Smart Mock AI: Interview engine and Job Recommendation Simulator

MS. M. Lavanya, Vinay Reddy, Shashi Vardhan, Ganesh, Rohith

Department of Information Technology

Keshav Memorial Institute of Technology, Hyderabad, Telangana, India

vinay949fxv@gmail.com

Abstract

Online recruitment has changed the way job seekers and employers interact, but most existing platforms still rely on static listings and manual screening processes that do little to help candidates prepare for actual interviews. This paper describes the development of an AI-Powered Job Portal integrated with a Resume-Based Smart Mock Interview System. The platform allows users to register, upload their resumes, and apply for job listings in a straightforward manner. What makes it different from conventional portals is the built-in mock interview module, which reads the user's uploaded resume, extracts relevant skills and experience, and generates interview questions tailored to that profile. Users can answer these questions within the interface, and the system evaluates their responses based on relevance, completeness, and keyword matching, then returns structured feedback. The goal is not just to help users find jobs, but to give them a practical way to prepare before facing real interviews. The system is built entirely using standard web technologies — HTML, CSS, JavaScript, Node.js, and a relational database — without depending on any heavy machine learning infrastructure. It is lightweight, deployable on shared hosting, and accessible to students and fresh graduates who are often most in need of interview preparation support.

Keywords — *Job Portal, Mock Interview, Resume Parsing, Question Generation, Response Evaluation, Career Preparation, Web Application, Natural Language Processing*

I. INTRODUCTION

The process of finding employment has moved almost entirely online over the past decade. Platforms such as LinkedIn, Naukri, and Indeed have made it easier for job seekers to discover openings and for companies to reach a wider pool of applicants. However, despite this convenience, a significant gap remains between getting shortlisted for a position and actually performing well in the interview. Many candidates, particularly fresh graduates and early-career professionals, struggle not because they lack technical knowledge but because they have had little or no practice at articulating their skills under interview conditions.

Interview preparation is often either expensive — through coaching classes or paid platforms — or generic, offering the same set of questions to everyone regardless of background. A computer science graduate with a focus on databases needs different practice than someone whose resume highlights front-end development or data analysis. Personalized preparation tools that adapt to individual profiles are relatively rare, and most of what exists either requires paid subscriptions or relies on complex AI infrastructure that is difficult to deploy and maintain at a student-project scale.

This work attempts to bridge that gap by embedding a resume-aware mock interview system directly into a job portal. The idea is that the same platform where a user searches and applies for jobs also helps them prepare for those jobs. When a user uploads a resume, the system extracts their listed skills and areas of experience and uses that information to generate relevant interview questions. The user answers those questions in the interface and receives immediate feedback on the quality of their responses. This closes a loop that most portals leave open.

The project is intentionally built with accessible technologies. Node.js handles the backend logic, a standard SQL database stores user and job data, and the frontend is built with HTML, CSS, and JavaScript. The question generation and response evaluation components use pattern-based text processing rather than external AI services. This makes the system easy to host, easy to extend, and honest about what it actually does — which matters for a project aimed at practical deployment in academic and career-development contexts.

II. PROBLEM STATEMENT

Most online job portals serve primarily as listing aggregators. They allow users to submit applications but do not offer any structured mechanism for candidates to assess their readiness for interviews. This creates a situation where a user can apply to dozens of positions but remain unprepared for any of them.

The specific problems this system addresses are as follows:

1. Generic interview preparation tools do not account for individual skill sets or work experience. A candidate with three years of backend development needs different practice than someone fresh out of college with only academic projects.
2. There is no widely available free tool that integrates job searching and interview preparation into a single platform. Users currently have to switch between multiple applications and services.
3. Feedback on practice interview answers is typically absent from free tools. Without feedback, candidates cannot identify weak points in their responses.
4. Many interview preparation platforms require subscriptions or depend on cloud-based AI services with usage limits, making them inaccessible to students in resource-constrained environments.

The proposed system aims to address these issues by building a combined platform that is functional, accessible, and honest in its design scope.

III. OBJECTIVES

The primary objectives of this project are:

- To develop a fully functional web-based job portal where users can register, manage profiles, upload resumes, and apply for posted job listings.
- To implement a resume parsing module that extracts relevant technical skills, experience areas, and educational background from uploaded documents.
- To design a question generation engine that creates interview questions specific to the skills and domains identified in a user's resume.
- To build a response evaluation module that assesses user answers against expected content using keyword matching and basic text analysis.
- To provide actionable feedback to users after each mock interview session, highlighting what was answered well and where improvement is needed.
- To ensure the system remains deployable on standard web hosting without requiring specialized hardware or paid AI API subscriptions.

IV. SYSTEM ARCHITECTURE

The system is organized into three logical layers that interact through defined interfaces. The presentation layer handles all user-facing interactions, the application layer manages business logic and data processing, and the data layer handles persistent storage of user profiles, job listings, resumes, and session records.

A. Presentation Layer

The frontend is built using HTML5, CSS3, and JavaScript. It includes separate views for user registration and login, a job listing and search interface, a resume upload form, and the mock interview screen. The mock interview screen presents questions one at a time, accepts typed responses, and displays feedback after each answer or at the end of a session. The design is kept functional and clean, with a focus on usability over visual complexity.

B. Application Layer

The backend is developed using Node.js with the Express framework. It exposes RESTful endpoints that the frontend calls for operations such as user authentication, job search, resume upload, question generation, and answer submission. The application layer also includes the core processing modules: the resume parser, the question generator, and the response evaluator. These components operate server-side and return structured JSON responses to the client.

C. Data Layer

A relational database stores all persistent data including user accounts, uploaded resume metadata, job listings, question banks, and interview session histories. The schema is normalized to avoid redundancy and to allow efficient querying. Resume files are stored in a designated server directory, and only the file path along with extracted metadata is saved in the database.

D. Processing Modules

Three specialized modules handle the intelligent functionality of the system:

- Resume Parser: Reads uploaded PDF or plain-text resume files and identifies sections such as skills, education, and experience using regular expressions and keyword dictionaries.
- Question Generator: Maps extracted skills to a pre-built question bank organized by domain and difficulty level, selecting appropriate questions for each session.
- Response Evaluator: Compares user answers to model responses using a keyword presence and coverage score, then assigns a rating and generates written feedback.

V. METHODOLOGY

The system follows a sequential workflow that begins when a user logs in and ends with a feedback report after a completed mock interview session. The steps are described below.

Step 1: User Registration and Profile Setup

A new user registers with basic information — name, email, and password. After registration, the user can fill in additional profile details and upload a resume in PDF or TXT format. The system stores the file and triggers the parsing process automatically upon upload.

Step 2: Resume Parsing

The uploaded resume is read line by line on the server. The parser uses a set of predefined keyword lists organized by category — programming languages, frameworks, databases, tools, soft skills, and academic domains. Each line of the resume is checked against these lists, and matches are recorded with their category labels. The parser also identifies section headings such as "Work Experience" and "Projects" to give contextual weight to certain keywords. The output is a structured JSON object containing identified skills and experience markers.

Step 3: Job Search and Application

The job listing interface allows users to search by keyword, location, or domain. Listings are fetched from the database and displayed with relevant details. Users can apply with one click; the application is stored in the database along with a timestamp and the user's profile reference. Employers or administrators can view applications through a separate backend dashboard.

Step 4: Mock Interview Session Initialization

When a user chooses to start a mock interview, the system retrieves the skill profile parsed from their resume. It selects a set of questions from the question bank that corresponds to the identified domains. Questions are chosen to cover different difficulty levels — introductory, intermediate, and slightly advanced — to give a rounded session. The session is initialized with a unique identifier and stored in the database.

Step 5: Question Delivery and Answer Collection

Questions are presented one at a time on the interview screen. The user reads each question and types their answer in a text area. There is no time limit enforced, as the intention is to support thoughtful practice rather than simulate pressure. Each answer is submitted individually before the next question appears.

Step 6: Response Evaluation

After each answer is submitted, the evaluator compares it against a stored model answer for that question. The model answer contains a set of expected keywords and phrases associated with a correct or strong response. The evaluation uses a simple scoring method: it counts how many expected terms are present in the user's answer, normalizes this against the total expected count, and produces a percentage-based score. It also identifies which important terms were absent and includes them in the feedback. The score is categorized into three bands: satisfactory, needs improvement, and insufficient.

Step 7: Feedback Report Generation

At the end of the session, a summary report is generated and displayed. It shows the questions asked, the user's answers, the scores assigned to each, and the specific terms that were missing from weaker answers. The report is also saved to the user's session history so they can review past attempts over time.

VI. IMPLEMENTATION

The system is implemented using a standard web development stack. The following technologies are used:

Frontend

- HTML5 and CSS3 for page structure and styling.
- JavaScript (vanilla) for dynamic interactions, form validation, and asynchronous API calls using the Fetch API.
- Session tokens stored in browser local storage for maintaining login state.

Backend

- Node.js runtime with the Express.js framework for API routing and middleware management.
- Multer middleware for handling resume file uploads.
- PDF-parse library for extracting text content from uploaded PDF resumes.
- bcrypt for password hashing and jsonwebtoken for session management.

Database

- MySQL is used as the primary database. Tables include users, jobs, applications, resumes, questions, sessions, and session_responses.
- Queries are parameterized to prevent SQL injection.

Question Bank

The question bank is stored as a JSON file organized by domain tags. Each entry contains the question text, the domain it belongs to, the difficulty level, the model answer, and a list of expected keywords. Currently the bank includes approximately 200 questions across domains such as web development, databases, data structures, operating systems, networking, and soft skills. The bank can be expanded without modifying any application code.

Deployment

The application is tested on a local Node.js server and is structured for deployment on standard VPS or shared hosting environments that support Node.js. No external API calls are made at runtime, which means there are no rate limits or costs associated with scaling up usage.

VII. RESULTS AND DISCUSSION

The system was tested with a group of final-year undergraduate students across two sessions. In the first session, participants uploaded their resumes and completed a mock interview round. In the second session, held one week later after participants had reviewed their feedback, the same group attempted another round.

Qualitatively, the results were encouraging. Most participants found the question selection relevant to their profiles. Students with data-heavy resumes received questions on SQL and Python, while those listing front-end projects were asked about HTML, CSS, and JavaScript fundamentals. This confirmed that the resume parsing and question mapping were working as intended for typical undergraduate resumes.

The feedback mechanism was well-received. Several students noted that seeing exactly which keywords were missing from their answers helped them understand what interviewers typically expect. This is a simple insight, but it is one that students rarely get outside of coached settings.

In terms of system performance, response times for question generation and answer evaluation remained under one second for typical resume sizes and answer lengths, even on modest hardware. The Node.js server handled concurrent sessions without observable degradation during testing.

There were some limitations in the evaluation quality. Short or poorly worded answers sometimes scored deceptively well if they happened to contain several expected keywords without demonstrating real understanding. Conversely, a well-structured answer that paraphrased expected content without using exact terms sometimes scored lower than it deserved. These edge cases point toward the known limitations of keyword-based evaluation, which is discussed further in the limitations section.

Overall, the system functions as intended as a practical tool for basic interview preparation. It fills a realistic gap — personalized question generation linked to a job portal — without overpromising on the sophistication of its AI components.

VIII. ADVANTAGES

- Personalization: Interview questions are drawn from the user's own skill profile rather than a generic pool. This makes each session more relevant than standard practice tools.
- Integration: Combining job search and interview preparation in a single platform reduces the friction of using multiple tools. A user can identify a job and begin preparing for it in the same session.
- Accessibility: The system does not require any paid subscriptions or API keys. It can be accessed by anyone with an internet connection and a browser, making it suitable for students in low-resource settings.
- Lightweight Architecture: Because all processing happens server-side with standard Node.js code, the system can be hosted on basic infrastructure without specialized hardware or cloud computing credits.
- Transparency: The feedback the system provides is based on clear criteria — keyword presence in model answers. This means users understand why they received a particular score rather than receiving an opaque rating.
- Extensibility: The question bank is stored in a simple JSON format and can be updated or expanded by anyone maintaining the project. New domains or difficulty levels can be added without touching the application logic.

IX. LIMITATIONS

Like any system built with a defined scope, this platform has several limitations that are important to acknowledge honestly.

- Keyword-Based Evaluation Only: The response evaluator does not understand sentence meaning or semantic relationships. An answer can score poorly despite being factually correct if it uses different vocabulary than the model answer. This is a fundamental limitation of term-matching approaches.
- Limited Resume Parsing for Complex Formats: The parser works reliably on plain-text resumes and simple PDFs. Heavily formatted resumes with tables, columns, or images may not parse correctly, leading to incomplete skill profiles.
- Static Question Bank: The questions are pre-authored and finite. The system does not generate entirely new questions on its own. Over time, users who attempt multiple sessions may encounter repeated questions.
- No Voice or Video Support: Real interviews often involve verbal communication and non-verbal cues. This system operates entirely through text, which means it does not help users practice spoken delivery, pace, or body language.
- No Employer-Side Features: The current system is designed for job seekers. Employer functionalities like posting jobs and reviewing applications are minimal and not the focus of the project.

X. FUTURE SCOPE

Several enhancements are planned for future iterations of this project.

- Semantic Answer Evaluation: Integrating a sentence similarity approach — such as cosine similarity over TF-IDF vectors or lightweight word embeddings — would improve the quality of response scoring without requiring heavy infrastructure.
- Dynamic Question Generation: Extending the question bank to generate novel questions programmatically by combining templates with skill terms extracted from the resume would reduce repetition for frequent users.
- Speech-Based Interview Mode: Adding a voice input option that transcribes spoken answers would allow users to practice verbal communication, which is closer to real interview conditions.
- Multilingual Support: Adding support for regional languages in the user interface would make the platform accessible to a broader demographic, particularly in non-English-speaking regions.
- Employer Dashboard: Developing a more complete employer module with job posting, application tracking, and candidate shortlisting features would make the portal useful on both sides of the hiring process.
- Resume Scoring and Suggestions: Alongside interview preparation, the system could analyze uploaded resumes and suggest improvements based on common employer expectations, such as missing sections or low keyword density for specific roles.

XI. CONCLUSION

This paper presented an AI-Powered Job Portal with a Resume-Based Smart Mock Interview System designed to address a practical gap in existing recruitment platforms. The system allows users to register, search and apply for jobs, upload their resumes, and take personalized mock interview sessions — all within one unified web application. Interview questions are selected based on the skills and domains identified from the user's resume, and responses are evaluated using keyword-based analysis with structured feedback.

The implementation uses accessible, standard technologies — Node.js, MySQL, and browser-side JavaScript — without relying on external AI services or cloud-based machine learning platforms. This makes it realistic to build, deploy, and maintain at the level of a student or small-team project. Testing with undergraduate students showed that the system produces relevant questions, provides understandable feedback, and performs reliably within the expected usage scope.

The platform does not claim to replicate the full depth of human-evaluated interview practice. What it does offer is a free, personalized, and integrated starting point that is better than generic question lists and more honest than tools that overstate their AI capabilities. With the enhancements described in the future scope, it has the potential to grow into a genuinely useful career preparation tool.

. ACKNOWLEDGMENT

The authors would like to thank the faculty of the Department of Information Technology at Keshav Memorial Institute of Technology for their guidance throughout the development of this project. We also acknowledge the open-source communities behind Node.js, Express.js, and the various libraries used in this work.

. REFERENCES

[1] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. O'Reilly Media, 2009.

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[3] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology behind Search, 2nd ed. Addison-Wesley, 2011.

[4] D. Flanagan, JavaScript: The Definitive Guide, 7th ed. O'Reilly Media, 2020.

[5] E. Brown, Web Development with Node and Express, 2nd ed. O'Reilly Media, 2019.

[6] A. Sweigart, Automate the Boring Stuff with Python, 2nd ed. No Starch Press, 2019.