ChatGPT

Africa Global Talent Platform – MVP Product Design

Context and Rationale

Africa's digital workforce is growing rapidly and is projected to expand by **130 % by 2030** [1]. Internet penetration in Africa increased from **39 % in 2019 to 43 % in 2023**, with over **600 million** mobile internet users [2]. The continent is demographically young – the median age is about **19**, and over **60 %** of people are under 25 [3] [4] – creating a deep talent pool. Governments in Kenya, Nigeria and South Africa are investing in digital skills and remote-work policies [5] [6], and hiring African professionals is cost-effective because of lower living costs while still offering high-quality expertise [7]. These trends validate building a global recruiting platform focused on African and diaspora talent.

This document translates the SEO-driven sitemap into a buildable Minimum Viable Product (MVP). It defines core features, data models, system architecture, job ingestion workflows, search and matching logic, and outlines where artificial intelligence (AI) will be introduced later. The goal is to enable a small engineering team to start building immediately while positioning the platform for future growth.

1. MVP Feature Definition

The MVP should support the essential flow: **Candidate discovers a suitable job → applies → Employer reviews → Employer responds**. We avoid feature bloat and defer non-critical functions to later phases.

Candidate-facing Features (IN MVP)

- **User registration and login** (email + password) with basic profile creation. Users identify as candidates.
- **Profile editor** with fields for name, location (country), professional summary, skills, experience, education, and a resume upload. Candidates can update their details and upload a PDF resume.
- **Job search** with keyword search and filters for country, remote-from-Africa, visa sponsorship, relocation assistance and job category/industry. Search results list jobs with key details and eligibility flags.
- **Job detail page** showing full description, eligibility tags (remote/visa/relocation), company overview and application deadline.
- **Apply to job** form capturing a cover letter and submitting the candidate's profile/resume to the employer. The application status is recorded.
- **Application tracker** where candidates see each application's status (applied, under review, shortlisted, hired, rejected).
- **Email notifications** for application submission and status updates.
- **Access to resources** (e.g., visa guidance and career guides) via static pages defined in the SEO plan.

Candidate-facing Features (NOT in MVP)

- AI-driven job recommendations or chatbots.
- In-platform messaging between candidates and employers.
- Community forums or networking features.
- Resume-building wizards or career coaching services.
- Video interview scheduling and live interviews.
- Multi-language UI (future internationalization can be added later).

Employer-facing Features (IN MVP)

- **Employer registration and login** with verification (email or admin approval). Employers create a company profile with name, industry, location, description and logo.
- **Job posting** interface that captures title, description, required skills, location (country or remote), eligibility flags (remote from Africa, visa sponsorship, relocation assistance), job category/industry, and application deadline. Posts are set to “pending” until approved by admin.
- **Employer dashboard** listing all posted jobs, status (pending, approved, closed), and number of applications.
- **Application review** page per job showing candidate profiles, resumes and cover letters. Employers can update an application’s status (under review, shortlisted, hired, rejected) and trigger an email notification to the candidate.
- **Basic candidate communication** via email replies outside the platform (no internal messaging).

Employer-facing Features (NOT in MVP)

- Searching the candidate database or proactively contacting candidates.
- Bulk job uploads or integrations with external ATS systems.
- Advanced analytics on job performance or diversity metrics.
- In-platform chat or interview scheduling.
- Employer branding pages or company blogs.

Admin / Internal Tools (IN MVP)

- **User management**: view and deactivate users (candidates/employers) and assign roles.
- **Job moderation**: approve or reject job postings based on eligibility (must be open to African candidates and not discriminatory). Admins can edit job details to normalise fields.
- **Content management** for static pages (resources, FAQs) defined in the SEO plan.
- **Basic reporting**: counts of users, jobs and applications.

Admin / Internal Tools (NOT in MVP)

- Automated moderation using AI or machine learning.
- Comprehensive analytics dashboards (e.g., conversion funnels, demographics).
- Payment or billing systems (free to use initially).
- Integration with HRIS/ATS products.

2. Core Domain Models

A clean, scalable schema supports the MVP while remaining extensible. The following entities are stored in a relational database (e.g., PostgreSQL). Foreign-key relationships enforce consistency.

Model	Key fields	Relationships	Purpose
User	`id`, `email` (unique), `password_hash`, `role` (enum: candidate, employer, admin), `created_at`, `last_login`	One-to-one with `CandidateProfile` or `Company` depending on role	Base authentication and authorisation entity. Separates user credentials from profiles.
CandidateProfile	`id`, `user_id`, `full_name`, `country`, `summary`, `skills` (array of strings), `experience_years`, `education`, `resume_id` (FK to Resume)	Belongs to `User`; has many `Application`	Stores personal and professional details for candidates. `skills` is a denormalised list to allow simple matching in MVP.
Resume	`id`, `file_path`, `file_name`, `uploaded_at`	Belongs to `CandidateProfile`	References a file stored in S3 (or equivalent). Separated to allow multiple versions later.
Company	`id`, `user_id`, `name`, `industry`, `country`, `description`, `website_url`, `logo_path`	Belongs to `User`; has many `Job`	Stores employer/company information. Allows reuse if multiple people from the same organisation join later.
Job	`id`, `company_id`, `title`, `description`, `required_skills` (array), `country`, `is_remote` (boolean), `visa_sponsorship` (boolean), `relocation_assistance` (boolean), `category` (e.g., engineering, marketing), `posted_at`, `application_deadline`, `status` (pending, active, closed)	Belongs to `Company`; has many `Application`	Represents a job posting. Eligibility flags (`is_remote`, `visa_sponsorship`, `relocation_assistance`) enable filtering and matching.

Model	Key fields	Relationships	Purpose
Application	`id`, `job_id`, `candidate_profile_id`, `cover_letter`, `status` (applied, under_review, shortlisted, hired, rejected), `applied_at`, `updated_at`	Belongs to `Job` and `CandidateProfile`	Captures a candidate's submission to a job. Tracks the application workflow and provides status updates to candidates.
JobEligibilityTag (optional in MVP)	`id`, `job_id`, `tag_type` (enum: remote_from_africa, visa_sponsorship, relocation), `value` (boolean)	Belongs to `Job`	Separate table to support flexible tagging and future expansion (e.g., hybrid roles, part-time). For MVP we may embed flags in `Job` for simplicity.

Why these models?

- **Separation of concerns:** Credentials (User) are separated from profiles (CandidateProfile and Company) for security and clarity. Candidate and employer data diverge in structure and life-cycle.
- **Structured eligibility:** Embedding boolean flags in the job model supports the specific filters required by the platform while allowing a separate tagging table later for more nuanced attributes.
- **Application entity:** Keeps the many-to-many relationship between candidates and jobs clean. It stores statuses and timestamps, enabling reporting and rule-based matching.
- **Resumes stored in file storage:** Large documents should not live in the database; storing metadata and a pointer to S3 keeps the database lean and simplifies file handling.

3. High-Level System Architecture

The architecture balances simplicity, scalability and cost. A monolithic backend supports all core functions but is modular enough to evolve into services later.

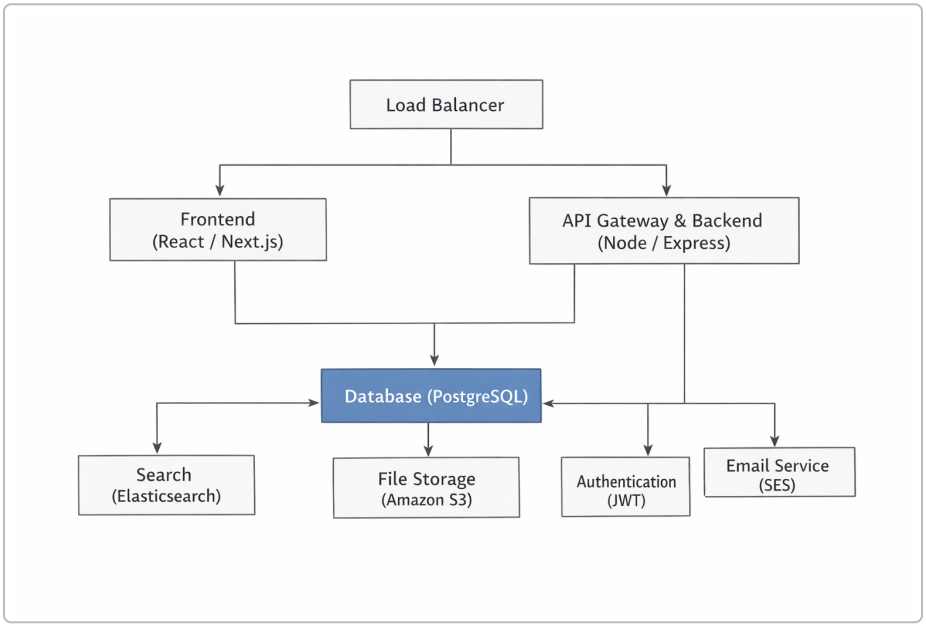
Components

1. **Frontend** – a single-page application built with **React/Next.js** for fast, interactive user experiences and server-side rendering for SEO. It communicates with the backend via RESTful JSON APIs over HTTPS.
2. **API & Backend** – implemented with **Node.js and Express** (or a comparable Python framework such as Django). It exposes endpoints for authentication, profiles, job postings, search, and applications. Business logic enforces validations and status transitions.
3. **Database** – **PostgreSQL** stores structured data and supports relational integrity. It offers robust indexing for common queries and can be hosted via a managed service.
4. **Search Engine** – **Elasticsearch** indexes job descriptions, skills and company names for full-text search and faceted filtering (country, remote, visa sponsorship, relocation, category). The backend syncs jobs to the search index on creation or update.

5. **File Storage** – **Amazon S3** (or equivalent object storage) stores resumes and company logos. Files are uploaded by the backend and served via pre-signed URLs to prevent unauthorised access.
6. **Email/Notification Service** – a managed service such as **Amazon SES** or SendGrid handles transactional emails (account verification, application receipts, status updates).
7. **Authentication** – JSON Web Tokens (JWT) stored in HTTP-only cookies provide stateless authentication. Passwords are hashed (e.g., bcrypt). An optional third-party provider (Auth0) can be integrated later.
8. **Hosting & Deployment** – initial deployment can use a PaaS (e.g., AWS Elastic Beanstalk or Heroku) or containerised Docker images on a small Kubernetes cluster. Continuous integration ensures code is tested and deployed automatically.

Architecture Diagram

Below is a simplified diagram of the proposed architecture. User requests hit the frontend, which communicates with the API server. The API server interacts with the database, search engine, file storage and email service. Authentication uses JWTs stored securely.



Justification of Technology Choices

- **React/Next.js**: widely adopted, developer-friendly and supports server-side rendering for SEO – important because many job searches originate from search engines.
- **Node.js/Express**: lightweight and non-blocking, ideal for I/O-heavy operations such as file uploads and searches. The JavaScript stack enables sharing validation logic between frontend and backend.
- **PostgreSQL**: mature open-source relational database with strong support for JSON fields (for skills arrays) and indexing. Familiar to most engineers and easily managed.
- **Elasticsearch**: designed for full-text search and faceted filtering. It scales horizontally and can handle large volumes of job postings as the platform grows.
- **S3 & SES**: managed services reduce operational overhead. S3 provides durable storage and easy integration; SES (or SendGrid) ensures reliable email delivery.

- **JWT**: stateless authentication fits a distributed architecture and simplifies scaling. Using HTTP-only cookies mitigates XSS attacks.

4. Job Ingestion Strategy

The platform must grow its job database ethically and sustainably while ensuring that roles are genuinely open to African candidates. Ingestion progresses through three phases.

Phase 1 – Manual Job Posting (MVP)

Employers manually create job postings through the employer dashboard. Fields are structured (title, description, location, eligibility flags, skills) to support search and matching. Posts remain in a pending state until an admin reviews and approves them. This ensures that early content aligns with the platform's mission and maintains quality control.

Phase 2 – Admin-curated Job Ingestion

As the job inventory needs to grow, internal staff or trusted partners curate roles from trusted sources (e.g., partner companies, remote job boards). Admins manually input these roles using the same structured form or through a secure back-office interface. During entry they normalise job titles, descriptions and required skills, add eligibility flags (remote, visa, relocation), and ensure that the job is open to African candidates. Duplicate detection compares new postings against existing jobs (matching on company name, title and location) and flags potential duplicates for review.

Phase 3 – External Job Aggregation (API-first)

Once the platform and brand are established, the team can integrate with external job feeds via APIs (with permission from partner boards). A background service fetches jobs, maps external fields to the platform schema, normalises text, and automatically flags remote and Africa-friendly attributes. Duplicate detection uses fuzzy matching (e.g., comparing hashed title + company + date) to prevent multiple copies. Jobs that pass validation go into a "requires approval" queue where admins can verify eligibility before publishing. Ethical considerations include: only aggregating jobs that allow African applicants; obtaining licences from source sites; and avoiding scraping without consent.

Job Normalisation & Eligibility Tagging

- **Normalisation**: Convert varied job titles and descriptions into a consistent format. For example, "Senior Software Engineer (Remote – Africa)" and "Software Engineer – Remote – Open to Kenyan Applicants" should both map to `title=Software Engineer`, `is_remote=true`, `visa_sponsorship=false`, `relocation_assistance=false`, `country=null`. Required skills are extracted via manual input or simple parsing and stored as arrays.
- **Eligibility Tagging**: Each job is tagged with booleans for `is_remote`, `visa_sponsorship` and `relocation_assistance`. Admins verify that remote jobs explicitly allow applicants from African countries. Visa sponsorship tags are applied when employers indicate they will sponsor work visas [8]. Relocation tags indicate roles that include relocation support.

- **Duplicate Detection**: When a new job is ingested, a hash of `company_name + title + location + posted_at` is compared against existing hashes. If a potential duplicate is found, the record is flagged for admin review to merge or discard.

5. Search and Filtering Logic

Search is central to candidate experience. A good search engine must be fast, accurate and flexible.

Filters and Queries

1. **Keyword search** – candidates can enter keywords (job title, skills, company). The search engine uses full-text fields in Elasticsearch to return results ranked by relevance.
2. **Country filter** – selects jobs located in a specific country (e.g., "/jobs/nigeria") or remote roles that explicitly accept candidates from that country. Implemented via a `country` field on the job document and a nested `eligible_countries` array for remote jobs.
3. **Remote from Africa** – a boolean filter on `is_remote=true` combined with `eligible_regions` containing "Africa". Only remote jobs open to African candidates are returned.
4. **Visa sponsorship** – filter on `visa_sponsorship=true`. This surfaces roles where employers offer work-visa support [8].
5. **Relocation assistance** – filter on `relocation_assistance=true`. Useful for candidates willing to relocate.
6. **Job category / industry** – filter on `category`. Categories are predetermined (e.g., software engineering, marketing, finance) to avoid free-text variance. Elasticsearch facets return counts for each category to populate UI filters.

Scalability and Accuracy

- **Indexed fields**: Each filterable attribute is stored as a keyword in Elasticsearch, enabling efficient filtering without scanning all documents.
- **Search–database consistency**: A change data capture mechanism (e.g., database triggers or application hooks) keeps the search index in sync with the database. Deleting or updating a job triggers an index update.
- **Synonyms & stemming**: For keywords, the search engine can implement stemming (e.g., "engineer" vs "engineering") and synonyms ("developer" vs "programmer") to improve recall. However, this can be configured conservatively in MVP.
- **Monitoring & tuning**: Search logs help identify zero-result queries or mismatches, enabling refinement of analyzers and synonyms.

6. Rule-based Matching Logic (Pre-AI)

Before introducing machine learning, the platform can provide explainable match scores between candidates and jobs. The rule-based system promotes transparency and fairness.

Steps to Compute a Match Score

1. **Skill Match (up to 50 points)** – Compare a candidate's `skills` array to a job's `required_skills`. Calculate the proportion of required skills that the candidate possesses (e.g., 3 out of 5 → 30 points). Optionally weight certain skills higher if marked as critical by the employer.
2. **Eligibility Match (up to 30 points)** – Check whether the candidate's country is permitted by the job's `country` or `eligible_countries` field. If the job is remote and open to all African countries, candidates from any African country score full points. If the job requires relocation and offers visa sponsorship, ensure the candidate is willing to relocate (a candidate-profile flag).
3. **Experience Match (up to 20 points)** – Compare the candidate's `experience_years` to the employer's desired range (if provided). Full points are awarded when the candidate's experience is within ±2 years of the required range; partial points if outside the range.

Output and Explanation

The backend returns a **match score** (0–100) along with a **breakdown** explaining how the score was derived. For example:

> *Match Score: 75/100.* **Skills:** You match 3 of the 5 required skills (30 points). **Eligibility:** Your country (Kenya) is eligible for this remote role (30 points). **Experience:** You have 4 years' experience vs. 5 years required (15 points). *Recommendation:* Add "TypeScript" to your skills to improve your match.

Reasons for mismatch (e.g., ineligible location) are explicitly presented so candidates understand why some jobs are not recommended. Employers see a similar breakdown when reviewing applicants. This transparency addresses fairness concerns and avoids black-box decisions.

7. AI Readiness (Future Phases)

Machine learning can enhance search and matching, but only after foundational data and processes are in place. Preparing for AI requires collecting structured, high-quality data and establishing governance.

Future AI Enhancements

1. **Personalised job recommendations** – models learn from candidate behaviour (clicks, applications) and suggest jobs beyond explicit searches.
2. **Resume parsing and skill extraction** – natural language processing (NLP) can extract structured information from uploaded resumes, reducing manual data entry and improving matching accuracy.
3. **Intelligent job ingestion** – AI can help classify job descriptions, detect eligibility for African candidates, and highlight potential biases.
4. **Chatbots and virtual assistants** – answer candidate questions or guide employers through posting jobs.

Data Requirements & Guardrails

- **Data collection**: Gather anonymised logs on search queries, application outcomes, and job-candidate interactions. Ensure consent is obtained and provide opt-out mechanisms.

- **Bias mitigation**: Monitor models for geographic or demographic bias. Do not train on sensitive attributes (race, ethnicity, religion, etc.). Only use relevant signals such as skills, experience and location.
- **Transparency**: Provide explanations for AI-driven recommendations. Users should be able to understand why a job or candidate was suggested.
- **Ethical sourcing**: Follow partner sites' terms when ingesting data. Avoid scraping without permission.

8. Next Engineering Steps

1. **Set up repositories and CI/CD** – Initialise code repositories (frontend and backend) and configure automated testing and deployment pipelines.
2. **Design database schema** – Implement the tables described above in PostgreSQL. Write migration scripts and ensure indexes on searchable fields.
3. **Implement authentication** – Build user registration, login and JWT issuance. Secure password storage with bcrypt.
4. **Develop basic CRUD endpoints** – Create API routes for users, profiles, companies, jobs and applications. Validate input and enforce role-based access control.
5. **Integrate search engine** – Set up an Elasticsearch cluster, define job index mappings and implement indexing logic in the backend. Build search endpoints with filters.
6. **Build frontend components** – Develop pages for registration/login, profile creation, job search and filters, job details, application forms and dashboards (candidate and employer).
7. **Set up file uploads** – Integrate S3 for resume and logo uploads with pre-signed URLs. Store metadata in the database.
8. **Implement admin tools** – Provide an admin interface for approving jobs and managing users. Basic dashboards can be built with off-the-shelf admin templates.
9. **Deploy and test** – Use a staging environment for functional testing. Conduct usability tests with a small group of candidates and employers. Iterate based on feedback.

By focusing on these foundational elements, the platform will deliver a compelling user experience while honouring the mission of connecting African talent to global opportunities. Subsequent phases can build on this solid base, incorporating AI, advanced analytics and richer features without compromising fairness or transparency.

---

1 2 4 How Africa's Talent Marketplace is Reshaping Remote Work | Breedj
https://www.breedj.com/how-africas-talent-marketplace-is-reshaping-remote-work/

3 5 6 7 Why Africa is the Next Big Destination for Remote Talent - Talenteum
https://talenteum.com/why-africa-is-the-next-big-destination-for-remote-talent/

8 Countries Offering Easy Work Visa Sponsorship | For Africans
https://visaliv.com/forum/main-forum/what-countries-can-you-easily-get-a-work-visa-sponsorship-as-an-african