PROJECT REPORT

SmartHire AI — Job Portal & Internship Management System

Full Stack Engineer (Intern) · DawoodTech NextGen · Remote · Apr 2026 – Jun 2026

Author: Numair Fahad · MERN stack + AI integration

1. Overview

SmartHire AI is a production-style MERN job portal and internship management system built for three distinct audiences — students, recruiters, and administrators. Alongside the standard hiring workflow (listings, applications, status tracking), it embeds AI directly into the product: resumes are scored and critiqued in-app, and recruiters generate first-draft job descriptions in seconds. The frontend is deployed on Vercel, the API on Microsoft Azure App Service, and data lives in MongoDB Atlas.

Resource	Link
Live application	`https://smart-hire-ai-job-portal-internship.vercel.app`
GitHub repository	`github.com/numair-2003/smartHireAI_JobPortal_InternshipManagementSystem`
API base (Azure)	`smarthire-ai-backend-numairfahad-fcacdxh4b2guehgc.uaenorth-01.azurewebsites.net/api`
Health check	`GET /api/health`

2. Problem & Objectives

Campus hiring platforms typically do one thing well and leave the intelligence to a separate chat tool. Students receive no structured feedback on their resumes, and recruiters rewrite near-identical job descriptions by hand. SmartHire AI closes that gap inside a single permission-scoped product.

- Deliver AI-driven, role-aware resume feedback inside the portal itself.
- Let recruiters generate accurate first-draft job descriptions and skill tags in seconds.
- Keep students, recruiters, and admins in fully separated, permission-scoped experiences.
- Ship with realtime notifications, email alerts, and production-grade deployment.

3. System Architecture

Frontend (Vercel): React.js with Redux Toolkit for state, React Router for role-scoped routes, Tailwind CSS for the UI layer, Axios for API access, Socket.IO client for live notifications, and a localStorage-backed brightness preference system (Bright / Soft / Night).

Backend (Azure App Service): Node.js + Express with Mongoose models per domain (User, Job, Application, Notification). JWT authentication with bcrypt password hashing, role-based middleware on every protected route, Multer + Cloudinary for file storage, Nodemailer for transactional email, Socket.IO rooms scoped per user, and a provider-agnostic AI service layer.

Layer	Technology
Frontend	React.js, Redux Toolkit, React Router, Tailwind CSS, Axios, Socket.IO client, React Hot Toast
Backend	Node.js, Express.js, Mongoose, JWT, Bcrypt.js, Multer, Socket.IO, Nodemailer
Data & storage	MongoDB Atlas, MongoDB Compass, Cloudinary (resumes, profile photos)
AI	Google Gemini (gemini-2.5-flash) with OpenAI (gpt-4o-mini) as an interchangeable provider
Delivery	Vercel (frontend), Azure App Service (API), Git & GitHub, environment-scoped secrets

4. Feature Set

Feature	Description
Authentication	JWT register/login with bcrypt hashing and protected routes
Role dashboards	Separate student, recruiter, and admin experiences
Job marketplace	Search and filter jobs/internships by keyword, type, and location
Recruiter tools	Post jobs, review applicants, update application status
Student tools	Upload profile photo and resume, apply to jobs, track applications
Admin panel	Manage users, roles, active status, and platform statistics
AI resume review	Model-scored resumes with strengths and concrete improvements
AI job generator	Generates descriptions, requirements, and skill tags, then publishes them
Uploads	Cloudinary-backed PDF/DOC/DOCX resumes and JPG/PNG/WEBP avatars
Notifications	Socket.IO live delivery plus stored notification history
Email alerts	Nodemailer notifications when application status changes
Demo data	Seed script with realistic users, jobs, applications, AI scores, notifications

5. API Surface

All protected endpoints expect `Authorization: Bearer <token>`. Errors return `{ success: false, message }`; stack traces are suppressed when `NODE_ENV=production`.

- **Auth** — register, login, current user, update profile, upload profile photo.
- **Jobs** — list active jobs (search / type / location filters), job details, recruiter listings, create, update, delete.
- **Applications** — apply to job, upload profile resume, view profile/application resume, my applications, job applications, update status.
- **AI** — resume review, job description generator.
- **Notifications** — list, mark one read, mark all read.
- **Admin** — platform stats, list users, update user, delete user.
- **Realtime** — Socket.IO notification channel scoped per authenticated user.

6. Security & Configuration

- JWT-signed sessions with bcrypt-hashed credentials; role-based middleware guards every privileged route.
- CORS restricted to known frontend origins via FRONTEND_URL / FRONTEND_URLS.
- All secrets (MONGO_URI, JWT_SECRET, Cloudinary keys, EMAIL_PASS, AI_API_KEY) held in environment configuration — never committed and never exposed to the frontend.
- AI keys live only in backend environment settings; prompts are templated server-side so clients never submit raw prompts.
- Graceful degradation: when the AI provider has no quota, the API returns demo fallback responses instead of failing the request.

7. Engineering Challenges

MongoDB Atlas SRV DNS failures. Local development intermittently failed with `querySrv ECONNREFUSED` due to unreliable router DNS. Diagnosed with `nslookup` against public resolvers and resolved by supporting a non-SRV connection URI plus a configurable `DNS_SERVERS` value, with Cloudflare WARP as a local fallback. Azure App Service uses its own network path, so Atlas Network Access rules were configured separately.

Provider lock-in on the AI layer. Resume review and JD generation were placed behind a single service interface, so switching between Gemini and OpenAI is a configuration change (`AI_PROVIDER`, `AI_MODEL`)

rather than a rewrite. Model output is validated before it reaches the client.

Notification fan-out. Broadcasting events and filtering client-side wasted bandwidth and leaked state; per-user Socket.IO rooms proved both correct and cheaper.

8. Results

- Three fully separated role experiences sharing one authentication and permission model.
- AI resume review and job-description generation running inside real production flows.
- Realtime notifications and email alerts wired across the full application lifecycle.
- Deployed end to end: Vercel frontend, Azure App Service API, MongoDB Atlas database, Cloudinary media.

9. Key Learnings

- Abstracting the AI provider behind a service interface paid for itself the first time the model was swapped.
- Role-based middleware is far easier to reason about than guard checks scattered through controllers.
- Infrastructure and DNS issues deserve documented, reproducible workarounds — they cost more time than feature work.
- Deployment constraints (CORS, environment scoping, cold starts) should be designed for from day one, not retrofitted.

Numair Fahad — AI / ML Engineer · numair1919@gmail.com · Report generated for recruiter review from the SmartHire AI project documentation.