Assignment: AI-Powered Job Tracker with Smart Matching

Timeline: 2–3 Days
Difficulty Level: Intermediate–Advanced
Focus Areas: Product Thinking · AI Integration · Frontend + Backend · System Design

---

Problem Statement

Build an **AI-powered job tracking platform** that:

- Fetches jobs from external sources
- Intelligently matches jobs with a user's resume using AI
- Tracks job applications with smart UX decisions
- Includes a conversational AI assistant that can **control UI filters in real time**

This assignment is designed to test **engineering skills**, **AI orchestration**, and **product thinking** — similar to what you'd build in a real startup environment.

Core Features

---

1. Job Feed & External Integration

Job Source

Fetch jobs from any providers such as Azduna, etc.

Job Feed UI

Display jobs in a clean, readable feed with:

- Job title
- Company name
- Location
- Job description
- Job type
- **Apply** button on each job card

Required Filters (All Must Work)

- **Role / Title** – text search
- **Skills –** multi-select (React, Node.js, Python, etc.)
- **Date Posted**
    - Last 24 hours
    - Last week
    - Last month
    - Any time
- **Job Type**
    - Full-time
    - Part-time
    - Contract
    - Internship

- **Work Mode**
    - Remote
    - Hybrid
    - On-site
- **Location** – city or region
- **Match Score**
    - High (>70%)
    - Medium (40–70%)
    - All

---

2. Resume Upload & Profile

- At login, prompt the user to upload a **resume (PDF or TXT)**
- Use the following **test credentials**:
    - **Email:** `test@gmail.com`
    - **Password:** `test@123`
- Only **one resume per user**
- User should be able to **replace/update** resume anytime
- Extract and store resume text for AI processing

---

3. AI-Powered Job Matching (Mandatory)

Requirements

- When jobs load, **automatically score each job** against the user’s resume
- Show a **match score (0–100%)** on every job card
- Color-coded badges:
    - □ Green: >70%
    - □ Yellow: 40–70%
    - □ Gray: <40%
- Display a **“Best Matches” section** at the top (6–8 highest scoring jobs)
- Show a short explanation:
    - Matching skills
    - Relevant experience
    - Keywords alignment

□ **LangChain is mandatory** for AI-based job matching.

---

4. Smart Application Tracking (Critical Thinking Section)

This feature tests your **UX and product decision-making**.

Flow

1. When the user clicks **Apply**
    - Open the job’s external link in a **new tab**
2. When the user returns to your app, show a popup:

“Did you apply to [Job Title] at [Company]?”

Options:

- Yes, Applied
- No, just browsing
- Applied Earlier

Behavior

- If **Yes** → save job as **Applied** with timestamp
- Allow status updates:
    - Applied → Interview → Offer / Rejected
- Display all applications in a **dashboard view**
- Include a **timeline** per application

---

5. AI Assistant (Mandatory)

Choose **one UI approach**:

- **Option A:** Floating chat bubble (bottom-right, expandable)
- **Option B:** Collapsible sidebar (slides in/out)

A) Natural Language Job Search

Users should be able to ask:

- “Show me React developer jobs with Node.js”
- “Find ML engineer roles using PyTorch and TensorFlow”
- “Remote frontend jobs”

- “Senior backend roles posted this week”

The AI should:

- Understand intent
- Return relevant jobs
- Respect existing filters
- Show match scores

B) AI Controls Frontend Filters (Very Important)

The AI **must directly update UI filters**, not just reply in chat.

Examples:

- “Show only remote jobs” → Applies *Remote* filter
- “Filter by last 24 hours” → Updates date filter
- “Only full-time roles in Bangalore” → Applies multiple filters
- “High match scores only” → Match score >70%
- “Clear all filters” → Resets UI

C) Product Help

AI should answer questions like:

- “Where can I see my applications?”
- “How do I upload my resume?”

- “How does job matching work?”

AI Architecture Requirements

- **LangGraph is mandatory**
- LangGraph must handle:
    - Intent detection
    - Action routing (search, filter update, help)
    - Conversation state
    - Tool/function calling for UI filter updates

Technical Stack

- **Frontend:** React
- **Backend:** Node.js + Fastify
- **AI Matching:** LangChain (Required)
- **AI Orchestration:** LangGraph (Required)
- **LLM:** OpenAI / Anthropic / Gemini
- **Storage:** In-memory or JSON (keep it simple)

Required Deliverables

1. Live Deployment (Mandatory)

- Application must be accessible via a public URL
- Should work on desktop and mobile

2. GitHub Repository (Mandatory)

- Clean folder structure
- Meaningful commits
- `.env.example` file
- **No secrets committed**

README.md (Mandatory)

Your README **must include**:

a) Architecture Diagram

Visual diagram showing:

- Frontend
- Backend
- External job API
- LangChain job matching
- LangGraph AI assistant
- Data flow

b) Setup Instructions

- Local setup steps

- Environment variables
- Prerequisites

c) LangChain & LangGraph Usage

- How LangChain is used for job matching
- LangGraph graph structure and nodes
- Tool/function calling for UI filter updates
- Prompt design
- State management approach

d) AI Matching Logic

- Scoring approach
- Why it works
- Performance considerations

e) Popup Flow Design (Critical Thinking)

- Why you designed it this way
- Edge cases handled
- Alternative approaches considered

f) AI Assistant UI Choice

- Why bubble or sidebar
- UX reasoning

g) Scalability

- How this handles:
    - 100+ jobs
    - 10,000 users

h) Tradeoffs

- Known limitations
- What you'd improve with more time

Evaluation Criteria

Must-Have

- Live link working
- GitHub repo public
- LangChain used for matching
- LangGraph used for AI assistant
- Filters work (manual + AI)
- Match scores visible
- Smart popup flow implemented
- AI can control UI filters

What We Look For

- Product sense
- Clean architecture

- Proper AI orchestration
- UX quality
- Code readability
- Real-world thinking

Bonus Points

- Advanced LangGraph usage (conditional edges, parallel nodes)
- Conversation memory
- Smooth UI animations
- Voice input
- Mobile-first design
- Creative extra features

Final Submission Checklist

- Live link works on desktop & mobile
- GitHub repo is public
- README includes architecture diagram
- LangChain implemented
- LangGraph implemented
- AI controls filters

- Match scores visible
- Application tracking works
- No secrets in code

Submission Guidelines

Please submit your completed assignment by sending the following details to:

Email: info@tcconsultingservices.in

Include in your email:

- ☐ **Live Application URL**
- ☐ **Public GitHub Repository Link**
- ☐ Brief note explaining:
  - Your tech stack choices
  - Any assumptions or limitations
  - Extra features (if any)

Email Subject Line:
AI-Powered Job Tracker Assignment – Your Full Name

□ **Submissions without a working live link or public GitHub repository will not be considered.**