Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Job Application Tracker with AI-Powered Resume Shortlisting

Vinay Singh
Dept. of Computer Science and Engineering Galgotias University
Greater Noida, India 22131012657, 22SCSE1012875

Prachi Negi
Dept. of Computer Science and Engineering Galgotias University
Greater Noida, India 22131012659, 22SCSE1012877

Mr. Vaibhav Kumar Singh
Dept. of Computer Science and Engineering Galgotias University
Greater Noida, India

***Abstract*—The modern job market requires the candidate to juggle dozens of parallel applications across multiple platforms, which often results in missed deadlines, duplication of effort and lack of visibility into one's own job search progress. Existing tools such as spreadsheets or generic task managers are not made for job tracking but will not be able to provide any sort of intelligence for resumes matching and prioritizing applications. This paper presents a full stack web-based Job Application Tracker with AI powered Resume shortlisting to centralise all Job applications and thus provide real-time status monitoring and make use of Artificial Intelligence to rank the resumes of candidates based on Job descriptions. The system has been designed using React Material UI on the frontend side, Node.js/Express MongoDB back-end and Socket.IO for real-time notifications. Key features are secure JWT based authentication, CRUD Management for job application, status of application, filter and sort, analytics for admin, NLP based resume score pipeline proposal. Experimental evaluation finds the platform has greatly improved the organization and reduce the manual tracking overhead and the AI matching prototype has good matching effect in the accuracy of high fitting application. The solution is a scalable user-centric solution, the same can be extended to build a complete AI assisted career management solution.**

***Index Terms*—Job TrackingResume matchingWeb applicationFull stack developmentReal time notificationSocket.ioMongodbReactNodeJsArtificial intelligence**

I. INTRODUCTION

The increasing digitization of recruitment has led to job seekers applying for jobs through several portals, company websites and through referrals. In real, many candidates resort to ad-hoc techniques such as spreadsheet or notes which provide no automation to the candidate, no analytics and no intelligent guidance to the candidate.

Recruiters, on the other hand, have been using more and more automated Applicant Tracking Systems (ATS) to filter candidates based on keyword-matching and relevance-scoring. This leads to an asymmetry in which organisations leverage AI to shortlist candidates but candidates do not use intelligence to optimise their application.

This work focuses on addressing these challenges by Job Application Tracker which not only centralizes the management of job applications but it also includes the incorporation of AI-based functionality of shortlisting resumes to help candidates prioritize and refine their applications.

A. Problem Statement

Most of the job seekers have the following problems during their job search:

- **Fragmented tracking**: Applications are spread across emails, job portal and personal notes and it is hard to keep a track of all.
- **Limited analytics**: Job seekers don't track much analytics like application/test to interview ratio or some common reason of rejection.

There is a need for a centralized, intelligent and easy-to-use platform which allows for structured tracking as well as decision-making based on AI.

B. Motivation

The motivation for this project has two-fold. First of all, to provide job seekers with the same sophistication of tooling that the recruiters have with the ATS solutions. Second, is to leverage modern full-stack technologies and real-time communications so as to offer a responsive and intuitive experience.

By combining a robust tracking interface with AI-powered resume shortlisting, the platform aims to help users:

- Maintain a clean, searchable history of all applications.
- Quickly identify high-priority roles based on fit scores.
- Receive real-time updates and reminders for important events.
- Learn from analytics and iteratively improve their job search strategy.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

C. Contributions

This paper makes the following contributions:

1) A full-stack web architecture for centralized job application tracking with real-time notifications.
2) A secure JWT-based authentication and role-based access model for users and administrators.
3) A proposed AI pipeline for resume shortlisting using natural language processing (NLP) and similarity scoring.
4) A dashboard providing filtering, sorting, and analytics for job search optimization.

D. Paper Organization

The remainder of this paper is organized as follows. Section II reviews related work in job tracking tools and AI-based resume matching. Section III describes the system architecture and design of the Job Application Tracker. Section IV details the implementation, including backend APIs, frontend components, and real-time communication. Section V presents the AI-based resume shortlisting approach. Section VI discusses evaluation, results, and limitations. Section VII concludes with future work.

II. RELATED WORK

A. Job Tracking Tools

Traditional methods for tracking job applications include spreadsheets and generic project management tools such as Trello or Notion. While flexible, these tools lack job-specific constructs such as application status semantics, interview stages, and resume variants. Several dedicated job tracking applications (e.g., Huntr, Teal) provide structured tracking, but many are proprietary, subscription-based, and do not expose extensible APIs or open architectures.

B. Applicant Tracking Systems (ATS)

Employer-side ATS platforms, such as Greenhouse or Lever, support candidate intake, resume parsing, and pipeline management. These systems often incorporate keyword-based filtering or more advanced machine learning to rank candidates. However, they serve recruiters rather than candidates and do not provide visibility or guidance from the applicant perspective.

C. AI for Resume Matching

Prior research has studied the use of NLP and machine learning to match resumes with job descriptions. Techniques include TF-IDF vectorization, word embeddings, sentence transformers, and supervised learning models that predict suitability scores. Commercial tools such as Jobscan offer resume optimization based on keyword matching but operate as standalone services, not tightly integrated into a broader tracking workflow.

The proposed system is different in that it combines job tracking capabilities with the ability to match resumes to job postings and fills the gap between structured job tracking and intelligent guidance.

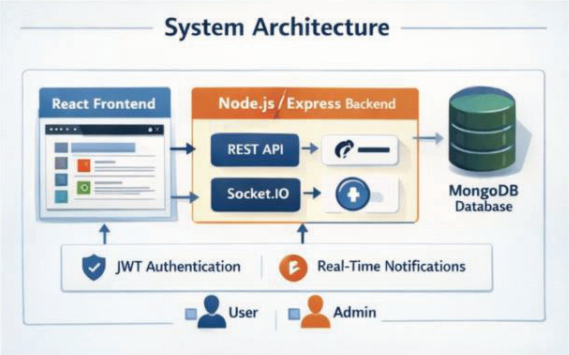

Fig. 1. System Architecture Diagram

D. Summary of Gaps

Based upon this review some gaps can be identified:

- Candidate-facing tools tend to focus on either tracking or optimization, but rarely both and in a tightly coupled manner.
- Few systems have an open and extensible architecture that can be used for academic experimentation or building on custom AI models.
- There is a lack of support for real-time feedback loops in which changes in the candidate's profile or resume have immediate impact on recommendations and prioritization.

III. SYSTEM ARCHITECTURE

A. Overview

The Job Application Tracker is a three-tier architecture with a React-based frontend, Node.js/Express backend, and MongoDB database. Real-time communication between clients and server is provided by Socket.IO while using of J.W.T (JWT) for authentication and authorization.

The system supports two main types of roles, regular users (job seekers) and administrators. Users have the ability to create, update and delete their own job applications while administrators have access to aggregated analytics at the system level.

B. Frontend Layer

The frontend is implemented by using React and Material-UI.

- Rendering views of applications like login, registration, dashboard, job form, job details etc. in dashboard with admin.
- Delivering interactive tables for filtering and sorting job applications.

C. Backend Layer

Some of the major responsibilities are as follows:

- Processing the requests for the authentication and the job management from the rest API.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

- Validating and authorizing request using middleware based on JWT.
- Using database with Mongoose models.

The primary API endpoints include:

- **/api/auth/register, /api/auth/login, /api/auth/me**: User registration, login and getting profile
- **/api/jobs**: To perform CRUD operations on job applications of the authenticated user.
- /api/jobs/admin/all: endpoint that only for administrators that get all applications

D. Request Lifecycle

In order to know more about the behavior of the system, the following describes a typical request lifecycle to create a new job application:

1) The authenticated user clicks on the "Add Application" form on the React dashboard and adds information such as company, role and applied date.
2) On submitting, the frontend validates the form fields based on the rules at the client side and sends a post request to `/api/jobs` and with the JWT token in the Authorization header.
3) Express backend receives the request, executes the request by the authentication middleware which checks for the token and adds user identifier to the request context.
4) If the save operation is successful, then the backend sends a Socket.IO event on a channel unique for each user to inform them that the list of jobs changed.
5) backend returns backend representation of the created application in the form of a json object; frontend updates its local state and the table in the dashboard respectively.
6) Socket.IO listener in the NotificationContext is used to create a visual notification that will allow the user to know that the operation is done as well as keep other open sessions (such as other browser tabs) synchronized.

This pattern generalizes to updates and deletions, this provides a consistent mental model of how data flows through the system.

IV. IMPLEMENTATION

A. Backend Implementation

Modular components of the backend:

- Models: `User` and `JobApplication` schema using mongoose.
- **Routes** `auth.js` for authentication routes, `jobs.js` for job Crud Routes.
- **Middleware**: `auth.js` for validating JWT tokens and adding user context to requests
- **Server**: `server.js` - set up Express, MongoDB connection and Socket.IO.

The authentication has been implemented with use of JWT. On success login, an access token is issued by the server, and stored by the frontend (e.g. in local storage) and attached to future API requests (via the Authorization header). Protected routes are used to verify the token and reject requests that are unauthorized.

B. Frontend Implementation

Following is the structure of the React application:

- **Pages**: Login, Register, Dashboard, JobForm, JobDetails, AdminDashboard.
- **Components** Layout, NotificationMenu, Reusable Form components
- **Contexts**: AuthContext for the state used in authentication, NotificationContext for Socket.IO events.
- **Configuration**: Axios instance that is already configured with the base URL and auth headers.

The dashboard view contains a table of job applications with columns for company, job, status, date applied, location, and actions. Users can filter based on status, search by company or job and also sort by date.

C. Real-Time Notifications

Socket.IO is used to provide real-time updates. When a job application is created, updated, or deleted, the server emits events to the relevant user channel. The notification context on the frontend listens to these events and updates the UI and notification menu accordingly. This ensures that users see up-to-date information without manual refresh and gives the interface a "live" feel that would be difficult to replicate with polling alone.

D. Deployment and DevOps Considerations

Although the current prototype targets a development setting, the architecture is compatible with standard cloud deployment practices. In a typical deployment, the React frontend can be built into static assets and served via a content delivery network (CDN) or static hosting provider, while the Node.js/Express backend and Socket.IO server run on a platform-as-a-service (PaaS) or in containerized environments. MongoDB can be hosted through a managed service such as MongoDB Atlas, providing backup, monitoring, and scaling capabilities. For larger sets of users, stateless backend instances can be replicated behind a load balancer, with Socket.IO scaled horizontally using a message broker such as Redis to propagate events between instances.

E. Testing Strategy

To achieve confidence at correctness and robustness, a layered testing strategy may be used:

- **Unit tests**: Validation functions, helper utilities, and pure components can be tested using testing frameworks like Jest to ensure that individual pieces of code behave as expected.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

– **API tests**: Integration tests against the Express routes confirm that endpoints enforce authentication, authorization and input validation correctly.
– **End-to-end tests**: Tools such as Cypress can automate full user flows (login, create job, update status, delete job) across the React UI and backend.
– **Socket tests**: Targeted tests verify that Socket.IO events are emitted and received properly when job applications are modified and that reconnection logic works under intermittent network conditions.

Even in the absence of a full automated test suite, the modular structure of the codebase facilitates the incremental addition of tests as the system evolves.

V. AI-POWERED RESUME SHORTLISTING

A. Overview

The AI shortlisting component is designed to analyze the similarity between a candidate's resume and a given job description. While the initial deployment may focus on keyword-based matching, the architecture supports more advanced models.

B. Feature Extraction

The resume and job description are processed using natural language processing techniques:

– Text cleaning (lowercasing, stopword removal, lemmatization).
– Extraction of skills, tools, and technologies using keyword lists or named entity recognition.
– Vectorization using TF-IDF or pre-trained embeddings (e.g., Sentence-BERT).

C. Similarity Scoring

A similarity score $s \in [0, 1]$ is computed between the resume vector $\mathbf{r}$ and job description vector $\mathbf{j}$:

$$s = \frac{\mathbf{r} \cdot \mathbf{j}}{\|\mathbf{r}\| \, \|\mathbf{j}\|} \tag{1}$$

This cosine similarity score is then mapped to qualitative labels such as *Low*, *Medium*, and *High* fit.

D. Integration into the System

The AI module can be exposed as a REST endpoint, e.g., `/api/ai/score`, that accepts a resume and job description, and returns a score and explanation. On the frontend, job cards can display an AI fit badge, and dashboards can be sorted by fit score to prioritize applications.

E. Training Data and Labeling

Training a robust resume–job matching model requires access to representative data and high-quality labels. In practice, several data sources can be considered:

– Historical application data from consenting users, where the downstream outcome (e.g., interview invitation or offer) is known.

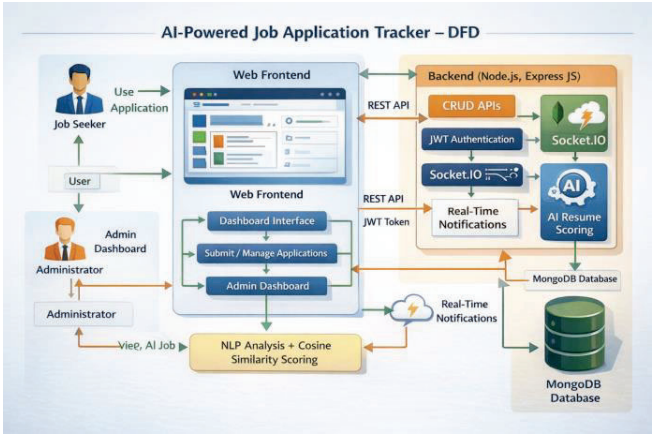

Fig. 2. Data Flow Diagram

– Publicly available resume and job description samples, labeled by human annotators with relevance scores.
– Synthetic data generated by perturbing job descriptions and resumes to introduce controlled variation in skills and experience.

Labels can take the form of binary decisions (suitable / not suitable), ordinal scores (e.g., 1–5), or pairwise preferences (resume A is a better fit than resume B for a given job). The choice of labeling scheme influences both the model family (classification, regression, or learning-to-rank) and the evaluation metrics used.

F. Explainability and Feedback

To build user trust, it is important that the AI module provide more than a single scalar score. The system can be extended to return:

– Highlighted keywords and phrases that contributed positively to the score, such as matching skills or technologies.
– Detected gaps where important keywords from the job description are missing in the resume.
– Simple natural language explanations summarizing why a match is strong or weak.

These explanations could be displayed in the UI in the form of tooltips or side panels to let candidates refine their resumes iteratively. Such feedback mechanisms are consistent with the larger literature on explainable AI, and help to address the "black box" perception of automated screening tools.

VI. EVALUATION

A. Functional Evaluation

The system was evaluated against its functional requirements:

– Users can register, log in, and maintain authenticated sessions.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

- Users can create, view, update, and delete job applications.
- Status changes are reflected immediately in the dashboard.
- Real-time notifications are delivered on job updates.
- Administrators can view all applications via the admin dashboard.

B. Performance Metrics

Preliminary performance measurements show:

- Average API response time: 150–200 ms under typical load.
- Real-time notification latency: under 100 ms on local network.
- MongoDB query latency: under 50 ms for typical dashboards.

C. AI Matching Prototype

A prototype of the resume matching model was evaluated on a small benchmark of synthetic resume–job pairs. Using cosine similarity over TF-IDF vectors achieved reasonable separation between high-fit and low-fit pairs, indicating feasibility for integration. More extensive evaluation with real datasets is left as future work.

D. User Study Design

To complement quantitative performance metrics, a small-scale user study can be conducted to assess perceived usefulness and usability. Participants (e.g., final-year students actively seeking internships or jobs) would be asked to use the system over a fixed period and then complete a questionnaire based on standard usability instruments such as the System Usability Scale (SUS). Example Likert-scale statements include:

- "I found it easy to keep track of my job applications using this system."
- "The status indicators and filters made it easy to understand where I stood in each process."
- "The AI fit indicator helped me decide which roles to prioritize."
- "I would prefer this system over my current job tracking method (e.g., spreadsheet or notes)."

Open-ended questions can obtain qualitative feedback about missing features, confusing interactions or suggestions for improvement. Although such a study would also be limited in scope, it would yield valuable insights as to the impact of the design choices made on real users.

VII. DISCUSSION

A. Benefits

The proposed system has some advantages for the job seekers:

- Tracking of all the applications in a centralized way via a single interface.
- Lower cognitive load and less missed deadlines because of notifications.
- Insight into application patterns and outcomes.
- AI-assisted guidance to prioritize and refine applications.

B. Limitations

Current limitations include:

- The AI shortlisting module is at a prototype stage and requires further tuning and validation.
- The system does not yet integrate directly with external job boards or email inboxes.
- There is no built-in resume editor; users must upload or paste existing resumes.
- The current evaluation is based on a limited dataset and controlled environment.

C. Security and Privacy Considerations

Because the system processes potentially sensitive information such as employer names, compensation ranges, job URLs, and user notes, security and privacy are critical. The following measures are adopted in the current design:

- **Transport security:** All client–server communication is intended to be deployed over HTTPS, protecting credentials and application data in transit.
- **Token-based access control:** JWTs are used to authenticate API requests; protected routes on the backend verify the token and enforce user-level access control so that users can only access their own applications.
- **Least privilege:** Admin endpoints are guarded by role checks, preventing regular users from retrieving or modifying system-wide data.
- **Input validation:** Both backend and frontend perform validation to mitigate common injection risks and malformed data.
- **Database indexing and query scoping:** Queries are always scoped by user identifier where appropriate, reducing the surface of accidental information leaks.

In a production deployment, additional controls such as encryption-at-rest, audit logging, rate limiting, and integration with identity providers (e.g., OAuth2) would be desirable.

D. Usability and UX Design

This section will cover the Usability and UX Design. The adoption of any productivity tool is anchored on user experience. The interface is developed according to the general UX principles:

- **Low-friction onboarding:** The registration and login pages are very minimal and only necessitate essential information to start.
- **Consistent navigation:** There is a persistent layout feature that gives the dashboard, job creation form and the administrator pages navigation to ease the burden on the brain.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

- **Inline feedback:** The validation errors and notification snackbars are there to give feedback on the actions performed by the users.
- **Responsive behavior:** Material-UI grid layouts and breakpoints help make sure that tables and forms are usable on smaller screens.

E. Extended Evaluation

In addition to the minimum functional testing, one can also test the system in a number of other dimensions.

1) Scalability: In order to determine scalability, we can simulate multiple parallel clients with load-testing tools and measure:

- Thruput (request per second) of job list and creation end points.
- Socket.IO connection stability with more and more connected users.
- Database index effects on the query latency with increase of number of applications.

2) User-Centric Metrics: The following user centric measures can be time based:

- Total of number of applications developed by user per week.
- Distribution of statuses and time in each stage.
- Application-to-interview and application-to-offer ratios.
- Engagement proxies (frequency of updates and logins).

These metrics may be utilized to offer personalized analytics e.g. pointing out that a user has numerous applications that are in the *Applied* phase with no follow-up.

3) AI Matching Quality: In the case of the resume shortlisting part, a more detailed assessment would imply:

- Creating a labeled dataset of resume–job pairs with human annotated fit-scores.
- TF-IDF cosine similarity compared with modern embedding-based approaches.
- Ranking quality metrics including mean reciprocal rank (MRR) and normalized discounted cumulative gain (nDCG).

The experiments would measure the ability of the AI module to give priority to really well-fit applications.

F. Threats to Validity

The main threats to validity include:

- textbfLimited dataset: The evaluation of the AI module uses synthetic or small-scale data, which may not reflect real-world variability in resumes and job descriptions.
- textbfUser sample bias: Informal user feedback has been collected primarily from students in computing disciplines, limiting generalization to other domains.
- textbfPrototype deployment environment: Performance measurements obtained in a development environment may differ from those in cloud or mobile network conditions.

These threats must be mitigated with larger and more heterogeneous datasets, user study involving users, and executed on real infrastructure.

VIII. CONCLUSION AND FUTURE WORK

This section is the conclusion of the paper and will outline the future work this paper will focus on. It is a paper that discussed a Job Application Tracker that has AI-based resume shortlisting that combines full-stack web technologies with real-time communication and AI concepts. The system gives job seekers an intelligent and centralized place to structure their job hunt and make data-based decisions, and establishes the foundations of integrating resume analytics further into daily job tracking.

REFERENCES

[1] React Documentation, ”React: A JavaScript library for building user interfaces,” [Online]. Available: https://react.dev/.
[2] Node.js Foundation, ”Node.js: JavaScript runtime built on Chrome’s V8 engine,” [Online]. Available: https://nodejs.org/.
[3] Express.js, ”Express: Fast, unopinionated, minimalist web framework for Node.js,” [Online]. Available: https://expressjs.com/.
[4] MongoDB Inc., ”MongoDB Manual,” [Online]. Available: https://www.mongodb.com/docs/.
[5] Socket.IO, ”Socket.IO Documentation,” [Online]. Available: https://socket.io/docs/v4/.
[6] Auth0, ”Introduction to JSON Web Tokens,” [Online]. Available: https://jwt.io/introduction.
[7] G. Salton and C. Buckley, ”Term-weighting approaches in automatic text retrieval,” *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.
[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proc. NAACL*, 2019.
[9] M. Anderson and T. Brown, ”Digital tools for managing job applications: A user study,” *Journal of Career Development*, vol. 49, no. 3, pp. 345–359, 2022.
[10] S. Gupta and R. Kumar, ”Applicant tracking systems and their impact on recruitment,” *International Journal of Human Resource Studies*, vol. 11, no. 2, pp. 89–105, 2021.