1. User Stories and Requirements

User Roles:

- **Company/Employer**: The user who will upload job posts and receive ranked resumes.
- **Admin (Optional)**: For managing the platform.

User Stories:

1. **Company User:**
    - As a company user, I want to create a job post with detailed criteria so that I can define the qualifications required.
    - As a company user, I want to upload multiple resumes in different formats (PDF, DOC) so that they can be analyzed.
    - As a company user, I want to receive a sorted list of resumes ranked by how well they match the job criteria so that I can quickly find the best candidates.
    - As a company user, I want to view individual resumes and their match score to assess the qualifications of the candidates.
2. **Admin (Optional):**
    - As an admin, I want to manage company users (create, edit, delete) so that I can control access to the platform.
    - As an admin, I want to oversee the parsing and ranking processes to ensure accuracy and fairness.

---

2. Core Features

1. **Job Posting**: Allow employers to create and submit job posts with detailed job requirements (skills, experience, education).
2. **Resume Upload**: Employers can upload multiple resumes in PDF or DOC format.
3. **Resume Parsing**: Automatically extract details (e.g., name, education, experience) from the uploaded resumes.
4. **AI Matching & Ranking**: Rank resumes based on how closely they match the job post's criteria using AI/NLP.
5. **Dashboard**: Display job posts, uploaded resumes, and the ranked list of candidates.

---

3. Technical Specifications

Frontend (Next.js)

- **UI Framework**: Tailwind CSS for responsive design.

- **Pages**:
    - **Login/Register**: Secure authentication for employers.
    - **Dashboard**: Overview of posted jobs and resumes.
    - **Job Post Form**: A form to create new job listings.
    - **Resume Upload**: A drag-and-drop or form-based interface to upload resumes.
    - **Resume Results**: A table/list displaying ranked resumes based on job criteria.

Backend (Nest.js)

- **Authentication**: JWT-based for secure login.
- **API Endpoints**:
    - **POST /jobs**: Create a new job post.
    - **GET /jobs**: Get a list of all job posts.
    - **POST /resumes**: Upload and parse resumes.
    - **GET /resumes/**
      : Get ranked resumes for a specific job post.
- **AI/NLP Integration**:
    - Use **spaCy** or **Hugging Face** for natural language processing.
    - Develop algorithms to analyze and rank resumes based on job criteria.

Database (Postgres)

- **Schema Design**:
    - **User**: Stores employer and admin details.
    - **JobPost**: Stores job posts and their criteria (e.g., skills, experience).
    - **Resume**: Stores parsed resumes, including skills, education, and work experience.
    - **ResumeJobMatch**: Stores the matching scores between resumes and job posts.

---

4. High-Level Architecture

1. **Frontend (Next.js)**:
    - Handles user interactions, form submissions (job post, resume upload), and displays results.
2. **Backend (Nest.js)**:
    - Serves the API to manage job posts and resumes.
    - Performs the AI-powered matching of resumes to job criteria using NLP.
3. **Database (Postgres)**:
    - Stores job post details, parsed resume data, and matching scores.
4. **NLP/AI Service**:
    - A service integrated within the backend to process resumes, analyze text, and return scores.

---

5. Next Steps for Implementation (Phase 2 onwards)

- Set up the **project architecture** (Next.js frontend, Nest.js backend, Postgres DB).
- Create basic API structure and design frontend pages based on user stories.
