I want to create a product for HR analytics.

Here’s an **action plan** for developing the FastAPI backend as a **client-facing product**, complete with a **minimal demo frontend** to showcase functionality:

## **🛠️ Phase 1: Build a Solid, API-First Backend**

1. **API-First Design**
   * Define resource-focused endpoints (e.g., /v1/items) using RESTful patterns—use nouns, support JSON, include pagination, filters, versioning, and proper error codes.
2. **FastAPI + SQLModel + PostgreSQL**
   * Continue with your stack: use Pydantic and SQLModel for well-typed models.
   * Organize project modularly: separate models, routers, services, config folders.
   * Add Alembic migrations from day one.
3. **Auto-Generated Documentation**
   * Leverage FastAPI’s /docs, /redoc, and /openapi.json.
   * Make your docs clear for client developers.

## **🚀 Phase 2: Prep for Clients & Maintainability**

1. **Polish Developer Experience**
   * Make your API intuitive and hard to misuse—consistent naming, predictable behavior, well-documented responses and errors.
2. **Auth & CORS**
   * Add secure authentication (preferably OAuth2 + JWT) for client integration.
   * Configure precise CORS policies for your demo origins only.
3. **Versioning & Backwards Compatibility**
   * Prefix URLs with /v1/ and plan version upgrades carefully.

## **🧪 Phase 3: Build the Demo Frontend**

1. **Minimal Interactive Frontend**
   * Use a small HTML+JS page (vanilla JS, HTMX, or React) to show API integration—one form to submit and one to display the JSON response.
2. **Avoid Over-Engineering**
   * Skip complex UI frameworks; focus on quick, understandable functionality—just enough to prove the API works.
   * Example: a simple page with a text box, submit button, and output area for JSON response.
3. **Deploy Demo Lightly**
   * Serve via S3 + CloudFront or alongside your backend behind a proxy.
   * Ensure scope-limited credentials (if any) and demo-only data to avoid leaks.

## **🔄 Phase 4: Integration Guidance for Clients**

1. **API Documentation & Examples**
   * Create an api.js or api.ts file exporting fetch/axios methods for each endpoint.
   * Provide code snippets in JS and Node, showing both client-side and server-side calls.
2. **Interactive API Explorer (Optional)**
   * Use Swagger UI built into FastAPI as a sandbox clients can use live.

## 

## **🧩 Phase 5: Testing + Feedback Loop**

1. **Manual Demo Testing**
   * Run through demo flows, confirm functionality and usability from end-users perspective.
2. **Gather Client Input**
   * Observe demos, solicit questions: Is the API too verbose? Missing features? Hard to integrate?
3. **Iterate & Refine**
   * Improve ergonomics, documentation, error messages, and demo interface based on feedback.

This will help me do the following tasks:

1. **Create Job Descriptions** (JD) based on the given user input using AI. I want to use LLMs via langchain and langgraph Every JD will have the following sections, some are mandatory and some are optional:
   1. Job Title (Mandatory)
   2. Location (Mandatory)
   3. Reporting Relationship (Mandatory) - The person to whom to report to
   4. Function (Mandatory) - For example, Marketing, sales, etc.
   5. Role overview (Mandatory)
   6. Key responsibilities (Mandatory)
   7. Qualifications (Mandatory)
   8. Skills and competencies (Mandatory)
   9. Our Company (Optional) - It will take the company`s website URL and fetch the required data
   10. Our Culture (Optional) - It will take the company`s website URL and fetch the required data

After creating the JD using the LLM, the user can regenerate the JD if they are not satisfied with it. They can also provide custom commands if necessary, but that's optional. They can also edit the generated JD on the website if they choose to. They should also be able to download it as a PDF if they want.

1. **Job Posting:** Now that the job description (JD) is created, I want to post it as a job. This will be done in the following places:
   1. Job Portals such as LinkedIn, Naukri, Indeed, etc.
   2. Directly through the company website
   3. Through the company website via referral
2. **Scanning JD for different sections based on the given table:** Now, I want to fetch the different criterion based on the following table that are applicable to the JD we have

| Category | Subcategory | Criteria | Points |
| --- | --- | --- | --- |
| Relevance to Job Description | Skill Match | Primary Skills | 30 |
| Secondary Skills | 10 |
| Tools & Technologies | 5 |
| Experience Level | Years of Experience | 5 |
| Relevant Experience | 10 |
| Education and Certifications | Degree Requirements | 6 |
| Certifications | 4 |
| Job-Specific Achievements and Impact | Total Points | 5 |
| Market-Driven Parameters | Cultural Fit | Company Values Match | 8 |
| Job Mobility and Stability | Job-Hopping Patterns | 5 |
| Gap in Experience | 2 |
| Other Screening Considerations | Resume Presentation Quality | Readability and Format | 2 |
| Grammar and Spelling | 2 |
| Geographic and Remote Work Flexibility | Location Proximity | 2 |
| Remote Work Experience | 2 |
| Additional Activities and Interests | Total Points | 2 |
| Total | | | 100 |

A sample json output for this could be as follows:

```json

{

"Relevance to Job Description": {

"Skill Match": {

"Primary Skills": [

"Salesforce",

"Apex",

"Visualforce",

"Lightning",

"API integration"

],

"Secondary Skills": [],

"Tools & Technologies": []

},

"Experience Level": {

"Years of Experience": [

"3+ years in Salesforce development",

"5+ years total experience"

],

"Relevant Experience": [

"Salesforce development"

]

},

"Education and Certifications": {

"Degree Requirements": [

"B.Tech.",

"MCA",

"MBA"

],

"Certifications": [

"Salesforce certifications"

]

},

"Job-Specific Achievements and Impact": {

"Total Points": []

}

},

"Market-Driven Parameters": {

"Cultural Fit": {

"Company Values Match": []

},

"Job Mobility and Stability": {

"Job-Hopping Patterns": [],

"Gap in Experience": []

}

},

"Other Screening Considerations": {

"Resume Presentation Quality": {

"Readability and Format": [],

"Grammar and Spelling": []

},

"Geographic and Remote Work Flexibility": {

"Location Proximity": [],

"Remote Work Experience": []

},

"Additional Activities and Interests": {

"Total Points": []

}

}

}

```

1. **Creating a CV repository:** Now, after creating all the job postings and getting the CVs. I want to have a proper database that will store all these resumes.The database should:
   1. Have a column to track the source of the resume such as job portal, website etc.
   2. Date and time of when the candidate applies must be noted.
   3. Store the Aadhar details of the candidate to uniquely identify them.
   4. If there are multiple instances of the same candidate, verifiable by Aadhar, then always consider the latest.
   5. If there is a CV which is suitable for a job, if it is more than one year old, then we ask that candidate to send the latest CV. We send a mail to the candidate to do that which will have a link to our job posting.
   6. Efficiency is of utmost importance over here, as we will be dealing with a vast number of CVs.
2. **CV Scoring:** I have a rule based table, that helps us to grade a CV based on the details fetched from the job description:

[Rule-Based Excel for CV Scoring](https://docs.google.com/spreadsheets/d/1D4TUKEzRX6SHP1KY55r4HCpQy2eHYaap/edit?usp=sharing&ouid=103531710442092277893&rtpof=true&sd=true)

A sample json output for this CV scoring wuld be as follows:

```json

{

"Relevance to Job Description": {

"Skill Match": {

"Primary Skills": 18.0,

"Secondary Skills": 0,

"Tools & Technologies": 3.75

},

"Experience Level": {

"Years of Experience": 4,

"Relevant Experience": 8

},

"Education and Certifications": {

"Degree Requirements": 6,

"Certifications": 4

},

"Job-Specific Achievements and Impact": {

"Total Points": 0

}

},

"Market-Driven Parameters": {

"Cultural Fit": {

"Company Values Match": 0

},

"Job Mobility and Stability": {

"Job-Hopping Patterns": 5,

"Gap in Experience": 2

}

},

"Other Screening Considerations": {

"Resume Presentation Quality": {

"Readability and Format": 4,

"Grammar and Spelling": 2

},

"Geographic and Remote Work Flexibility": {

"Location Proximity": 2,

"Remote Work Experience": 0

},

"Additional Activities and Interests": {

"Total Points": 0

}

}

}

```

The user can determine what percentage should be a cutoff to accept the CVs. Let`s say those with scores above 70%. Even among people with similar scores, the priority will be decided on First Come First Serve basis.