Evidi

AI-powered assistant for sourcing, summarizing, and optimizing job responses with clarity and automation.

ROBIN, Héloïse
Dept. Computer Science
Hanyang University
Seoul, S. Korea
heloiserobin@hanyang.ac.kr

KHEYAR¸ Adel
Dept. Computer Science
Hanyang University
Seoul, S. Korea
adelkheyar@hanyang.ac.kr

ARM, Jules
Dept. Computer Science
Hanyang University
Seoul, S. Korea
julesarm@hanyang.ac.kr

DESJONQUERES, Nicolas
Dept. Computer Science
Hanyang University
Seoul, S. Korea
nicolasdesjonqueres@hanyang.ac.kr

***Abstract*—The Evidi Job Response Assistant is a web-based intelligent system that aims to streamline the job application process through automation and artificial intelligence. In the current employment landscape, job seekers face the repetitive task of manually searching, reviewing, and responding to numerous job offers across multiple platforms. This project proposes an integrated and automated workflow capable of retrieving job postings, filtering them based on personalized user criteria, summarizing descriptions using language models, and generating optional draft responses.**

The proposed solution integrates several state-of-the-art technologies: FastAPI serves as the backend API for data management, MongoDB Atlas as the cloud database, n8n as the automation and orchestration engine, and React + TypeScript as the frontend interface. In addition, the project leverages OpenAI's GPT-4 API for natural language summarization and generation. Deployed entirely in the cloud (Render, Vercel, and n8n.cloud), the system demonstrates how AI-driven automation can simplify job searching while maintaining flexibility, scalability, and transparency in a student-deployable environment.

***Index Terms*—Artificial Intelligence, Workflow Automation, FastAPI, n8n, MongoDB Atlas, React, Job Search, Natural Language Processing**

I. Role Assignments

TABLE I: Role Assignments

Role	Name	Task Description
User	Nicolas	Use the app as a real user. Provide feedback on its use of the app. Report bugs, and suggest improvements.
Customer	Jules	Provide requirements and feedback. Validate the functional design and user experience. Evaluate deliverables during milestones (UI mockups, MVP demo). Ensure the project meets academic or business goals.
Software Developer	Adel	Implement frontend in React (UI, API calls). Develop backend API (authentication, DB, CRUD endpoints). Design and integrate the database (PostgreSQL/Supabase). Configure n8n workflows for automation and AI processing. Test, debug, and deploy components on cloud platforms.
Development Manager	Héloïse	Define project scope, timeline, and milestones. Assign tasks to team members and manage version control (Git). Ensure coherence between frontend, backend, and automation. Supervise testing, deployment, and documentation. Communicate with the customer and ensure project quality.

II. Introduction

A. Motivation

In today's hyperconnected society, the process of job seeking has evolved into a complex digital endeavor. With the exponential growth of online recruitment platforms and professional networks, users are exposed to thousands of potential opportunities daily. While this abundance theoretically broadens access to employment, it also introduces new forms of cognitive overload. Candidates must continuously navigate through heterogeneous data (job descriptions, skill requirements, company cultures, and application procedures) dispersed across numerous websites and communication channels.

Traditionally, this process demands substantial time and emotional investment. Job seekers often spend hours manually reviewing listings, extracting relevant qualifications, and tailoring cover letters for each posting. Over time, this repetitive task can lead to fatigue, demotivation, and ultimately, missed opportunities. Meanwhile, organizations receive massive volumes of generic applications, highlighting the inefficiency and asymmetry in the modern recruitment pipeline.

With the rapid progress in **Artificial Intelligence (AI)**, particularly in **Natural Language Processing (NLP)** and workflow automation, there is a unique opportunity to reimagine this process. By automating the collection, analysis, and synthesis of job offers, it becomes possible not only to increase efficiency but also to provide more equitable and intelligent access to employment information. The motivation behind this research is therefore to develop a transparent, modular, and accessible tool that empowers individuals to leverage automation ethically and effectively in their job search journey.

B. Problem Statement (User Needs)

Despite the apparent convenience of existing online job boards such as **LinkedIn**, **Glassdoor**, or **Indeed**, users continue to face several pain points. These platforms offer keyword filters and alerts, yet they remain fundamentally passive, incapable of adaptive reasoning or personalized contextualization. Users must still screen each listing, interpret nuanced requirements, and manually generate application materials.

Automation services such as **Zapier** or **Make (Integromat)** provide general workflow integration, but they are not designed for employment-specific contexts. They lack semantic understanding of job-related data, leading to rigid automation pipelines. Furthermore, while advanced AI systems like **ChatGPT** can produce natural language text, they require extensive manual prompting and lack connectivity with dynamic job feeds or filtering logic.

From an educational and research perspective, this absence of a unified, open, and domain-specific framework creates a barrier for both practitioners and students wishing to experiment with AI-driven automation in real-world contexts. Hence, there is a clear need for a customizable, intelligent, and transparent ecosystem that integrates job data retrieval, summarization, and response generation into a single, cloud-ready platform.

C. Existing Solutions

A comparative analysis of current tools reveals several partial solutions, yet none fully address the multidimensional needs of job seekers.

Platforms such as **Simplify** and **LoopCV** attempt to streamline the application process by automating repetitive form-filling or submission tasks. However, their infrastructures are proprietary and non-extensible, limiting opportunities for customization or academic exploration. Similarly, **Huntr** provides efficient tracking for ongoing applications but lacks an AI-driven decision layer capable of analyzing or ranking offers intelligently.

On the other hand, low-code automation platforms such as **Zapier** and **n8n** allow users to design workflows visually, connecting data sources and web APIs. While powerful, they operate as generic middleware and do not incorporate domain-specific heuristics such as keyword extraction, offer classification, or motivation-letter personalization. Finally, AI-driven assistants like **ChatGPT** or **Claude** can generate text upon request but cannot autonomously interact with job data sources or maintain persistent user contexts across sessions.

This review highlights the fragmented nature of current technological ecosystems and underscores the necessity of an integrated, open-source framework where AI models, automation logic, and user interfaces converge seamlessly.

D. Proposed Solution

The **Evidi Job Response Assistant** is designed to address these gaps by combining the flexibility of modern cloud computing with the intelligence of advanced language models. The proposed system features a modular architecture comprising four key layers: **data ingestion**, **AI-driven processing**, **backend management**, and **interactive visualization**.

Through **n8n**, the system automatically retrieves job offers from diverse sources such as RSS feeds, email inboxes, and public APIs. These offers are then filtered through user-defined criteria (including domain, skills, or salary range) and stored in a **MongoDB Atlas** database managed via a **FastAPI** backend. The backend exposes RESTful endpoints that feed into a **React + TypeScript** frontend, where users can visualize offers, summaries, and application drafts.

An **AI layer**, powered by **OpenAI GPT-4**, performs advanced summarization and generates personalized application responses. The combination of these technologies results in a robust workflow that minimizes manual intervention, enhances precision, and supports reproducible research. Beyond its technical contributions, this project aims to demonstrate a scalable model for **AI-assisted decision-making** and to provide an educational reference for integrating automation, data engineering, and language intelligence in real-world employment contexts.

III. Requirements

A. User Authentication and Management

The system must provide secure and user-friendly mechanisms for registration, authentication, and session management.

- **Registration:** New users must create an account using a valid email address and password. Upon submission, an email verification code is sent automatically. Only verified accounts gain access to the main interface.
- **Password Security:** All user passwords are hashed using a robust cryptographic algorithm (e.g., **SHA-256** or **bcrypt**) prior to database storage. Plaintext passwords are never stored or transmitted.
- **Login:** The login process validates the provided credentials against stored hash values. Upon success, a **JWT (JSON Web Token)** is issued to manage authenticated sessions securely across the frontend and backend.
- **Account Recovery:** In case of forgotten credentials, the system provides a password reset flow via email-based verification. Expired or invalid tokens are automatically rejected to maintain security.

B. User Criteria Management

Users can define personalized job search criteria that guide the system's filtering and retrieval mechanisms.

- **Criteria Creation:** Users specify parameters such as target keywords, job titles, industries, required skills, employment type (e.g., remote, full-time, internship), and geographic location.
- **Criteria Persistence:** Each user's preferences are stored in the cloud database under a unique user identifier, allowing the same filters to be reused automatically for subsequent job searches.
- **Dynamic Modification:** The user interface enables modification or deletion of existing criteria. Any change

triggers an immediate synchronization across the backend and automation workflows.
- **Validation:** Input validation ensures the accuracy of filter definitions, preventing empty or malformed entries before committing data to storage.

C. Job Offer Ingestion System

The ingestion module is responsible for automatically collecting job offers from multiple external sources in a structured and scalable manner.

- **Sources of Data:** The system supports several channels:
  - RSS feeds from public job boards.
  - REST APIs from professional platforms (e.g., LinkedIn, Indeed, Welcome to the Jungle).
  - Email inbox parsing for newsletters and subscriptions.
- **Automation Pipeline:** Workflows are executed through the **n8n** automation platform. Each workflow consists of nodes for fetching, transforming, and forwarding job data to the backend.
- **Data Standardization:** Collected job data is normalized into a unified JSON structure with standardized fields such as `title`, `company`, `location`, `skills`, `description`, and `source`.
- **Scheduling:** The ingestion process operates at configurable intervals (e.g., every 2 hours) or can be triggered manually by the user through the frontend.

D. Offer Filtering Module

Once data is ingested, the filtering module compares each offer against the user's criteria to identify relevant matches.

- **Matching Algorithm:** A hybrid approach combining keyword search and semantic similarity (e.g., cosine similarity via sentence embeddings) determines the relevance of each offer.
- **Scoring System:** Each offer receives a numerical relevance score between 0 and 1. Only offers above a user-defined threshold (e.g., 0.7) are retained for summarization.
- **Duplicate Detection:** Hash-based identifiers prevent repeated storage of identical job listings from multiple sources.
- **Result Storage:** Filtered offers are saved in the MongoDB database and marked with their corresponding user ID, timestamp, and relevance score.

E. AI Summarization Engine

The summarization engine leverages **OpenAI GPT-4** to generate concise and structured job summaries.

- **Input Data:** The engine receives the normalized job description text and associated metadata from the filtering module.
- **Prompt Template:** A predefined template guides the model to produce summaries containing the following sections: *Position Title, Company Overview, Key Responsibilities, Required Skills, and Application Insights.*
- **Output Format:** Summaries are returned as structured text blocks and stored alongside the original job offers in the database.
- **Quality Control:** The backend verifies that all expected fields are present before saving the result. In case of missing or malformed output, a re-generation is automatically triggered.

F. AI Letter Draft Generation (Optional)

An optional functionality enables users to generate personalized cover letter drafts for selected job offers.

- **Input Context:** The model combines three sources of information: (1) the summarized job offer, (2) the user's stored profile data, (3) previously defined motivation style preferences.
- **Prompt Structure:** The AI is instructed to generate a professional and context-aware letter draft with three sections: introduction, motivation, and conclusion.
- **User Review:** Generated letters are displayed in the frontend editor, allowing the user to modify, approve, or export them in text or PDF format.

G. Data Management Layer

All information produced by the system (job offers, summaries, and user data) is stored and managed within a secure cloud database.

- **Database Engine:** The system uses **MongoDB Atlas**, chosen for its scalability, flexibility, and document-based structure that fits dynamic job data.
- **Data Schema:** Each document includes nested structures for offer metadata, AI-generated summaries, and associated user identifiers.
- **Backup and Retention:** Automatic backup policies ensure data persistence. Obsolete or expired job offers are archived to a secondary collection for future analysis.

H. Notification and Alert System

To enhance user engagement, the system automatically informs users of new or relevant job opportunities.

- **Notification Triggers:** Alerts are generated whenever a newly ingested job offer exceeds the user's relevance threshold.
- **Delivery Channels:** Notifications are dispatched through email or third-party integrations such as Slack or Telegram via **n8n** connectors.
- **Content Format:** Each notification includes the job title, company name, and a short excerpt of the AI summary with a link to view full details on the dashboard.

I. Frontend Visualization Dashboard

The user interface provides access to all system functionalities in an organized and interactive manner.

- **Technology Stack:** Developed using **React** and **TypeScript**, ensuring responsiveness, modularity, and cross-platform accessibility.

- **Views and Components:** The dashboard consists of multiple pages: login, profile settings, job feed, AI summaries, and letter drafts.
- **Interaction Design:** Users can filter, search, and sort offers; review AI-generated content; and trigger workflow actions directly (e.g., "generate letter" or "refresh offers").

J. System Integration Workflow

This component ensures smooth interoperability between all subsystems: automation, backend, database, and frontend.

- **Backend API:** Built with **FastAPI**, it exposes RESTful endpoints for data retrieval, filtering, and AI model invocation.
- **Authentication Flow:** Communication between frontend and backend is protected by JWT-based session tokens.
- **Automation Orchestration:** The **n8n** workflows continuously synchronize external data sources with the backend database, ensuring real-time updates.
- **Deployment Infrastructure:** Services are containerized using **Docker** and deployed across cloud providers (e.g., Vercel for frontend, Render for backend, MongoDB Atlas for data storage).

IV. Development Environment

A. Choice of Software Development Platform

The **Evidi Job Response Assistant** is developed as a distributed, cloud-based web application integrating automation, AI, and workflow orchestration. The project aims to streamline the job search process by automating the retrieval, filtering, and summarization of job offers using AI-driven methods. Given the short project timeline (two months) and the need for a scalable, low-maintenance architecture, we carefully selected a modern and cloud-native technology stack.

Our backend is developed using **FastAPI (Python 3.11)**, chosen for its asynchronous design and native compatibility with RESTful APIs. The database layer relies on **MongoDB Atlas**, a cloud-based NoSQL service optimized for unstructured text data such as job listings. The automation component leverages **n8n.cloud**, which provides a visual workflow platform to connect APIs and automate data ingestion. The frontend is built with **React 18** and **TypeScript**, ensuring maintainability and rapid UI prototyping from **Figma AI** designs. The system's intelligence layer is powered by the **OpenAI GPT-4 API**, responsible for summarization and draft letter generation.

For deployment, the backend is hosted on **Render**, the frontend on **Vercel**, and automation workflows on **n8n.cloud**, ensuring distributed yet coordinated operation. This configuration provides a balance between scalability, modularity, and ease of collaboration among the four-member development team.

TABLE II: Tools and Language Choice

Tools and Language	**Reason**
FastAPI (Python 3.11)	FastAPI is a modern, high-performance web framework designed for building APIs with Python. It supports asynchronous I/O through ASGI, allowing concurrent processing of multiple requests—critical for handling AI summarization calls and workflow webhooks. Its native OpenAPI integration and Pydantic validation enhance security and documentation, while automatic type checking ensures robust development.
MongoDB Atlas	MongoDB Atlas is a fully managed NoSQL database ideal for storing dynamic and unstructured data. Its document-oriented model allows flexible schema design for job listings with varying fields. The cloud-hosted service offers high availability, free-tier scalability, and easy integration with Python backends, eliminating the need for local database maintenance.
n8n (Workflow Orchestration)	n8n provides a visual low-code automation platform that connects data pipelines across APIs, databases, and AI services. It automates RSS ingestion, filtering, and message forwarding to FastAPI, significantly reducing manual scripting. Its cloud-based deployment ensures reliability and transparency for non-technical stakeholders.
React + TypeScript	React offers a component-based architecture suitable for modular and responsive web interfaces. TypeScript adds static typing and compile-time checking, reducing bugs and improving maintainability. Together, they enable rapid frontend development aligned with modern design standards and seamless integration with backend APIs.
OpenAI GPT-4 API	GPT-4 provides advanced natural language understanding and summarization. It transforms long job descriptions into concise summaries and assists in generating personalized cover letter drafts. API-based integration simplifies scalability and experimentation with prompt engineering.
Render + Vercel + n8n.cloud	Render hosts the FastAPI backend with continuous deployment from GitHub. Vercel automates frontend builds and deployment for React. n8n.cloud hosts automation workflows without local setup, maintaining always-on task execution across the stack.

B. Cost Estimation

Ensuring the reliability and scalability of the **Evidi Job Response Assistant** requires cloud services that minimize operational overhead while providing sufficient performance. We opted for free-tier and low-cost cloud options to maintain budget efficiency during the prototype phase while retaining professional-grade features.

Render hosts the FastAPI backend under a free-tier plan that supports continuous integration (CI/CD) and SSL-secured endpoints. **Vercel** similarly provides free hosting for static React applications, ensuring seamless deployment from GitHub. For workflow orchestration, **n8n.cloud** offers a community-tier plan that provides adequate execution capacity for RSS and email automation. **MongoDB Atlas**'s M0 cluster tier supports sufficient storage for several thousand job entries with automated backups.

Overall operational costs remain minimal, with the only recurring expense being **OpenAI API usage**, estimated between 5–10 USD per month during testing, depending on the volume of summarization and draft generation requests.

TABLE III: Hosting and AI Tools

Tools and Language	**Reason**
Render (Backend Hosting)	Render provides an accessible Platform-as-a-Service (PaaS) for deploying Python web apps directly from GitHub repositories. Its automatic builds, HTTPS support, and continuous deployment streamline backend updates with zero configuration cost.
Vercel (Frontend Hosting)	Vercel automates the deployment process for React applications and optimizes performance through global edge caching. Its integration with GitHub and focus on frontend projects make it ideal for CI/CD of the web interface.
n8n.cloud (Automation Hosting)	n8n.cloud manages workflows in a hosted environment without server setup. Its scalability and integration options ensure continuous operation of data ingestion and automation processes, critical for real-time job updates.
MongoDB Atlas (Database Hosting)	The M0 free cluster offers fully managed cloud storage, automated backups, and monitoring. It eliminates the need for local servers while ensuring reliable performance for the prototype.
OpenAI API (GPT-4)	While usage incurs minimal costs, the GPT-4 API offers unparalleled summarization quality, ensuring high-value AI outputs. Its pay-per-use model aligns well with project scalability and limited testing budgets.

C. Software in Use

1) Existing Systems / Tools:
Simplify (Simplify Copilot) is a browser-based tool that automates repetitive job application tasks. It provides features such as auto-filling application fields, tracking applications, tailoring resumes, and identifying missing keywords in one's CV. While Simplify excels at streamlining the manual data entry portion of applications, it does not (publicly) provide a unified backend, cross-source ingestion pipeline, or AI summarization embedded in a web app.

LoopCV is another job search automation platform that matches job seekers with listings and automates part of the application process. It uses AI to optimize CVs, track applications, and even directly apply on behalf of the user. LoopCV's strength lies in its end-to-end workflow (search $\rightarrow$ apply $\rightarrow$ track), but it is a closed, commercial product with limited transparency into its internal pipelines.

Huntr offers features for job application tracking, AI-assisted resume and cover letter generation, and auto-filling application forms. However, Huntr is primarily a productivity / tracking tool; it does not appear to automate ingestion from RSS or provide full workflow orchestration with external services like n8n.

LazyApply automates job applications across job platforms via AI, handling form filling and submission. Its value is in applying at scale, but challenges include handling custom fields, CAPTCHA, and job boards with complex forms.

From the research perspective, **ResumeFlow: An LLM-facilitated Pipeline for Personalized Resume Generation and Refinement** introduces a pipeline that takes job descriptions and resumes and produces tailored, optimized CVs using LLMs. This aligns with our AI-driven summary / draft generation module. It demonstrates the viability of leveraging LLMs for alignment between job descriptions and user profiles.

2) Comparison & Gap Analysis:

- **Scope of ingestion:** Existing tools like Simplify or LazyApply typically rely on browser extension or manual input, whereas our project plans to support ingestion via RSS feeds, APIs, and emails through automation workflows.
- **AI summarization / drafting:** While tools offer resume optimization or cover letter suggestions, few expose summarization of full job descriptions or draft generation from user profile + job content. Our approach explicitly integrates that.
- **Transparency and extensibility:** Commercial tools are black-box; you cannot inspect or customize their pipelines. We provide modular architecture (n8n + API backend) that is open, testable, and extensible.
- **Integration & orchestration:** Our system orchestrates ingestion, filtering, AI processing, and persistence together. Tools like Simplify partly automate, but lack seamless end-to-end pipeline control integrated with a web app interface.
- **Flexibility & deployment:** Because ours relies on open stack (FastAPI, MongoDB, n8n, React), we can adapt features, scale, modify workflows, or replace components, which is typically impossible with off-the-shelf tools.

V. Specifications

A. Requirement 1 – User Management

Goal: Allow users to register, authenticate, and manage their profile and preferences.

Implementation:

- **Frontend:** Login and Register pages with full name, email, password, and confirmation fields. Settings page for profile update, password change, and notification preferences. Axios handles communication with backend.
- **Backend:** FastAPI endpoints `/register`, `/login`, `/user/preferences`, `/user/profile`, and `/user/password`. JWT-based authentication and bcrypt password hashing.
- **Database:** MongoDB collection `users` storing { `_id, full_name, email, password_hash, preferences, settings` }.
- **Security:** JWT tokens for authentication, password validation, and session protection.

Pseudocode:

```
POST /register:
  receive {full_name, email, password}
  hash = bcrypt.hash(password)
  insert into db.users({full_name, email, hash})
  return success

POST /login:
  check email exists
  verify bcrypt(password, hash)
  return JWT_token
```

B. Requirement 2 – Criteria & Filter Management

Goal: Allow users to define, update, and store job search preferences.

Implementation:

- **Frontend:** Filters page with tech stack tags, experience level, include/exclude keywords, location preferences, and job type options.
- **Backend:** FastAPI endpoints `/criteria/update` and `/criteria/get`.
- **Database:** MongoDB collection `criteria` linked to user ID.
- **Integration:** n8n workflows retrieve criteria for dynamic filtering.

Pseudocode:

```
POST /criteria/update:
  user_id = JWT_token.user
  update db.criteria where user_id
  return success
```

C. Requirement 3 – Job Offer Ingestion

Goal: Retrieve and store job offers automatically from public and integrated sources.

Implementation:

- **Automation:** n8n workflows scheduled every 3 hours to fetch from RSS feeds, APIs, or email triggers.
- **Backend:** FastAPI webhook `/webhook/jobs` receives job payloads and stores them in MongoDB.
- **Frontend:** Sources page to add, edit, or remove sources with manual sync and status indicators.

Pseudocode:

```
RSS Trigger -> Filter (new posts only)
-> HTTP POST to FastAPI /webhook/jobs
-> Insert into db.jobs
```

D. Requirement 4 – Offer Filtering

Goal: Match job offers against user-defined criteria.

Implementation:

- **n8n:** “IF” nodes filter job payloads based on keywords and location.
- **Backend:** Python regex fallback for keyword matching.
- **Frontend:** Jobs page filter dropdown (All, Matched, Applied, Rejected) with match badges.

Pseudocode:

```
for job in new_jobs:
  if any(keyword in job.description for keyword in
  user.criteria):
    insert into db.jobs_filtered
```

E. Requirement 5 – AI Summarization

Goal: Generate concise AI summaries for each job description.

Implementation:

- **n8n:** HTTP request node calls OpenAI API (GPT-4/4o-mini).
- **Backend:** FastAPI endpoint `/ai/summarize` for manual trigger.
- **Frontend:** Job Detail Modal’s “AI Summary” tab shows results with regeneration option.
- **Database:** MongoDB collection `summaries`.

Pseudocode:

```
POST /ai/summarize:
  input = job.description
  prompt = "Summarize in 5 bullet points"
  response = openai.ChatCompletion(prompt)
  db.summaries.insert({job_id, summary: response})
```

F. Requirement 6 – AI Letter Draft Generation

Goal: Automatically generate a motivation letter based on user’s CV and job description.

Implementation:

- **n8n:** Uses OpenAI API with user profile and job data.
- **Backend:** Endpoint `/ai/draft` for manual regeneration.
- **Frontend:** “Cover Letter” tab in Job Detail Modal with edit, regenerate, copy, and download options.

Prompt Example:

```
"Write a short motivation paragraph for
this position based on user’s experience
and the job description."
```

G. Requirement 7 – CV Upload & Analysis

Goal: Analyze uploaded CVs to extract skills and job preferences.

Implementation:

- **Frontend:** CV Analysis page with drag-and-drop upload, file card, and AI analysis results.
- **Backend:** Endpoint `/cv/analyze` using OpenAI model for extraction.
- **Database:** Stores extracted skills, experience level, and preferences in `cv_analysis` collection.

Workflow:

```
Upload CV -> Analyze with AI -> Extract
Skills and Preferences -> Store in db.cv_analysis
-> Apply to Filters
```

H. Requirement 8 – Notification System

Goal: Notify users of new job matches via multiple channels.

Implementation:

- **n8n:** Slack, Email, or Push notification nodes.
- **Backend:** Endpoint `/api/notify` for message dispatching.
- **Frontend:** Settings page toggles for email, push, and weekly digest.

Message Example:

```
"New job matching your skills: Data Engineer at
XCorp."
```

I. Requirement 9 – Dashboard & Analytics

Goal: Display user's job search metrics and activity overview.

Implementation:

- **Frontend:** Dashboard with stats cards (Total Jobs, Matched, Applied, Response Rate), recent activity feed, and quick actions.
- **Backend:** Endpoint `/api/dashboard` aggregates metrics.
- **Database:** Activity logs stored for history display.

J. Requirement 10 – Frontend Dashboard and Navigation

Goal: Provide a modern, responsive interface for all application modules.

Implementation:

- **Framework:** React + TypeScript + Tailwind CSS.
- **Global UI:** Header with logo, theme switcher, settings, logout.
- **Navigation:** Tabs for Dashboard, Jobs, Sources, Filters, CV Analysis, and Settings.
- **Theme Support:** Default, Dark, Deep Blue, and Green themes.

K. Requirement 11 – Data Storage

Goal: Securely persist all user and job-related data.

Implementation:

- **Database:** MongoDB collections for users, criteria, jobs, summaries, letters, CV analyses, and notifications.
- **Access Control:** JWT authentication required for all write operations.

Schema:

```
users: { _id, full_name, email,
password_hash, preferences, settings }
criteria: { user_id, tech_stack[],
keywords_include[], keywords_exclude[],
location[], job_type[], experience_level[] }
jobs: { job_id, title, company, description,
tags[], location, source, matched }
summaries: { job_id, summary, ai_model,
updated_at }
letters: { job_id, user_id, content, updated_at }
cv_analysis: { user_id, extracted_skills[],
experience_level, preferences }
notifications: { user_id, message, type,
timestamp, read }
```

L. Requirement 12 – Logging, Monitoring & Error Handling

Goal: Enable observability and debugging across all services.

Implementation:

- **FastAPI:** Middleware logs requests and responses.
- **n8n:** Built-in execution and error logs.
- **Cloud:** Render logs webhook failures and server errors.
- **Frontend:** Toast notifications and retry logic for network errors.

M. Requirement 13 – Testing and Validation

Goal: Ensure application stability through automated testing.

Implementation:

- **Backend:** Pytest for unit testing and Postman for integration tests.
- **Frontend:** React Testing Library for component testing.
- **Automation:** GitHub Actions CI pipeline for continuous testing.

N. Requirement 14 – Integration Workflow

Goal: Maintain full interoperability among system components.

Architecture Overview:

- **n8n:** Manages ingestion, filtering, and AI summarization.
- **FastAPI:** Handles authentication, validation, and persistence.
- **MongoDB:** Centralized data storage.
- **OpenAI API:** Provides summarization and letter generation.
- **React Frontend:** Displays user-facing data and controls.

Data Flow:

```
[RSS / API / Email Source]
          ↓
     [n8n Workflow]
          ↓
 [AI Summarizer Node / Letter Draft Node]
          ↓
 [FastAPI Webhooks (/webhook/jobs, /ai/...)]
          ↓
       [MongoDB]
          ↓
 [React Frontend Dashboard & Pages]
```

O. Requirement 15 – UX Enhancements (Cross-Page Features)

Goal: Enhance user experience through modern interaction patterns.

Implementation:

- Toast notifications for success, error, and info messages.
- Loading indicators, skeleton loaders, and progress feedback.

- Responsive mobile design with touch-friendly inputs.
- Keyboard accessibility for modals and navigation.
- Smooth theme transitions and persistent preferences.