

Placement Management System

Harshita Priyal¹, Saloni Kumari², Basudeo Mahato³

Department of Computer Science and Engineering

³Assistant Professor, RTC Institute of Technology, Ranchi, Jharkhand, India

^{1,2}B.Tech, RTC Institute of Technology, Ranchi, Jharkhand, India

Abstract - The report aims at describing the architecture design for a detailed Placement Management System (PMS) for higher education. We begin with a literature survey of academic studies and commercial offerings, identifying common features (e.g. student/recruiter profiles, job postings, eligibility filtering, interview scheduling, offer tracking, and analytics) and gaps in existing practice. System requirements are obtained in Methodology and layered architecture (UI, Application Server, Database) is proposed and relation data model (ER-Diagram is introduced as a relational data model). We discuss technology stack options (e.g Web framework, database, cloud etc.), and highlight security/privacy (encryption, access control, FERPA/GDPR compliance). A plan of testing and deployment including unit/integration tests, user acceptance and CI/CD Items to consider during implementation: UI responsiveness, ease of integration with student information/LMS and HR/ERP systems (in order to access current academic/recruiter data), scalability and maintenance. We have estimated a per module development effort (refer table & chart below) and would like to describe our pricing choices (in-house / commercial / fully custom). A modern PMS by virtue of stakeholder process and information integration provides realistic administrative benefits as well as improved placement results.

Introduction

Statement of the Problem

Student career placement, one of the major campus events, remains on manual job-tracking and campus placement at most campuses. Announcements and applications are often handled via email, bulletin boards or ad-hoc spreadsheets, causing delays and confusion. In the absence of a way to automatically weed out unqualified students, people who don't meet the criteria may be allowed to submit applications. This will obviously waste the coordinators' time. As

Priyanga et al, "traditional placement strategies prove inefficient and error **prone**", **resulting in a loss of opportunities**. This 2019 case study found that placement officers had difficulty "manually checking, searching and updating" the students records and that students couldn't see the services. All in all, the problem of disparate data and manual processes results in constraints on volume of placement and on the workload on placement officers.

Objectives

The goal is to create a web-based Placement Management System for automating the campus placement activities for the college/university.

The new student portal system's primary uses will be as

1. Student, employer and job data together
2. Self-service system for students, employers and administrators.
3. Automated online handling of application, interviews and job offer.
4. Online and real-time analysis of data and reporting.
5. System compatibility with existing university systems (i.e. SIS/LMS, ERP, etc) for the update of student profile. Once achieved these should result in increased efficiency of placement cycles, better communication between interested parties and, in the end, an increase in placement rates.

Scope

The system will cover the end-to-end placement process:

- Managing student data: building resume, eligibility check, application status and interview status.
- Company management: company registration, job posting, eligibility criteria specification.
- Placement office: application monitoring, interview scheduling (tests, GDs, etc). The platform should support effective communication channels

(notifications, reminders), as well as results reporting (offers, acceptances).

- **Analytics/Reporting:** generation of placement statistics (rates, averages, sector distribution) for accreditation and decision-making.
- It will be a multi-user system (students, placement officers, recruiters, administrators) And role-based access, both web and mobile access.

The **stakeholders** to this system are:

- **Students:** they would be registering themselves in the system and uploading their resume, they can apply to campus placement drives.
- **Placement Officer/Administrator:** they would be in-charge of the placement drive, would approve companies and also see the status of the drives and access the reports.
- **Externals:** Recruiters/Companies, who would be posting vacancies and who could view candidates applying and could respond with interview feedback.
- The final category of users is the **University Management**, who need reports related to placement and progress (necessary for funding, accreditation, and other purposes. Benefits of this system are seen across different stakeholders like, student gain visibility, reminders, opportunity to present profile in a professional way. Whereas recruiters can find out the best candidates efficiently and also use scheduling tools. Placement officers save a lot of paper work. Providing data that placement administrators can access).

I.Literature Review

Academic Studies

Through this experiment, I learned that doing this by hand takes too much time and automation approaches were thought as followings. Balondo et al. (2019) studied a university's placement office and found a "very high" need for a digital system to reduce manual work and centralize records. Similarly, Priyanga et al. (In 2025, a group of students suggested a "Smart and Automated College Placement System", which will use the existing web technologies (React, Laravel and cloud services) and found that the campus placement processes at colleges (through mails or notice boards instead of following the criteria of eligibility) is becoming a hassle. Their system uses automated filters so only qualified students are notified, and provides dashboards and real-time analytics to placement officers.

Another paper, Pradeepa et al., 2026 "Smart Placement Management system with Intelligent Career support" three modules; student, recruiter, and officer module. It utilizes data analytics such that academic backgrounds and skills acquired by student is analyzed to suggest suitable job profiles or highlight skill gaps while recruiter is presented with a list of candidates in a ranked format based on qualifications. The architecture of the system is standard 3-tier model (Presentation, application, database) and it accumulates data on "job matching and skill gap analysis." These show that a trend toward an AI-empowered placement and the fact that, placement systems should be data-driven to provide good results is a commonly stated observation.

Existing Commercial Systems

A variety of vendors are coming out with campus placement software solutions. Most feature integrated dashboards, automated matching and run on cloud / SaaS platforms. For example, OpenEduCat (an open source Odoo based solution) mentions "manage campus recruitment, match students to jobs offers by academics & skill profiles, scheduling interview, tracing offer, placement reporting" from one integrated portal. Generate lists of eligibility by GPA, courses or attendance. Trace candidates across a number of interview stages and review interviewer feedback. "Cloud based campus recruitment solutions with scalable dashboards and secure workflow" is the solution that Orell Placify promotes. Essential features of the Orell system are role-based access, automated notifications, management of resumes, application tracking, and audit logs. Security (encrypted data, managed approval of jobs) and integration (ability for businesses to upload jobs and responses through the portal) is also a focus.

Creatrix Campus (Student ERP) Placement module allows students to 'Build their job profile, search & apply for jobs/internships in real time eligibility'. It can enable the placement teams to have a central database for all the companies and extract the student-company data. The candidates can be seen on the recruiters account, they can be filtered by skill sets, gpa and certificates. Creatrix, like everyone else, gives placement analytics and reporting, company engagement information.

Camu Digital Campus (a learning platform) also promotes an intelligent placement system. According to its blog, good PMS needs to integrate student eligibility tracking, automatic scheduling (including text reminders to students), SIS integration and

reporting, and powerful analytics. There is an explicit requirement from Camu to integrate with the SIS, where GPA, attendance and course can drive eligibility filters. They further stress data security and compliance, noting that “placement data is sensitive” and must meet protection standards (i.e. encryption and secure storage).

The functionalities presented by vendors under each of these systems are similar. Each system offers students and recruiters a portal, a mechanism to post and filter job opportunities by eligibility, to schedule interviews, and to verify conflicting interviews. There is also tracking for the offers that are extended and statistics regarding where students have been placed.

Comparison of 5 illustrative solutions, Table 1 gives a comparison.

Table 1: Example placement management systems (feature sets are taken from vendor documentation)

Feature	Open Edu Cat	Oréll Placify	Creatrix Campus	Vidyalaya LMS
Student portal (profile/resume)	Yes	Yes	Yes	Yes
Recruiter portal	Yes	Yes	Yes	Yes
Job posting & management	Yes	Yes	Yes	Yes
Eligibility matching/filtering	Yes (GPA, courses)	Yes (smart matching)	Yes (filters by GPA/skills)	Yes
Interview scheduling	Yes (conflict-checked)	Yes (multi-round, automated reminders)	Yes (scheduling interface)	Yes
Automated notifications	Yes (email/SMS alerts)	Yes (SMS/email)	(implied)	(Yes)
Analytics & reporting	Yes (placement stats)	Yes (dashboards, performance reports)	Yes (placement & student reports)	(Yes)

Feature	Open Edu Cat	Oréll Placify	Creatrix Campus	Vidyalaya LMS
Integration (LMS/ERP)	SIS integration options	(Not explicitly stated)	SIS/LMS via ERP suite	LMS (as part of ERP)
Security & compliance	(Role-based access)	Encryption, role control, audit trails	(Secure data capture)	(Confidential, GDPR-aware)
Deployment model	Cloud/SaaS or on-premise	Cloud & on-premise options	Cloud-based SaaS	Cloud/LMS suite

(Note: Empty cells denote features the placement package should contain, but are not in sources. The suppliers compared these for us)

Literature Gaps

While a lot of these systems are very well featured, it has been observed in academic circles that most are lacking in some way or other. These existing solutions tend to deal primarily with the central workflow and are deficient on the area of the career guidance aspect. Thus, we have skill-gap analysis part in Pradeepa et al. Is using AI, this application is not adopted commercially yet. Also, very few sources discuss privacy or compliance in depth, even though student data is sensitive. (Data security is mentioned in one paper, but in a general, unqualified way.) A further gap relates to research on scalability, with only Priyanga et al. In reference to scalability in the cloud. Thus, our design should ensure future extensibility (AI modules, mobile apps, etc.) and a robust data governance process that far surpasses what's typically in a vendor specification.

II. Methodology

Functional Requirements

An effective PMS needs to support these functions:

- **User management:** Authenticate and apply role-based authorization for students, recruiters, officers, admins.
- **Student Profile:** creation/editing of academic/skill profiles and resume uploads.
- **Company/Job management:** companies register or are approved by staff; they post job/internship openings with eligibility criteria.
- **Application Processing:** Process the application, the students view eligible job postings, apply to the

posting; the system record application (date and status), ensure students only submit unique and timely applications.

- **Interview Scheduling:** system which would enable placement officers (and recruiters) to schedule interview slot/test. This would automatically check it against the timetable of the student. Notifications (email/SMS/app push) alert all parties of interview details.
- **Offer Tracking:** Record job offer details (pay, dates), send out offers, keep track of responses and accept/reject/no offer. Send reminders and chase up outstanding work.
- **Reporting & Analytics:** dashboards for placement officers and deans: placement rates, average salary, sector distribution, etc. Real-time metrics for accreditation bodies (e.g. NAAC, NBA).
- **Integration:** APIs or connectors to pull student records (grades, attendance) from SIS/LMS, and to push placement outcomes back. Optional SSO (OAuth/SAML) with campus IAM.
- **Communications:** includes native email/SMS gateway for broadcast of announcements and approvals. These functional points are in line with the literature and the vendor checklist (see e.g. student registries, application management, scheduling).

Non-Functional Requirements

- **Performance:** The system should handle large cohorts (e.g. thousands of students, hundreds of jobs) with sub-second response for key operations. Effective query capabilities are necessary for filtering the eligibility on the fly.
- **Usability:** UI/UX should be easy to use by non-technical users (students, admins). Interfaces should be mobile-responsive or a dedicated app for students on the go. Some of the campuses may need multi-language facility.
- **Scalability:** Handle traffic growth by adding horizontally (more servers) or vertically. Use cloud infrastructure (e.g. AWS/Azure/GCP) with auto-scaling as in Priyanga et al. A microservices or modular architecture will help scale components (e.g. separate service for scheduling).
- **Security & Privacy:** Needs to protect all data (encrypted at rest and in transit) and comply with the regulations (FERPA for US and GDPR for world). Essential role-based access control (RBAC); e.g. students see only their own info; recruiters see only their own applicant pool; admins see all. Record all accesses to, and all changes to-for audit purposes.

Several vendors stress encrypted protocols and audit logs.

- **Reliability:** Aim for high availability (e.g. 99.5% uptime) using redundant servers/DB replicas. There should be backups & disaster recovery processes available (all vendors state that backups are an option).
- **Maintainability:** Software should have a well-defined structure & good documentation, preferably using industry standard frameworks so there are benefits of support. Use containerization or PaaS if you want something easy to maintain.
- **Portability:** software must be platform independent (browser-based and not dependent on one operating system) and if to be installed on premise, to be deployable on standard configurations (Linux machines, etc.).
- The NFRs are developed on basis of general design patterns for educational software and additionally through specific vendor statements, like Orell claim: "fully protected with complete data backup and security".

System Architecture

- A 3-tier client server system. (see Figure 1 below). The presentation layer consists of web (or mobile) clients; the application layer contains business logic and APIs; the database layer holds all data. This corresponds to existing layouts.

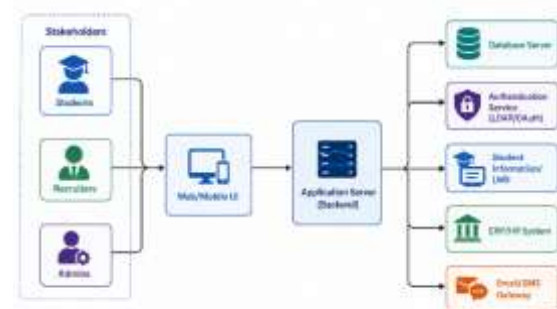


Figure 1: System architecture (high level) Users (students, recruiters, admins) are fronted by the UI. Backend implements logic and ties in to SIS/ERP and authentication.

Highlights:

- The backend can be implemented as a web framework (e.g. Django, Spring Boot, or Node.js) exposing RESTful APIs. The role of this component is the enforcement of policies and coordination of the modules (e.g. application processing, interview scheduling).

- The frontend is a rich web application (e.g. React/Vue/Angular) or mobile apps. It consumes backend APIs and provides dashboards and forms.
- The database should be a robust RDBMS (PostgreSQL, MySQL, etc.) And it's ACID-compliant. It will store users, profiles, applications, schedule slots, etc. Figure 2 (below, ER diagram) specifies the schema of the data.
- External integration services: We would recommend integrating with the campus SIS/LMS (to get real-time grades, course completions) and ERP/HR (to pull the corporate contacts, placement records) via secured APIs. Can be simplified the login with SSO provider (SAML/OAuth), the students can log in with campus id. The notifications will be sent by an SMTP/SMS gateway service.

Data Model (ER Diagram)

The schema design considers the main entities that are: STUDENT, COMPANY, JOB, APPLICATION, INTERVIEW, OFFER.

The relationship has been illustrated in Fig 2 below

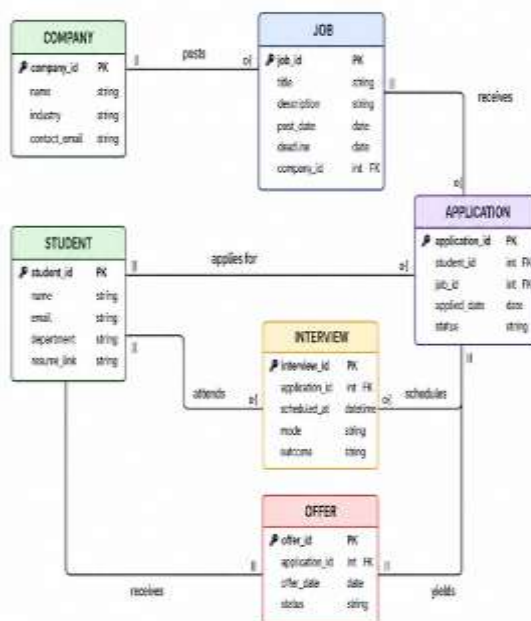


Fig 2: ER diagram of Placement Management System.

Candidates take applications from jobs using the APPLICATION table which creates planned INTERVIEWS and possibilities for OFFERS. Each JOB is posted by a COMPANY. Primary keys (PK) and foreign keys (FK) are indicated.

This is the relational model and this guarantees consistency of the data because of the relationship between APPLICATION student_id and STUDENT

student_id. (not shown are other tables like USERS, user role and log tables).

What tech stacks do we use:

There isn't any required language or platform to use, but here are a couple of possibilities:

- **Web framework:** Django (Python), ruby on rails or node.js/express. A fast way to generate RESTful APIs, IJERT cases: Laravel/PHP, React.
- **Database:** PostgreSQL or MySQL for relational storage; optionally a NoSQL (e.g. MongoDB) for unstructured data like activity logs.
- **Frontend:** Develop the responsive UI by an up-to-date JS framework like React/Vue/Angular. Alternatively, server-side rendering (e.g. ASP.NET, Spring MVC).
- **Mobile:** React Native or Flutter for cross-platform apps if mobile access is required.
- **Host:** Cloud (AWS, Azure, GCP) offers Scalability which could reach to any amount. Priyanga et al. suggest AWS for elasticity. Containerization (Docker) and Kubernetes or managed services (AWS Elastic Beanstalk, Azure App Services) can simplify deployment.
- **Login Type:** Security is improved when University single sign-on is used.
- **Notification:** Email server (SMTP) or services (SendGrid) as well as SMS gateways (Twilio) for alerts.
- **Version Control/CI:** Git repositories with CI/CD pipelines (e.g. Jenkins and GitHub Actions) can take care of automated tests and deployments.

Safety and security problems

Since the PMS handles sensitive student and corporate data, security is paramount. This is required:

- **Encryption:** Encrypt sensitive fields (e.g. personal IDs) in the database. The encryption of data in transit and rest are discussed by Orell and Camu.
- **Security of access:** Implement an granular RBAC, which authenticates users to allow them to see authenticate data only. For example, a student can access his own applications and offers; placement officers can access students' placements information. All admin functions are logged on an audit trail (this is Orell's system).
- **Compliance:** Most regions have rules surrounding educational data (FERPA, GDPR), and data should be exportable and deletable upon request. Data should also be compliant with all third party services (SIS, email provider).

- **Privacy:** Personal Identifiable Information (PII) not shown where it does not need to be. Sharing student's emails with recruiter should be by explicit consent only. Data retention policies should be defined (e.g. how long to keep alumni placement records).
- **Develop in a secure way:** Avert common vulnerabilities like SQL Injection, XSS, CSRF, keep up with the OWASP best practices. Conduct security testing (penetration tests, code scans) prior to deployment.
- This matches the vendor advice about secure recruitment sites as well as education system best practice.

Test Plan

Full test should cover:

- **Unit Testing:** Each module's functions (e.g. application submission logic) are verified with automated tests.
- **Integration Testing:** Test the APIs and data flow between modules, especially integrations (e.g. Extracting grades from the SIS or sending reminders).
- **System Testing:** End-to-end tests of workflows (student applies → admin schedules interview → interviewer provides feedback). Use testing tools (e.g. Selenium for UI, Postman for APIs).
- **User Acceptance Testing (UAT):** Perform a pilot study with real users (students, officers, recruiters) in order to collect the necessary feedback, and confirm the UI works appropriately for them.
- **Performance Testing:** The most important actions (many people login simultaneously, or many applications run simultaneously) should be performance tested to satisfy scalability objectives. These can be tested by using tools such as JMeter.
- **Security Testing:** Perform scanning for vulnerability and potentially external audit before going live.

Deployment Plan

The PMS should be deployed in phases:

1. **Development Environment:** Developers work with local servers and mock data.
2. **Cloud/staging Server:** Use CI/CD pipelines to deploy code automatically to staging on commits.
3. **Production-**The plan would be to deploy to production servers, probably in a load-balanced cluster. Run zero-downtime deploys by implementing techniques such as blue/green deployments or canary releases. Set up monitoring (application logs and infra

metrics) to know what has broken when it has been deployed.

4. **Data Migration** If you are migrating data from a system you are currently using, outline how you will be migrating your existing data (student profiles, previous placement records) into the new system.

5. Make sure that there is enough documentation and training material for the system to be administered after launch, and for students and recruiters to understand how to access and navigate the new portal.

As for implementation:

Interface and UX

The interface must be easy to use and fast responding, for the variety of users it serves:

- **Clarity:** Eliminate the unnecessary. For instance, the placement officer should be able to see a clear display of the ongoing drives and the task list pending at his end. Students should easily find jobs relevant to their profile. The Camu guide stresses "user-friendly interface" and ease of navigation for first-time users.
- **Role-Specific Views:** Provide custom dashboards, e.g. a recruiter's eye displays the incoming applications and the interview slots and a student's eye displays the applied jobs and their upcoming schedules.
- **Mobile Access:** A major requirement is to have mobile support for students. They are able to log in using mobile website/mobile app, and post/view notifications using their mobile phones.
- **Accessibility:** This section explains that, to be truly usable, a UI must be accessible. It recommends including alternate text (alt text) for images and providing good labels for interactive elements so that every user can interact with and find their way through it.
- **Feedback and assistance:** Show tooltips, help screens and interactive error messages (e.g. All set! It's been successfully submitted.). The confirmations dialog boxes for important actions (such as withdrawing an application) are a good idea.
- Vendor examples (e.g. such as the Vidyalaya's LMS FAQ) show the importance of providing ease of access to data, and a sense of "confidence" in the process to the user.

Integration with LMS/ERP Systems

- The integration is one of the most differentiating factors. It should not be a siloed system: PMS should retrieve student data (current GPA, credits, certificates etc). This would avoid the necessity of human verification on eligibility. For example, Camu and

EDU in turn mention SIS connection: "[Eligibility] pages are created from real-time academic data. There is no 6-8 hour data merging with spreadsheets". Similarly, when posting new job opportunities, an enterprise system's alumni module ERP can retrieve the required competencies from the records of HR or from alumni placement data.

- In terms of technology, that would mean to use API's or to establish a (secured) direct link towards institutional data base.
- **SIS/LMS:** API endpoints to get the GPA, attendance, passed subjects. (LMS with a REST API or by using ODBC to connect to the database.)
- **HR/Finance ERP:** Company/recruiter info could be synchronized from an ERP vendor list. Allowances paid to students would feed into finance modules from placement data.
- **Learning modules:** The PMS could link to an e-learning system for placement test administration if coding tests or the like are offered as part of a placement, and record test results.
- Well integrated will result in a considerable improvement in accuracy. EDU ERP states that when student data changes, (e.g. If GPA decreases) the eligibility list is updated automatically and flagged. EDU ERP states that when student data changes, (e.g. If GPA decreases) the eligibility list is updated automatically and flagged.

Scalability

- This system must be able to handle numerous users and amount of data. Cloud deployment (AWS/GCP/Azure) and on demand scalable resources. The application layer must be stateless so that instances may be added, and it must make use of a load balancer.
- **Horizontal Scaling:** is the addition of more application servers behind the load balancer. Use a managed database cluster (Read replicas and sharding if applicable).
- **Microservices (Optional) :** If you see that you would need this in future for scale, add services (Auth service, scheduling service, notification service).
- Use caches (e.g. Redis or Memcached) for data that is queried more often, like a students profile details and eligibility calculation.
- **Asynchronous jobs:** Heavy processes such as batch sending emails, report creation should be dispatched to job queue/worker processes.
- The Priyanga et al. Also emphasize the cloud (AWS) to make the solution scalable. A well-designed deployment strategy (e.g. Containerised micro-services

in Kubernetes) will allow the system to handle high load in placement drives.

Maintenance and Upgrades

- Once deployed, the system will need a certain amount of maintenance.
- OS/frameworks/libraries patching to correct bugs and address security holes.
- **Support:** A helpdesk or ticket system to track and solve user reported problems. Comprehensive documentation (user manuals, FAQs) lowers the number of requests for support.
- **Logging/Monitoring:** set up application and error logging, as well as New Relic/Cloud Watch so we can find issues as they arise.
- **Back Up & Recovery:** The database will be backed up each day automatically. Regular restore tests will be performed. This was a selling point for vendors.
- **Archiving of data:** Historical placement data could be archived as the years go on (in order to reduce main DB size) and yet remain accessible.
- Use Git for version control on all code and implement well-defined branching practices. All changes are to go through code review.

Cost Estimates

- It is worth noting here that the development cost will depend on the methodology you choose to use, as the nature of development tasks can vary a lot. We outline rough scenarios (assuming undergraduate project vs. commercial implementation).
- **Low-Cost (Education/Volunteer Development):** A small team (2-3 students or faculty) could use open-source technologies to produce a minimum viable system in 200-300 person-days. Hardware costs are minimal if using open-source DB and free-tier cloud. Maintenance relies on volunteers. The most costly part is developer time.
- **Medium-Cost (Commercial SaaS or Consultancy):** Hiring a third-party developer/team or purchasing a license (e.g. Orél/Creatrix) might cost in the low six figures (USD) annually for licensing/support. Customization adds to cost. The professional services & SLAs are on hand in this case.
- **High-Cost (Enterprise Solution):** A full-scale custom build with high-end hosting (dedicated servers, advanced security audits) could run into the mid to high six figures or more. This includes staff for 12-18 months of development and procurement of reliable infrastructure.

Table 2 below (and the chart alongside) shows a module-by-module breakdown of effort estimates under a moderate development environment (approx 5-6 people over a period of 1 year). *True costs will vary based on location, team rates and project size.*

Module	Estimated Effort (person-days)
Authentication/Roles	10
Student Profile & Resume	15
Job Posting & Management	15
Application Workflow	10
Interview Scheduling	20
Analytics/Reporting	15
Notification System (Email/SMS)	5
SIS/LMS & ERP Integration	15
Admin Dashboard	10
UI/UX Design & Frontend	15

Table 2: Proposed modules and approximate effort for development.

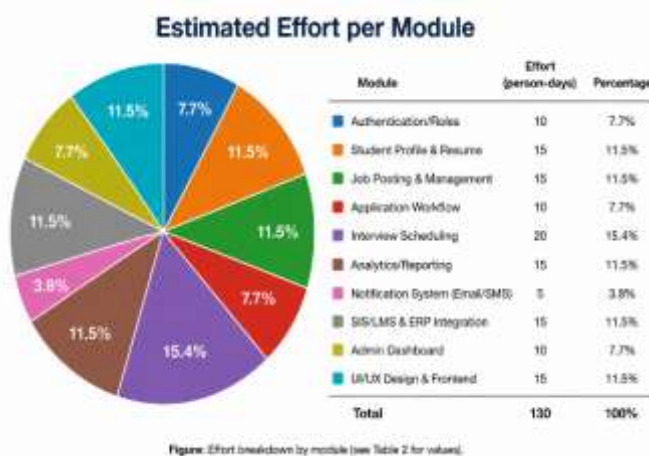


Figure: Chart of effort breakdown by module (see Table 2 for values).

Note: The above efforts assume a moderately complex system. We have to put more effort in critical modules (scheduling, integration and UI). Tests and project management and buffers (~20%) are also needed. (Not shown).

Conclusion

By implementing a separate Placement Management system campus recruitment can move from a disjointed, manual process to a strategic plus point for the college. Institutions also save time and decrease error as student jobs can be placed with a single

database that then manages all aspects of the matching, interview, and offer process. Analytics and transparency create increased accountability (e.g. for accrediting bodies), and richer student profiles with integrated LMS data lead to better matching and higher placement rates.

We have established a structured methodology to be used in creating such a system: specifying requirements, research on available solutions, system architecture proposal and consideration for other implementation issues (security, integration, scalability). Diagrams (architecture and ER) and tables of features/effort support the design. Vendors supply most of these components ready to use but an institution may decide on a closed source or open-source system to satisfy its needs. In all these instances careful planning, like the one shown, has been critical. To sum up, good PMS has less paperwork but positive results. The gains for the student (better careers advice), the recruiter (checked applicants) and the university (good reputation and statistics) far outweigh the cost of developing an updated, comprehensive placement portal.

References

- Liva Jorge. "Transforming Campus Recruitment with Placement Management Systems." Medium, Mar. 30, 2023. (Defines core features of PMS: profiles, postings, scheduling, etc.).
- Pradeepa R. *et al.* "Smart Placement Management System with Intelligent Career Support." *IJERT*, vol. 15, no. 04, Apr. 2026. (Proposes AI-driven career guidance in PMS; discusses architecture and benefits.).
- K. Priyanga *et al.* "Smart and Automated College Placement System." *IJERT*, vol. 13, issue 05, June 2025. (Discusses inefficiencies of manual placement; presents React/Laravel/AWS architecture with analytics.).
- Christian Balondo, Kristine Samson. "The Need to Develop a Career and Placement Management System..." Univ. of Nueva Caceres (research study, 2019). (Case study highlighting manual challenges and need for PMS.).
- OpenEduCat. "Placement Management Software." (Feature page describing campus recruitment portal with eligibility, scheduling, reports.).
- Oréll Software (Placify). "Orell Placement Management Software." (Product page; highlights end-to-end campus recruitment solution with secure workflows.).

- Creatrix Campus. *“Placement Management System.”* (Solutions page; details student, placement team, and company features.).
- Camu Digital Campus. *“How a smart Placement Management System simplifies Campus Recruitment.”* (Blog; outlines key PMS features and integration points.).
- Vidyalaya LMS. *“Online Training and Placement Management”* (FAQ; explains features of placement module – profiles, scheduling, tracking, analytics.).
- EDU ERP (Indian ERP vendor). *“Placement Management System for Colleges & Universities.”* (Marketing page; discusses centralization, AI-driven matching, and ERP/LMS integration benefits.).