Techie Dating Backend – Detailed SRD & Development Prompt

1. Project Goal

Build a backend for a dating platform targeting techies and engineers, starting with MVP features and allowing future expansion.

---

2. Tech Stack

- Language: Java 17+
- Framework: Spring Boot 3.x
- Database: PostgreSQL (Flyway for migrations)
- ORM: Spring Data JPA (Hibernate)
- Authentication: Spring Security + JWT
- Object Mapping: MapStruct
- File Storage: Cloudinary (Free Tier)
- Testing: JUnit 5, Mockito, Testcontainers
- API Docs: Springdoc OpenAPI (Swagger UI)
- Data Generation: Java Faker
- Password Hashing: BCrypt
- Validation: Hibernate Validator (JSR-380)
- Optional Caching: Redis

---

3. Database Schema

app_user

- id (UUID, PK)
- email (VARCHAR, unique, not null)
- password (VARCHAR, not null)
- created_at (TIMESTAMP, default now())

profile

- id (UUID, PK)
- user_id (UUID, FK → app_user)
- display_name (VARCHAR, not null)
- bio (TEXT)
- gender (VARCHAR)
- dob (DATE)

- latitude (DOUBLE)
- longitude (DOUBLE)
- experience_yrs (INT)

skill

- id (SERIAL, PK)
- name (VARCHAR, unique, not null)

user_skill

- user_id (UUID, FK → app_user, PK part)
- skill_id (INT, FK → skill, PK part)
- level (INT, 1–5)

photo

- id (UUID, PK)
- user_id (UUID, FK → app_user)
- url (VARCHAR, not null)
- is_primary (BOOLEAN, default false)

like_event

- id (UUID, PK)
- from_user (UUID, FK → app_user)
- to_user (UUID, FK → app_user)
- created_at (TIMESTAMP, default now())

match_pair

- id (UUID, PK)
- user_a (UUID, FK → app_user)
- user_b (UUID, FK → app_user)
- matched_at (TIMESTAMP, default now())

conversation

- id (UUID, PK)
- match_id (UUID, FK → match_pair)

message

- id (UUID, PK)
- conversation_id (UUID, FK → conversation)
- sender_id (UUID, FK → app_user)
- content (TEXT, not null)
- sent_at (TIMESTAMP, default now())

---

4. API Endpoints & Data Contracts

Base Path: `/api/v1`

Auth

POST `/auth/register`

```
{
  "email": "user@example.com",
  "password": "Secret123",
  "displayName": "Alice",
  "dob": "1995-05-12",
  "gender": "Female"
}
```

Response:

```
{"userId": "uuid", "message": "Registration successful"}
```

POST `/auth/login`

```
{"email": "user@example.com", "password": "Secret123"}
```

Response:

```
{"accessToken": "jwt-token", "refreshToken": "jwt-refresh-token"}
```

Profile

GET `/users/me` → Returns logged-in profile. PUT `/users/me` → Update profile.

Skills

GET `/skills` → List all skills. POST `/users/me/skills` → Add skills.

Search

GET `/search?skills=java,spring&distanceKm=50&minExp=2&maxExp=10` Returns profiles matching filters.

Likes & Matches

POST `/likes` → Like a user. GET `/matches` → List matches.

Messages

POST `/conversations/{id}/messages` → Send message in a match conversation.

---

5. Matchmaking Logic

- Score = (shared skills ÷ total unique skills) × 100
- Filter by distance & active profiles.

---

6. Free Integrations

- Cloudinary: profile pictures.
- Gravatar: default avatars.
- Mapbox Geocoding API: location lookup.
- Faker: fake seed data.

---

7. Development Extras

- CORS enabled for frontend.
- Swagger UI via Springdoc OpenAPI.
- Flyway migrations.
- Testcontainers for integration tests.

This document serves as both the SRD and implementation prompt for building the complete backend application.