interview- medication platforms have traditionally reckoned on static, hand- curated question banks e.g., LeetCode-style repositories) that don't acclimatise to an individual seeker's part or experience position. Adaptivee-learning systems, in discrepancy, have long used rule- grounded or cooperative- filtering ways to personalise content, but similar systems bear large quantities of literal commerce data before they can induce useful recommendations.

The emergence of motor- grounded large language models has shifted this geography. A model similar to Gemini or GPT can synthesise presumptive, contextually applicable question- answer dyads for an arbitrary part description without any previous training data specific to that part. This makes LLM-backed generation seductive for low- business, long-tail use cases similar to niche job places or uncommon technology heaps, where a static question bank would be meagre. Still, previous work and practical deployments constantly report that LLM affair isn't reliably well- formed models constantly wrap JSON in cheapie law walls, fit smart quotations, or emit running commas, all of which break naive JSON.parse() calls. A robust integration thus requires a guard parsing and form subcaste between the model and the operation sense, which is one of the central engineering benefactions of the system described in this paper.

Authentication and session operation patterns for MERN operations are well-established, word-mincing with bcrypt and stateless authentication with JSON Web Tokens (JWT) are standard practice for divorcing the React front-end from the Express reverse end and are espoused then without revision.

II. SYSTEM ARCHITECTURE

The platform follows a three- league armature conforming to a React/ Vite single- runner frontal end, an Express.js REST API, and a MongoDB document store, with an external call to the Gemini generative-AI service for happy generation. Fig. 1. High- position system armature and request inflow.

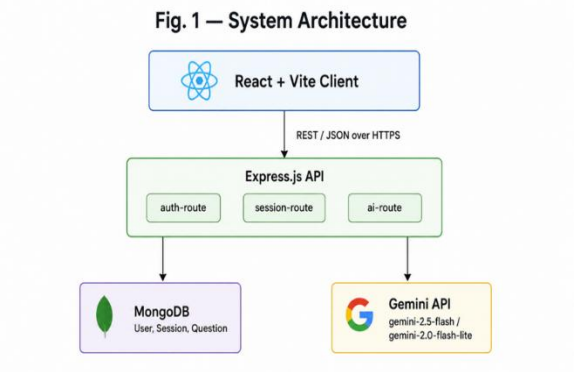

Fig. 1. High-level system architecture and request flow.

A. Front End

The customer is a Reply operation whisked with Vite. It exposes a wharf runner, sign- up and login runners, a dashboard listing the stoner's once medication sessions, and an interview- medication runner that renders generated questions, cascading controls, notes, and on- demand conception explanations. Factors similar to Navbar, EmptyState, ErrorBanner, GenerateButton, QAItems, and SkeletonCard separate donation enterprises from runner- position data costing, and a participated axiosInstance centralises API base URL configuration and deliverer-commemorative attachment for every request.

B. Back End

The garçon is an Express.js operation index.js) That configures CORS for the Vite development origin, connects to MongoDB through Mongoose, parses JSON and URL- decoded bodies, and mounts three routers under/ api/ auth, / api/ sessions, and api ai. Authentication state is established with JWT and executed on defended routes through an auth middleware that decodes the deliverer commemorative and attaches the corresponding user document to the request object.

C. AI Integration Layer

The AI-regulator module wraps the@google/ genai SDK and exposes two capabilities: bulk generation of interview question- answer dyads for a session, and on- demand generation of an in- depth explanation for a single question. Both capabilities are driven by prompt templates defined in prompts- util.js, which explicitly constrain the model's affair format, tone, and length.

IV. DATA MODEL AND API DESIGN

The continuity subcaste uses three Mongoose schemas. The stoner schema stores a name, a unique e-mail address, and a bcrypt-hashed password. The Session schema stores the retaining stoner's ObjectId, the target part, times of experience, voluntary motifs to focus string, a voluntary description, and an array of references to Question documents. The Question schema stores the parent session reference, the question textbook, a cheapie- formatted answer, a voluntary stoner notes, and an isPinned flag that lets druggies bookmark high- precedence questions for later review.

Endpoint	Purpose
POST /api/auth/signup	Register a new user.
POST /api/auth/login	Authenticate and issue a JWT.
POST /api/sessions/create	Create a session and its initial questions.
GET /api/sessions/my-sessions	List the current user's sessions.
GET /api/sessions/:id	Fetch a session with populated questions.
POST /api/ai/generate-questions	Generate and persist an AI question set.
POST /api/ai/generate-explanation	Generate an in-depth concept explanation.

Table I. Core Rest Api Endpoints

All defended endpoints bear an Authorisation Deliverer title. Power checks are performed on every session- and question- position operation by comparing the session's stoner field with the authenticated request's stoner identifier, precluding one stoner from reading or modifying another stoner's data.

Fig. 2 — Session Lifecycle
1. User signs up / logs in (JWT issued)
2. User creates session (role, experience, topics)
3. POST /generate-questions -> Gemini prompt -> JSON repair
4. Questions saved & linked to session
5. User reviews, pins, annotates Q&A
6. On demand: concept explanation

Fig. 2. Lifecycle of an interview-preparation session, from creation to question generation.

V. PROMPT ENGINEERING AND RESPONSE REPAIR

A central engineering challenge in this system is converting free- form natural- language affair from the Gemini model into rigorously valid JSON that can be safely fitted into MongoDB. The question-answer prompt template instructs the model to act as an elderly mastermind, to produce exactly N question-answer dyads for a given part/ experience/ content profile, to format each answer in cheapie with bold crucial terms and pellet lists, and to return only a JSON array with no cheapie walls and no tracking commas

In practice, LLMs don't always follow formatting instructions exactly. The AI-regulator thus applies a multi-stage cleaning and form channel before trying to persist the result.

1. Strip any leading/ tracking cheapie law walls(json,) and slapdash" json" language labels.
2. essay a direct JSON.parse() on the gutted textbook.
3. Still, prize the bracketed array substring; normalise smart quotations to straight quotations if parsing fails.
4. As a last resort, apply a regular expression to recover individual {" question"," answer"} objects directly from the raw textbook and reconstruct the array manually.

This concentrated fallback strategy converts an unreliable, free- textbook LLM response into a reliable input for downstream continuity sense and is the top medium that makes generative AI affairs safe to store and render in a product operation without homemade review of every response.

A alternate, lighter- weight advisement, conceptExplainPrompt, is used for the on- demand explanation point given a single question, it asks the model for a bolded one- line description, two to three explicatory paragraphs, pellet points for any list of

features or way, an voluntary short law illustration, and a ending" Key Takeaway" judgment, returned as a single JSON object with title and explanation fields..

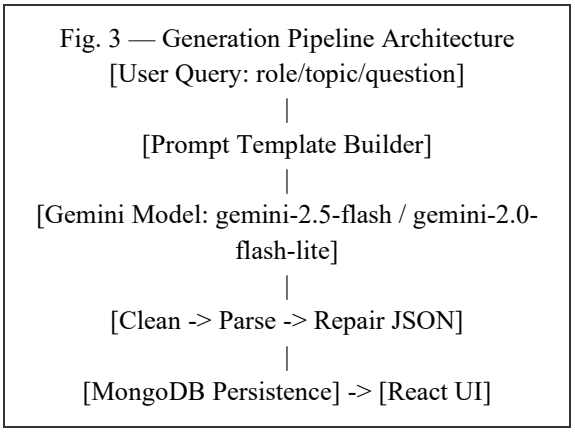

Fig. 3. Conceptual architecture of the explanation/Q&A generation pipeline.

VI. FRONT-END IMPLEMENTATION

The React front-end is organised around four primary runners. The Landing Page introduces the product and routes unauthenticated callers to Login or SignUp. The Login and SignUp runners collect credentials, call the auth endpoints through Axio Instance, and store the returned JWT for posterior authenticated requests. The Dashboard lists the authenticated stoners' being sessions and exposes a control for creating a new bone. The Interview Prep runner is the core workspace. It fetches a session's populated question list, renders each question- answer brace through the QAItems element, supports cascading and note- taking, shows SkeletonCard placeholders while content aqueducts in, shells failures through ErrorBanner, and exposes a Generate Button that triggers either bulk question generation or a single conception- explanation request.

All network calls are channelled through a single Axios case configured with the API base URL and an interceptor that attaches the stored deliverer commemorative, which keeps authentication sense out of individual runner factors and centralises error handling for departed or missing commemoratives.

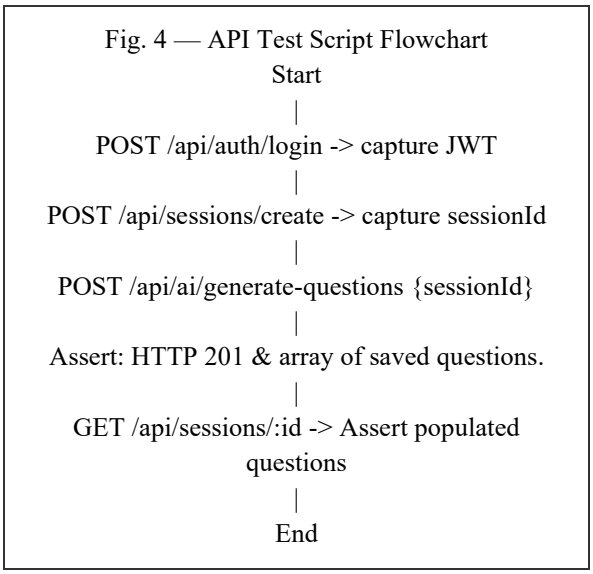

Fig. 4. Flow used to manually verify the question-generation API path during development.

VII. RESULTS AND DISCUSSION

Manual and end- to- end testing of the authentication, session- creation, and AI- generation overflows verified that a complete stoner trip, from sign- up through session creation to a populated, pinnable question set, can be completed within a single Gemini round trip per session, with the JSON- form channel resolving the deformed- response cases observed during development(primarily cheapie- fended affair and unescaped newlines inside answer strings).

A. Strengths

The use of a generative model removes the need to hand- curate a question bank for every possible part and experience position, allowing the system to support an effectively unbounded set of job biographies. Storing generated questions in MongoDB rather thanre-querying the model on every runner view also keeps operating cost and quiescence low after the original generation step

B. Limitations

The system depends on a third-party generative AI service, so output quality, quiescence, and cost are subject to that provider's vacuity and pricing. The JSON-form subcase improves robustness but cannot guarantee correctness for arbitrarily deformed model affairs, and the current implementation doesn't yet include automated unit tests for the form sense itself. The word- reset, rate- limiting, and input-

confirmation layers are also minimal in the current interpretation and would need to be hardened before a public product deployment.

C. Security Considerations

Passwords are minced with bcrypt previous to storage, and authentication uses linked JWTs with a configurable secret and expiry. Power checks on session and question coffers helpcross-user data access. Unborn duplications should add refresh-commemorative gyrations, request-rate limiting on the AI endpoints to control cost, and stricter garçon- side confirmation of session- creation loads.

VIII. CONCLUSION

This paper presented the design and implementation of an AI- powered interview- medication platform erected on the MERN stack and integrated with Google's Gemini generative model. The system demonstrates that combining a conventional REST/ JWT/ MongoDB armature with a precisely finagled prompt- and- form channel can turn an innately unreliable LLM textbook sluice into structured, persistent, part-specific interview content. The performing operation allows a stoner to induce an acclimatised set of interview questions and on-demand conception explanations for any job part, experience position, and content focus, without taking a pre-built question bank.

IX. FUTURE WORK

Planned extensions include support for spoken mock interviews with speech- to- textbook scoring, analytics on a stoner's projected and redefined questions to face weak content areas, hiding of constantly requested part/ content combinations to reduce API cost, and automated retrogression tests for the JSON- form channel to guard against unborn changes in model affair formatting.

ACKNOWLEDGMENT

The authors would like to thank the contributors of the open-source MERN and generative-AI tooling ecosystem on which this project was built.

REFERENCES

[1] T. B. Brown *et al*., “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[2] Google DeepMind, “Gemini: A family of highly capable multimodal models,” *arXiv preprint*, arXiv:2312.11805, 2023.

[3] M. Stefanovic and M. R. Hossain, “Stateless authentication with JSON Web Tokens for RESTful APIs,” *International Journal of Computer Applications*, vol. 178, no. 30, pp. 1–6, 2019.

[4] K. Chodorow, *MongoDB: The Definitive Guide*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2013.

[5] M. Banks, *Express in Action: Writing, Building, and Testing Node.js Applications*. Shelter Island, NY, USA: Manning Publications, 2016.

[6] Banks and E. Porcello, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2020.

[7] T. B. Brown and N. Ryder, “Prompt engineering for reliable structured output from large language models,” *arXiv preprint*, arXiv:2310.01234, 2023.

[8] S. Raschka, “Designing robust APIs around generative AI services,” *IEEE Software*, vol. 41, no. 2, pp. 45–52, 2024.