Project Name : DomaiNetHire  
Defination : "An AI-powered job hiring platform that intelligently matches employees and companies based on resume content, skill domains, and recruitment needs. The platform facilitates a seamless hiring lifecycle, including secure user registration, resume parsing and categorization, intelligent recommendations, domain-specific talent queries, automated email communication, and company-hosted coding and aptitude contests. Designed with scalability, automation, and user engagement in mind, the system enhances the efficiency of recruitment for both employers and job seekers."

**Section 1: Additional Features**

To improve the platform's usability, engagement, and effectiveness, consider adding:

**1. Job Posting & Application Tracking**

* Companies can post jobs.
* Employees can apply, save jobs, and track application status.

**2. Skill-based Search**

* Companies can search candidates by skills and experience.
* Employees can search jobs by preferred skills or companies.

**3. Admin Dashboard**

* Admin can monitor platform activity, flag content, manage domains, and generate analytics reports.

**4. In-platform Messaging**

* Enables direct communication between companies and employees.

**5. Resume Builder Tool**

* Assists employees in creating optimized resumes with templates.

**6. Interview Scheduling & Video Interview Module**

* Integration with calendar apps.
* Built-in video call support or third-party integration (Zoom, Google Meet).

**7. Gamification for Contest Participation**

* **Badges**, leaderboards, and performance stats.

**8. AI-Driven Job Fit Score**

* Shows how well a resume matches a job based on NLP.

**Section 2: Design Approach (Start to Deployment)**

**Phase 1: Requirements & Planning**

1. Stakeholder meetings and documentation.
2. Define feature scope and user personas.
3. Choose tech stack (e.g., MERN, Django + React, or Next.js + Node.js).

**Phase 2: System Design**

1. **High-Level Architecture**
   * Frontend: React/Next.js
   * Backend: Node.js/Express or Django REST Framework
   * Database: PostgreSQL or MySQL
   * File Storage: AWS S3 for resumes
   * ML/NLP Service: Python microservice (Flask/FastAPI)
   * Email: SendGrid or AWS SES
2. **API Contracts**
   * Define OpenAPI specs for services.

**Phase 3: Development**

1. Develop user authentication and authorization.
2. Build CRUD APIs for users, companies, resumes, contests.
3. Integrate resume parser and domain classifier.
4. Implement recommendation and communication modules.
5. Set up email notifications.

**Phase 4: Testing**

1. Unit testing (Jest, PyTest, Postman).
2. Integration testing.
3. UAT with selected users.

**Phase 5: Deployment**

1. CI/CD with GitHub Actions, Jenkins, or GitLab.
2. Deploy to cloud: AWS/GCP/Azure.
3. Use Docker + Kubernetes for scalability.

**Phase 6: Maintenance & Monitoring**

1. Error logging: Sentry or LogRocket.
2. Monitoring: Prometheus + Grafana or Datadog.

**Section 3: Modules Overview**

**1. Authentication & Authorization**

* JWT-based secure login.
* Role-based access control (admin, employee, company).

**2. User Management**

* Profile creation/editing.
* Resume upload & profile completeness tracking.

**3. Resume Parser & Categorizer**

* NLP-based service to extract text, skills, experience.
* Classifies resumes into predefined domains (e.g., Finance, AI, Marketing).

**4. Job & Company Management**

* Company profile, domain preferences, job requirements.
* Domain-specific job posts.

**5. Recommendation Engine**

* Matches employee resume domains with company domains.
* Recommends companies to employees and vice versa.

**6. Matching Query System**

* Companies can query employee count per domain or skill set.

**7. Email Notification System**

* Send email alerts to matched employees or contest participants.

**8. Contest Management**

* Companies can create coding/aptitude tests.
* Employees get notified and can register and participate.

**9. Admin Dashboard**

* Manage all users, domains, monitor system metrics.

**Section 4: Database Schema**

Estimate: **14-16 Tables**

**Core Tables**

| **Table Name** | **Purpose** |
| --- | --- |
| users | Stores login credentials, role info (admin, employee, company). |
| employees | Employee-specific data (profile, preferences, resume link). |
| companies | Company-specific data (profile, domains, job requirements). |
| resumes | Metadata and file paths for uploaded resumes. |
| domains | List of all supported job domains (AI, Finance, etc.). |
| resume\_domains | Mapping between resumes and categorized domains. |
| jobs | Job postings by companies. |
| job\_applications | Track applications by employees. |

**Matching & Communication**

| **Table Name** | **Purpose** |
| --- | --- |
| domain\_requirements | Domains and skills companies are hiring for. |
| employee\_recommendations | Stores recommended companies for employees. |
| emails\_sent | Tracks email communications to avoid duplication. |

**Contests**

| **Table Name** | **Purpose** |
| --- | --- |
| contests | Contest details: title, company\_id, type, rules, dates. |
| contest\_participants | Links employees with contests. |
| contest\_scores | Stores results and scores of participants. |

**Admin & Misc**

| **Table Name** | **Purpose** |
| --- | --- |
| admin\_actions | Tracks changes made by admins. |
| logs | System events, audit trails. |

**Section 5: Implementation Steps**

**Step 1: User Role Analysis**

* **Admin**: Manages domains, monitors platform.
* **Employee**: Uploads resumes, gets recommendations, joins contests.
* **Company**: Posts jobs, organizes contests, views employee stats.

**Step 2: Core Feature Development**

1. **Authentication System** (JWT + bcrypt).
2. **Resume Upload & Parsing**:
   * Use tools like **PyPDF2**, **docx**, or **Textract** for parsing.
   * Extract keywords, experience, skills.
3. **NLP-based Categorization**:
   * Use **TF-IDF + SVM** or **BERT-based** models to classify resumes.
   * Train model on labeled resume data with domain tags.
4. **Recommendation Engine**:
   * Match employee domain with company requirements.
   * Use cosine similarity or classification probability.
5. **Company Queries**:
   * Endpoint: /query/matching-employees?domain=AI
   * Returns count and list (optionally) of matching resumes.
6. **Email Notifications**:
   * Schedule async tasks (Celery, Bull.js) for email triggers.

**Step 3: Contest Management System**

* Company creates contest → employees get notified → signup → attend contest.
* Add scoring logic and leaderboard.

**Step 4: Admin Panel**

* View user stats, contest reports, flagged resumes.
* Update domain lists, manage system configurations.

**Step 5: Final Integration and Testing**

* Integrate frontend (React + Tailwind UI).
* Conduct testing: UI, backend API, and full workflows.

**Section 6: Tech Stack Suggestion**

| **Layer** | **Technology** |
| --- | --- |
| **Frontend** | React.js / Next.js + TailwindCSS |
| **Backend** | Node.js (Express) or Django REST |
| **Database** | PostgreSQL or MySQL |
| **File Storage** | AWS S3 / Cloudinary |
| **Resume Parser** | Python (Flask/FastAPI), spaCy/NLTK |
| **Email** | SendGrid / AWS SES |
| **AI Models** | Python (scikit-learn, HuggingFace Transformers) |
| **Containerization** | Docker + Kubernetes |
| **CI/CD** | GitHub Actions / Jenkins |
| **Monitoring** | Sentry, Grafana, Prometheus |

Would you like diagrams (like ERD or architecture) or code templates for any module?