WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

InternAI: Learn. Grow. Get Hired.

Diyansh Jain, Aadarsh Sharma, Yuvraj Gandhi, Shoaib Shaikh, Prof. Dhanashree Tamhane*

Department of Information Technology, Vidyalankar Institute of Technology, Mumbai, India
**Under the Guidance, Department of Information Technology, Vidyalankar Institute of Technology*

ABSTRACT

–Campus placement and internship recruitment at most engineering institutes still depend on manually shared spreadsheets, general job-portal listings, and resumes that are screened by hand before a recruiter even reviews them. This creates two connected problems: students get little practical feedback on how well their resume or skills fit a particular role, while recruiters spend a large amount of time filtering applications that do not meet basic requirements. InternAI is a web-based internship and placement platform developed to address this gap. The system uses a Next.js (React and TypeScript) frontend, a Node.js and Express backend organized as a modular monolith, and a MongoDB Atlas document store, connected through a JSON Web Token based authentication layer. InternAI provides students with resume upload and parsing, an integrated Applicant Tracking System (ATS) style scoring module, a skill-gap analysis tool, and a roadmap generator that maps their current profile to the requirements of a target role. On the recruiter side, the platform provides a dashboard for candidate matching, a structured hiring pipeline, and analytics for applicant flow. This paper presents the system architecture, the matching and scoring methodology, the security and accessibility measures, and the results of automated testing, a layout check across three browser engines, and a cross-validated evaluation of the resume classifier, along with the limits of that evidence.

Index Terms–internship matching, resume screening, applicant tracking system, skill gap analysis, placement platform, recommendation engine, explainable scoring, web application security.

I. INTRODUCTION

Placement cells at most institutes manage internship and placement drives through a combination of email threads, spreadsheets, and third-party job boards that are not designed around a college's specific requirements. A student looking for an internship usually has no reliable way to see how closely their resume fits a posting before applying, while a recruiter has no automated way to rank submissions beyond their arrival order. Since students can't accurately assess their fit with a company in advance, the application process creates an inefficient system for students and recruiters. Students apply broadly, and recruiters spend hours sorting and screening resumes, a task that research on talent analytics has long tried to automate [1].

The issue is compounded by the high volume of applications. A single internship posting for a mid-sized institute can draw hundreds of applications (our estimate), the majority of which are broad, unstructured PDFs and scanned image resumes with random, varying fonts and order of sections. Recruiters can spend hours screening resumes for required keywords and formats, which in practice comes down to checking a small set of keywords. As a result, a qualified student can be filtered out for cosmetic reasons – such as a missing section heading or an unconventional resume layout – almost as easily as for an actual skill mismatch.

The problem is not specific to one institute. The India Skills Report 2025 puts overall graduate employability at 54.81 percent, against 51.25 percent a year earlier [2], so a large share of candidates still reach employers short of what a role asks for. The student, who is the person most affected by a screening score, rarely gets to see it. That is the part InternAI tries to change.

InternAI was developed to address this specific gap. Instead of acting as a general job board, the platform follows the internship hiring workflow for students, companies and administrators: a student profile carries resume, skill, and academic information through the process, while the recruiter workflow allows a company to post an opening, receive ranked candidates, and move them through a defined hiring pipeline without leaving the platform. Students also receive direct and actionable feedback through an ATS-style resume score, a list of missing or weak skills for a selected role, and a generated roadmap showing which skills to work on next, rather than having to guess why an application was rejected.

This paper makes four contributions. It describes a modular architecture in which students, companies and administrators share one scoring and ranking core. It gives the scoring method in full, with every weight written out, so that a score can be traced back to its parts. It adds a cross-check that lowers the reported confidence when two independent estimates of the same quantity disagree. And it reports what the automated tests, the layout checks and a cross-validated evaluation of the resume classifier do and do not show, including the places where the evidence is weak.

The rest of this paper is organized as follows. Section II reviews existing internship and placement tools and the gap addressed by this work. Section III describes the system architecture, data model, and platform modules. Section IV explains the matching, scoring and classification methods. Section V discusses implementation details, including the frontend structure, authentication, security, real-time features, the language-model gateway, accessibility, and testing strategy. Section VI reports the test, layout and classifier results and the limits of the evidence. Finally, Section VII concludes the paper and describes planned future work.

II. RELATED WORK

A. General and Internship-Focused Job Portals

General job portals such as LinkedIn and Naukri serve the open market. They list openings from many employers and let anyone apply, which is not how an institute's internship season works: a closed group of students deals with a closed group of recruiters over a few months, and the college wants to see what happens to each of them. Internshala is closer to the need, since it concentrates on internships and entry-level jobs [3], but it is still mainly a listing service. Students search and apply, and they generally get no signal about how well they fit before they do.

B. Applicant Tracking Systems and Resume Parsing

From the recruiter side, Applicant Tracking Systems (ATS) automate many aspects of a recruiter's job. Production versions of ATS focus on parsing and ranking incoming resumes. They provide little to no information to applicants. Meanwhile, academic research examining resume-to-job matching has largely focused on information retrieval and classification, by defining a similarity metric using a job's description keywords [4], [5]. Generally, these methods can efficiently rank a large pool of applicants, but do not tend to incorporate a mechanism which allows applicants to view and impact the score prior to submitting an application.

Person-job fit has also been treated as a learning problem. Zhu et al. [6] learn a joint representation of resumes and postings from past applications, and Qin et al. [7] add an ability-aware network that models the separate abilities a posting asks for. Surveys of job recommender systems [8], [9] cover content-based, collaborative and hybrid approaches. Learned and collaborative methods need logs of earlier applications or hires, which a new platform at a single institute does not have, so InternAI starts from explicit rules and keeps them readable.

Resume reading is the first step. Retyk et al. [10] have considered resume parsing to be a sequence labeling problem of lines as well as tokens, and compared different models for this purpose. InternAI takes a plainer route: lists of common section headings, pattern matching for contact information, and an alias table for skills. This approach is much simpler to inspect and correct, although it is less adaptable to peculiar formats, which is why every parse has a confidence factor (Section IV-A).

C. Fairness, Explainability and Language Models

The use of automated tools for recruiting presents issues that cannot be overlooked in a product targeting students. In particular, Raghavan et al. [11] investigated the vendors' claims of decreasing the bias of algorithmic recruiting and noticed the lack of publicly available information about testing such claims. Black and van Esch [12] review the hiring processes where AI is currently employed and the list of criteria that need to be verified before its use. Rudin [13] notes that for high stakes decisions, an interpretable model is preferable to a black box followed by an explanation.

This line of thought has influenced one of the decisions made for InternAI: the scoring algorithm of the CV takes into account seven named dimensions, so a student or a recruiter can see exactly which dimension was the weakest and why. The large language models are applied only at the periphery for parsing, application insight and portfolio analysis, and always with a local alternative. Large language models can generate fluent text unsupported by the input data [14], so the score is calculated by an explicitly stated set of rules and a model never generates the score number.

D. Positioning of InternAI

As illustrated in Table I, InternAI falls into both groups. It is not a generalized marketplace platform nor a recruitment-specific screening tool. From a student perspective, InternAI is a skill-gap analyzer and recommender. From a recruiter perspective, despite the fact that candidate applications remain unstructured, InternAI allows a framework wherein candidates are forwarded in a structured fashion. This work tries to fill the gap outlined above with a two-pronged strategy involving offering structured feedback to students as well as structured candidate review to recruiters.

The table comparison in Table I is intentionally broad. It refers to what is typically offered by each type of service rather than a feature list of a particular vendor whose features change frequently. The only two columns relevant to this study are whether the applicant receives any feedback before applying and what the recruiter can do with the tool.

Platform	Scope	Feedback to Applicant	Recruiter-Side Workflow
LinkedIn / Naukri	Open job market	Not typical	Search and apply
Internshala	Internship listings	Not typical	Listing plus applicant inbox
Standalone ATS	Enterprise hiring	Not typical	Automated resume ranking only
InternAI	Internship hiring (students, companies, administrators)	Resume score, skill gap, roadmap	Ranked candidates, structured pipeline

TABLE I. COMPARATIVE POSITIONING OF EXISTING PLATFORMS

III. SYSTEM ARCHITECTURE

A. Overview

InternAI adopts a three-level architecture. The presentation level comprises a Next.js 15 application coded using React 19, TypeScript, and Tailwind CSS [15]. The application level consists of an Express 4 service hosted on Node.js 20 and consisting of 18 modules per domain and a core layer and provides a REST API [16] to the frontend layer. The database level employs a MongoDB Atlas cluster [17] since the system needs to store semi-structured data, such as resumes, skillsets, and hiring pipeline stage data, which do not naturally fit in a relational database schema. JSON Web Tokens (JWT) [18] serve as the authentication and session management mechanism in the system since JWTs are issued during login and validated at every protected route in the backend level via middleware. These middleware functions also perform role-based authorization between the three user types (student, recruiter, administrator).

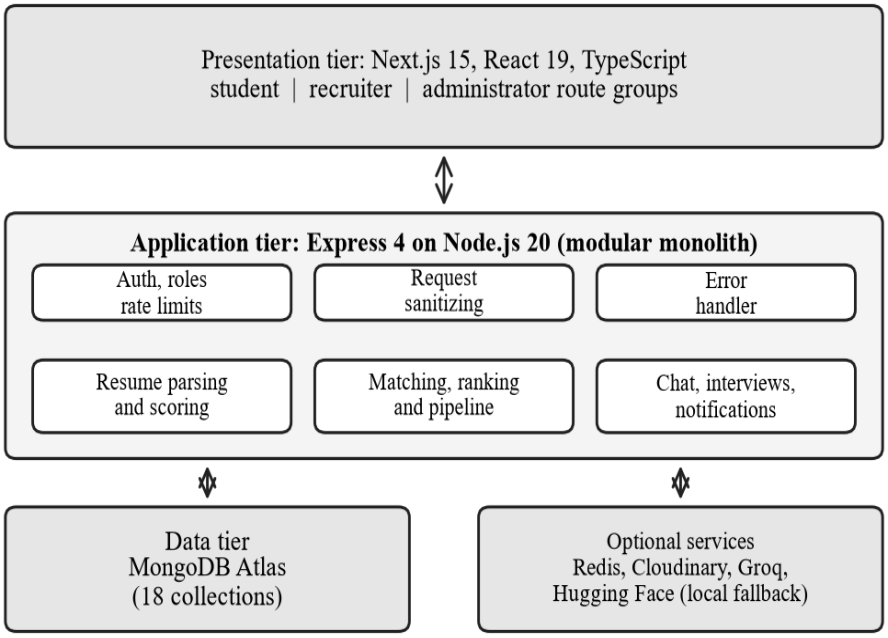


Fig. 1. Three tiers of InternAI and its optional services.

Application tier is the monolithic modular design, which refers to a single deployable application that consists of separate modules for the specific domain., each with its own routes, controllers, services and models. Splitting every domain into its own service would add network calls, separate deployments and distributed failure modes [19], and a team of four with one database and one hosting target gets little back for that cost. Starting with a well-factored monolith also keeps the option of splitting a module out later [20].

B. Data Model

A separate MongoDB collection contains each individual core platform entity [21]. These include users (student, company and administrator accounts, distinguished by a role field), intern profiles (academic record, skill list, and parsed resume), internships (a listing with its required and priority skills), and applications (the link between a student and an internship, including the current pipeline stage). Further collections hold chat conversations and messages, and there are 18 collections in all. Applications refer to both an intern profile and an internship by identifier instead of embedding either document. This is because

the new score or match will be generated with the latest information and not the need for re-writing all the application details. In the case of MongoDB, the common problem faced by people while designing the model is that of embedding and referencing [21]. Since data which is read together and changes together is embedded, the skills and parsed resume is embedded into the intern profile. Records that grow over time or are queried on their own, such as applications, conversations and messages, are separate collections.

C. Module Description

Table II summarizes the platform's functional modules and the role each one plays in the overall workflow.

Module	Function
Student Dashboard	Unified view of resume status, skill-gap summary, recommended internships and application history.
Recruiter Dashboard	Company-side view of postings, incoming applications and pipeline stage per candidate.
Resume Upload and Parsing	Reads a PDF, DOCX or TXT resume into structured fields and reports how reliable the parse was.
Resume Score / ATS Analysis	Scores a resume against a target role and flags weak or missing sections.
Skill Gap Analysis	Compares recorded skills against a role's skill profile and returns the delta.
Roadmap Generator	Orders the skill gap into a sequence of topics to acquire next.
Internship Matching	Ranks open internships against a student's skills using required and preferred skill match (80:20).
Hiring Pipeline	Recruiter-facing stage tracker: applied, under review, shortlisted, interview scheduled, technical round, HR round, selected or rejected (with on hold).
Messaging	Real-time chat between a student and a company.
Analytics and Reports	Shows applications per posting, stage conversion and recruiter performance.

TABLE II. PLATFORM MODULES

The student- and recruiter-facing modules use the same underlying data layer. Skill and role information taken from the parsed resume, the student's profile and the recruiter's posting form feeds the same scoring and ranking logic used by the Resume Score, Skill Gap Analysis, Roadmap Generator, and Internship Matching modules. As a result, a change to a student's profile is reflected consistently across every module that depends on that information.

D. Request Flow

The process followed for a request (for example, for a student to update his/her resume score) is the same for each module. The frontend makes an authenticated request with the JWT [18] that was issued to the student at login. Middleware in the backend layer validates the token and attaches the resolved user and role to the request context. The target router then loads the student's existing profile and the relevant posting/role definition from MongoDB. The scoring function described in Section IV is used to compute the result. The reply is in the format described in Section V-A and is delivered back to the client-side. The same path is followed by all the modules. So, for adding new scored and ranked features, we do not need to change the authentication and formatting layers of the response.

E. Technology Justification

Each layer of the stack was selected to meet a particular requirement rather than simply being used by default. React with TypeScript was preferred over a plain JavaScript frontend so that the shape of each API response is typed on the client [15]. This will help prevent a misnamed field from being found by the browser. The Express framework on Node.js [22] is used to ensure that all the technologies are consistent in their language. In addition, the middleware layer of the Express framework will help to logically sequence the security steps (header check, CORS check, sanitization, rate limiting, and authentication). The Node.js I/O system [23], which does not block, is also suitable for the matching and analytics endpoints that have to make several MongoDB requests per request. The use of MongoDB Atlas [17] is better than a relational database as the core document (a resume, skill list of a posting, or chat logs) can be inconsistent between different users.

In addition to the functionality shown in Table II, the system was designed with a small number of non-functional goals [24]. The time of computing of scores and matching endpoints were guaranteed to be independent of total size of the applicant pool since they were constrained to process just one role-student pair instead of recomputing over all applicants for the particular position. This guarantees constant time of processing a request for computing resume score regardless of how many students had already applied for the position. The horizontal scaling of the data layer is handled via MongoDB Atlas service [17] where the size of the cluster can be resized without changing the application code. Role-based access control system discussed in Section III-A provides also an availability feature of the platform: the malfunction in the pipeline for recruiter analytics cannot prevent the student from submitting his/her resume since each request is processed independently.

IV. MATCHING AND SCORING METHODOLOGY

A. Skill Representation

The skills of the candidate and the required/ preferred skills in the posting are considered to be lists. The parsed skills are matched against an alias list in order to ensure normalization of skills such as "JS" and "JavaScript" which should be considered as the same skill. The parser recognizes eight different section types: summary, skills, projects, education, experience, certifications, achievements and additional information. The parser identifies the above sections through list of common names used for each section and detects the email address, phone number and link using regular expression matching. A quality check is made on the parsing result for six criteria - no email address, no phone number, less than 3 skills, no projects, no education entries and no summary or summary too short. No issues will give 0.95 parse confidence, one or two issues gives 0.82, three or four issues gives 0.68 and five or six issues gives 0.55.

B. Scoring Function

Equation 1 is used to calculate the resume score of a student for a job position:

$$S = \Sigma\,(wi \times di),\ i = 1 \text{ to } 7 \quad (1)$$

where di represents the score (0 to 100) of dimension i and wi its weight: keyword match 0.35, technical skills 0.15, projects and achievements 0.15, experience 0.10, education 0.10, structure and formatting 0.10, and certifications 0.05. In the case that the parsing confidence c is less than 0.75, the score is multiplied by c. When matching internships, the required skill match is R = (number of matched required skills/total number of required skills) × 100, while the overall match is O = 0.8R + 0.2P, where P is the preferred skill match.

Dimension	Weight	What is checked
Keyword match	0.35	Required and preferred skill keywords found in the resume text, weighted 80:20
Technical skills	0.15	Matched skills that are backed by projects or certifications
Projects and achievements	0.15	Relevance of projects (75 percent) and of achievements (25 percent)
Experience	0.10	Experience entries and how well they fit the role
Education	0.10	Education entries and how well they fit the role
Structure and formatting	0.10	Section coverage, formatting, readability and length
Certifications	0.05	Certifications relevant to the target role; unrelated ones are listed but add nothing

TABLE III. SCORING DIMENSIONS AND WEIGHTS

Table III provides the dimensions. The weights are rationalized, not estimated: keyword match gets the highest weight as it reflects clearly what the posting needs, while certifications get the lowest weight as they are extras, not requirements. Each dimension also offers a description and list of recommendations in plain language, so that any deduction is linked to a cause. This is in line with the argument in favor of transparent scoring in high stakes situations [13], though the weights themselves have not been validated yet. Keyword matching in the first dimension is an extension of traditional term-based retrieval [25]. Figure 2 illustrates the entire process of information flow.

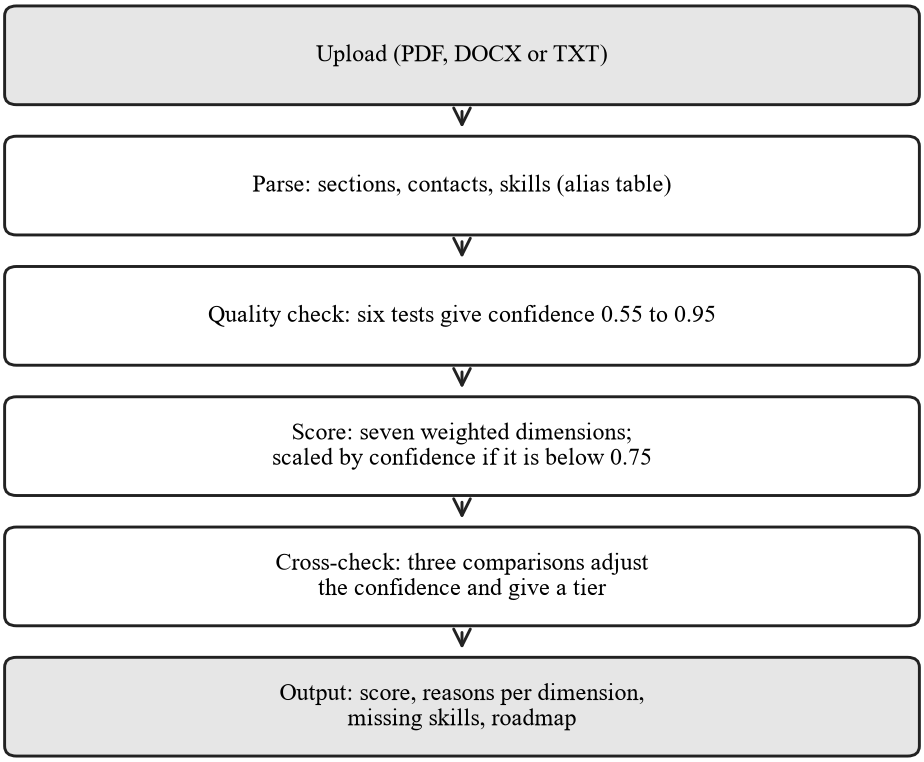


Fig. 2. Resume scoring flow from upload to output.

C. Roadmap Ordering

The roadmap generation component considers the set of skills that are lacking from the student for the targeted job, arranges them according to their positions in the skill set of the job. The priority will depend on the position, skills number 1 to 3 will be considered high priority, number 4 to 6 medium and the rest low.

D. Worked Example

Now consider the resume evaluated using the scoring engine twice. It scores 84 for a general Full Stack Engineer position. But when evaluated against an actual posting for the Backend Developer Intern job where the requisite skills include Node.js, Express, MongoDB, PostgreSQL, Git, Docker, SQL, and API design, it scores 74 since it doesn't have the last two skills mentioned above.

E. Ranking Applicants for a Posting

As soon as an internship opportunity is posted by the company, the backend will order the candidates based on the weighted average of three numbers: 0.55 multiplied by the required skill match, 0.20 by the resume rating, and 0.25 by a project rating. The required skill has the highest weighting while the latter two allow for a candidate with a lower match score to get some additional advantage from their strong resume and projects. For students, recommended internships are ordered based on the match O.

F. Cross-Checking the Score

One score alone can be wrong without it being detected, so there will be multiple independent estimations of the same. There are three pairs of comparison. The score of the resume is compared to the sum of recruiter's readiness, project's strength, technical strength, and the resume's ATS compatibility. Additionally, it is compared to the parsing result itself, and the number of skills parsed from the resume is compared to the number of those recognized during scoring or identified as skill gaps. The tolerance values are 10, 12, and 8 points. In case the difference between the two

numbers is less than 3 points, the confidence is increased by 10 points. In case the difference is greater than tolerance, the confidence is decreased by the size of the difference, while in case the difference is larger than twice the tolerance, it is decreased by twice the difference with the maximum being 35 points, with the report of a conflict if the confidence is lowered below 50. The resulting confidence is then converted to a level displayed together with the score.

G. Optional Resume Classifier

In addition to the rule-based scoring algorithm, the repository contains an optional classifier for predicting which out of 20 job categories the resume belongs to. It converts resumes to TF-IDF representation [26], [25], where the weight of each word is the frequency of its occurrence in the document multiplied by how infrequent the word is overall, and then uses it for classification via one-vs-rest k-nearest neighbors [27] implemented using scikit-learn [28]. This classifier is not required for operation of the platform.

V. IMPLEMENTATION

A. Backend and Data Layer

The backend is modularized into 18 modules for each functionality (authentication, students, companies, internships, applications, interviews, chats, etc.) using Express. These are mounted at /api and /api/v1. All modules give out responses in the same response envelope using a single helper function, and all errors are returned in the form of a single AppError and dealt with by a single error handling middleware.

All cross-cutting concerns are encapsulated in a middleware. Security headers are supplied using Helmet, cross origin requests are restricted to the allowed origins, body parsing and sanitization is done and there is an overall limit of 400 API requests per client for 15 minutes. The limits for login/signup, one time password creation and verification are set to 30, 8 and 20 respectively.

B. Frontend Architecture

The front end is structured using role-based routing groups [15]: student, recruiter, and administrator. The routes load independently from each other, which means that when an account logs in, it only loads the code it needs. UI atoms like forms, pipeline board, and chart elements in the analytics view are kept in a common components library, which is used across all three routing groups. This allows for element consistency, for instance, form controls in the resume view or any other view.

C. Security

Sanitization occurs prior to execution in the API. In the case of the NoSQL sanitizer that strips query operators out of request data in order to prevent NoSQL injection [29], it must come after the body parser: initially mounted before the body parsers, the module saw no body and thus did not strip anything, but moved after the parser corrected the flaw. As for uploads, file names are based only on the base name, whitelisting safe characters and extensions, while files can be uploaded only to a predefined set of folders, thus preventing path traversal (OWASP Top 10 [30]). Input handling complies with standard database security practices [31].

Token-based authentication is performed according to standard tokenization procedure. Passwords are hashed with bcrypt using cost factor of 12 [32], making every attempt to guess password costly on purpose. Access and refresh tokens are JSON Web Tokens [18], and the latter is stored in the database only as SHA-256 hash, which means that any database backup will not give anyone valid sessions. Access token is delivered in Bearer header, while there is no way for the server to authenticate using cookies, eliminating the possibility of forgery of request with the help of cookies, though sacrificing token storage in client browser. Role checking is performed using 25 permission keys, which allows restricting access to certain screens for employees and administrators.

D. Accessibility

Accessibility improvements were implemented in a targeted fashion, based on WCAG 2.1 [33]. Dialogs trap and return focus on dialog close, tabs react to keyboard navigation, active page within navigation menu has aria-current applied, form errors are associated with corresponding form elements, and muted text colors were changed to comply with a 4.5:1 contrast ratio. Some problems are still unresolved, such as missing heading level in the admin dashboard.

E. Real-Time Communication

The Chat application uses Socket.IO, which establishes WebSocket connections [34] whenever possible. The video interview is a peer-to-peer WebRTC session [35] with the backend being the signaling mechanism, using the protocol stack described in the RFC 8825 [36]. All peers connect to all others, a mesh topology that is easy to implement and doesn't require a media server, although it is suitable only for small conferences. No TURN server is used, hence the call may fail if both users are located behind symmetric NAT.

F. Language-Model Gateway

Among them are parsing assistance, application knowledge and portfolio text. The gateway routes each call to Groq, which provides Llama 3.1 8B Instant by default [37], then to Hugging Face, which provides Mistral 7B Instruct [38], and follows the local rules in case there is no key provided, in case AI is disabled or in case the providers do not respond. It is also possible to configure the gateway to try following the local rules first. The responses are cached and each user has a limit of calls per minute. Due to the fact that text generated by models can include claims not covered by the prompt [14], model output is considered to be suggestions only.

G. Testing Strategy

There are 139 tests on the backend side (65 unit tests, 63 integration tests and 11 regression tests), while there are 97 tests on the frontend side, and all of them succeeded. Tests were conducted according to standard unit, integration and regression testing methodology [24]. Testing includes login, role and ownership verification, file uploading policy, the hiring pipeline rules, skills matching and the calendar. There are browser level tests in the repository, but they have not been performed for the purposes of this paper.

H. Deployment and Configuration

The front-end and back-end are built and deployed as separate artifacts. The front-end artifact is built as a Next.js app for Netlify or Vercel, while the back-end artifact is deployed as a standalone Express service with the render.yaml file defining it. Values specific to each environment, such as the MongoDB Atlas connection string, JWT sign-in secret, and origins allowed for cross-origin request handling, are injected into the build process via environment variables instead of being checked in with the source code. In this way, the same artifact can be used to connect to a development, staging, or production database without changes to the code base. Keeping the two artifacts separate also provides the flexibility of redeploying only the front-end artifact in case of changes in styling or workflow.

I. Challenges and Design Trade-offs

There were two design choices that have been revisited in development. First of all, it was the choice of reading resumes. The system accepts a PDF, DOCX or text file uploaded by students, then tries to parse it. However, parsing may fail if the resume layout is peculiar in some way. Thus, every parsing attempt gets assigned some confidence score, and having a bad confidence score lowers the final score. Second, the question was whether student accounts and recruiter accounts would use one users collection or different collections. The single users collection split by roles, as it has been stated in Section III-B, has been chosen in order not to duplicate the code related to authentication and role-based access control.

VI. RESULTS AND DISCUSSION

A. Tests, Reviews and Manual Checks

This platform has been tested through three methods. Firstly, automatic unit and integration testing has been introduced to test the API functionality and the logic behind the matching process. Second, manual functional testing was performed to evaluate each of the modules defined in Section III, with respect to their functionality, starting from student registration through a recruiter moving a candidate in the hiring process. Last, the interface was reviewed manually for accessibility, and its layout was checked in Chromium, Firefox and WebKit at widths of 375, 768, 1366 and 1920 pixels on seven key pages, with no horizontal overflow or script errors. Table IV summarizes the outcome.

Module / Area	**Validation Method**	**Outcome**
Authentication and sessions	Integration and unit tests (33)	33 of 33 passed
Role and ownership checks	Integration tests (14)	14 of 14 passed
File-upload rules	Integration tests (17)	17 of 17 passed
Hiring pipeline rules	Integration and unit tests (15)	15 of 15 passed
Skill normalization and matching	Unit tests (27)	27 of 27 passed
Application relevance	Unit tests (7)	7 of 7 passed

Module / Area	Validation Method	Outcome
Calendar	Integration tests (12)	12 of 12 passed
Frontend logic and components	Vitest (97)	97 of 97 passed
Upload path traversal and sanitizer order	Direct reproduction, then fixed	Fixed

TABLE IV. VALIDATION SUMMARY

The modules presented in Table II were tested using the automated testing performed in Table IV and manual functional testing. A student can upload a resume, get a score and a skill-gap breakdown, and see recommended internships. A recruiter can see applicants in ranked order and move them through the pipeline stages. Errors now share one format, so the frontend can handle them in one place.

In terms of the security review, the file upload path traversal vulnerability and an order problem within the request-sanitization middleware were detected and fixed. Replication confirmed the resolution. As for the accessibility review, fixes related to keyboard usage, focus management and contrast of text were identified and applied.

The most significant qualitative improvements in comparison with the manual, spreadsheet-and-email approach from Section I are observed in those places in the workflow where no feedback was provided before. Namely, students get their resume score and a prioritized list of skill gaps prior to applying rather than getting those in the rejection email. At the same time, recruiters get a ranked list of candidates to be considered with tracking in the recruiting pipeline rather than an unranked inbox list that needs manual sorting. Those benefits are not evaluated against the live placement drive in this project since such comparison would involve implementation of the platform together with the institute's placement procedure during the whole recruitment period.

The key disadvantage is that the weighting factor for the score is derived manually, rather than through an algorithm based on the data on outcomes, since the latter does not yet exist. The ranking will therefore be less flexible to account for certain patterns that are not accounted for in the predetermined weightings. The matter is further elaborated in relation to future work in Section VII. InternAI is currently unable to analyze image resumes due to the lack of an OCR function.

B. Layout Check Across Browsers

The layout check tested three browser engines (Chromium, Firefox and WebKit) at four widths (375, 768, 1366 and 1920 pixels) on seven pages: the landing page, the sign-in page, the student dashboard, the resume score page, the company dashboard, the hiring pipeline and the administrator overview. There were 84 combinations. In each combination, a script logged in using a seeded demo account, waited for the page to load, checked whether the document was wider than the viewport and whether an element overflown the right edge outside a scrollable container like the pipeline board. Script errors were also counted. None of the pages overflown and none of them generated script errors. The test tells nothing about user-friendliness of the layouts. Narrow widths were tested by changing the viewport size in desktop browsers, not using a mobile device.

C. Resume Classifier Evaluation

Of all components within InternAI, it is only the optional classifier that can be tested by external labeled data. For that we took a public resume collection from Kaggle [39]: 962 rows spread over 25 categories. A first look showed that just 166 of those texts differ from one another, so splitting the rows at random would leave copies of the same resume in both training and test sets and flatter every result. We deleted the duplicates before continuing any further. There are 20 out of 25 classes that the classifier recognizes; if we use only those, we have 806 entries, which boil down to 135 unique resumes, having from 4 to 13 each class.

We posed two distinct questions. The first one was how the shipped model files react to this data set. Nothing is known about their training data, so the result is not a proper held-out test, and in our runs the files loaded only under scikit-learn 1.4.2, while versions 1.5 to 1.7 failed. On all 806 rows they got 74.9 percent right, and on the 135 distinct resumes 71.9 percent. The second question was what the same design can do when it is trained and tested on different resumes. We kept the shipped TF-IDF settings, trained a one-vs-rest k-nearest-neighbor classifier, and ran four-fold cross-validation five times over the 135 distinct resumes. Accuracy came to 80.7 percent (standard deviation 5.1) with a macro F1 of 75.2 percent. As a yardstick we also trained a linear support vector machine on identical features, which scored 85.6 and 81.8, while a rule that always names the most frequent category scored 8.2 percent accuracy. Table V lists the results.

Method	Accuracy (%)	Macro F1 (%)
Most common category (CV)	8.2 ± 1.3	0.8 ± 0.1
Retrained TF-IDF + k-NN (CV)	80.7 ± 5.1	75.2 ± 6.6
Retrained TF-IDF + linear SVM (CV)	85.6 ± 4.2	81.8 ± 5.4
Shipped model, 135 distinct resumes (not held out)	71.9	67.3
Shipped model, all 806 rows (not held out)	74.9	69.5

TABLE V. RESUME CLASSIFIER RESULTS (20 CATEGORIES, 135 DISTINCT RESUMES)

Read these figures with care. Only 135 resumes stand behind them, which is why the folds vary so much. Most resumes in the collection are tidy documents stuffed with keywords for one field, so they tell us little about the messy resumes students actually write or about the roles that InternAI's recruiters post. The categories of HR, Arts, Sales and others are wide areas of study and not internships. This is one of the primary reasons why the classifier remains an advisory model. The support vector machine performed better than the k-NN but was used only to benchmark and no changes were made to the existing platform. None of the above evaluates the resume score, match percent or ranking.

D. Threats to Validity

These results have certain restrictions. All demonstration statistics are faked and all resumes used in this statistics are short ones, therefore none of the statistics numbers is real and corresponds to real students. These tests were conducted only once and these tests are done by the same persons that implemented the system code. Statistics related to classifier accuracy was obtained based on 135 unique resumes in one public dataset. User study was not performed by us, therefore any claim regarding usability of the platform for students and recruiters will be based only on design assumptions and user observations. There are several weak spots left in the system itself. Access token is stored in browser storage. Multiple unsuccessful login attempts for one account are not blocked and correspond to the authentication failures item of the OWASP list [30]. Uploading files is based only on the file type and its extension without checking the beginning of file content. Interviews can not establish connection if both parties are located behind the strict symmetric NAT. Backend is implemented as one single instance that stores socket state in memory, therefore can not be scaled.

VII. CONCLUSION AND FUTURE SCOPE

In this paper, we proposed the InternAI, a platform for hiring interns designed based on the workflow of students, companies and administrators instead of the market-based system used in generic sites that have a listing system. In combining the resume uploading/parsing, ATS scoring, skill gap analysis and internship matching for the students and hiring pipeline and analytics for the recruiters, the platform provides data that would otherwise be missed with a simple spreadsheet and email-based process. While students get an indication of their match score, recruiters can get a ranked and pipeline-tracked applicant pool.

The boundaries of the modules as defined in Table II are not dependent on the specific format and vocabulary of a particular institute. In theory, this means that the same platform could be connected to another institute's database and list of skills without modifying the modules. Further validation of this portability as well as the change in the modeling technique mentioned above will be done in the future.

Planned future work includes replacing the reasoned score weights with a model trained on historical placement outcomes once enough labeled outcome data has been collected through use of the platform; extending the analytics module with predictive metrics such as expected time-to-placement; and packaging the student-facing views as an installable mobile application to reduce reliance on desktop access during placement drives.

Aside from the model change itself, a number of smaller issues arise after Section VI-D. The use of exact keywords ignores skills that have no common words, like REST APIs and API design, while the addition of sentence embeddings [40] as a secondary metric to the keyword score is another possibility. A test for fairness, inspired by the work of Raghavan et al. [11], involving comparisons of scores between groups of candidates in a sample that has been properly consented to, needs to be performed before making any decisions about hiring based on the score. Engineering-wise, TURN support would allow interviews to happen under strict NAT, account lockout after multiple login failures, first bytes validation of uploads, OCR to parse scanned resumes, and a controlled experiment with students and placement officers will determine if the feedback affects behavior.

ACKNOWLEDGMENT

The authors would like to thank Prof. Dhanashree Tamhane of the Department of Information Technology, Vidyalankar Institute of Technology, for guidance and feedback throughout the development of this project.

REFERENCES

[1] C. Qin et al., "A comprehensive survey of artificial intelligence techniques for talent analytics," arXiv:2307.03195, 2023.

[2] Wheebox, "India Skills Report 2025," Wheebox in association with CII, AICTE and partner organizations, Dec. 2024.

[3] Internshala, "Fresher jobs, internships and courses." [Online]. Available: https://internshala.com/ (accessed Oct. 21, 2026).

[4] F. Ricci, L. Rokach, and B. Shapira, Eds., Recommender Systems Handbook. New York, NY, USA: Springer, 2011.

[5] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval, 2nd ed. Boston, MA, USA: Addison-Wesley, 2011.

[6] C. Zhu, H. Zhu, H. Xiong, C. Ma, F. Xie, P. Ding, and P. Li, "Person-job fit: Adapting the right talent for the right job with joint representation learning," ACM Trans. Manage. Inf. Syst., vol. 9, 2018.

[7] C. Qin, H. Zhu, T. Xu, C. Zhu, L. Jiang, E. Chen, and H. Xiong, "Enhancing person-job fit for talent recruitment: An ability-aware neural network approach," in Proc. 41st Int. ACM SIGIR Conf. Research and Development in Information Retrieval, 2018, pp. 25–34, doi: 10.1145/3209978.3210025.

[8] C. de Ruijt and S. Bhulai, "Job recommender systems: A review," arXiv:2111.13576, 2021.

[9] M. N. Freire and L. N. de Castro, "e-Recruitment recommender systems: A systematic review," Knowl. Inf. Syst., vol. 63, no. 1, pp. 1–20, 2021.

[10] F. Retyk, H. Fabregat, J. Aizpuru, M. Taglio, and R. Zbib, "Résumé parsing as hierarchical sequence labeling: An empirical study," in Proc. 3rd Workshop on Recommender Systems for Human Resources (RecSys in HR 2023), 17th ACM Conf. Recommender Systems, Singapore, Sep. 2023.

[11] M. Raghavan, S. Barocas, J. Kleinberg, and K. Levy, "Mitigating bias in algorithmic hiring: Evaluating claims and practices," in Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT*), 2020, doi: 10.1145/3351095.3372828.

[12] J. S. Black and P. van Esch, "AI-enabled recruiting: What is it and how should a manager use it?," Business Horizons, vol. 63, no. 2, pp. 215–226, 2020, doi: 10.1016/j.bushor.2019.12.001.

[13] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," Nature Mach. Intell., vol. 1, no. 5, pp. 206–215, 2019, doi: 10.1038/s42256-019-0048-x.

[14] Z. Ji et al., "Survey of hallucination in natural language generation," ACM Comput. Surv., vol. 55, no. 12, 2023, arXiv:2202.03629.

[15] Vercel, "Next.js Documentation." [Online]. Available: https://nextjs.org/docs

[16] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

[17] MongoDB, Inc., "MongoDB Atlas Documentation." [Online]. Available: https://www.mongodb.com/docs/atlas/

[18] M. B. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force, May 2015.

[19] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in Present and Ulterior Software Engineering, M. Mazzara and B. Meyer, Eds. Cham, Switzerland: Springer, 2017, pp. 195–216.

[20] M. Fowler, "MonolithFirst," martinfowler.com, Jun. 2015. [Online]. Available: https://martinfowler.com/bliki/MonolithFirst.html

[21] MongoDB, Inc., "Data Modeling Introduction," MongoDB Manual. [Online]. Available: https://www.mongodb.com/docs/manual/core/data-modeling-introduction/

[22] StrongLoop and IBM, "Express: Fast, unopinionated, minimalist web framework for Node.js." [Online]. Available: https://expressjs.com/

[23] OpenJS Foundation, "About Node.js." [Online]. Available: https://nodejs.org/en/about

[24] I. Sommerville, Software Engineering, 10th ed. Boston, MA, USA: Pearson, 2015.

[25] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[26] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Inf. Process. Manage., vol. 24, no. 5, pp. 513–523, 1988.

[27] T. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE Trans. Inf. Theory, vol. 13, no. 1, pp. 21–27, 1967.

[28] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[29] A. Ron, A. Shulman-Peleg, and E. Bronshtein, "No SQL, no injection? Examining NoSQL security," in Proc. 9th Workshop on Web 2.0 Security and Privacy (W2SP), 2015, arXiv:1506.04082.

[30] OWASP Foundation, "OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks," OWASP Foundation, 2021.

[31] E. Bertino and R. Sandhu, "Database Security–Concepts, Approaches, and Challenges," IEEE Trans. Dependable Secure Comput., vol. 2, no. 1, pp. 2–19, 2005.

[32] N. Provos and D. Mazières, "A future-adaptable password scheme," in Proc. USENIX Annual Technical Conf., 1999.

[33] World Wide Web Consortium (W3C), "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, 2018.

[34] I. Fette and A. Melnikov, "The WebSocket protocol," RFC 6455, Internet Engineering Task Force, Dec. 2011.

[35] World Wide Web Consortium (W3C), "WebRTC 1.0: Real-time communication between browsers," W3C Recommendation, Jan. 2021.

[36] H. Alvestrand, "Overview: Real-time protocols for browser-based applications," RFC 8825, Internet Engineering Task Force, Jan. 2021.

[37] A. Grattafiori et al., "The Llama 3 herd of models," arXiv:2407.21783, 2024.

[38] A. Q. Jiang et al., "Mistral 7B," arXiv:2310.06825, 2023.

[39] G. Dutta, "Resume dataset," Kaggle. [Online]. Available: https://www.kaggle.com/gauravduttakiit/resume-dataset

[40] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. Conf. Empirical Methods in Natural Language Processing and 9th Int. Joint Conf. Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 2019, pp. 3982–3992.