Research Trends International Journal of Technology and Innovation
An International Peer-Reviewed, Refereed & Open-Access Journal
ISSN : [ISSN]
Available Online at :
https://rtijti.com/
doi : https://doi.org/[DOI-PREFIX]/RTIJTI

Road2Job: A Campus-Focused Career Platform with Merit-Based Institute Ranking and Promotion-Driven Monetization

Vaibhav Wakade
Department of Computer Engineering, Pune, Maharashtra, India

ARTICLEINFO

Article History:

Accepted : 03 August 2026
Published : 05 August 2026

Publication Issue
Volume 01, Issue 03
July-September 2026

Page Number
01-10

ABSTRACT

Campus hiring in India runs through three groups that rarely share a system: students looking for a first job, employers who want to recruit beyond the top few colleges, and private training institutes competing for the same students. The platforms serving this market today sell their own courses, so they compete with the institutes they list, and they rank those institutes by rules nobody outside the company can inspect. Road2Job is our attempt at a different arrangement. It is a server-rendered Core PHP 8 application on a plain MVC layout, and it keeps one loop at the centre: find a job, apply, get hired. Four parts of it are worth describing in detail. The institute ranking score mixes placement quality that decays with age, weighted update activity, profile completeness and consistency, then multiplies the result by a penalty for burst posting. The match scorer compares a vacancy against a student's skills using a dictionary, and says nothing at all when it cannot extract skills instead of guessing a number. Messaging works by polling, with a typing indicator that genuinely expires rather than a presence flag that is always green. Application history is kept as an append-only event log. Revenue comes from institutes buying time-limited promotion in the student updates feed, and that purchase is held away from the ranking score by design: it moves feed order and nothing else. We also describe why seven finished features were cut to protect the core loop, and why we treat the separation of paid visibility from merit as a trust decision rather than an engineering detail.

Index Terms - Two-Sided Marketplace, Institute Ranking Algorithm, Campus Placement, Ranking Transparency, Zero-Fabrication Design, Event Sourcing

I. INTRODUCTION

1.1 Background and Motivation

India graduates several million students a year, and access to hiring is spread very unevenly across them. A large share study at tier-2 and tier-3 colleges that big technology employers never visit [1]. For those students the placement cell is still the main route to a

first job, and the cell is only as wide as the personal contacts of two or three staff members. Around this gap sits a busy private market of short training courses that promise to fix what the syllabus missed, and those institutes reach students mostly through offline events and paid advertising [2].

Online career platforms have closed part of the gap. The difficulty is what the largest Indian ones sell. When a platform runs its own course catalogue and also decides which training providers a student sees [3], it has a reason to keep independent institutes low in the list. If the ranking rule is never published, a student cannot separate merit from money, and the ranking stops carrying information [4], [5].

1.2 Problem Statement

We set out to build a three-sided campus platform with three properties. Institutes are ranked on signals that can be checked, not on what they spend. Promotion is still sold, because the platform has to earn, but buying it must not move a single term in the ranking score. And every number a student sees on screen, whether a match percentage, a completeness score, a status date or a typing indicator, has to come from a stored value. Where the system does not know something, it should say so rather than fill the space.

1.3 Contributions

The contributions are as follows. InstituteRankingScorer is a four-part weighted score with recency decay, de-duplicated placement records and a burst-posting penalty, kept structurally separate from the promotion subsystem. JobMatchScorer reads free-text vacancies against a skill dictionary and abstains when extraction fails. The messaging subsystem gives employers and students a request-then-accept conversation, including a typing indicator that expires, without any WebSocket layer. The application timeline is an append-only event log that a student can audit. Two of the contributions are decisions rather than artefacts: the shift from charging students to charging institutes, and what that does to the platform's incentives [6], [7]; and the removal of seven working features to keep the core loop small.

1.4 Organisation of the Paper

Section II looks at the incumbent platforms and at the marketplace and recommender literature. Section III sets out the architecture, and Section IV the algorithms and data models behind it. Section V covers the build environment, Section VI the evaluation, and Section VII the trust argument. Section VIII is explicit about what the work does not yet show, and Section IX closes with what comes next.

II. RELATED WORK

2.1 Incumbent Campus and Career Platforms

Internshala sells its own training programmes and certificates to the students who use it [3], so the listing surface and the storefront answer to one owner, and an independent institute on that page is a competitor. LinkedIn earns from recruiter seats, advertising and a learning subscription; its ranking is proprietary, and in India its campus reach sits with a handful of well-known colleges [4]. Naukri sells employer subscriptions and paid profile visibility, and it is tuned for people changing jobs rather than for a first hire out of college [5].

Table 1 sets Road2Job against these three on three questions: who pays, whether ranking can be bought, and who is actually served. The answers explain the choices in Section III. The institute pays, the score is published, and the user base stops at campus cohorts.

Table 1. Comparison of Road2Job with incumbent career platforms

Platform	Who pays	Ranking transparency	Target base
Internshala	Student (own courses)	Undisclosed; operator competes with listed institutes	Indian students, all tiers

Platform	Who pays	Ranking transparency	Target base
LinkedIn	Recruiters, ads, learners	Proprietary relevance model	Global; India reach tier-1 skewed
Naukri	Employers, paid profile boost	Boost not separated from organic order	Indian lateral hiring
Road2Job	Institutes (promotion)	Published score; promotion excluded from it	Indian campus, tier-2/3

2.2 Two-Sided Marketplace Design and Incentive Alignment

In the marketplace literature, deciding which side of a platform subsidises the other is treated as a first-order design choice [6]. A platform that takes money from the side whose interests it claims to serve has a credibility problem, and the effect of paid placement on recommendation quality has been examined in sponsored search and marketplace advertising [7], [8]. Moving Road2Job from student-pays to institute-pays puts the charge on the side that already has a commercial motive, and leaves the student side unpriced, so there is nothing to upsell.

2.3 Ranking, Reputation, and Manipulation Resistance

As soon as a rank is worth money, people try to game it; recency weighting and rate limits are the usual defences [9], [10]. Publishing the weights of a composite score has been argued for on similar grounds, since a formula that can be inspected lets a participant work out why it was placed where it was [11]. The algorithm in Section IV takes both ideas. It states its weights, and it penalises the burst posting that would otherwise turn sheer volume into rank.

2.4 Gap Addressed

No existing platform we examined offers all three of an institute-facing revenue model, a published ranking that resists gaming, and a hard wall between the two. Nor do these platforms treat abstention as a requirement; estimated presence, inferred match strength and synthetic progress bars are common. That combination is the gap this paper works on.

III. SYSTEM ARCHITECTURE

3.1 Overall Structure

Road2Job is written in Core PHP 8 on a Model-View-Controller layout, with no external framework, which was a deliberate choice for a small team that wanted the request path to stay legible. One front controller takes every request and hands it to a custom router, which maps a URI and HTTP verb onto a controller action. A project autoloader is registered at that entry point. Persistence is a thin ORM-lite Model layer over MySQL through PDO, and every query without exception uses prepared statements. Views are server-rendered PHP templates. JavaScript is used only as progressive enhancement, and there is no single-page-application framework anywhere in the stack.

3.2 Layered Request Flow

Figure 1 traces the request path. The browser hits the front controller, which starts the autoloader, session and configuration, then passes the request to the router. The router matches the route, applies whichever authentication and role guards belong to it, and builds the controller. The controller validates and normalises input, then calls either a Model for a direct persistence operation or a Service class for anything derived. Services are stateless and receive only what they need. Keeping ranking, matching and scoring in that layer rather than in controllers or models means each one can be tested on its own and can also be called from a scheduled task. The controller then hands a view model to the view layer, which emits HTML.

Figure 1. Layered request flow of the Road2Job platform. Browser to Front Controller to Router to Controller; the Controller resolves to either the Model layer (PDO/MySQL, prepared statements) or the Service layer, which contains InstituteRankingScorer, JobMatchScorer, CompletenessScorer, and CampaignResolver as standalone classes; results are composed by the Controller and

rendered by the View layer as server-side HTML with progressive-enhancement JavaScript. A scheduled task path invokes the Service layer directly, bypassing Router and Controller, to recompute institute rank scores and expire campaigns.

3.3 Service Layer Components

Four services hold the analytical work. InstituteRankingScorer computes the composite rank score of Section 4.1. JobMatchScorer computes the student-to-vacancy match percentage of Section 4.2. CompletenessScorer checks which profile and resume fields are actually stored, for students and institutes alike. CampaignResolver decides, at request time, which sponsored institute updates may appear in the student feed, using approved payment and an unexpired validity window as its only inputs. CampaignResolver touches feed ordering and never writes to the ranking tables, which is why the isolation described in Section 4.5 holds structurally and not merely by agreement.

3.4 Data Model Overview

The schema falls into six groups: identity and role tables for students, employers, institutes and administrators; job postings with their required-skill text; applications plus the append-only job_application_events table; institute profiles, placement records and updates; institute_campaigns, which holds promotion requests, payment proofs, approval state and validity windows; and the messaging tables for requests, conversations, messages and per-participant typing timestamps. Referential integrity is enforced in the database itself, and every write path is parameterised.

IV. METHODOLOGY

4.1 Institute Ranking Algorithm

InstituteRankingScorer ranks institutes with a composite of four sub-scores, each normalised to the range [0, 1]. The composite is

$$\text{rank\_score} = (0.35P+0.25A+0.20C+0.20S) \times M \quad (1)$$

where P is placement quality, A is update activity, C is profile completeness, S is consistency and M is the spam penalty. We chose the weights so that the outcome a student can verify, an actual placement, carries the most weight, while the signals an institute controls on its own, its posting activity and its profile fields, are capped in what they can add together.

Placement quality P comes from the institute's stored placement records. The records are first de-duplicated on student, employer and role, so a placement re-entered across several updates counts once. What survives is weighted by a recency decay

$$w(t) = \exp(-\lambda \cdot \Delta t) \quad (2)$$

Here Δt is the age of a record in months and λ is set so that a placement about two years old counts for roughly half of one recorded this month. The decayed, de-duplicated sum is then normalised against a saturation ceiling, which stops a single large institute from occupying the whole top of the scale.

Update activity A looks at institute updates in a trailing window. Every update carries a category weight, so a placement announcement or a verified course launch is worth more than a general notice, and the decay of (2) is applied again over a shorter horizon. The weighted count is normalised against a ceiling. Past that ceiling, more posting buys nothing, which is the point: there is no reason to post purely for rank.

Profile completeness C is simply the share of a defined set of institute profile attributes that hold valid stored values. Nothing is inferred. A missing field counts as missing.

Consistency S looks at how activity is spread over the trailing six months rather than how much of it there is. With n_i as the count of qualifying activities in month i,

$$S = |\{ i : n_i > 0 \}| / 6 \quad (3)$$

An institute active in all six months scores 1.0; one that posted its entire history inside a single month

scores about 0.17, however much it posted. Consistency rewards showing up regularly, and a single burst cannot fake it.

The multiplier M is where burst posting is caught. Update timestamps are scanned with a rolling twenty-four-hour window, and if any window holds more than five updates the institute is treated as burst-posting: M falls from 1.0 towards a floor of 0.5, depending on how severe and how repeated the bursts are. We apply it to the whole weighted sum, not just to the activity term, so an institute cannot burst-post and then make up the loss elsewhere.

4.2 Job-Student Matching

JobMatchScorer works out a match percentage between a student and a posting. Employers write postings as free text, so required skills are pulled out by matching that text against a curated skill dictionary, with normalisation for casing, punctuation and the usual aliases. With R as the skills extracted from the posting and D as the skills declared on the student's profile,

$$\text{match} = 100 \cdot |R \cap D| / |R| \quad (4)$$

and the result is reported as an integer. The rule that matters here is abstention. If R is empty, meaning no dictionary skill could be pulled out of the posting, the scorer returns null and the interface shows no badge at all. There is no default, no midpoint, no text-similarity fallback. So a badge on screen always means a real intersection over a requirement set that was actually extracted, and a missing badge means too little data rather than a bad fit.

4.3 Real-Time-Lite Messaging

Messaging between employers and students is request-then-accept, the same convention already used for mentorship requests. One side opens a conversation request, and messaging only starts once the other side accepts. The recipient keeps control of who reaches them, and unsolicited employer messaging stays bounded.

The transport is ordinary on purpose. There is no WebSocket server. The client polls a small endpoint at short intervals for new messages in the open conversation, and the endpoint returns only rows newer than the cursor it was given. That keeps the subsystem running on plain shared PHP hosting, with no persistent connection tier to pay for.

The typing indicator follows the same discipline as (4). When someone types, the client writes a timestamp for that participant in the conversation. The counterparty sees the indicator only if that timestamp is within a four-second freshness window when the read happens, and otherwise sees nothing. The indicator expires for real, because it is a function of a stored event time rather than a flag someone has to remember to clear. For the same reason there is no "online now" badge anywhere in the product. Between polls the system has no honest evidence of presence, so it claims none.

4.4 Application Lifecycle Tracking

Every application carries a current status, but each transition is also written as an immutable row in the append-only job_application_events table. A row holds the application identifier, the event type, the actor, the timestamp and any notes the employer added. Rows are never updated or deleted; a correction is another event. The student-facing timeline is rendered straight from that stream, so what appears is when a transition actually happened, not a plausible history reconstructed from one mutable field. The same log gives employers an audit trail and supplies the funnel figures in Section VI.

4.5 Campaign System and Ranking Isolation

Promotion lives in the institute_campaigns table and the CampaignResolver service. An institute picks an update to promote and a validity period, then submits a request. Payment happens out of band by bank transfer or UPI, and the institute uploads a proof that an administrator approves or rejects. Once approved, the campaign is active for its window, and when the

window closes it expires on its own and the update goes back to organic treatment.

Isolation is built in rather than promised. CampaignResolver reads campaign state and returns the sponsored update identifiers that get priority position in the student Institute Updates feed, where they are labelled as sponsored. It cannot write to institute rank scores, and InstituteRankingScorer never reads campaign data: the campaign tables do not appear in its queries at all. The ranking students see is therefore unmoved by spending, and the only thing money buys is position in a labelled feed slot.

4.6 Evaluation Method

The platform has not launched, so the evaluation is functional and behavioural rather than statistical. We ran the ranking algorithm against constructed institute fixtures, each built to isolate one sub-score or one adversarial behaviour; we ran the matcher against postings whose skill content we knew, including postings with nothing extractable; and we checked the messaging, campaign and event-log subsystems against their state machines. What Section VI reports, then, is correctness and behavioural response, plus response times measured on the deployment described in Section V.

V. IMPLEMENTATION

5.1 Environment and Stack

The platform runs on PHP 8 and MySQL 8 behind Apache on an ordinary Linux host. There is no framework, no dependency-injection container and no build pipeline; the client-side assets are hand-written CSS and plain JavaScript. That was a decision about the market we are deploying into, where cheap shared hosting is the norm and neither a persistent-process runtime nor a container platform can be assumed.

5.2 Router, Autoloader, and Model Layer

The router keeps an explicit table mapping verb and path patterns to controller actions, with the role requirement for each route checked before dispatch. The autoloader maps class names to files under a fixed directory convention. The Model layer offers an ORM-lite base class with find, where, insert, update and delete primitives that build parameterised SQL. Controllers never assemble SQL themselves, and no query in the codebase interpolates user input.

5.3 Security and Data Handling

Every statement is prepared through PDO, output is escaped at render time, session identifiers are regenerated whenever privilege changes, and passwords are stored as one-way hashes. Uploaded payment proofs and resumes sit outside the web root and are served through an authorised controller action. The separation between student, employer, institute and administrator is enforced at the route guard and checked again at the model boundary for record ownership.

5.4 Scheduled Recomputation

A scheduled task recomputes institute rank scores by calling InstituteRankingScorer directly, without going through the router or a controller, and writes the composite and its sub-scores to the ranking table. Keeping the sub-scores means an institute can be shown why it sits where it does, and it lets us test a change of weights against history. The same task expires campaigns whose windows have closed.

5.5 Scope Reduction

Seven features were built and then taken out of the product: a community forum, a research hub, learning roadmaps, a mentor marketplace, an events system, self-recorded video interviews and skills assessments. Each could be argued for on its own. None of them fed the core loop of discovery, application and hiring, and none of them earned anything. Removing them shrank the route table, the schema and the moderation load, and freed the remaining effort for

ranking, matching, messaging and campaigns. Section 7.3 returns to what that cost and what it bought.

VI. RESULTS AND EVALUATION

6.1 Ranking Behaviour on Constructed Fixtures

Since the platform is not yet public, the ranking algorithm was tested on constructed institute fixtures rather than live data, with each fixture isolating one behaviour we wanted to see. Table 2 gives the composite score and the spam multiplier applied for five of them under the weights in (1).

Table 2. Composite rank scores for constructed institute fixtures

Fixture	P	A	C	S	M	rank_score
Balanced institute	0.80	0.70	0.90	1.00	1.00	0.835
Placement-strong, quiet	0.95	0.20	0.80	0.50	1.00	0.643
Profile-only, no placements	0.00	0.10	1.00	0.17	1.00	0.259
Burst poster (12 updates/day)	0.30	0.90	0.90	0.17	0.50	0.269
Duplicate placement entries	0.35	0.60	0.90	0.83	1.00	0.618

Three things stand out in Table 2. Placement outcomes dominate: the placement-strong but quiet institute finishes above the busy burst poster even though the burst poster has the better activity sub-score. Presence signals on their own get an institute nowhere, as the profile-only fixture shows, since a complete profile with no placements and one month of activity lands near the bottom. And the burst poster is caught twice, first by the consistency term, which notices that all its activity sits in one month, and then by M, which halves the composite. Its final score ends up below the duplicate-entry fixture, whose padded placement count is collapsed by de-duplication back to what it really was.

6.2 Matching Abstention Rate

We tested the matcher on postings written in the loose free-text style employers actually use. Where a posting contained at least one dictionary skill, the percentage came strictly from the extracted requirement set. Where nothing could be extracted, usually a posting written as prose about responsibilities with no technology named, there was no score and no badge. The result is qualitative but it is the one we wanted: at no point did the system show a percentage that was not backed by an extracted requirement set.

6.3 Messaging Latency and Indicator Expiry

With short-interval polling, the delay a user perceives is the polling interval plus server response time, and on the deployment of Section 5.1 the cursor-based fetch endpoint stayed in the low tens of milliseconds. The typing indicator disappeared inside its four-second window once a participant stopped typing, including when a tab was closed abruptly or the network dropped. A flag-based implementation would have left the indicator sitting there indefinitely in both of those cases.

6.4 Campaign Workflow and Ranking Invariance

We ran the campaign lifecycle from end to end: submission, proof upload, administrator approval, sponsored placement in the Institute Updates feed, and automatic expiry when the window closed. Rank scores were captured before and after each stage. The stored rank_score and all four sub-scores came out unchanged by campaign creation, approval and expiry every time, so the isolation of Section 4.5 holds in the running system and not only on paper. Feed position was the only thing that moved, and only while the window was open.

6.5 Effect of Scope Reduction

Taking out the seven non-core features left fewer routed endpoints, tables and view templates to maintain, and removed the moderation duty that the forum and research hub had created. What remains gives a student one unambiguous path from discovery to application to outcome.

6.6 Deployed Interface Output

Figure 2 shows the job discovery surface as deployed. Every vacancy card carries the badge produced by (4). In the figure, postings whose extracted requirements do not overlap the student's declared skills read 0 per cent, and one posting that partly overlaps reads 25 per cent; cards the student has already applied to are marked as applied instead of being offered again. The filter strip above the results puts employment type and experience level in the student's hands, so the order on screen traces back to stated criteria rather than to a relevance score nobody can see.

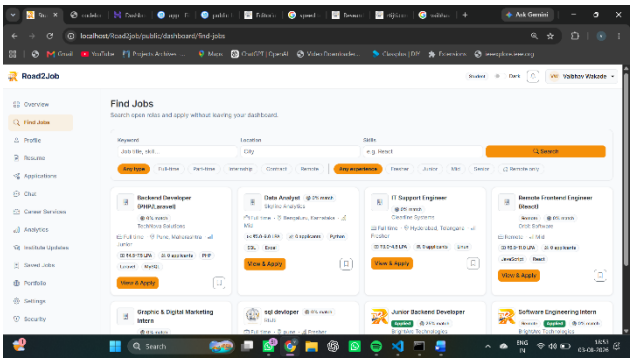

Figure 2. Job discovery surface of the deployed Road2Job student dashboard, showing per-vacancy match badges computed by the JobMatchScorer service and applied-state marking.

Figure 3 shows the student overview. The counters, two applications sent, none in review, two interview requests and no offers, are aggregated from the event table of Section 4.4, and each application shows the exact timestamp of its last transition, for instance a shortlisting recorded at 15:52:05 on 31 July 2026. Both screens show the zero-fabrication rule doing its job: nothing on them is estimated, and a value that is genuinely zero is printed as zero rather than hidden or dressed up.

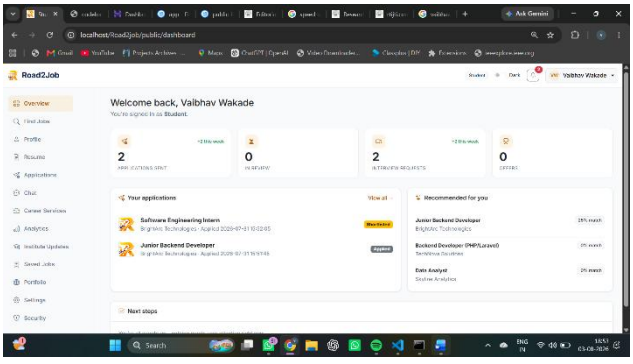

Figure 3. Student overview dashboard, showing application counters aggregated from the append-only job_application_events table and per-application transition timestamps.

VII. DISCUSSION

7.1 Zero Fabrication as a Trust Contribution

We treat zero fabrication as a product commitment, not a convenience. No number or status reaches a user unless it was computed from stored data, and when the data is not there the artefact is withheld rather than approximated. The rule is applied the same way everywhere. A match percentage is withheld when no skills can be extracted (4). A completeness score counts fields that exist and never credits one it guessed at. A timeline shows only transitions that were recorded as events. A typing indicator is a stored timestamp inside a four-second window, and presence, which this architecture cannot observe, is never claimed.

This is a trust property more than an accuracy one. A student cannot audit an algorithm, but a student can certainly notice a platform that shows an eighty-one per cent match on a posting that names no skills, or a recruiter marked online when nobody is there. Each of those is small on its own and damning together, because it teaches the user that the interface will assert things it does not know. Abstention costs something: emptier screens, badges that sometimes do not appear, and the engagement a fake presence cue would have bought. We think that is the right trade, because what a career platform really owns is whether a student believes what is on the screen at the moment they decide where to apply.

7.2 Incentive Alignment of Advertiser-Pays Monetization

The pivot changes who the platform is working for. When the operator sells its own courses, the student is both the audience for a recommendation and the target of a sale, and every institute in the list competes with the operator's catalogue [3], [6]. On Road2Job the student pays nothing, the institute buys labelled

visibility for its own programmes, and the operator has no catalogue to protect. What is then good for the business, plenty of engaged students and plenty of institutes, is also what makes honest ranking worth keeping.

Isolation is what keeps that alignment from decaying. If money could lift rank_score, the advertiser-pays model would recreate the problem it was meant to solve, only now selling rank instead of courses [7], [8]. Keeping campaign state out of the ranking service's inputs, and confining paid effects to a labelled slot in a feed, lets the platform make a claim it can defend: the institute ranking cannot be bought, and the updates feed is partly paid for and says so on the item.

7.3 Scope Discipline in an Early-Stage Marketplace

The seven removals are worth recording because each feature had been built in answer to a reasonable request. But a two-sided marketplace fails in one main way, which is not enough activity in its core transaction. Anything that does not deepen that transaction eats the capacity needed to deepen it, and with two cold-start problems running at once it also splits the attention of both sides. The forum, research hub, roadmaps, mentor marketplace, events, video interviews and assessments all served adjacent needs while the core loop was still thin. Concentrating on that loop, and on the promotion engine that pays for it, treats breadth as something to buy after liquidity rather than before.

VIII. LIMITATIONS

8.1 Payment Processing

There is no payment gateway. Campaign payment happens by bank transfer or UPI outside the platform and verification is manual, with the institute uploading a proof and an administrator approving or rejecting it. That puts a human delay between paying and going live, leaves room for reconciliation errors, offers no automated refund or chargeback route, and will not scale past a handful of concurrent campaigns.

8.2 Scalability

The deployment is a single database with no read replication, no sharding and no horizontal application scaling, and rank recomputation is a full batch rather than an incremental update. That is fine for a bounded set of campuses, but it has not been shown under concurrent load, and polling makes request volume grow linearly with the number of open conversations.

8.3 Evaluation Scope

Section VI rests on constructed fixtures and functional checks, not production data. So the weights in (1) have not been validated against real student outcomes, the decay constant in (2) has not been tuned empirically, the skill dictionary's coverage of employer vocabulary has not been measured at scale, and no user study of the zero-fabrication interface has been run.

8.4 Client Coverage

There is no native mobile app. The interface is responsive and server-rendered, which rules out offline use and push notification and the background refreshing that a mobile-first student population will expect.

IX. CONCLUSION AND FUTURE WORK

9.1 Conclusion

This paper described Road2Job, a campus career platform for Indian students, employers and training institutes, built as a Core PHP 8 MVC application with a PDO/MySQL model layer and server-rendered views. Its main technical piece is the institute ranking algorithm, which combines recency-decayed and de-duplicated placement quality, category-weighted activity, profile completeness and six-month consistency under published weights of 0.35, 0.25, 0.20 and 0.20, then multiplies by a burst-posting penalty floored at 0.5. On constructed fixtures the

orderings came out as intended: placement outcomes beat presence signals, and a burst poster with the best activity sub-score is pushed below a duplicate-entry fixture once consistency and the multiplier are applied.

The second contribution is architectural. Revenue comes from selling institutes time-bounded, labelled promotion inside a student updates feed, with the campaign subsystem kept out of the ranking service's inputs, and rank scores were checked as unchanged across campaign creation, approval and expiry. Together with the zero-fabrication rule applied to match percentages, completeness scores, timelines and the four-second typing indicator, that gives a platform where every displayed quantity is either derived from stored data or withheld. The seven withdrawn features stand alongside as a small case study in scope discipline.

9.2 Future Work

Four extensions follow straight from Section VIII. The first is a payment gateway with programmatic settlement, automatic campaign activation and a refund path, replacing manual proof checks. The second is empirical tuning of the weights and the decay constant against real placement outcomes once enough production data exists, along with showing institutes their own sub-scores. The third is scaling work: read replication, incremental instead of batch rank recomputation, and moving from polling to server-sent events or a WebSocket tier if conversation volume ever justifies it. The fourth is a native mobile client with push notification for status events, which the event-sourced timeline can already feed. We also plan a longitudinal study of whether a ranking that cannot be bought measurably changes student trust and institute participation.

REFERENCES

[1] Ministry of Education, Government of India, All India Survey on Higher Education (AISHE) 2021-22. New Delhi, India: Department of Higher Education, 2023.

[2] Wheebox, Taggd and Confederation of Indian Industry, India Skills Report 2023. Gurugram, India: Wheebox, 2023.

[3] Internshala, "Internshala: Internships, courses and placement training for students." [Online]. Available: https://internshala.com. [Accessed: 03-Aug-2026].

[4] LinkedIn Corporation, "LinkedIn: Professional network, jobs and hiring solutions." [Online]. Available: https://www.linkedin.com. [Accessed: 03-Aug-2026].

[5] Info Edge (India) Ltd., "Naukri.com: Job search and recruitment platform." [Online]. Available: https://www.naukri.com. [Accessed: 03-Aug-2026].

[6] J.-C. Rochet and J. Tirole, "Platform competition in two-sided markets," Journal of the European Economic Association, vol. 1, no. 4, pp. 990-1029, 2003.

[7] A. Ghose and S. Yang, "An empirical analysis of search engine advertising: Sponsored search in electronic markets," Management Science, vol. 55, no. 5, pp. 1605-1622, 2009.

[8] B. Edelman, M. Ostrovsky and M. Schwarz, "Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords," American Economic Review, vol. 97, no. 1, pp. 242-259, 2007.

[9] P. Resnick, K. Kuwabara, R. Zeckhauser and E. Friedman, "Reputation systems," Communications of the ACM, vol. 43, no. 12, pp. 45-48, 2000.

[10] J. Kleinberg, "Bursty and hierarchical structure in streams," Data Mining and Knowledge Discovery, vol. 7, no. 4, pp. 373-397, 2003.

[11] N. Tintarev and J. Masthoff, "A survey of explanations in recommender systems," in Proc. IEEE 23rd Int. Conf. Data Engineering Workshop, Istanbul, Turkey, 2007, pp. 801-810.