



CampusHire: An AI-Powered Training and Placement Management System with Explainable Skill Matching

Amit Hirole, Om Talokar, Sejal Chandekar, Sakshi Hage, and Komal Tiwari

Department of Computer Science and Engineering
Guru Nanak Institute of Engineering and Technology
Nagpur, Maharashtra, India

How to Cite this Article:

Hirole, A., Talokar, O., Chandekar, S., Hage, S. & Tiwari, K. (2026). CampusHire: An AI-Powered Training and Placement Management System with Explainable Skill Matching. International Journal of Creative and Open Research in Engineering and Management, 2(10), 1-9. <https://doi.org/10.55041/ijcope.v2i10.037>

License:

This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.



<https://doi.org/10.55041/ijcope.v2i10.037>

Abstract—Campus training and placement (T&P) activities in many engineering institutions are still coordinated through spreadsheets, e-mail and notice boards, which makes eligibility checks, shortlisting and placement reporting slow and error-prone and gives students little guidance on which opportunities suit them. This paper presents CampusHire, a web-based T&P management system that brings students, recruiters, training and placement officers and administrators onto a single role-based platform and adds an explainable skill-matching engine. The engine normalizes the skills recorded in each student profile and job posting, filters postings on academic eligibility and application deadline, and ranks the remaining postings by a skill-coverage score: the percentage of a job's required skills that the student already holds. Because the score is computed from the set of missing skills, the same computation produces a skill-gap report that tells each student what to learn for a target role. The system is implemented with HTML5, CSS3, JavaScript and Bootstrap on the client, PHP on the server and a relational MySQL schema, and protects data with bcrypt password hashing, prepared SQL statements, session hardening and role-based access control. A worked example run on the implemented matching functions shows that ineligible postings are excluded before ranking and that, unlike the Jaccard index, the coverage score is not lowered when a student lists additional unrelated skills. CampusHire offers institutions a transparent, low-cost foundation that can be extended with semantic skill matching and placement analytics.

Index Terms—Campus recruitment, content-based recommendation, explainable matching, job recommendation, placement management system, role-based access control, skill-gap analysis, web application.

I. INTRODUCTION

The training and placement (T&P) cell of an engineering institution connects students with employers. Its work spans the entire recruitment cycle: collecting student records, announcing drives, checking eligibility, forwarding applications, coordinating interviews and reporting final placements. In many institutions these activities are still managed with spreadsheets, e-mail threads, messaging groups and notice boards. This approach works for a few recruiters but does not scale as student intake, the number of visiting companies and the variety of job roles grow.

Manual coordination creates problems for every stakeholder. Placement officers spend much of their time checking eligibility criteria and compiling lists by hand. Recruiters receive applications filtered only on coarse criteria

such as branch and CGPA rather than on the skills a role requires. Students learn about opportunities late, receive little guidance on which roles fit their profile, and get no specific feedback on the skills they lack. Institutions, in turn, lack consolidated data on placement trends and cohort skill gaps with which to plan training.

Research on job recommender systems [1], [2] and person-job fit [3], [4] has produced accurate matching models. However, most of this work targets large online job markets with extensive interaction histories, and the resulting models are difficult for a college to train, deploy and explain. Studies on campus placement prediction [5], [6] estimate whether a student is likely to be placed but do not manage the placement process itself. An institution therefore needs an integrated system that runs on modest infrastructure, covers the complete placement lifecycle, and matches students to jobs in a way that students and placement officers can understand and verify.

This paper presents CampusHire, an AI-powered T&P management system designed for this setting. The main contributions are as follows:

- 1) A role-based, three-tier architecture and relational data model that support the complete placement lifecycle, from registration to final selection, for students, recruiters, placement officers and administrators.
- 2) An explainable skill-coverage matching method that combines eligibility filtering, job ranking and skill-gap analysis in a single computation, together with a formal statement of its properties.
- 3) An implementation on a widely available PHP and MySQL stack, with security controls mapped to the OWASP Top 10.
- 4) A worked evaluation of the matching method against the Jaccard index, and a functional test plan for the platform.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes the system architecture and data model, and Section IV formalizes the matching method. Section V presents the implementation, Section VI discusses the results, and Section VII concludes with directions for future work.



II. RELATED WORK

A. Placement Management Systems

Web-based placement portals adopted by institutions digitize student registration, company information, drive announcements and placement records. They reduce paperwork and give students a single point of access, but the placement officer usually still checks eligibility and shortlists candidates, and matching is limited to filters on fields such as branch and CGPA. Such portals therefore serve mainly as information-management systems rather than decision-support tools.

B. Job Recommender Systems

Siting et al. [1] and de Ruijt and Bhulai [2] survey job recommender systems and group them into content-based, collaborative-filtering, knowledge-based and hybrid approaches. Malinowski et al. [7] observed that job matching is bilateral: a good recommendation must satisfy both the candidate's preferences and the employer's requirements. Paparrizos et al. [8] learned job recommendations from the career transitions recorded in user profiles, and the RecSys Challenge 2016 [9] benchmarked recommenders on interaction logs from a professional networking platform. These methods depend on large volumes of historical interaction data. A single college has no such history at the start of each placement season. This cold-start condition favours content-based methods, which compare item attributes directly with a user profile [10].

C. Person–Job Fit and Skill Extraction

Zhu et al. [3] proposed joint representation learning of résumés and job postings to predict person–job fit. Qin et al. [4] extended this idea with an ability-aware neural network that uses hierarchical attention to weigh job requirements against candidate experience. On the input side, Yu et al. [11] extracted structured information from résumés with a cascaded hybrid model, and Guignani and Misra [12] identified implicit skills from document embeddings for job recommendation. These techniques are accurate, but they need labelled training data and considerable computation, and their scores are hard to explain to an individual student.

D. Placement Prediction and Adaptive Training

Thangavel et al. [5] built a classifier that assigns students to placement categories from academic data, and Aravind et al. [6] compared several machine-learning models for predicting the placement status of undergraduate students. Kabudi et al. [13] reviewed AI-enabled adaptive learning systems that personalize training from learner data. These studies give institutions useful insight, but they are stand-alone analyses rather than part of an operational placement workflow.

E. Transparency in Algorithmic Hiring

Raghavan et al. [14] examined vendors of algorithmic pre-employment assessments and highlighted the need for transparency and validation when algorithms influence hiring decisions. On a campus, where a recommendation can shape a

student's career, an interpretable scoring rule is easier to audit and to justify than an opaque model.

Table I summarizes how CampusHire relates to these approaches. It combines the lifecycle management of placement portals with a content-based, explainable matching method that works from the first placement season, without historical data.

TABLE I
COMPARISON OF PLACEMENT-SUPPORT APPROACHES

Capability	Manual	Portal	ML rec.	CampusHire
Centralized student and job records	No	Yes	–	Yes
Role-based access control	No	Yes	–	Yes
Automatic eligibility check	No	Partial	–	Yes
Skill-based job ranking	No	No	Yes	Yes
Skill-gap feedback to student	No	No	Partial	Yes
Score explainable to the student	–	–	Partial	Yes
Usable from the first season	Yes	Yes	No	Yes
Application status tracking	Partial	Yes	–	Yes

Manual: spreadsheets and e-mail; Portal: conventional T&P portal; ML rec.: learned recommenders such as [3], [4], [8]; –: not applicable.

III. SYSTEM ARCHITECTURE AND DESIGN

A. Stakeholders and Access Model

CampusHire serves four primary roles. Students build academic and skill profiles, upload résumés, view recommended jobs and apply. Recruiters publish job openings and manage applicants. Training and placement officers (TPOs) coordinate drives, monitor applications and maintain placement records. Administrators manage user accounts and system configuration. Faculty mentors can also follow the progress of the students assigned to them. Each account carries exactly one role, and every protected operation checks this role on the server before it executes. Table II shows the resulting role–permission matrix.

TABLE II
ROLE–PERMISSION MATRIX

Operation	Stu.	Rec.	TPO	Adm.
Maintain profile and upload résumé	Yes	–	–	–
View ranked jobs and skill gap	Yes	–	–	–
Apply and track application status	Yes	–	–	–
Post and edit job openings	–	Yes	Yes	–
Review applicants, update status	–	Yes	Yes	–
Monitor drives, generate reports	–	–	Yes	Yes
Manage users and recruiters	–	–	Yes	Yes
System configuration	–	–	–	Yes

B. Layered Architecture

Fig. 1 shows the three-tier architecture. The presentation layer, built with HTML5, CSS3, JavaScript and Bootstrap, renders role-specific dashboards in any standard browser. The application layer, written in PHP, contains the authentication and access-control logic; the modules for profiles, jobs,



applications, reports and notifications; and the matching engine described in Section IV. The data layer consists of a MySQL database and a file store for uploaded résumés. All browser traffic uses HTTPS, and every database access from the application layer uses prepared statements.

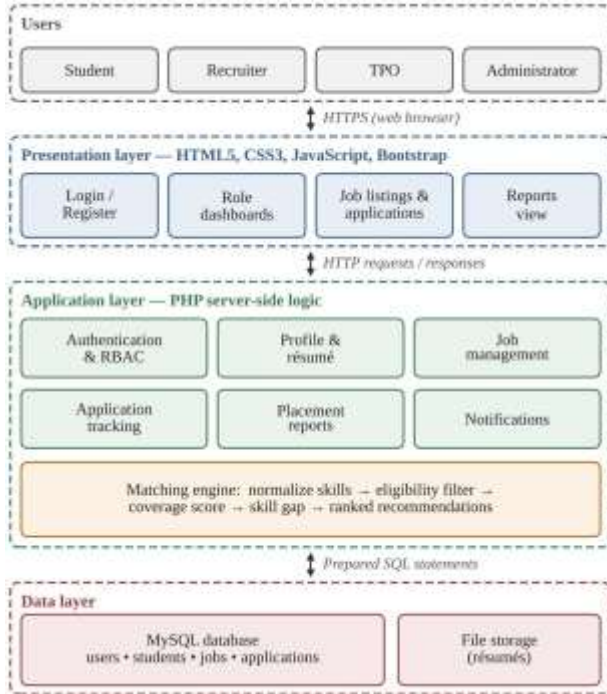


Fig. 1. Three-tier architecture of CampusHire.

C. Data Model

Table III summarizes the core relational schema [15]. Identity attributes common to all users are kept in the *users* table. Student-specific attributes are kept in a *students* table whose primary key is also a foreign key to *users*, which models a one-to-one specialization. Each job references the recruiter who posted it and stores the minimum CGPA, the required skills and the application deadline. The *applications* table links students to jobs. Its composite UNIQUE constraint on (job_id, student_id) prevents duplicate applications at the database level, and it stores the match score computed at the time of application, so recruiters can see how well each applicant fits the role.

TABLE III
CORE RELATIONAL SCHEMA

Table	Attributes	Keys and constraints
users	user_id, name, email, password, role, created_at	PK user_id; email UNIQUE; role ∈ {student, recruiter, tpo, admin}; password stored as bcrypt hash
students	student_id, branch, cgpa, skills, resume_path	PK student_id, also FK → users
jobs	job_id, recruiter_id, title, company, min_cgpa, skills, deadline	PK job_id; FK recruiter_id → users
applications	app_id, job_id, student_id, match_score, status, applied_on	PK app_id; FKs → jobs, students; UNIQUE (job_id, student_id)

D. Placement Workflow

The placement workflow has seven steps: 1) a user registers or logs in according to role; 2) the student creates a profile with academic details and skills and uploads a résumé; 3) a recruiter or TPO posts a job with eligibility criteria and required skills; 4) the system matches the student profile against active jobs and recommends suitable ones; 5) the student applies and tracks the application; 6) the recruiter shortlists candidates, and the interview and selection stages are recorded; and 7) final results update the placement records from which reports are generated. The status attribute of each application follows the lifecycle in Fig. 2. An application moves from *applied* to *shortlisted*, *interview* and *selected*, and can be *rejected* at any stage before selection.

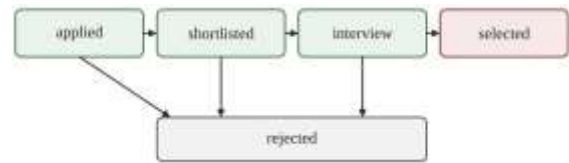


Fig. 2. Lifecycle of the application status attribute.

IV. SKILL-MATCHING AND RECOMMENDATION METHOD

The matching engine is a content-based recommender [10] that compares the skills in a student profile with those required by each job. It was designed to meet three requirements: it must work without historical placement data, respect the institution's eligibility rules, and produce a score whose meaning a student can check. Fig. 3 outlines the pipeline.

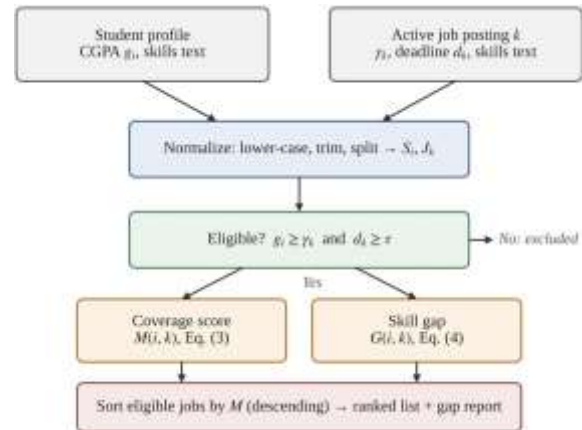


Fig. 3. Eligibility-aware skill-coverage matching pipeline.

A. Notation

Let P be the set of job postings. Student i has a CGPA $g_i \in [0, 10]$ and a free-text skill list. Posting $k \in P$ has a minimum CGPA γ_k , an application deadline d_k and a free-text list of required skills. Let τ denote the current date.



B. Skill Normalization

Skill lists are entered as comma-separated text. A normalization function v converts each token to lower case and trims surrounding white space, so that “Java ” and “java” map to the same skill. The normalized skill set of student i is

$$S_i = \{v(t) : t \in T_i, v(t) \neq \varepsilon\} \quad (1)$$

where T_i is the set of comma-separated tokens in the student’s skill list and ε is the empty string. The required skill set J_k of posting k is obtained in the same way.

C. Eligibility Filtering

Institutional eligibility rules are treated as hard constraints and applied before any scoring. The set of postings for which student i is eligible is

$$E_i = \{k \in P : g_i \geq \gamma_k \wedge d_k \geq \tau\} \quad (2)$$

so a posting whose CGPA criterion the student does not meet, or whose deadline has passed, is never recommended, however well the skills match.

D. Coverage Score and Skill Gap

For each eligible posting, the match score is the percentage of the job’s required skills that the student holds:

$$M(i, k) = 100 \cdot \frac{|S_i \cap J_k|}{|J_k|} \quad (3)$$

with $M(i, k) = 0$ when $J_k = \emptyset$. The skill gap is the set of required skills that the student lacks:

$$G(i, k) = J_k \setminus S_i \quad (4)$$

Since $|S_i \cap J_k| = |J_k| - |G(i, k)|$, the score and the gap are two views of the same computation:

$$M(i, k) = 100 \left(1 - \frac{|G(i, k)|}{|J_k|} \right) \quad (5)$$

This identity makes every score explainable in the student’s own terms. A score of 75% for a posting with four required skills means that exactly one required skill is missing, and the system names it.

E. Ranking

The recommendation list R_i contains the eligible postings sorted by $M(i, k)$ in descending order, each accompanied by its gap set $G(i, k)$. Algorithm 1 gives the complete procedure.

Algorithm 1: Eligibility-aware skill-coverage recommendation

Input: student i with CGPA g_i and skill text; postings P with γ_k, d_k and skill text; current date τ
Output: ranked list R_i of tuples (k, M, G)
1: $S_i \leftarrow \text{NORMALIZE}(\text{skills of } i)$; $R_i \leftarrow \emptyset$
2: **for each** $k \in P$ with $d_k \geq \tau$ **do**
3: **if** $g_i < \gamma_k$ **then continue** // ineligible
4: $J_k \leftarrow \text{NORMALIZE}(\text{skills of } k)$
5: **if** $J_k = \emptyset$ **then** $M \leftarrow 0$; $G \leftarrow \emptyset$
6: **else** $M \leftarrow \text{round}(100 \cdot |S_i \cap J_k| / |J_k|, 1)$
 $G \leftarrow J_k \setminus S_i$
7: append (k, M, G) to R_i
8: **end for**
9: sort R_i by M in descending order; **return** R_i
NORMALIZE(x): split x on commas; lower-case and trim each token; discard empty tokens.

F. Properties

The coverage score has four properties that suit the campus setting:

(P1) *Boundedness*: $0 \leq M(i, k) \leq 100$, and $M(i, k) = 100$ if and only if $J_k \subseteq S_i$.

(P2) *Monotonicity*: if $S_i \subseteq S_{i'}$, then $M(i, k) \leq M(i', k)$; acquiring a skill can never lower a student’s score.

(P3) *Independence from unrelated skills*: skills outside J_k do not affect $M(i, k)$.

(P4) *Constraint safety*: by (2), no ineligible posting can enter R_i .

Property (P3) distinguishes the coverage score from the Jaccard index [16], [17], a common set-similarity measure:

$$Jac(S_i, J_k) = \frac{|S_i \cap J_k|}{|S_i \cup J_k|} \quad (6)$$

Because the denominator of (6) grows with every skill in $S_i \setminus J_k$, the Jaccard index penalizes a student with a broad profile, even though the employer is indifferent to those extra skills. Section VI-B quantifies this effect.

G. Computational Complexity

With hash-based set operations, computing (3) and (4) for one posting costs $O(|S_i| + |J_k|)$. A recommendation request therefore costs $O(\sum_k (|S_i| + |J_k|))$ for scoring plus $O(|E_i| \log |E_i|)$ for sorting. At campus scale, with short skill lists and at most a few hundred active postings in a season, recommendations can be computed on demand without precomputation or caching.

V. IMPLEMENTATION

A. Development Environment and Methodology

CampusHire was developed with a stage-wise methodology [18]: requirement analysis, system design, database design, module implementation, integration of the AI-based features, implementation of the recruitment workflow, and finally testing and deployment. Visual Studio Code was used for coding, XAMPP (Apache, MySQL and PHP) provided the local server environment, phpMyAdmin was used to administer the database, and Google Chrome and Microsoft Edge were used for cross-browser testing. Development requires a computer with at least an Intel Core i5 or equivalent processor,



8 GB of RAM and 256 GB of SSD storage. Users only need a device with a dual-core processor, 2 GB of RAM and a modern web browser.

B. Frontend

The frontend comprises the home, login, registration, dashboard, job-listing, profile and application pages. Bootstrap's responsive grid adapts the layout to desktops, tablets and phones. JavaScript validates forms before submission. For example, the registration form rejects malformed e-mail addresses, passwords shorter than eight characters and CGPA values outside the range 0–10, and displays all errors together. Fig. 4 and Fig. 5 show representative screens.



(a) Registration with role selection



(b) Student profile with preferred locations and target roles

Fig. 4. Student-facing screens of CampusHire.

C. Backend and Matching Module

The backend is organized as one PHP script per function. The scripts share a configuration script that opens a UTF-8 MySQL connection through the mysqli extension and starts the session. After successful authentication, the login script redirects the user to the dashboard for that user's role. The recommendation module implements Algorithm 1 directly: it retrieves the student's CGPA and skills, fetches the jobs whose deadline has not passed, skips postings whose minimum CGPA exceeds the student's, computes the score and the missing skills with PHP's array intersection and difference functions, and sorts the results by score. When a student applies, the score is stored in the *applications* table and shown next to the applicant in the recruiter's view (Fig. 5).



Fig. 5. Recruiter dashboard with open roles, applications, review queue, offers and pipeline summary.

D. Security

Security controls were mapped to the OWASP Top 10:2021 categories [19], as summarized in Table IV. Passwords are stored only as bcrypt hashes produced by PHP's password-hashing API [20]. All queries use prepared statements with bound parameters, which keeps SQL code separate from user data. The session identifier is regenerated on login to prevent session fixation, and every protected script checks the role held in the server-side session; for example, the job-posting script redirects any user who is not a recruiter to the login page. A failed login always produces the same generic message, so the response does not reveal whether an e-mail address is registered.

TABLE IV
SECURITY CONTROLS MAPPED TO OWASP TOP 10:2021

Risk category	Control in CampusHire
A01 Broken access control	Role held in server-side session and checked by every protected script
A02 Cryptographic failures	bcrypt password hashes (password_hash / password_verify); HTTPS for data in transit
A03 Injection	mysqli prepared statements with bound parameters for all user input
A07 Identification and authentication failures	Session ID regenerated at login; one generic error for any failed login
Input validation (all)	Client-side checks repeated on the server before data is stored

E. Deployment

For production, the code is versioned on GitHub and deployed to a PHP-capable web host or a cloud virtual machine, for example on AWS or Google Cloud. The schema is imported into a server-side MySQL instance, the database credentials and domain name are configured, and an SSL/TLS certificate is installed to enable HTTPS before the live site is tested. Users then need nothing beyond a web browser. The main deployment risks are server misconfiguration, database connection errors and dependence on network connectivity.

VI. RESULTS AND DISCUSSION

A. Functional Outcome

The implemented system replaces scattered spreadsheets, e-mail and notice boards with a single platform that each stakeholder reaches through a role-specific dashboard. Students maintain profiles and target roles (Fig. 4), recruiters manage postings and follow their applicant pipeline (Fig. 5), and TPOs



and administrators oversee drives, users and placement records. Every application is recorded with its match score and status, so the progress of each drive can be traced from a single source of data.

B. Worked Example of the Matching Method

To illustrate the method, consider a student with a CGPA of 7.80 and the skill set $S = \{\text{java, python, sql, html, css, git}\}$, together with the five postings in Table V. The values in Table V were obtained by running the system's matching functions on these inputs. Posting J4 is excluded at the eligibility step because its minimum CGPA of 8.0 exceeds the student's, even though its coverage score of 66.7% would have ranked it second. The remaining postings are ranked J5 (100%), J1 (75%), J3 (50%) and J2 (40%). Each score is explained by its gap set: for the backend role J1, the student meets three of the four required skills and lacks only Spring Boot, which is a specific and actionable training target.

TABLE V
WORKED EXAMPLE: STUDENT WITH CGPA 7.80

Job (min. CGPA)	Required skills	M (%)	Jac. (%)	Skill gap G
J1 Backend Developer (7.0)	Java, Spring Boot, SQL, Git	75.0	42.9	spring boot
J2 Data Analyst (6.5)	Python, SQL, Excel, Power BI, Statistics	40.0	22.2	excel, power bi, statistics
J3 Frontend Developer (6.0)	HTML, CSS, JavaScript, React	50.0	25.0	javascript, react
J4 Software Engineer (8.0)†	Java, Python, Data Structures	(66.7)	(28.6)	data structures
J5 Web Developer Intern (6.0)	HTML, CSS, Git	100.0	50.0	–

Student skills: Java, Python, SQL, HTML, CSS, Git. †Ineligible (7.80 < 8.0): excluded before ranking; scores shown for reference only. Resulting ranking: J5 > J1 > J3 > J2.

The example also illustrates property (P3). Adding four skills unrelated to J1 (Photoshop, Tableau, C++ and Linux) to the student's profile leaves the coverage score for J1 unchanged at 75.0%, whereas the Jaccard index falls from 42.9% to 27.3%. In this example both measures produce the same ranking, but the Jaccard values are lower and depend on skills that the employer does not require. Under the Jaccard index, a student who broadens a profile would therefore appear to fit a role less well, an effect that the coverage score avoids.

C. Functional Testing

Table VI lists the functional test cases derived from the requirements. They cover registration and input validation, authentication, role-based access control, job posting, eligibility-aware recommendation, prevention of duplicate applications, status tracking and resistance to SQL injection. Integration testing checks the data flow between modules, for example that a status change made by a recruiter appears on the student's dashboard, and database testing checks referential integrity and the uniqueness constraint on applications.

TABLE VI
FUNCTIONAL TEST CASES

ID	Test scenario	Expected result
TC1	Register a student with valid details	Account created; user can log in
TC2	Register with malformed e-mail, 6-character password or CGPA 11	Submission blocked; all errors shown
TC3	Log in with correct credentials	Session created; role dashboard opened
TC4	Log in with a wrong password	Generic error; no session created
TC5	Student opens the job-posting page	Redirected to login page
TC6	Recruiter posts a job with skills and deadline	Job stored and shown to eligible students
TC7	Student below a job's minimum CGPA views recommendations	Job not listed
TC8	Student applies twice to the same job	Second application rejected by UNIQUE key
TC9	Recruiter sets status to shortlisted	New status shown on student dashboard
TC10	Enter 'OR '1'='1 as the login e-mail	Treated as literal text; login fails

D. Discussion and Limitations

Compared with manual coordination, CampusHire automates the most repetitive parts of a placement officer's work, namely eligibility checking, initial skill-based screening and status tracking, and it gives students a transparent account of where they stand for each role. The system runs on commodity LAMP-stack hosting and needs no training data, which keeps the cost of adoption low.

The current method also has limitations. First, skills are matched as exact strings after normalization, so synonyms and abbreviations are not recognized: a student who lists "JS, React" scores 50% for a job that requires "JavaScript, React". Second, all required skills carry equal weight; the method neither distinguishes mandatory from preferred skills nor captures proficiency levels. Third, skills are self-reported in the profile; résumés are stored, but their text is not yet parsed to verify or enrich the skill list. Fourth, the matching is one-sided: it measures how well a student meets a job's requirements but does not model the student's own preferences, which bilateral approaches take into account [7]. Finally, the method has not yet been evaluated against actual placement outcomes, which requires data collected over at least one placement season.

VII. CONCLUSION AND FUTURE WORK

This paper presented CampusHire, a web-based training and placement management system that brings students, recruiters, placement officers and administrators onto a single role-based platform and adds an explainable, eligibility-aware skill-matching engine. The coverage score is bounded, monotone, unaffected by unrelated skills and tied directly to a skill-gap report, so every recommendation can be explained to the student it concerns. Built on HTML5, Bootstrap, PHP and MySQL, with security controls aligned to the OWASP Top 10, the system gives institutions a low-cost, transparent foundation for digitizing campus placement.



Future work will address the limitations identified above. Skills will be normalized against a standard taxonomy such as ESCO [21], and semantic matching with sentence embeddings [22] will recognize skills expressed in different words. Mandatory and preferred skills will be weighted separately, and skills will be extracted automatically from résumés [11], [12]. Placement-outcome data collected by the system will support predictive analytics [5], [6] and periodic fairness audits of the recommendations [14]. Planned functional extensions include AI-assisted mock interviews with personalized feedback, automated interview scheduling, online assessments, a mobile application with push notifications, and a user study to measure time savings and how often students and recruiters act on the recommendations.

ACKNOWLEDGMENT

The authors thank Dr. Sushma Telrandhe, Head of the Department of Computer Science and Engineering, and Dr. Kishor Porate, Principal, Guru Nanak Institute of Engineering and Technology, Nagpur, for their constant encouragement and support, and the teaching and non-teaching staff of the department for their help throughout this project.

REFERENCES

- [1] Z. Siting, H. Wenxing, Z. Ning, and Y. Fan, "Job recommender systems: A survey," in *Proc. 7th Int. Conf. Comput. Sci. Educ. (ICCSE)*, Melbourne, VIC, Australia, 2012, pp. 920–924, doi: 10.1109/ICCSE.2012.6295216.
- [2] C. de Ruijt and S. Bhulai, "Job recommender systems: A review," 2021, *arXiv:2111.13576*.
- [3] 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, no. 3, pp. 1–17, 2018.
- [4] 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. Res. Develop. Inf. Retrieval*, Ann Arbor, MI, USA, 2018, pp. 25–34, doi: 10.1145/3209978.3210025.
- [5] S. K. Thangavel, P. D. Bkaratki, and A. Sankar, "Student placement analyzer: A recommendation system using machine learning," in *Proc. 4th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Coimbatore, India, 2017, doi: 10.1109/ICACCS.2017.8014632.
- [6] T. Aravind, B. S. Reddy, S. Avinash, and G. Jeyakumar, "A comparative study on machine learning algorithms for predicting the placement information of under graduate students," in *Proc. 3rd Int. Conf. I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, Palladam, India, 2019.
- [7] J. Malinowski, T. Keim, O. Wendt, and T. Weitzel, "Matching people and jobs: A bilateral recommendation approach," in *Proc. 39th Annu. Hawaii Int. Conf. Syst. Sci. (HICSS'06)*, Kauai, HI, USA, 2006, vol. 6, p. 137c.
- [8] I. Paparrizos, B. B. Cambazoglu, and A. Gionis, "Machine learned job recommendation," in *Proc. 5th ACM Conf. Recommender Syst. (RecSys)*, Chicago, IL, USA, 2011, pp. 325–328.
- [9] F. Abel, A. Benczúr, D. Kohlsdorf, M. Larson, and R. Pálóvics, "RecSys Challenge 2016: Job recommendations," in *Proc. 10th ACM Conf. Recommender Syst. (RecSys)*, Boston, MA, USA, 2016, pp. 425–426.
- [10] P. Lops, M. de Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Boston, MA, USA: Springer, 2011, pp. 73–105.
- [11] K. Yu, G. Guan, and M. Zhou, "Resume information extraction with cascaded hybrid model," in *Proc. 43rd Annu. Meeting Assoc. Comput. Linguistics (ACL'05)*, Ann Arbor, MI, USA, 2005, pp. 499–506, doi: 10.3115/1219840.1219902.
- [12] A. Gugnani and H. Misra, "Implicit skills extraction using document embedding and its use in job recommendation," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 8, 2020, pp. 13286–13293, doi: 10.1609/aaai.v34i08.7038.
- [13] T. Kabudi, I. Pappas, and D. H. Olsen, "AI-enabled adaptive learning systems: A systematic mapping of the literature," *Comput. Educ.: Artif. Intell.*, vol. 2, Art. no. 100017, 2021, doi: 10.1016/j.caeai.2021.100017.
- [14] M. Raghavan, S. Barocas, J. Kleinberg, and K. Levy, "Mitigating bias in algorithmic hiring: Evaluating claims and practices," in *Proc. Conf. Fairness, Accountability, Transparency (FAT*)*, Barcelona, Spain, 2020, pp. 469–481.
- [15] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [16] P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.
- [17] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [18] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [19] OWASP Foundation, "OWASP Top 10:2021." [Online]. Available: <https://owasp.org/Top10/>
- [20] The PHP Group, "PHP manual: Password hashing." [Online]. Available: <https://www.php.net/manual/en/book.password.php>
- [21] European Commission, "ESCO: European skills, competences, qualifications and occupations." [Online]. Available: <https://esco.ec.europa.eu/>
- [22] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. Conf. Empirical Methods Natural Lang. Process. and 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, Hong Kong, China, 2019, pp. 3982–3992.