give me data base structure

Below is a clean PostgreSQL database structure for the web app MVP, written in plain terms with suggested column types, keys, and constraints. It follows standard relational design for a job portal with users, companies, jobs, resumes, and applications.[1] [2]

Conventions

- Use UUIDs for primary keys; timestamps are UTC; enums via CHECK constraints or PostgreSQL enums; jsonb only where flexibility helps (skills, profile sections).[2] [3]

users

- Purpose: All accounts (candidate, company user, admin).[4] [5]
- Columns:
  - id UUID PK, email TEXT UNIQUE NOT NULL, hashed_password TEXT NOT NULL, role TEXT CHECK (role IN ('candidate','company','admin')) NOT NULL, status TEXT CHECK (status IN ('active','disabled')) DEFAULT 'active' NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[3] [2]

organizations

- Purpose: Company records.[6] [1]
- Columns:
  - id UUID PK, name TEXT NOT NULL, website TEXT, verification_status TEXT CHECK (verification_status IN ('pending','approved','suspended')) DEFAULT 'pending' NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[1] [2]

org_users

- Purpose: Many-to-many link between users and organizations, with an org-specific role.[7] [8]
- Columns:
  - id UUID PK, org_id UUID NOT NULL FK → organizations.id ON DELETE CASCADE, user_id UUID NOT NULL FK → users.id ON DELETE CASCADE, org_role TEXT CHECK (org_role IN ('owner','recruiter','viewer')) NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[2] [7]
- Constraints: UNIQUE (org_id, user_id) to avoid duplicates.[3] [2]

candidate_profiles

- Purpose: Extra profile info for candidates (1:1 with users).[4] [1]
- Columns:
  - user_id UUID PK FK → users.id ON DELETE CASCADE, full_name TEXT, city TEXT, summary TEXT, skills_json JSONB, education_json JSONB, experience_json JSONB, links_json JSONB, updated_at TIMESTAMPTZ DEFAULT now() NOT NULL.[1] [2]

resumes

- Purpose: Uploaded or AI-generated resumes with structured content.[2] [1]
- Columns:
  - id UUID PK, user_id UUID NOT NULL FK → users.id ON DELETE CASCADE, storage_path TEXT NOT NULL, parsed_json JSONB, template_id UUID NULL, source TEXT CHECK (source IN ('uploaded','generated')) NOT NULL, version_label TEXT, active BOOLEAN DEFAULT false NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[1] [2]
- Index: (user_id, active) for quick default-resume lookup.[3] [2]

jobs

- Purpose: Job postings by organizations.[6] [1]
- Columns:
  - id UUID PK, org_id UUID NOT NULL FK → organizations.id ON DELETE CASCADE, title TEXT NOT NULL, location_type TEXT CHECK (location_type IN ('remote','hybrid','onsite')) NOT NULL, employment_type TEXT CHECK (employment_type IN ('full-time','part-time','intern','contract')) NOT NULL, experience_min INT, experience_max INT, skills_required_json JSONB, skills_preferred_json JSONB, jd_text TEXT NOT NULL, status TEXT CHECK (status IN ('draft','open','closed')) DEFAULT 'open' NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[2] [1]
- Indexes: org_id, status, GIN on skills_required_json for filters.[3] [2]

applications

- Purpose: Candidate applications to jobs.[1] [2]
- Columns:
  - id UUID PK, job_id UUID NOT NULL FK → jobs.id ON DELETE CASCADE, user_id UUID NOT NULL FK → users.id ON DELETE CASCADE, resume_id UUID FK → resumes.id ON SET NULL, stage TEXT CHECK (stage IN ('applied','shortlisted','interview','offer','hired','rejected')) DEFAULT 'applied' NOT NULL, decision TEXT NULL CHECK (decision IN ('offer','reject')), created_at TIMESTAMPTZ DEFAULT now() NOT NULL, updated_at TIMESTAMPTZ DEFAULT now() NOT NULL.[2] [1]

- Indexes: job_id, user_id, (job_id, stage) for pipeline views.[3] [2]
- Trigger: updated_at auto-update on row change (optional).[2] [3]

scores

- Purpose: Store fit scoring per application.[5] [2]
- Columns:
  - application_id UUID PK FK → applications.id ON DELETE CASCADE, keyword_score INT CHECK (keyword_score BETWEEN 0 AND 100), semantic_score INT NULL CHECK (semantic_score BETWEEN 0 AND 100), fit_flag BOOLEAN, explanation TEXT.[3] [2]

notes

- Purpose: Recruiter notes on applications.[1] [2]
- Columns:
  - id UUID PK, application_id UUID NOT NULL FK → applications.id ON DELETE CASCADE, author_user_id UUID NOT NULL FK → users.id ON DELETE SET NULL, text TEXT NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[1] [2]
- Index: application_id for quick retrieval.[2] [3]

admin_audit

- Purpose: Audit trail for admin (and sensitive) actions.[4] [2]
- Columns:
  - id UUID PK, actor_user_id UUID NOT NULL FK → users.id ON DELETE SET NULL, action TEXT NOT NULL, target_type TEXT CHECK (target_type IN ('user','org','job','application')) NOT NULL, target_id UUID, payload_json JSONB, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[3] [2]

templates (optional, for resume rendering)

- Purpose: Manage resume templates.[2] [3]
- Columns:
  - id UUID PK, name TEXT NOT NULL, html_template_path TEXT NOT NULL, css_path TEXT, status TEXT CHECK (status IN ('active','inactive')) DEFAULT 'active' NOT NULL, created_at TIMESTAMPTZ DEFAULT now() NOT NULL.[3] [2]

Integrity and performance checklist

- Mandatory uniques: users.email UNIQUE; org-user pair UNIQUE(org_id, user_id).[2] [3]
- Foreign keys with ON DELETE to maintain referential integrity and prevent orphans.[7] [3]
- Useful indexes: users.email; jobs.(org_id,status); applications.(job_id,user_id,stage); JSONB GIN on skills fields if used for filters.[3] [2]

Example creation order (SQL migration 001_initial.sql)

- Create tables in dependency order: users → organizations → org_users → candidate_profiles → resumes → jobs → applications → scores → notes → admin_audit → templates.[9] [10]
- Wrap in a transaction for atomic migration; add down migration to drop in reverse order.[10] [9]

Environment and migrations

- Use a migration tool (Alembic/Prisma/TypeORM) to apply 001_initial.sql and track future changes (002_indexes.sql, 003_constraints.sql, etc.).[9] [10]
- Maintain an ERD and short data dictionary in the repo for clarity across the team.[11] [12]

This schema is normalized, enforces relationships, and supports core queries for a job portal while leaving room for growth (semantic scores, AI resume generation, and richer search) via additional migrations.[1] [2]

⁂

1. https://www.geeksforgeeks.org/sql/how-to-design-a-relational-database-for-online-job-portal/
2. https://www.cockroachlabs.com/blog/database-schema-beginners-guide/
3. https://aloa.co/blog/database-schema
4. https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5
5. https://www.geeksforgeeks.org/dbms/how-to-design-a-database-for-web-applications/
6. https://vertabelo.com/blog/designing-a-database-for-an-online-job-portal/
7. https://www.acceldata.io/blog/database-relationships-explained-key-concepts-and-best-practices
8. https://www.geeksforgeeks.org/dbms/database-design-in-dbms/
9. https://www.dbvis.com/thetable/introduction-to-database-migration-a-beginners-guide/
10. https://dev.to/dbvismarketing/how-to-start-with-database-migrations-1ce4
11. https://miro.com/diagramming/how-to-design-database-schema/
12. https://www.manifest.ly/use-cases/software-development/database-design-checklist