SMART INTERNSHIP FINDER: AN AI-POWERED PLATFORM FOR PERSONALIZED INTERNSHIP DISCOVERY

Suguna.M,
Indian,
Assistant professor,
Department of Computer Science and Engineering,
SNS College of Technology,
sugunasae@gmail.com.

Hema.S.J,
Indian,
Student,
Department of Computer Science and Engineering,
SNS College of Technology,
Hem020424@gmail.com.

Haries Prasad.S.K,
Indian,
Student,
Department of Computer Science and Engineering,
SNS College of Technology,
Hariesprasad2007@gmail.com.

Jeeva.S,
Indian,
Student,
Department of Computer Science and Engineering,
SNS College of Technology,
s.jeeva1700@gmail.com.

Gokulakrishnan.L,
Indian,
Student,
Department of Computer Science and Engineering,
SNS College of Technology,
Ggokul93811@gmail.com.

Abstract: **Smart Internship Finder is a full-stack web application developed to help engineering students discover and apply for internships using their skills, interests and career goals. The platform integrates internship search, student profiling, resume analysis, skill extraction, intelligent matching, skill-gap analysis, application tracking and role-based management within a unified environment. The documented implementation is divided into three independent services: a React 18, Vite and Tailwind CSS frontend; a Node.js, Express and MongoDB backend; and a Python FastAPI artificial-intelligence service using spaCy and scikit-learn. The AI service can analyze PDF and DOCX resumes, extract structured information and identify technical skills. Internship and candidate information are converted into TF-IDF representations and compared using cosine similarity to generate a normalized match score. In addition to the overall match, the platform identifies matched, missing and bonus skills so that students can understand their readiness for an internship. Students can filter opportunities by location, course, compensation and certificate type, save internships, submit applications and monitor application notifications. Companies can publish internship opportunities, maintain company profiles and manage applications, while administrators can verify companies and moderate internships, users and applications. The repository also documents deployment options for the frontend, backend, AI service and MongoDB Atlas, together with integration testing and performance measurements. The proposed system demonstrates how full-stack web engineering and natural language processing can be combined to reduce fragmentation in internship discovery and provide a more informative student-centered application workflow.**

Keywords: Smart Internship Finder, Internship Recommendation, Resume Parsing, Skill Extraction, TF-IDF, Cosine Similarity, Skill Gap Analysis, Natural Language Processing, React, FastAPI, MongoDB

I. INTRODUCTION

Internships provide engineering students with opportunities to connect academic learning with practical work experience. However, internship discovery is frequently distributed across multiple sources, and students may have to search separately for vacancies, understand eligibility requirements, prepare

resumes, compare companies and track applications. This fragmented process becomes more difficult when a student is unsure whether the technical skills listed in an internship description are already present in the student's profile.

The Smart Internship Finder project addresses this problem through a unified web platform. The repository describes the application as a full-stack system that helps engineering students discover and apply for internships using AI-powered resume analysis, skill-gap detection and intelligent matching [1]. Rather than treating internship search as a simple listing problem, the project connects the student's profile and resume with the requirements of an internship. This makes the platform useful both for finding opportunities and for understanding the skills needed to improve an application.

The application is organized as three independent services. The frontend is implemented using React, Vite and Tailwind CSS. The backend is implemented using Node.js, Express and MongoDB, while the AI service uses Python, FastAPI, spaCy and scikit-learn [1]. This separation creates a clear boundary between the user interface, application and database logic, and machine-learning/NLP processing.

The student-facing feature set includes an AI resume parser, smart matching, skill-gap analysis, advanced search, application tracking, saved internships and notifications. The company side provides internship posting, application management and company profiles, while administrators are given verification, analytics and moderation functions [1]. The platform consequently represents a complete internship lifecycle rather than a single recommendation component.

The main technical contribution of the matching system is the use of TF-IDF vectorization and cosine similarity. The matching guide in the repository describes a process in which candidate and internship text are transformed into vectors, compared and normalized to a 0–100 match score [2]. The system also provides a skill-level explanation by comparing required skills with skills extracted from the student's resume or profile.

The principal objectives of the project are: (1) to simplify internship discovery; (2) to reduce repetitive manual resume information entry; (3) to provide an interpretable match score; (4) to identify skill gaps before application; (5) to provide a structured application tracking mechanism; (6) to give companies a controlled environment for publishing opportunities and reviewing applications; and (7) to provide administrators with platform-level management and verification.

The project is therefore positioned at the intersection of web application engineering, natural language processing, information retrieval and recommendation systems. Its current implementation also provides a practical baseline for future improvements such as semantic transformer models, cloud document storage, WebSocket-based notifications, enhanced search and mobile delivery [1].

II. LITERATURE SURVEY

Internship recommendation can be considered a specialized information-retrieval and matching problem. In a conventional search system, a student enters keywords and receives opportunities containing related terms. Such an approach can return a large number of results but does not necessarily explain how closely an opportunity matches a candidate. The Smart Internship Finder attempts to extend the search process by incorporating the candidate's profile and resume into the matching workflow.

TF-IDF is a classical information-retrieval representation that assigns higher importance to terms that are informative within

a document while reducing the effect of very common terms [4]. When candidate and internship descriptions are represented as TF-IDF vectors, cosine similarity can be used to estimate the directional similarity between the two vectors. This approach is computationally practical and comparatively transparent, which makes it suitable as a baseline for an application where the matching result should be understandable.

Resume processing introduces a document-understanding challenge because resumes can contain different structures, headings, formats and writing styles. The project documentation describes extraction of sections such as personal information, education, experience, projects, certifications, achievements, interests and skills [5]. PDF and DOCX processing is incorporated into the resume workflow, while image-based processing is also documented in the expanded resume system.

Natural language processing provides another important layer because skills may appear in different textual contexts. The Smart Internship Finder uses spaCy and a technology-skill database to identify and standardize technical skills before the matching process [2]. This makes the system less dependent on exact formatting and allows the extracted skill list to be reused for profile synchronization and skill-gap analysis.

Recommendation systems can also benefit from explanations. A single similarity score may not tell a student why an

opportunity is considered suitable or unsuitable. The project's skill-gap workflow addresses this by separating skills into matched, missing and bonus categories. The result can therefore be interpreted as a preparation guide as well as a recommendation signal.

The platform also incorporates the operational features expected in a multi-role recruitment application. Students can apply and track applications; companies can create and manage internships; administrators can verify companies and manage content [1]. This role separation is important because the same internship record has different lifecycle actions depending on the actor.

The repository identifies the current TF-IDF approach as a limitation because semantic relationships that do not share the same words may be missed. It proposes deep-learning or transformer-based approaches such as BERT or RoBERTa as future improvements [1]. Other documented limitations include local resume storage, polling-based real-time updates, basic company/admin interfaces, absence of email notifications, basic MongoDB text search and the current web-only delivery model [1].

III. PROPOSED SYSTEM

The proposed Smart Internship Finder architecture follows a layered service model. The presentation layer provides responsive pages, dashboards and forms. The backend provides authentication, authorization, REST APIs, business logic and database operations. The AI layer provides resume parsing, skill extraction, matching and skill-gap analysis. MongoDB stores the persistent application records.

Figure 1: Smart Internship Finder System Architecture

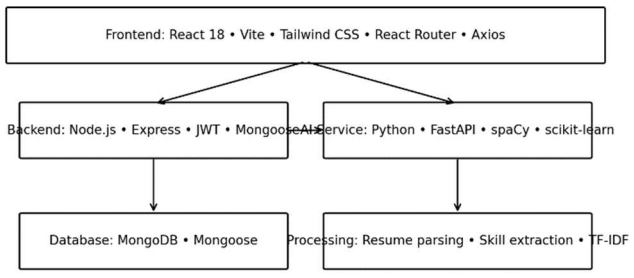


Figure 1: Smart Internship Finder System Architecture

The frontend is implemented using React 18 with Vite and Tailwind CSS. React Router supports route navigation and Axios is used for communication with backend services. The documented project structure contains reusable components, an authentication context, layouts, pages, service functions and utility modules [1]. Dashboard visualizations use Recharts. The architecture documentation also identifies responsive design, protected routes, lazy loading, form validation and search state synchronization as part of the frontend design [2].

The backend is implemented with Node.js and Express and uses Mongoose for MongoDB access. The project follows a model-controller-route organization with middleware for authentication and role protection. The backend exposes separate API groups for authentication, internships, students, companies and administrators [1]. This structure allows the application to enforce different permissions while maintaining a consistent REST interface.

Authentication includes student, company and administrator login and registration flows. The backend uses JWT-based authentication and bcrypt password hashing. The repository security notes specify bcrypt salt rounds of 10, a strong JWT secret requirement, restricted CORS configuration and file-upload validation [1]. The design also separates role and ownership checks so that users cannot freely modify resources belonging to another role or account.

The database layer is based on MongoDB. The documented model structure contains entities for users, student profiles, companies, internships, applications, resumes, saved internships, notifications, reviews, learning resources and skills [2]. This data model supports the full workflow from registration through profile construction, internship discovery, application submission and status tracking.

The AI service is implemented using Python and FastAPI. The repository structure identifies separate modules for resume parsing, skill extraction and matching [1]. spaCy provides NLP processing, while scikit-learn provides TF-IDF vectorization and similarity calculation. The service exposes endpoints for resume analysis, skill extraction, match scoring, batch matching and skill-gap analysis.

The matching process begins by collecting relevant candidate information and internship requirements. Text is normalized and transformed into TF-IDF vectors. Cosine similarity is then calculated between the candidate representation and internship representation. The resulting value is normalized into a 0–100 score. The documented interpretation labels scores of 90 or above as Excellent, 75–89 as Strong, 60–74 as Moderate and values below 60 as Low [2].

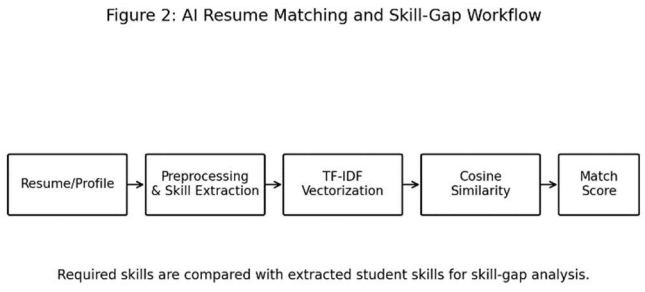


Figure 2: AI Resume Matching and Skill-Gap Workflow

The match score is complemented by explicit skill-gap analysis. If an internship requires skills such as Python, SQL or React, the system compares these requirements with the student's extracted and profile skills. Skills present in both sets are treated as matched, required skills absent from the student's set are identified as missing, and additional candidate skills can be identified as bonus skills. This creates an explanation layer that can help a student prioritize learning before applying.

The resume subsystem is an important input to the AI workflow. The documented system accepts PDF, DOC and DOCX resume formats and includes file-size restrictions. Resume analysis extracts structured sections and technical skills. The repository also documents an AI resume-builder workflow in which students can create or edit resume information and export the resulting document [5]. The extracted skills can be synchronized with the student profile, reducing repeated manual entry.

The internship search module supports multiple filters. According to the project README, students can filter by location, course, compensation and certificate type, along with other listing attributes [1]. Internship records include detailed information such as role, company, location, duration, mode, compensation, start date, deadline and skill requirements. The project also provides endpoints for all internships, paid internships, unpaid internships, course lists and internship statistics.

The student application module connects discovery with action. A student can submit an application, inspect previous applications, withdraw an application where supported, save or unsave an internship and receive notifications. The documented API includes endpoints for application creation, withdrawal, saved internships and notification read states [1].

The company module provides the opposite side of the recruitment workflow. A company can maintain its profile, create internships, update them, delete them and retrieve applications. Company users can also change application status through the corresponding API. Administrator verification provides an additional control before a company receives a verified status [1].

The administrator module is intended to protect the quality and consistency of the platform. Administrators can view statistics, list and manage users, verify or unverify companies, manage internships and inspect applications. The repository also documents review management and content moderation capabilities [1].

The deployment model separates the frontend, backend, AI service and database. The repository documents Vercel or Netlify for frontend deployment, Render/Railway/Heroku-style deployment for the Node backend, Render/Railway for the Python AI service and MongoDB Atlas for cloud database storage [1]. The provided project website is deployed at the Vercel address associated with the repository [1].

IV. IMPLEMENTATION AND WORKFLOW

The implementation begins with user registration and authentication. A student creates an account and can subsequently maintain a profile containing academic and career-related information. The authentication layer retrieves the current user and protects dashboard routes. Companies and administrators use separate authentication paths, ensuring that role-specific pages and operations are not mixed.

After authentication, the student can construct a profile and upload a resume. The resume is sent to the backend as multipart form data and passed to the AI service for analysis. The parser extracts textual content, detects relevant sections and sends the result to the skill-extraction component. The identified skills can then be used in the matching workflow and synchronized with the student's profile.

The internship discovery stage retrieves listings through the internship API. Filters can narrow the result set by course, location, compensation and certificate type. Internship details provide the information required for the student to decide whether to investigate an opportunity further. The AI matching service adds a candidate-specific match score, while skill-gap analysis explains the technical overlap.

Once an internship is selected, the student can apply directly through the application workflow. The application record stores the relationship between the student and the internship. Companies can retrieve applications for their internships and update their status. Students can then monitor changes through the dashboard and notifications.

Saved internships provide a separate mechanism for opportunities that are not immediately applied to. This supports a two-stage workflow in which students first collect potential opportunities and later return to compare them. Notifications complement the process by providing application-related updates and read/unread status management.

The company workflow starts with company registration and profile configuration. After administrative verification, the company can publish internship listings with required skills and other opportunity details. Applications can then be reviewed through company-specific endpoints. The administrator can monitor the platform at a higher level and perform verification or moderation actions.

The system also includes a learning-resource concept associated with skill-gap analysis. The purpose is to connect missing skills with preparation rather than treating a missing requirement as a final rejection. This makes the system more useful for students who are still developing their technical profile.

V. RESULTS AND DISCUSSION

The repository contains an integration report covering the major application workflows. The documented test areas include registration, authentication, profile operations, resume upload, internship search, applications, saved internships, notifications and role protection. Company and administrator functionality are also included in the project's API and feature documentation [7].

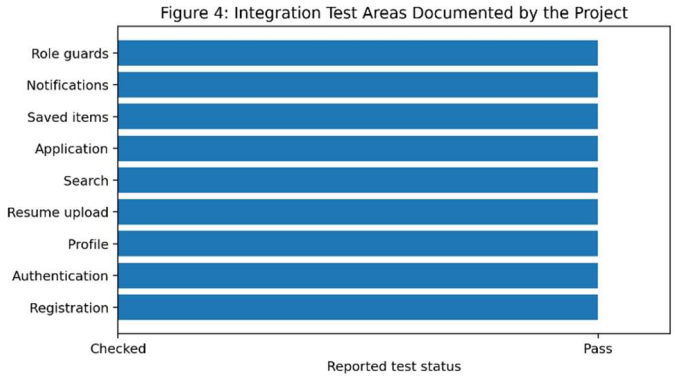


Figure 4: Integration Test Areas Documented by the Project

The project documentation reports operational measurements for page loading, API response, resume parsing and database query performance. The reported values include approximately 1.5 seconds for page loading at the documented LCP measurement, approximately 150 ms for API response at p95, approximately 2 seconds for resume parsing and approximately 30 ms for database queries [7]. These values are project-reported measurements and should be interpreted in relation to the test environment and workload rather than as universal performance guarantees.

The matching workflow provides an important result because it combines a numeric similarity score with an explanation. TF-IDF and cosine similarity offer a relatively lightweight mechanism for converting unstructured resume and internship text into comparable representations. The normalized score can be displayed consistently across internship listings, while the skill-gap result provides a more concrete explanation of missing requirements.

The resume subsystem reduces the amount of manual data entry expected from a student. Instead of entering every technical skill individually, the system can extract skills from an existing document. The project documentation describes structured extraction of education, experience, projects, certifications, achievements, interests and skills [5]. This information can then contribute to the student's profile and matching process.

The advanced search mechanism addresses the practical problem of information overload. By allowing filtering based on attributes such as location, course, compensation and certificate type, students can reduce the number of irrelevant opportunities before applying AI-assisted matching. The combined workflow is therefore not solely an AI recommendation system; it is an information-management system with AI as one layer.

The role-based design is another significant implementation result. A student should be able to manage personal applications and saved opportunities but should not be able to change another student's records. A company should be able to manage its own internships and applications but not system-wide users. Administrators require broader privileges for verification and moderation. The repository's middleware and API organization are designed around these distinctions [1][2].

From a maintainability perspective, the separation into frontend, backend and AI services allows independent development and deployment. The frontend can be deployed through Vercel or Netlify, the Node.js backend through a backend hosting service, and the Python AI service separately [1]. This separation also allows the AI component to be upgraded without requiring a complete rewrite of the user interface.

The current system has limitations that are explicitly acknowledged in the repository. The AI matcher currently uses TF-IDF cosine similarity and can be enhanced with deep-learning models. Resume files are stored locally and can be moved to cloud storage for production. Real-time updates currently use polling and can be replaced by WebSockets. The company and admin portals are described as basic, email notifications are not implemented, search is based on MongoDB text search, and the application is currently web-only [1].

These limitations also define a clear future development path. A transformer-based semantic matching model could improve recognition of related skills and job descriptions even when wording differs. A dedicated search engine could support richer filtering and ranking. Cloud object storage could improve resume availability and scalability. WebSockets could provide live status updates. Email integration could extend notifications beyond the web dashboard, while a mobile client could make the platform accessible through a dedicated application.

For evaluation, future work should also introduce a labeled internship-outcome dataset. The current repository documents engineering tests and operational measurements, but it does not provide evidence that the reported match score predicts actual internship selection outcomes. A future evaluation could compare the system's recommendations against human judgments or historical application outcomes using precision, recall, ranking metrics and user satisfaction measures. This would distinguish software integration performance from recommendation quality.

VI. SECURITY, DEPLOYMENT AND SCALABILITY

Security is incorporated into the architecture through authentication, password hashing, JWT-based sessions, role protection and input/file validation. The repository recommends a strong random JWT secret of at least 32 characters, bcrypt with salt rounds of 10, allowed-origin CORS and exclusion of environment files from version control [1]. Resume uploads are restricted by file type and size, reducing unnecessary exposure to unsupported files.

The deployment architecture separates public frontend delivery from backend and AI services. The frontend uses a Vercel-compatible production build with the dist output directory. The backend requires production environment variables and a cloud MongoDB connection. The AI service can be deployed with Python 3.10+ and started through Uvicorn. MongoDB Atlas is identified as a cloud database option [1].

Scalability is supported by the service separation because individual components can be scaled according to demand. Internship search and dashboard requests primarily use the backend, while resume analysis and batch matching consume AI-service resources. Separating these workloads prevents document processing from being tightly coupled to the web server process.

However, scalability in a production deployment would require further engineering. Local resume storage should be replaced by object storage, polling should be replaced by push-based notifications, and the basic search layer could be replaced by a dedicated search service. The AI matching component could also require caching or batch processing when the number of internships and users increases [1].

VII. FUTURE ENHANCEMENTS

The first future enhancement is semantic matching. Transformer-based language models could capture relationships between terms that do not share the same lexical form. For example, a model could learn that related technologies or job responsibilities may express similar concepts even when the exact words are different. The repository itself identifies deep-learning models as a possible improvement to the current TF-IDF approach [1].The second enhancement is real-time communication. WebSockets could replace polling for application-status changes and notifications. This would allow students to receive updates as soon as a company changes an application state.The third enhancement is scalable resume management. Cloud object storage such as S3 or a similar service could replace local file storage. This would support distributed deployment and reduce dependence on the filesystem of a single backend instance.

The fourth enhancement is search modernization. Elasticsearch or another dedicated search technology could support more advanced filtering, full-text ranking and scalable search across large internship collections. This could be combined with AI matching so that lexical retrieval and semantic ranking operate together.The fifth enhancement is broader accessibility. A React Native mobile application could extend the current web-only platform to mobile devices. Email services could complement in-app notifications. Company analytics and a richer administrator dashboard could also be developed as the number of users increases.

VIII. CONCLUSION

Smart Internship Finder demonstrates a practical integration of full-stack web development, natural language processing and

recommendation techniques for internship discovery. The platform combines student profiles, resume analysis, skill extraction, internship search, AI matching, skill-gap analysis, applications, saved opportunities and notifications in a single workflow. The company and administrator modules provide the complementary functions required to publish, verify and manage internship opportunities.

The architecture is separated into React/Vite/Tailwind frontend, Node.js/Express/MongoDB backend and Python/FastAPI AI service. This organization provides a maintainable boundary between user interaction, application logic and AI processing. The AI component uses TF-IDF and cosine similarity to produce a normalized match score and supplements the score with skill-gap information.The project documentation reports integration testing across the core workflows and provides initial operational measurements. These results demonstrate that the system has been implemented as an end-to-end application, while the documented limitations identify areas for continued development. In particular, semantic transformer matching, cloud resume storage, WebSocket notifications, richer search, mobile delivery and email notifications can extend the system.

The system should be regarded as a software platform and matching aid rather than a guaranteed predictor of internship selection. Its current value lies in reducing search fragmentation, structuring student information, highlighting skill requirements and connecting discovery with application tracking. With larger real-world datasets and controlled recommendation evaluation, the platform could be further assessed for recommendation quality and user impact.

REFERENCES

[1]. Hema-sj, “Smart Internship Finder,” GitHub repository, project README and implementation documentation, accessed September 2026.

[2]. Hema-sj, “Smart Internship Finder – AI Match Guide and Architecture Documentation,” GitHub repository, accessed September 2026.

[3]. Hema-sj, “Admin Dashboard Features,” GitHub repository, accessed September 2026.

[4]. G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988.

[5]. Hema-sj, “Resume System Documentation,” GitHub repository, accessed September 2026.

[6]. Hema-sj, “Student Company View Features,” GitHub repository, accessed September 2026.

[7]. Hema-sj, “Integration Report,” GitHub repository, accessed September 2026.

[8]. C. D. Manning, P. Raghavan and H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.