Database Design Document

Job Portal with Smart Resume–Job Matching

1. Overview

This document details the database design for the Job Portal application. The database selected is **PostgreSQL**. It stores users (candidates & recruiters), resumes, job postings, applications, AI matching results, and authentication tokens. The design focuses on data integrity, security, and performance for a web-scale mini-project.

2. Database Technology

- Primary DBMS: PostgreSQL (12+ recommended)
- Access: via SQLAlchemy ORM (FastAPI integration) or async ORM (Tortoise/SQLModel)
- Backups: Regular pg_dump exports and point-in-time recovery if using WAL archiving.

3. Entity-Relationship (ER) Overview

Main entities:
- users (candidate or recruiter)
- resumes (one per candidate, or multiple revisions)
- jobs (posted by recruiters)
- applications (candidate applies to a job)
- recommendations (AI similarity results stored for fast retrieval)
- tokens (refresh tokens / sessions)

Relationships:
- users (1) — (M) resumes
- users (recruiter) (1) — (M) jobs
- users (candidate) (1) — (M) applications — (M) jobs (many-to-many via applications)
- jobs (1) — (M) recommendations (each job can have many recommended candidates)

4. Table Schemas

Below are recommended table definitions with columns, types, constraints, and indexes. Use UUIDs for primary keys in production if preferred.

4.1 users

CREATE TABLE users (id SERIAL PRIMARY KEY, role VARCHAR(20) NOT NULL CHECK (role IN ('candidate','recruiter','admin')), name VARCHAR(200) NOT NULL, email VARCHAR(255) UNIQUE NOT NULL, password_hash VARCHAR(255) NOT NULL, profile_json JSONB, -- flexible profile fields is_active BOOLEAN DEFAULT TRUE, created_at TIMESTAMP WITH TIME ZONE DEFAULT now(), updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()); -- Indexes CREATE INDEX idx_users_email ON users(email);

4.2 resumes

CREATE TABLE resumes (id SERIAL PRIMARY KEY, user_id INTEGER REFERENCES users(id) ON DELETE CASCADE, filename VARCHAR(255), storage_path VARCHAR(512), -- S3/Server path text_content TEXT, -- extracted resume text parsed_skills TEXT[], -- optional array of skills uploaded_at TIMESTAMP WITH TIME ZONE DEFAULT now(), is_active BOOLEAN DEFAULT TRUE); CREATE INDEX idx_resumes_user ON resumes(user_id); CREATE INDEX idx_resumes_skills ON resumes USING GIN(parsed_skills);

4.3 jobs

CREATE TABLE jobs (id SERIAL PRIMARY KEY, recruiter_id INTEGER REFERENCES users(id) ON DELETE SET NULL, title VARCHAR(255) NOT NULL, description TEXT, required_skills TEXT[], -- array of skills location VARCHAR(255), employment_type VARCHAR(50), created_at TIMESTAMP WITH TIME ZONE DEFAULT now(), updated_at TIMESTAMP WITH TIME ZONE DEFAULT now(), is_active BOOLEAN DEFAULT TRUE); CREATE INDEX idx_jobs_recruiter ON jobs(recruiter_id); CREATE INDEX idx_jobs_skills ON jobs USING GIN(required_skills);

4.4 applications

CREATE TABLE applications (id SERIAL PRIMARY KEY, job_id INTEGER REFERENCES jobs(id) ON DELETE CASCADE, candidate_id INTEGER REFERENCES users(id) ON DELETE CASCADE, resume_id INTEGER REFERENCES resumes(id) ON DELETE SET NULL, cover_letter TEXT, status VARCHAR(50) DEFAULT 'applied', -- applied, shortlisted, rejected, interviewed, hired applied_at TIMESTAMP WITH TIME ZONE DEFAULT now()); CREATE INDEX idx_applications_job ON applications(job_id); CREATE INDEX idx_applications_candidate ON applications(candidate_id);

4.5 recommendations

CREATE TABLE recommendations (id SERIAL PRIMARY KEY, job_id INTEGER REFERENCES jobs(id) ON DELETE CASCADE, candidate_id INTEGER REFERENCES users(id) ON DELETE CASCADE, resume_id INTEGER REFERENCES resumes(id), similarity_score REAL NOT NULL, computed_at TIMESTAMP WITH TIME ZONE DEFAULT now()); CREATE INDEX idx_recommendations_job ON recommendations(job_id); CREATE INDEX idx_recommendations_candidate ON recommendations(candidate_id); CREATE INDEX idx_recommendations_score ON recommendations(job_id, similarity_score DESC);

4.6 tokens

CREATE TABLE tokens (id SERIAL PRIMARY KEY, user_id INTEGER REFERENCES users(id) ON DELETE CASCADE, refresh_token_hash VARCHAR(255) NOT NULL, user_agent TEXT, ip_address INET, created_at TIMESTAMP WITH TIME ZONE DEFAULT now(), expires_at TIMESTAMP WITH TIME ZONE); CREATE INDEX idx_tokens_user ON tokens(user_id);

5. Relationships & Constraints

- Foreign keys enforce referential integrity (ON DELETE behaviours as suggested).
- Use CHECK constraints for enums (roles, application status).
- Use JSONB for flexible profile fields and future-proofing.
- Use GIN indexes for array/JSONB fields to speed up skill-based queries.

6. Security & Optimization

Security best practices:
- Use parameterized queries / ORM to prevent SQL injection.
- Principle of least privilege: database user has minimal rights (no superuser for app).
- Encrypt sensitive data at rest and in transit (TLS).
- Hash refresh tokens and passwords (bcrypt) — never store plaintext.
- Validate and sanitize all inputs in backend.

Optimization:
- Add indexes on frequently queried columns (email, job_id, candidate_id).
- Use GIN indexes for full-text search or array fields (skills).
- Consider materialized views for expensive aggregations (e.g., top jobs).
- Partition large tables if data grows (applications, recommendations).
- Use EXPLAIN ANALYZE to optimize slow queries.

7. Backup & Maintenance

- Regular full backups: pg_dumpall or pg_basebackup.
- Point-in-time recovery (PITR) via WAL archiving for critical systems.
- Routine vacuum & analyze to maintain stats.
- Monitor disk, slow queries, connections; set alerts.
- Test backups periodically by restoring to staging environment.

8. Example Queries

-- 1. Fetch top 10 recommended candidates for a job (from recommendations table) SELECT r.candidate_id, u.name, r.similarity_score FROM recommendations r JOIN users u ON u.id = r.candidate_id WHERE r.job_id = 10 ORDER BY r.similarity_score DESC LIMIT 10; -- 2. Get recommended jobs for a candidate (join recommendations and jobs) SELECT j.id, j.title, rec.similarity_score FROM recommendations rec JOIN jobs j ON j.id = rec.job_id WHERE rec.candidate_id = 5 ORDER BY rec.similarity_score DESC; -- 3. Full-text search on job descriptions (requires tsvector column/index) SELECT id, title FROM jobs WHERE to_tsvector('english', description) @@ plainto_tsquery('python developer');

9. FastAPI Integration (SQLAlchemy models)

Use SQLAlchemy (or SQLModel) to define ORM models. Example SQLAlchemy model for Job and Recommendation simplified below: from sqlalchemy import Column, Integer, String, Text, ARRAY, ForeignKey, Float, DateTime, func from sqlalchemy.orm import relationship from database import Base class Job(Base): __tablename__ = 'jobs' id = Column(Integer, primary_key=True, index=True) recruiter_id = Column(Integer, ForeignKey('users.id')) title = Column(String, nullable=False) description = Column(Text) required_skills = Column(ARRAY(String)) created_at = Column(DateTime(timezone=True), server_default=func.now()) recommendations = relationship('Recommendation', back_populates='job') class Recommendation(Base): __tablename__ = 'recommendations' id = Column(Integer, primary_key=True, index=True) job_id = Column(Integer, ForeignKey('jobs.id')) candidate_id = Column(Integer, ForeignKey('users.id')) resume_id = Column(Integer) similarity_score = Column(Float) computed_at = Column(DateTime(timezone=True), server_default=func.now())

10. Closing Notes

This design aims to be practical for a 3rd-year mini-project while following best practices for security, performance, and maintainability. You can extend the schema with additional tables for interviews, company profiles, analytics, and audit logs as needed.