

InterAI: An AI-Assisted Multi-Round Web Platform for Integrated Interview Preparation with Fallback-Based Question Generation

¹Ramesh B E, ²Shruthi M K, ³Vaishnavi Kattimani, ⁴Vidya mv, ⁵Shambhavi G S, ⁶G R GOWRI

^{1 & 2} Associate Professor, ^{3, 4, 5 & 6} Final Year Students, Department of Computer Science and Engineering

SJM Institute of Technology, Chitradurga, Karnataka, India

Abstract—Preparing for a software job normally means moving between one website for aptitude tests, another for coding practice and a third for mock interviews, and the candidate has to join the scores together alone. This paper presents InterAI, a web-based platform that brings these three activities into one sequential workflow with a single login and a final consolidated report. The backend is written in Python with FastAPI, the frontend uses HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and JavaScript, data is kept in SQLite, and access is controlled with JWT (JSON Web Token) authentication and hashed passwords. Aptitude questions can be generated by an optional AI (Artificial Intelligence) provider, and a local fallback dataset keeps the round running when no provider is reachable. The coding round selects three questions according to the technology stack chosen by the candidate, and the interview round delivers stack specific prompts and stores the responses. We describe the layered architecture, the module design, the database and the implementation, and we report functional testing with fourteen test cases that cover authentication, the three rounds, storage and reporting. All fourteen cases passed in a local environment. The platform is meant for practice and self-assessment, not for hiring decisions, and its present limits and planned extensions are discussed.

Key words: Interview preparation, FastAPI, JWT authentication, language model, question generation, web application, SQLite, self-assessment.

I. INTRODUCTION

Getting a software job is rarely about a single skill. A candidate is usually asked to clear an aptitude test first, then a coding round, and finally a technical and behavioral interview. Most students prepare for each of these on a different website or application, each with its own login, its own scoring and no common record. After a few weeks of practice it is hard to say where one really stands, because the results are scattered over several places.

The recent growth of transformer based language models [1], [2] has made it practical to produce practice material automatically, and their use in education is now studied widely [3]. A platform that combines such generation with ordinary assessment logic can reduce the dependence on fixed

question banks. At the same time, an AI service can be unreachable, can run out of quota, or may simply not be configured on a college laboratory machine. A practical tool should therefore not stop working when that happens.

In this paper we present InterAI, a multi-round interview preparation platform that was developed as a final-year project in our department. A registered user passes through an aptitude round, a technical coding round and a mock interview round, and finally receives one report that combines the stored results. The main contributions of the work are the following. First, the three rounds are placed in one web application with a common login and a common database. Second, the aptitude round uses AI generated questions only as an option, and a local dataset takes over automatically. Third, the coding and interview content is chosen from the technology stack saved in the profile of the user. Fourth, the backend is divided into separate modules and its routes are protected with JWT (JSON Web Token) authentication and hashed passwords. Fifth, a reporting module builds a consolidated summary with improvement recommendations from the stored sessions.

The rest of the paper is arranged as follows. Section II reviews related work. Section III describes the proposed architecture and workflow. Section IV explains the implementation. Section V reports testing and results, Section VI discusses limitations, and Section VII concludes the paper with future work.

II. RELATED WORK

A. AI in Recruitment and Automated Interviews

Köchling and Wehner [4] reviewed 36 articles on algorithmic decision-making in recruitment and staff development. They found that such systems can bring efficiency and objectivity, but they also found repeated concern about discrimination and perceived unfairness. Langer et al. [5] studied highly automated job interviews and observed that the reaction of candidates depends on the stakes of the interview. These two studies influenced a design decision in our work. InterAI is positioned as a tool for practice and self-assessment, and it does not rank or select candidates for employment. An earlier study by the author [6] dealt with resume analysis together with interview

preparation. The present work concentrates on the assessment rounds themselves and keeps the session records in a database.

B. Question Generation

Elkins et al. [7] asked human raters to judge questions produced by a large language model for educational use and reported that the questions were useful and varied enough to be of practical value, though they still need checking. Bulathwela et al. [8] adapted pre-trained language models to educational text in their EduQG approach and showed better question quality, which supports self-assessment at scale. These findings encouraged us to use generated questions in the aptitude round. They also explain why we kept a fixed local dataset beside the generator.

C. Programming Content and Assessment

Sarsa et al. [9] used large language models to create programming exercises, sample solutions, test cases and code explanations, and found that most of the output was new and sensible but not always correct. Savelka et al. [10] tested GPT models on assessments from Python courses and found that the models could not pass the whole range of assessment types at that time. Orvalho et al. [11] presented GitSEED, an automated assessment tool for programming courses built around Git, which gives scalable and personal feedback. Taken together, these works show that generating programming content and judging submitted code are two separate problems. For that reason InterAI keeps a dedicated coding module with its own evaluation logic and does not leave the judgement to a language model.

D. Speech and Perception

Radford et al. [12] described Whisper, a speech recognition system trained with large scale weak supervision, and Lugaresi et al. [13] described MediaPipe, a framework for building perception pipelines. Both are useful for interview systems that listen to or watch the candidate. We mention them as background only. The present version of InterAI does not use speech or camera input.

E. Gap Addressed

The studies above treat recruitment fairness, question generation, code assessment and interview technology mostly as separate topics. What a student actually needs is one place where all the preparation stages are linked and where the scores are kept together. InterAI addresses this practical gap, and it does so with a design that still works when the AI service is not available.

III. PROPOSED SYSTEM

A. Design Goals

We set the following goals before starting the design: a single sequential workflow for the three rounds; separation of the user interface from the application logic; secure registration and login; persistent storage of every session; optional AI generation with automatic fallback; and a modular backend that can be extended later without rewriting the whole application.

B. System Architecture

The system follows a layered client-server architecture, shown in Fig. 1. The browser is the client. It talks to the server over HTTP (HyperText Transfer Protocol) using REST (Representational State Transfer) style requests [14] that exchange JSON (JavaScript Object Notation) data. The four layers are described below.

The *presentation layer* contains static HTML, CSS and JavaScript pages for login, registration, the dashboard, the three assessment rounds, the result view and the final report. The *application layer* is a FastAPI [15] server running on Uvicorn. It verifies the token of the user, routes each request to a module and returns a JSON response. The *data layer* uses SQLite [16] for users and assessment sessions, and JSON files for coding questions and interview prompts. The *AI layer* is optional. It holds the connection to the external providers OpenRouter, a Grok or OpenAI compatible service, and Gemini [20], and it sits next to the local fallback dataset. Only the aptitude module talks to this layer.

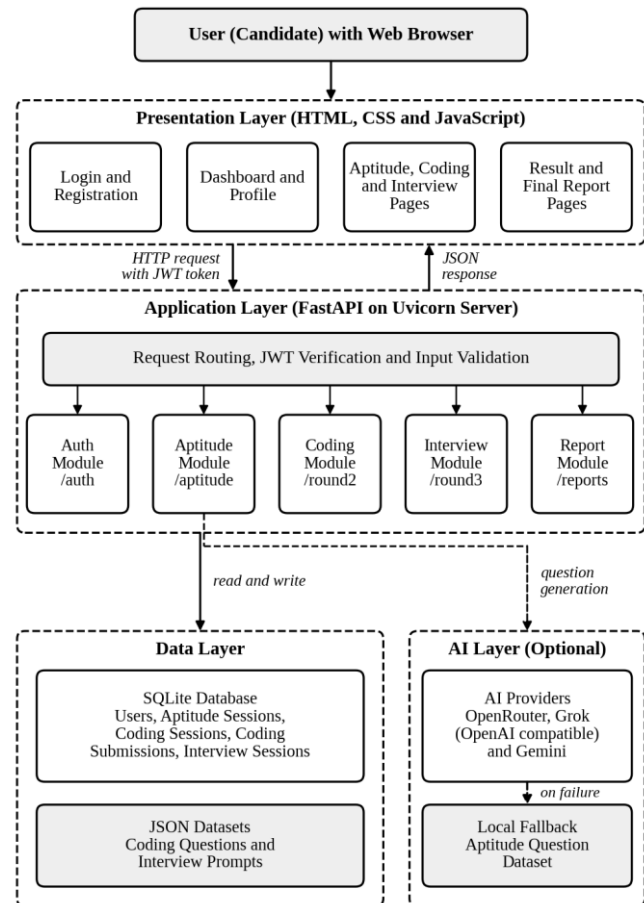


Fig. 1. Layered architecture of InterAI.

C. Module Description

The backend is split into six modules, as listed in Table I. Each one has a clear job and its own route group, so a change in one assessment round does not disturb the others. The main application file initializes the database, loads the configuration, registers the routers and serves the frontend pages.

TABLE I
BACKEND MODULES AND ROUTE GROUPS

Module (file)	Route	Responsibility
Authentication (auth.py)	/auth	Registration, login, token issue, profile update
Database (database.py)	None	Connection, schema creation, demo user
Aptitude (aptitude.py)	/aptitude	Fetch or generate questions, hide answers, grade
Coding (round2.py)	/round2	Session start, three questions, evaluation, storage

Interview (round3.py)	/round3	Stack specific prompts, response tracking
Reporting (reports.py)	/reports	Aggregate sessions, final report, advice

D. System Workflow

The workflow is shown in Fig. 2. The user registers or logs in, and a failed login is rejected and can be retried. After a valid login the server issues a token and opens the dashboard, where the technology stack stored in the profile is used later. In round one the aptitude module checks whether an AI provider is configured and whether generation succeeds. If it does, the generated questions are used. If not, the local dataset is loaded. The user answers, the server grades the answers and the session is stored. Round two selects three coding questions for the chosen stack, accepts the solutions, evaluates them and stores the submissions. Round three delivers interview prompts for the same stack and records the responses. Finally the reporting module reads the stored sessions and prepares the final report with recommendations.

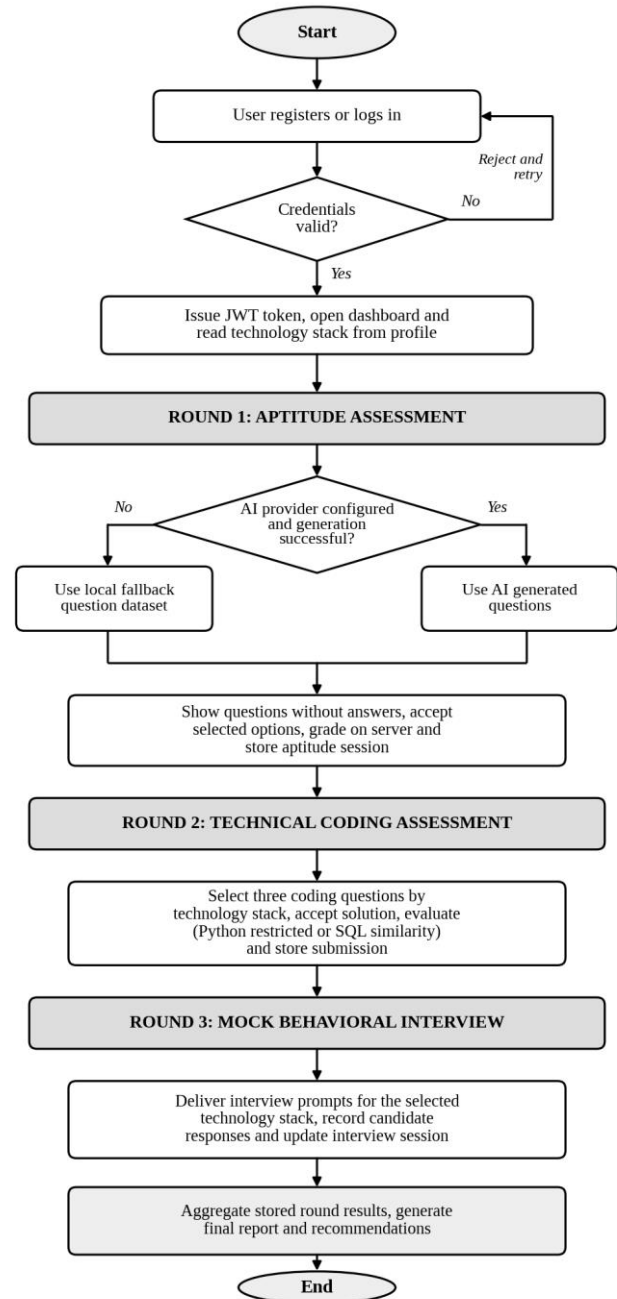


Fig. 2. Flow of the multi-round assessment process.

IV. IMPLEMENTATION

A. Technology and Environment

The platform was developed for local execution with Python 3.8 or later. Table II lists the main technologies. The configuration values and the keys of the AI providers are read from an environment file and are not written into the source code.

TABLE II
TECHNOLOGIES USED

Item	Technology and purpose
Language	Python 3.8 or later, backend logic
Framework	FastAPI with Uvicorn, REST routes and server
Frontend	HTML, CSS and JavaScript, static pages
Database	SQLite, users and assessment sessions
Security	PyJWT for tokens, Passlib for PBKDF2 hashing
AI providers	OpenRouter, Grok or OpenAI compatible, Gemini (optional)
Data files	JSON, coding questions, interview prompts, fallback questions
Testing	Python HTTP test scripts and manual checks

B. Authentication and Security

A new user gives a name, an email address, a password and a technology stack. The password is never stored as plain text. It is processed with PBKDF2 (Password-Based Key Derivation Function 2) using SHA-256 (Secure Hash Algorithm, 256 bit), which is described in [18], and only the hash is saved. On a successful login the server creates a JWT [17] that carries user related claims and an expiry time. Every protected route reads this token to find out which user is making the request. The server also keeps the correct answers of the aptitude questions on its own side and removes them from the response sent to the browser. These measures address two risk categories that are listed in the OWASP Top 10 [19], namely broken access control and cryptographic failures. Protection against network level attacks is outside the scope of this work, although earlier work in our department on machine learning based cyber attack detection [21] shows why web services need such protection.

C. Aptitude Round

Each aptitude question has an identifier, a category, a difficulty level, the question text, the options, the correct answer and an explanation. When a provider key is present in the environment, the module asks the provider to generate questions. If the call fails, or if no key is configured, the module reads the local fallback dataset instead, so the round always has questions to show. Generated questions may be cached for later use. When the user submits the answers, the server compares them with the stored answers, calculates the score and saves the aptitude session.

D. Coding Round

Coding questions are kept in a JSON file. A question has an identifier, a title, a language, a problem statement, starter code and test information. A coding session contains three questions that match the technology stack of the user. The submitted solution is evaluated by the coding module. Python answers are run with a restricted set of built-in functions, and SQL answers are judged by similarity matching with the expected query. The session progress and every submission are saved. This evaluation is a simple demonstration mechanism. It is not a secure sandbox for running untrusted code, and we do not present it as one.

E. Mock Interview Round

The interview module holds technology specific prompts. Each prompt has an identifier, a category, a title, the question text, a list of expected keywords and an example answer. The module sends the prompts for the stack of the user, accepts the

typed responses and updates the state of the interview session. This round is prompt based. It does not analyze speech, facial expression or gaze in the present version.

F. Database and Reporting

The database has five tables, which are users, aptitude sessions, coding sessions, coding submissions and interview sessions. A demonstration user is added when the database is created for the first time. The reporting module is reached through the /reports/final route. It reads the stored sessions of the logged-in user, joins the results of the three rounds and returns a summary with suggestions for improvement, which the final report page then displays.

V. TESTING AND RESULTS

A. Test Setup

The application was run locally and tested with unit level checks of the backend functions, integration tests along the paths frontend to backend to database and assessment module to database to result, functional tests with valid and invalid inputs, and basic usability checks of navigation. The HTTP interface was exercised with Python test scripts and with manual use of the browser pages. In total fourteen functional test cases were prepared and are grouped in Table III.

TABLE III
SUMMARY OF FUNCTIONAL TEST CASES

Group	Cases	Scenarios checked	Passed
Authentication	TC01–05	Valid registration, duplicate email rejected, valid login, wrong password rejected, protected feature without login	5 of 5
Aptitude	TC06–07	Questions displayed, answers evaluated and score produced	2 of 2
Coding	TC08–09	Coding session started, solution submitted and evaluated	2 of 2
Interview	TC10–11	Question displayed, response recorded	2 of 2
End to end and storage	TC12–14	All rounds completed, stored session retrieved, final report displayed	3 of 3

B. Results

All fourteen test cases gave the expected result. Registration, login and token based access worked as designed, and invalid credentials as well as unauthenticated access were refused. The aptitude, coding and interview rounds could be started, answered and stored, and the records could be read back when the report was requested. The aptitude round continued to serve questions from the local dataset when AI generation was not available. During local use the common operations such as login, question retrieval, answer submission and result retrieval completed without a noticeable delay. We did not record timing measurements, so we make no numerical claim about speed.

Table IV compares the main features of InterAI with the usual practice of using separate platforms. The comparison is qualitative and is based on the design of the system, not on a controlled user study.

TABLE IV
QUALITATIVE FEATURE COMPARISON

Feature	Separate platforms	InterAI
Single login for all rounds	Not usual	Yes
Aptitude, coding and interview together	No	Yes
AI assisted question generation	Varies	Yes, optional
Works without AI service	Not applicable	Yes, local fallback
Content by technology stack	Varies	Yes
One consolidated report	No	Yes

Against the objectives set at the start, the work provides user registration and authentication, an aptitude round with optional AI generation and local fallback, a coding round, a mock interview round, persistent session storage in SQLite and a consolidated final report.

VI. DISCUSSION AND LIMITATIONS

The three-round order moves from general reasoning to technical problem solving and then to communication, which is the order in which many recruitment processes are arranged. Keeping the three rounds under one account has a practical benefit, since the user does not have to collect scores from different places. The fallback design makes the application usable in a laboratory with limited internet access, which is a common situation on college machines.

The work also has clear limits. First, the coding evaluation is basic and is not a secure execution service. Second, SQLite suits a local prototype but would have to be replaced for many simultaneous users. Third, AI generated questions depend on the availability and configuration of the providers, and the generated items were not checked by a formal review process in this study. Fourth, the interview round does not score the quality of an answer automatically. It stores the response and keeps the expected keywords and the example answer for reference. Fifth, we have not carried out a user study or measured the learning benefit, so the results here show that the system works as specified and do not prove that it improves interview outcomes. Because of the fairness concerns discussed in [4] and [5], the platform should be used for practice and not as an employment decision tool.

VII. CONCLUSION AND FUTURE WORK

We have described InterAI, a web platform that joins aptitude assessment, technical coding assessment and mock interview practice into one workflow with secure login, persistent sessions and a final report. The optional AI generated aptitude questions together with a local fallback dataset keep the application usable when external services fail. The functional testing showed that all fourteen planned test cases passed in the local environment.

Several extensions are planned. A language model can be used to evaluate typed interview answers and give personal feedback. Speech input and output can be added for spoken interviews, using work such as [12] as a starting point. The coding round needs a proper sandbox with many test cases and more programming languages. A dashboard of scores over repeated attempts would show progress, and the stored history

can drive topic recommendations for each learner. A study with student users would then measure the real benefit of the platform.

ACKNOWLEDGMENT

The author thanks the final-year students of the Department of Computer Science and Engineering who carried out the project work, and the management of SJM Institute of Technology, Chitradurga, for the facilities provided.

REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.
- [2] T. Brown *et al.*, "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.
- [3] E. Kasneci *et al.*, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learn. Individ. Differ.*, vol. 103, Art. no. 102274, 2023.
- [4] M. Köchling and M. C. Wehner, "Discriminated by an algorithm: A systematic review of discrimination and fairness by algorithmic decision-making in the context of HR recruitment and HR development," *Bus. Res.*, vol. 13, no. 3, pp. 795–848, 2020.
- [5] M. Langer, C. J. König, and M. Papathanasiou, "Highly automated job interviews: Acceptance under the influence of stakes," *Int. J. Sel. Assess.*, vol. 27, no. 3, pp. 217–234, 2019.
- [6] B. E. Ramesh, "AI-powered resume analyzer and interview preparation system," *Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET)*, 2025.
- [7] S. Elkins, E. Kochmar, I. Serban, and J. C. K. Cheung, "How useful are educational questions generated by large language models?," arXiv:2304.06638, 2023.
- [8] S. Bulathwela, H. Muse, and E. Yilmaz, "Scalable educational question generation with pre-trained language models," arXiv:2305.07871, 2023.
- [9] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proc. ACM Conf. Int. Comput. Educ. Res. (ICER)*, 2022, pp. 27–43.
- [10] J. Savelka, A. Agarwal, C. Bogart, Y. Song, and M. Sakr, "Can generative pre-trained transformers (GPT) pass assessments in higher education programming courses?," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2023, pp. 117–123.
- [11] P. Orvalho, M. Janota, and V. Manquinho, "GitSEED: A Git-backed automated assessment tool for software engineering and programming education," arXiv:2409.07362, 2024.
- [12] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 202, 2023, pp. 28492–28518.
- [13] C. Lugaresi *et al.*, "MediaPipe: A framework for building perception pipelines," arXiv:1906.08172, 2019.
- [14] 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.
- [15] S. Ramírez, "FastAPI documentation." [Online]. Available: <https://fastapi.tiangolo.com>

- [16]SQLite Consortium, "SQLite documentation." [Online]. Available: <https://www.sqlite.org/docs.html>
- [17]M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, May 2015.
- [18]K. Moriarty, B. Kaliski, and A. Rusch, "PKCS #5: Password-based cryptography specification version 2.1," IETF, RFC 8018, Jan. 2017.
- [19]OWASP Foundation, "OWASP Top 10: The ten most critical web application security risks." [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [20]Gemini Team, Google, "Gemini: A family of highly capable multimodal models," arXiv:2312.11805, 2023.
- [21]B. Arpitha, R. Sharan, B. M. Brunda, D. M. Indrakumar, and B. E. Ramesh, "Cyber attack detection and notifying system using ML techniques," *Int. J. Eng. Sci. Comput. (IJESC)*, 2021.