ChatGPT

Database Schema Notes

- **Enumerated types** are used for `user_role`, `job_status`, and `application_status` to enforce valid values and simplify validation. They must be created before tables.
- **users** holds authentication credentials and a role. Email is unique; password hashes are stored securely.
- **candidate_profiles** maintains personal data for each candidate and has a one-to-one relationship with `users` (enforced by a `UNIQUE` constraint on `user_id`). The `skills` column uses the `TEXT[]` array type because skills lists are short and homogenous; a JSON column is unnecessary at this stage. `resume_id` is a foreign key to `resumes` but may be null if no resume is uploaded.
- **resumes** stores metadata about uploaded resumes. Each candidate can have at most one resume (`UNIQUE` on `candidate_profile_id`). File content resides in object storage; `file_path` stores its path.
- **companies** stores employer information and has a one-to-one relationship with `users` (unique `user_id`). Companies may expand to multiple users later; this schema can be adjusted by removing the unique constraint.
- **jobs** captures job postings. `required_skills` is stored as a `TEXT[]` array for simple inclusion checks. `eligible_countries` allows remote jobs to specify accepted countries. A `search_vector` column and trigger maintain a full-text index over `title`, `description`, and `required_skills` using PostgreSQL's `tsvector` type for efficient keyword search.
- **applications** links candidates to jobs and records their status. A unique constraint on `(job_id, candidate_profile_id)` prevents duplicate applications.
- **Indexes**: B-tree indexes on frequently filtered fields (`country`, `category`, `is_remote`, `visa_sponsorship`, `relocation_assistance`, `status`) accelerate queries and filter operations. GIN indexes on `skills` and `search_vector` enable efficient array and full-text search. Foreign-key columns (`job_id`, `candidate_profile_id`) also have indexes for fast joins.

Migration Plan

1. **Create enumerated types** (`user_role`, `job_status`, `application_status`). These cannot be dropped if data exists.
2. **Create tables** in order of dependencies: `users`, `candidate_profiles`, `resumes`, `companies`, `jobs`, then `applications`.
3. **Create full-text search function and trigger** after `jobs` table.
4. **Create indexes** for filtering and full-text search.
5. If making changes later (e.g., adding columns or types), use `ALTER TABLE` statements and create new enum values with `ALTER TYPE ... ADD VALUE`.
6. Use a migration tool (e.g., Flyway, Liquibase, or Knex) to version-control these scripts and ensure environments (dev/staging/prod) apply the same migrations.

These scripts align with the API contracts. For instance, the job `status` enum matches the job moderation API; application statuses mirror the lifecycle defined in the API spec. Developers should keep these enums and constraints synchronized when updating the application logic.