AI Job Application Agent

Overview

The **AI Job Automation Agent** is an autonomous job application engine that discovers, evaluates, and applies to roles across multiple job platforms with minimal human intervention. It is designed around three full end-to-end runs per week, where each run performs job discovery, scoring, decisioning, application execution, and logging within a single controlled batch.

The system targets **300–600 unique jobs per week**, discovering **100–200 jobs per run** and automatically applying to approximately **50%** of high-confidence roles while queuing the remaining **50%** as high-ROI, “dream” opportunities for manual or assisted follow-up. It emphasizes deterministic logic, platform-aware rate limiting, and explicit human oversight so that automation remains both effective and safe.

Core components include: job discovery via JobSpy and SerpApi across a curated set of job boards, a scoring and routing layer powered by LLMs and RAG over a 7-resume profile set, orchestration through n8n and MCP, site-specific Playwright flows for auto-apply, a SaaS-grade Chrome extension for advanced manual/assisted applications, and a consolidated Postgres + Notion tracking stack.

Scope & Objectives

The **scope** of this project is to build a single-user, production-ready AI job application engine that runs **3 fully automated batches per week**, each processing **100–200 discovered roles** from a fixed list of job platforms (LinkedIn, Indeed, Glassdoor, Google Jobs, ZipRecruiter, Wellfound, Remotive, WeWorkRemotely, RemoteOK, Jooble, SimplyHired, Stack Overflow, YC Startup Jobs, Hiring Cafe). It covers job discovery, scoring, routing, auto-application, manual-assist workflows, and centralized tracking, but excludes downstream interview scheduling, offer management, or multi-candidate recruiter tooling. The primary **objective** is to autonomously apply to approximately **50%** of discovered roles per run (high-confidence matches) end to end, while queuing the remaining **50%** as high-ROI, “dream” roles with rich context for manual or assisted follow-up. The system must remain platform-aware (rate limits, anti-bot, ToS risk), provide deterministic and observable decision paths, and allow the user to pause automation or override decisions at any point. Secondary objectives include minimising infrastructure complexity (Postgres + Notion as core data stores), leveraging a **RAG layer** over a 7-resume profile set for

personalised form filling, and using **n8n + MCP** as the orchestration backbone for clear, maintainable workflows.

TL;DR

A production-grade AI engine that, three times per week, discovers **100–200 jobs per run**, scores and routes them using an LLM + RAG layer over a 7-resume profile, and then auto-applies end to end to roughly half of the high-confidence roles while queuing the rest as high-ROI, “dream” opportunities. Each run executes the full pipeline—discovery, normalization, scoring, decisioning, Playwright/Chrome-extension application, and logging—within strict per-platform limits and clear human override points. The system favors a lean stack (Postgres + Notion, site-specific Playwright flows, SaaS-grade Chrome extension, n8n and MCP for orchestration) and is optimized for reliability, platform safety, and transparent operational control rather than raw application volume.

Tech Stack & Integrations

This section documents the core technology stack and external integrations used by the AI Job Automation Agent. The stack is intentionally lean: Python for backend logic, n8n and MCP for orchestration, Playwright plus a Chrome extension for automation, Postgres as the primary data store (including embeddings), and Notion as the primary tracking surface. LLM access is routed through NVIDIA NIM, OpenRouter, and Perplexity where appropriate, with SerpApi and JobSpy handling job discovery across the selected platforms.

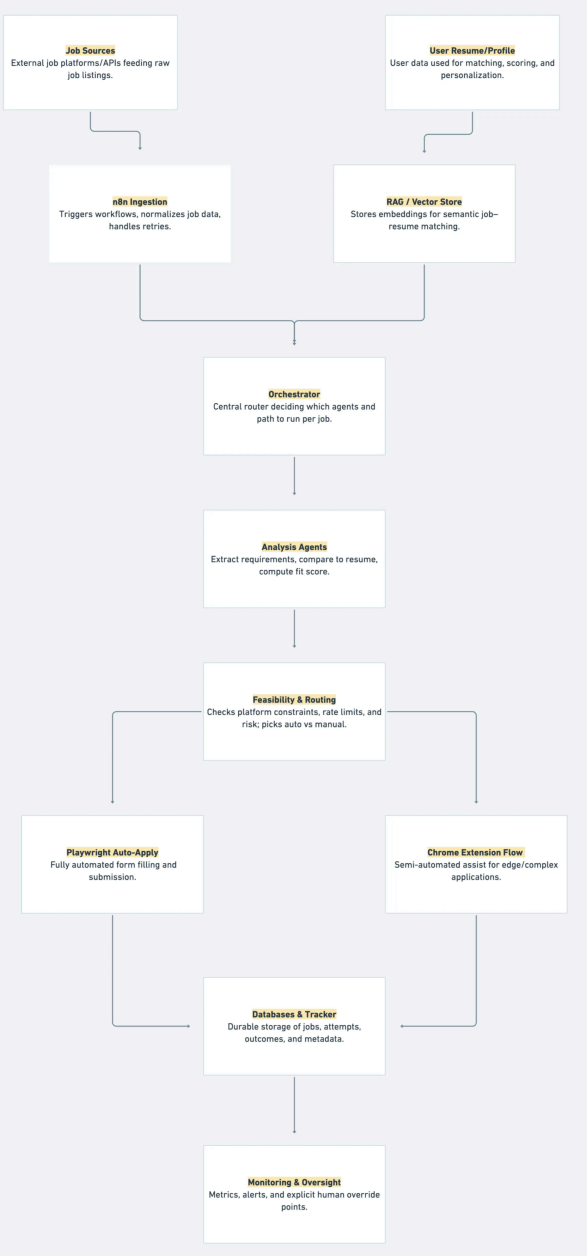
AI Job Agent

Aa Layer	Component	Technology / Tool	Purpose
Core Language	Backend	Python	Primary language for scraping
AI / LLM Layer	Reasoning Engine	LLM APIs (Gemini / OpenRouter fallback, NVIDIA NIM, Perplexity Sonar for select tasks).	Job analysis
Agent Architecture	Multi-Agent System	Orchestrator, Analyzer, Application, Intelligence Agents	Modular AI decision flow.
Workflow Orchestration	Automation Engine	n8n	End-to-end workflow control
Web Automation	Browser Automation	Playwright	Auto-apply
Scraping	Job Discovery	JobSpy, Jooble API, SerpApi (Google Jobs)	Job sourcing from multiple platforms.
Fallback Automation	Manual Assist	Chrome Extension (Manifest V3)	Autofill plus manual apply logging.
Backend API	Service Layer	FastAPI	Webhooks
Databases	Primary DB	PostgreSQL	Structured job and application data.
Databases	Cache / Queue	Redis	Task queuing (Optional/later phase)
Databases	Vector Store	**Postgres (pgvector)**	Resume and context embeddings for RAG stored via pgvector extension.
RAG System	Embeddings	text-embedding-3-small	Semantic search and resume matching.

Layer	Component	Technology / Tool	Purpose
Tracking & Ops	Application Tracker	Notion API	Centralized job and application tracking.
Email Integration	Notifications	Gmail API	Status updates, alerts, and optional outreach.
Infrastructure	Containerization	Docker, Docker Compose	Local and production deployment.
CI/CD (Later Phase)	Automation	GitHub Actions	Build
Monitoring	Observability	Prometheus, Grafana, Sentry	Metrics
Security	Secrets & Limits	Env vars, rate limiting, proxies	Secrets management, rate limiting, proxy rotation, and account safety controls.
Dev Environment	Tooling	VS Code, Git, macOS	Development and version control.

Architecture Overview

The system uses a **modular, agent-driven architecture** orchestrated by n8n and MCP, with a single end-to-end pipeline that runs three times per week. Each run begins with JobSpy and SerpApi discovery across the fixed job-board list, normalizes and stores **100–200** postings in Postgres, and then routes them through an Orchestrator Agent that applies deterministic filters and LLM/RAG-based scoring to decide between auto-apply and high-ROI queueing. High-confidence roles flow into site-specific Playwright automations that handle login, form detection, resume/answer insertion (powered by the 7-resume profile RAG system), submission, and structured logging, while complex or high-value roles are packaged with generated assets and handed off to a SaaS-grade Chrome extension for assisted completion. All outcomes and decisions are written to Postgres and mirrored into Notion for human-friendly tracking, with observability hooks (metrics, structured logs, error codes) exposed from each major node—discovery, scoring, automation, and extension—to support debugging, rate-limit safety, and explicit human override controls.



System Requirements & Constraints

This section captures the core functional and non-functional requirements for the AI Job Automation Agent. Each requirement is scoped around the 3-runs-per-week model (100–200 jobs per run, ~50% auto-apply, ~50% high-ROI queue) and emphasizes a lean data stack (Postgres + Notion), platform-safe automation, and observability across the entire pipeline.

System Requirements

Aa Name	Description	Priority	Status	Type
FR-1	System runs 3 end-to-end batches per week, and each batch performs discovery, scoring, decisioning, application, and	Must	Planned	Functional

Name	Description	Priority	Status	Type
	logging for that run.			
FR-2	Each batch discovers at least 100 and at most 200 unique jobs from the fixed platform list (LinkedIn, Indeed, Glassdoor, Google Jobs, ZipRecruiter, Wellfound, Remotive, WeWorkRemotely, RemoteOK, Jooble, SimplyHired, Stack Overflow, YC Startup Jobs, Hiring Cafe).	Must	Planned	Functional
FR-3	For every batch, approximately 50% of discovered jobs are auto-applied end to end with no user intervention, and the remaining 50% are queued as high-ROI opportunities with generated assets attached.	Must	Planned	Functional
FR-4	All auto-applied and queued jobs are logged to Postgres and mirrored into Notion, including job metadata, decision reason, status, and timestamps.	Must	Planned	Functional
NFR-1	System operates as a largely hands-off automation engine, requiring only periodic configuration changes and review of queued high-ROI jobs.	Must	Planned	Non-Functional
NFR-2	Data stack remains lean: Postgres as primary system of record (including embeddings via pgvector) and Notion for tracking and review; Redis is optional and added only if needed.	Should	Planned	Non-Functional
NFR-3	Scraping and automation respect per-site rate limits and anti-bot constraints, with configurable caps per platform and per run (e.g., max applications per site per day, randomized delays, proxy rotation).	Must	Planned	Non-Functional
NFR-4	System handles common failures (layout changes, captchas, network errors) by routing cases to the Chrome extension/manual flow with clear error codes instead of uncontrolled retries.	Must	Planned	Non-Functional
NFR-5	Personal data (resumes, profile attributes, platform cookies, API keys) is stored securely, with encryption at rest where possible and no secrets written to logs.	Must	Planned	Non-Functional
NFR-6	System exposes basic observability: metrics for jobs discovered, auto-applied, queued, failed, and retried per run, with enough detail to debug platform or logic issues.	Should	Planned	Non-Functional

Run Model & Capacity Plan

The AI Job Automation Agent operates in three discrete end-to-end runs per week. Each run is responsible for discovering 100–200 jobs, scoring and routing them, auto-applying to roughly half of the roles, and queuing the remainder as high-ROI opportunities. Capacity planning, rate limits, and resource allocation are all sized around this 3-runs-per-week model, not continuous 24/7 operation.

Run Model

Name	Notes	Value
Runs per week	Fixed scheduled full batches (for example: Monday, Wednesday, Friday).	3

Name	Notes	Value
Jobs discovered per run (min)	If fewer are found, the run is under-filled and discovery rules may need tuning.	100
Jobs discovered per run (max)	Hard cap to control cost, scraping load, and automation risk.	200
Jobs auto-applied per run (target)	Approximately 50% of discovered jobs that pass high-confidence thresholds.	~50–100
Jobs queued per run (target)	Remaining ~50% high-ROI, dream roles stored with generated assets.	~50–100
Weekly discovery volume	Aggregate across 3 runs based on 100–200 jobs per run.	~300–600
Weekly auto-apply volume	Assumes roughly 50% of discovered jobs are auto-applied.	~150–300
Max applications per platform per run	Configured caps to respect rate limits and anti-bot rules.	Site-specific (e.g., LinkedIn ≤ 25, others ≤ 10–20)
Max concurrent Playwright sessions	Concurrency tuned to proxy bandwidth and system resources.	3–5

Target Platforms & Compliance

The AI Job Automation Agent targets a fixed set of job platforms and job boards. Each platform has different rate limits, anti-bot protections, and terms of service expectations, so the system is configured with per-site limits, automation strategies, and escalation rules. This section tracks the supported platforms, how they are accessed (scraping vs API vs manual assist), and any compliance notes that influence run volumes and automation depth.

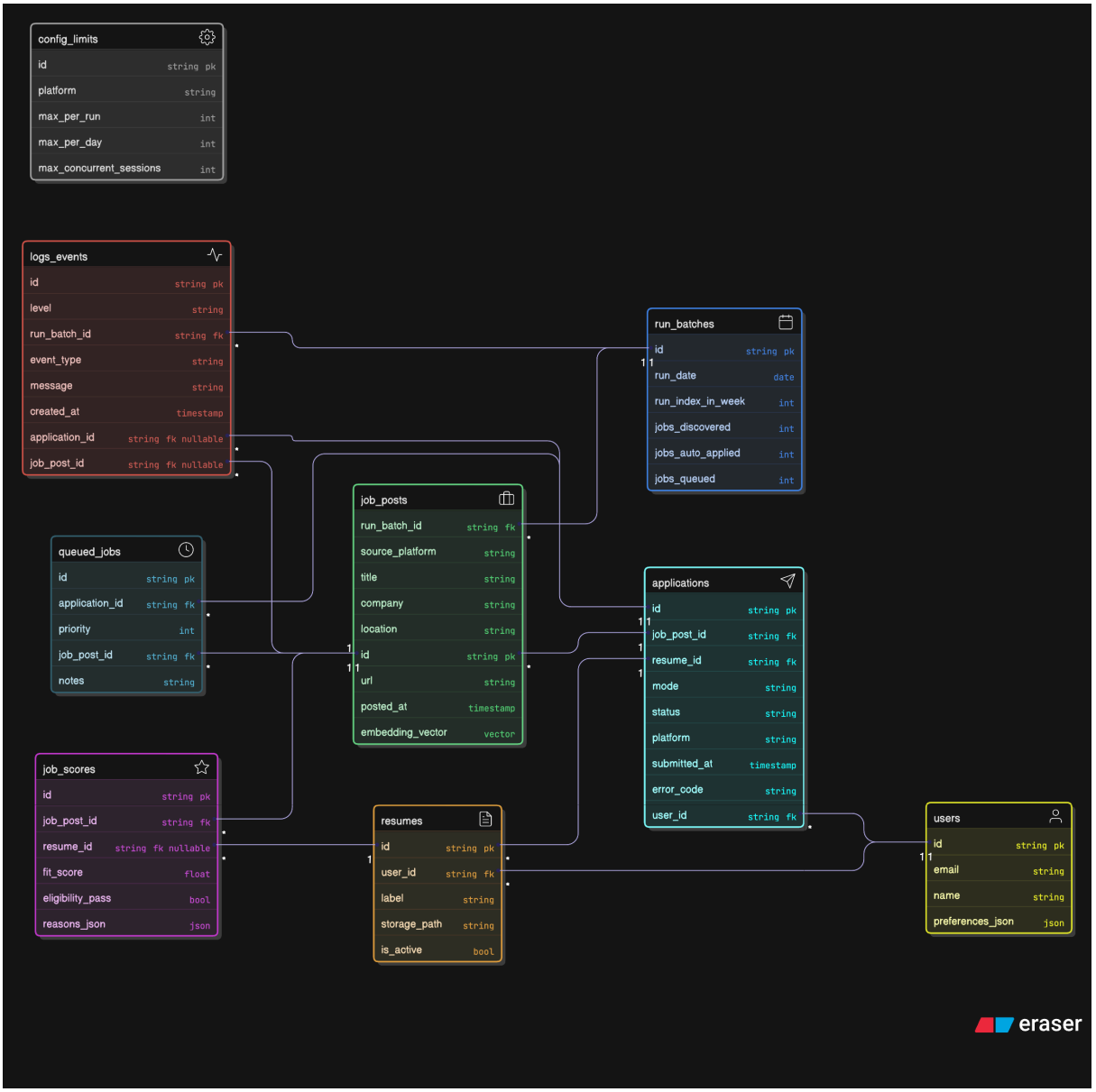
Target Platforms

Name	Access Method	Category	Compliance / Notes	Per-Run Cap	Primary Automation Path
Linkedln	JobSpy / site scraping	Job Board / Network	High anti-bot sensitivity; must respect daily caps, random delays, and avoid obvious scraping patterns.	<=25 applications	Playwright auto-apply for Easy Apply; Chrome extension for complex flows
Indeed	JobSpy / site scraping	Job Board	Monitor layout changes; avoid aggressive parallel sessions.	<=20 applications	Playwright auto-apply where forms are stable; extension fallback
Glassdoor	JobSpy / site scraping	Job Board	Some content behind login; treat carefully with proxy rotation.	<=15 applications	Playwright auto-apply; extension fallback
Google Jobs	SerpApi	Aggregator	Use SerpApi quotas; real applications occur on downstream sites.	N/A at aggre	Indirect: link out to source site, then

Name	Access Method	Category	Compliance / Notes	Per-Run Cap	Primary Automation Path
				gator level	Playwright/extension
ZipRecruiter	JobSpy / site scraping	Job Board	Respect pagination and request pacing.	<=15 applications	Playwright auto-apply
Wellfound (AngelList)	SerpApi Site Scraping	Startup Board	Startup-focused; forms may vary by company.	<=15 applications	Playwright auto-apply; extension for custom flows
Remotive	SerpApi Site Scraping	Remote Board	Primarily remote roles; usually simpler forms.	<=10 applications	Playwright auto-apply
WeWorkRemotely	SerpApi Site Scraping	Remote Board	Many jobs redirect to company sites; treat redirects as separate platforms.	<=10 applications	Playwright auto-apply
RemoteOK	SerpApi Site Scraping	Remote Board	Similar to other remote boards; simple but monitor for anti-bot.	<=10 applications	Playwright auto-apply
Jooble	SerpApi Site Scraping	Aggregator / Board	Mixture of direct and redirected applications.	<=10 applications	Playwright auto-apply or redirect to source
Simply Hired	SerpApi Site Scraping	Job Board	ATS redirects common; handle via site-specific flows.	<=10 applications	Playwright auto-apply
Stack Overflow Jobs	SerpApi Site Scraping	Tech Board	Tech-focused; may link to external ATS.	<=10 applications	Playwright auto-apply; extension fallback
YC Startup Jobs	SerpApi Site Scraping	Startup Board	High-ROI early-stage roles; consider higher scoring weight.	<=10 applications	Playwright auto-apply
Hiring Cafe	SerpApi Site Scraping	Curated Board	Curated list; treat as high-ROI by default.	<=10 applications	Playwright auto-apply

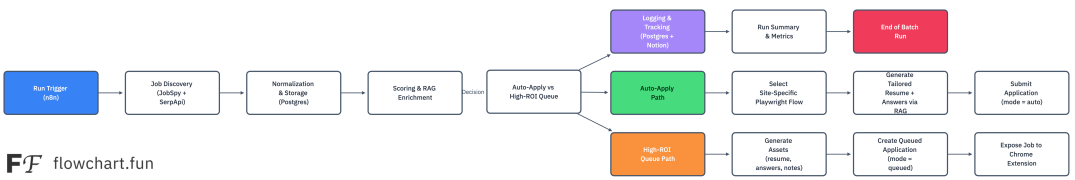
Data Model & Storage Strategy

The AI Job Automation Agent uses PostgreSQL as the primary system of record and Notion as the main human-facing tracking surface. Postgres stores jobs, applications, user profile data, resume variants, configuration, logs, and embeddings (via pgvector) required for the RAG layer. Notion mirrors a curated subset of this data—primarily application status and notes—so that day-to-day review and edits can happen in a familiar interface without coupling operational logic to Notion's APIs. Redis is optional and reserved for future queueing and caching needs.



System Workflow

Each run of the AI Job Automation Agent is a self-contained pipeline that goes from job discovery to application logging. Three times per week, n8n triggers a batch that discovers 100–200 jobs across the configured platforms, normalises and stores them in Postgres, scores and routes them via the Orchestrator and RAG layer, auto-applies to roughly half of the roles using Playwright/Chrome extension, and queues the remaining high-ROI roles for manual/assisted follow-up while synchronizing outcomes to Notion.



Agents & Responsibilities

This section defines the core agents that make up the AI Job Automation Agent: orchestration, discovery, scoring, decisioning, automation execution, manual assist, logging, and monitoring. Each agent has clear inputs, outputs, and fallback behavior so that failures are isolated, decisions are traceable, and the 3-runs-per-week batches can complete reliably with minimal human intervention.

Agents

Name	Failure / Fallback Behavior	Inputs	Outputs	Responsibilities	Type
Orchestrator Agent	On unexpected errors, mark job as 'orchestrator_failed', log event, and skip to next job without retry storm.	Run context, job_posts, job_scores config limits.	Routing decisions (auto-apply vs queue), per-job execution plan.	Coordinates each batch run; routes jobs through scoring, decisioning, and execution paths based on thresholds and platform rules.	Orchestrator
Discovery Agent	If a site fails or rate-limit hits, logs error, reduces that site's volume, and continues with remaining sources.	Platform search configs, previous seen/applied job history.	Normalized job_posts linked to the current run_batch.	Fetches 100–200 jobs per run from JobSpy and SerpApi across all configured platforms and normalizes them.	Analysis
Scoring Agent	If LLM or embedding service fails, marks job as 'unscored', logs error, and routes job to high-ROI queue for manual review.	job_posts, user profile, resumes, embeddings.	job_scores records with fit_score, eligibility_pass, and reasons.	Computes eligibility and fit_score for each job using rules plus LLM/RAG over the 7-resume profile set.	Analysis
Decision Agent	On inconsistent data, defaults to 'queued' and records a warning log.	job_scores, config_limits, platform counters.	Decision label (auto, queued, skipped) stored with job/application.	Decides per job whether to auto-apply, queue as high-ROI, or skip based on thresholds, risk flags, and platform caps.	Analysis
Auto-Apply Agent (Playwright)	On automation failure, retries once for transient issues; otherwise marks as 'auto_failed' and routes job into high-ROI queue.	Routing decisions, job_posts, generated resume + answers, platform credentials.	applications records (mode=auto, status=submitted or failed), detailed logs_events.	Executes site-specific Playwright flows to submit applications end to end for high-confidence jobs.	Execution
Manual Assist Agent (Chrome Extension)	If form detection fails, surfaces errors to user, logs layout issues, and allows fully manual completion.	queued_jobs entries, generated assets, user actions in browser.	applications updates (mode=manual/queued → submitted), enriched notes.	Supports semi-automated applications for queued high-ROI jobs via a SaaS-grade extension with advanced form detection and UX.	Execution
Logging & Tracking Agent	If Notion API is unavailable, queues sync operations for retry while keeping	events from all agents, application state changes.	logs_events rows, updated applications,	Writes structured logs and application statuses to Postgres and syncs	Support

Name	Failure / Fallback Behavior	Inputs	Outputs	Responsibilities	Type
	Postgres as the source of truth.		Notion pages/rows.	summarized views into Notion for tracking.	
Monitoring Agent	If metric export fails, logs a non-fatal warning and continues core processing.	run_batches, logs_events, applications.	Dashboards, alerts, and run health summaries.	Aggregates metrics per run (discovered, auto-applied, queued, failed) and exposes them to Prometheus/Grafana/Sentry.	Support

Job Discovery & Normalization

Job discovery is responsible for finding 100–200 new roles per batch run across the configured platforms and transforming them into a consistent internal format. This stage uses JobSpy and SerpApi with platform-specific queries, deduplicates results, applies basic eligibility filters, and stores normalized records in Postgres as `job_posts` linked to the current `run_batch`.

- **Sources:** LinkedIn, Indeed, Glassdoor, Google Jobs, ZipRecruiter, Wellfound, Remotive, WeWorkRemotely, RemoteOK, Jooble, SimplyHired, Stack Overflow, YC Startup Jobs, Hiring Cafe.
- **Tools:** JobSpy for direct scraping, SerpApi for Google Jobs and supported engines.
- **Per-run target:** 100–200 unique jobs after deduplication and basic filtering (location, title keywords, blacklist/whitelist).
- **Normalization fields:** platform, title, company, location, job type (remote/hybrid/onsite), posting URL, raw description, posting date, source run_batch_id.
- **Deduplication:** same company + title + location + canonical URL are treated as one job; prior "seen/applied" history prevents re-processing.

Scoring, Ranking & Decision Logic

This stage evaluates each normalized job against the 7-resume profile set to decide whether to auto-apply, queue as a high-ROI opportunity, or skip. It combines deterministic eligibility rules with LLM/RAG-based semantic scoring, then applies thresholds and platform caps so that roughly 50% of per-run jobs are auto-applied and 50% are queued.

- **Eligibility filters (hard rules):**
  - Location / remote preference, work authorization, seniority, salary band, blacklist/whitelist companies, excluded keywords.
  - Jobs failing any hard constraint are marked **skipped** with reasons.
- **Fit scoring (soft rules + AI):**
  - Skill and tech-stack overlap between JD and chosen resume variant.
  - Title/seniority similarity.
  - Semantic similarity via embeddings and/or LLM classification ("strong fit / moderate / weak").
  - Output stored as `fit_score` (0–1) plus `reasons_json`.

- **Decision thresholds (per job):**
  - **Auto-apply:** `eligibility_pass = true`, `fit_score ≥ 0.8`, job not flagged risky, platform caps not exceeded.
  - **High-ROI queue:** `0.5 ≤ fit_score < 0.8` or dream-company / dream-role tags, or auto-apply blocked by complex form.
  - **Skip:** `fit_score < 0.5` or fails eligibility.
- **Volume control:**
  - Enforce per-run targets: ~50–100 auto-apply, ~50–100 queued, via sorting by `fit_score` and `priority` and truncating above per-platform caps.

Automation Execution (Playwright)

Automation execution is responsible for taking high-confidence jobs marked for auto-apply and submitting full applications via site-specific Playwright flows. Each supported platform has its own script, form-mapping configuration, and rate limits so the system can behave like a careful human user while still running fully unattended during each batch.

- **Scope:**
  - Handles only jobs with decision = **auto-apply** and within per-platform caps for the current run.
  - Supports LinkedIn, Indeed, Glassdoor, ZipRecruiter, and other boards where forms are stable enough for scripted flows.
- **Per-site modules:**
  - One Playwright module per platform (e.g., `linkedin_auto_apply`, `indeed_auto_apply`).
  - Each module defines login strategy, navigation steps, selectors for key fields, and success criteria (confirmation page / toast).
- **Input data per job:**
  - Normalized job_post record (title, company, URL, platform).
  - Selected resume variant + RAG-generated answers for screening questions.
  - User/platform credentials and proxy to use.
- **Execution pattern:**
  - Jobs grouped by platform; each platform processed sequentially or with limited concurrency (e.g., 3–5 sessions).
  - Human-like delays between steps, randomized think times, and strict max-applications-per-run per platform.
- **Error handling:**
  - Transient errors (network, timeouts) retried once with backoff.
  - Hard failures (captcha, layout mismatch, blocked account) recorded with specific error codes and the job re-routed into the High-ROI queue for manual/extension handling.
  - All attempts create or update an `applications` row with `mode = auto` and detailed status.

Manual Assist & Chrome Extension

The Chrome extension provides a SaaS-grade, manual-assist path for high-ROI queued jobs and any applications that fail automated Playwright flows. It detects job forms in the browser, pulls pre-generated assets from the backend, autofills fields where safe, and lets the user review and finalize submissions with a rich visual UI.

- **Role in the system:**
  - Handles all jobs with decision = **queued** plus any auto-apply failures re-routed from Playwright.
  - Designed for complex ATS flows, unconventional forms, or dream roles where human review is desired.
- **Key capabilities:**
  - Page and form detection (identify company/job pages, input fields, select boxes, and textareas).
  - Fetch queued_job details from the backend (job metadata, chosen resume variant, suggested answers, notes).
  - Autofill non-risky fields (contact info, basic questions) and propose AI-generated text for longer answers and cover letters.
  - Provide clear visual indicators of which fields were autofilled, which need review, and which are left blank.
- **User experience:**
  - Extension popup or side panel shows job context, fit reasons, and recommended actions.
  - One-click “Apply with suggested answers” plus an option to edit any field before final submit.
  - After submission, the extension reports the outcome back to the backend to update `applications`, `queued_jobs`, and logs.
- **Reliability and safety:**
  - If form detection is incomplete or layout is unknown, the extension degrades gracefully: highlight what it can fill, label unknown fields, and avoid guessing.
  - All API calls are authenticated per user, and no secrets (cookies, tokens) are logged.

RAG & Resume Personalization

The RAG layer personalizes each application by selecting the best resume variant from a 7-resume set and generating context-aware answers using job descriptions plus your profile as retrieval context. It ensures applications stay truthful while strongly aligned to each role’s requirements.

- **Knowledge base:**
  - 7 resume variants stored in Postgres (1 generic + 6 domain-specific), plus structured profile data (skills, experience, achievements, preferences).
  - Embeddings created for resume sections, key bullets, and important profile facts using a shared embedding model (e.g., text-embedding-3-small).
- **Retrieval step per job:**
  - Use job title, tags, and description to predict the most relevant resume variant (e.g., backend, data, ML).

    - Retrieve top-N matching bullets and profile facts via vector search; pass these as context to the LLM.
- **Generation policies:**
    - LLM may rephrase, reorder, and highlight existing experience but must not fabricate new employers, roles, or credentials.
    - Outputs:
        - Recommended resume variant ID.
        - Tailored summary / headline.
        - Draft answers for common screening questions.
        - Optional short personalization snippet for cover letters or recruiter notes.
- **Integration into pipeline:**
    - RAG runs during **Scoring & RAG Enrichment** and again just before **Auto-Apply** or **Queue** to produce final text.
    - Generated content and chosen resume ID are stored alongside `job_scores` or `applications` so both Playwright and the Chrome extension can reuse them.

MCP & Orchestration Layer

MCP plus n8n act as the control plane for the AI Job Automation Agent, exposing scraping, scoring, RAG, automation, and logging tools behind a consistent interface. This allows higher-level agents and workflows to call capabilities in a modular way while keeping state and business logic centralized in the backend.

- **MCP role:**
    - Wraps core capabilities as tools: `discover_jobs`, `score_job`, `decide_route`, `auto_apply`, `queue_job`, `log_event`, `sync_notion`.
    - Provides a stable API for orchestrator/analysis agents, making it easier to change underlying implementations (e.g., swap JobSpy config or LLM provider) without rewriting agent prompts.
- **Interaction with n8n:**
    - n8n handles scheduling (3 runs/week), branching, retries, and data passing between steps.
    - Workflow nodes call MCP tools via HTTP or native integrations, ensuring all heavy logic lives in the backend services, not in n8n itself.
- **Benefits:**
    - Clear separation of concerns: n8n = workflow graph; MCP tools = executable capabilities.
    - Easier testing and simulation (MCP tools can be called directly), plus better observability because each tool call is logged with inputs/outputs.

n8n Workflow Design

Short page description

> The n8n workflow defines the concrete sequence of steps that implement one batch run: triggering, discovery, scoring, decisioning, execution, and logging. It is intentionally linear with explicit branches for auto-apply vs high-ROI queue and clearly defined failure paths.

Suggested bullet content

- **Main workflow nodes (high level):**
  1. **Cron / Manual Trigger** – starts one of the 3 weekly runs and creates a `run_batch` record.
  2. **Discovery Node** – calls MCP `discover_jobs` (JobSpy + SerpApi) until 100–200 unique jobs are collected.
  3. **Normalization Node** – writes/updates `job_posts` in Postgres.
  4. **Scoring & RAG Node** – calls MCP `score_job` for each posting; stores `job_scores`.
  5. **Decision Node** – runs MCP `decide_route` to label jobs as auto / queued / skipped and enforce per-platform caps.
  6. **Auto-Apply Subworkflow** – for auto jobs, calls MCP `auto_apply` (Playwright) with concurrency and retry limits.
  7. **Queue Subworkflow** – for queued jobs, calls MCP `queue_job` to create `queued_jobs` entries and pre-generate assets.
  8. **Logging & Notion Sync Node** – calls `log_event` and `sync_notion` to push run results into dashboards.
  9. **Run Summary Node** – updates `run_batches` metrics and sends any alerts.
- **Failure handling in n8n:**
  - Each critical node has an **error branch** that records a log event and either skips the job or routes it to the queue, rather than failing the whole run.
  - Global error handler sends a notification if a run aborts early or hits unexpected exceptions.

Observability, Logging & Notion Tracking

Observability ensures each batch run is transparent and debuggable. Core metrics, structured logs, and Notion views make it easy to see what happened in a run (discovered, auto-applied, queued, failed) and to drill into individual jobs or failures without digging through raw code.

- **Metrics & dashboards:**
  - Per-run metrics in `run_batches`: jobs discovered, auto-applied, queued, skipped, failed, and retried.
  - Export counters and timings to Prometheus/Grafana (e.g., applications per platform, success rate, average time per job, error rates).
  - Optional alerts via Sentry or similar when error rate, captcha rate, or platform blocks exceed thresholds.
- **Structured logging:**
  - All significant events (tool calls, Playwright actions, extension submissions, errors) create `logs_events` rows with `run_batch_id`, optional `job_post_id` / `application_id`, level, event_type, message, and timestamp.
  - Logs are written in a machine-readable JSON format so they can be filtered by job, platform, or failure mode.

- **Notion tracking:**
  - A Notion database mirrors key fields from `applications` and `queued_jobs`: job title, company, platform, mode (auto/queued/manual), status, run date, and notes.
  - n8n (via MCP tools) keeps Notion in sync after each run, enabling lightweight review and annotations in Notion while Postgres remains the source of truth.

Security, Privacy & ToS Considerations

Security and privacy controls protect user data and platform accounts, while ToS-aware behavior reduces the risk of bans or legal issues. The system is designed for a single user but follows patterns that can scale to stricter environments.

- **Data protection:**
  - Store resumes, profile data, platform cookies, and API keys in Postgres with encryption at rest where supported; never log secrets.
  - Access to credentials is scoped per service (scraping, Playwright, extension) and loaded into memory only when needed.
- **LLM and API usage:**
  - Only send necessary, minimized snippets of job descriptions and resume content to LLM providers (NVIDIA NIM, OpenRouter, etc.).
  - Clearly document which external providers are used and allow the user to disable or swap providers.
- **Platform ToS & safety:**
  - Respect rate limits and usage guidelines of each job board; enforce per-platform caps and human-like timing to avoid aggressive behavior.
  - Avoid bypassing explicit anti-bot mechanisms; if a platform blocks automation or introduces heavy captchas, route jobs to manual/extension mode or disable that source.
- **User control & consent:**
  - Provide a master “panic stop” to halt all automation immediately.
  - Make it explicit that the system applies on the user’s behalf; allow per-platform opt-in/opt-out and easy deletion of stored data on request.

Failure Modes & Fallback Strategies

The system anticipates common failure points (scraping blocks, LLM hallucinations, Playwright crashes, platform changes) with layered fallback strategies that prioritize queueing over hard failures, ensuring jobs aren't lost while maintaining run momentum.

- **Scraping & discovery failures:**
  - If JobSpy/SerpApi fails or returns <20 jobs, fall back to cached jobs from prior runs or secondary sources (LinkedIn RSS, RSS feeds).
  - Rate limit exceeded? Pause discovery for 30min and resume; log as `discovery_throttled`.

- **Scoring & decisioning failures:**
  - LLM scoring fails or scores are inconsistent? Default to `queued` for jobs above basic filters (location, keywords) rather than skipping.
  - `decide_route` errors route all remaining jobs to manual queue with a note for review.
- **Application failures:**
  - Playwright crashes/captchas: max 2 retries per job, then queue with `auto_failed` status.
  - Platform-specific issues (e.g., LinkedIn "something went wrong"): immediate queue + disable platform for 24h.
  - Extension submission fails: mark as `extension_pending` and notify via email/Slack.
- **Global recovery:**
  - Run-level failures (Postgres outage, n8n crash) trigger resume from last checkpoint using `run_batch_id`.
  - Daily cleanup reprocesses `stuck` jobs (pending >48h) via a separate n8n cron.

Future Enhancements & Scaling

Planned improvements focus on multi-user support, advanced AI capabilities, and platform expansion while maintaining the core single-user reliability. Scaling patterns enable team use cases without major rewrites.

- **AI & decisioning upgrades:**
  - Fine-tune scoring models on historical apply success data to improve auto-apply precision.
  - Add multi-agent debate for edge-case decisions (e.g., "is this worth manual review?").
  - Integrate computer vision for screenshot-based form filling on tricky platforms.
- **Platform & source expansion:**
  - Add Indeed, Glassdoor, AngelList via dedicated scrapers or official APIs.
  - Support international job boards (e.g., Naukri for India) with locale-aware scoring.
  - Browser extension v2: real-time job scoring during manual browsing.
- **Multi-user & enterprise:**
  - User namespaces in Postgres for team accounts with shared templates but isolated credentials.
  - Role-based queues: auto-apply for juniors, manual-only for seniors.
  - SaaS mode: per-user pricing based on applies/month, with white-label Notion dashboards.
- **Infrastructure scaling:**
  - Containerize MCP tools for Kubernetes; use Redis for job queues and state.
  - Distributed Playwright clusters with sticky sessions per platform account.
  - Cost optimization: dynamic LLM provider routing based on price/performance.

Conclusion

Conclusion

Name	Dependencies	Key Components	Key Specs	Limits/Constraints	Technologies
Future Enhancements	Additional scrapers infra	Fine-tune scoring multi-agent debate CV form filling multi-user Kubernetes	Team namespaces per-user pricing International boards	Enterprise: $99/user/month	MLflow Weights&Biases Redis Playwright Grid
Cost Structure	Credit cards API keys	$25-50/month: Supabase $20 SerpApi $10 JobSpy $10 LLM $5-10	Scale to $100/month 10x volume	Monitor via Grafana alerts	Usage-based: SerpApi 1000 queries JobSpy 3 runs/week LLM 200 jobs x $0.01
Deployment	Domain SSL certificates	Docker Compose local dev Railway/Render production	Single server: 8GB RAM 4 cores Local dev: Docker Compose	Weekly backup Postgres dump	Postgres 12GB Supabase n8n self-hosted MCP FastAPI
Notion Integration	Notion API token	Mirror applications queued_jobs to Notion DB	Auto-sync after each run User annotations in Notion	Notion as review layer Postgres=source of truth	n8n Notion node MCP sync_notion
Browser Extension	Chrome WebStore MCP API	Real-time job detection form prefill manual override	Detects 80% standard forms Prefills 60% fields	Manual override always available	Chrome extension content script background MCP API
Platforms Supported	Selectors credentials	LinkedIn LinkedIn Easy Apply Greenhouse Lever Ashby Greenhouse others	Per-platform caps: LinkedIn 20/run others 10/run	Disable on captcha/block	Browser extension Chrome Playwright selectors
Auto-Apply Engine	Playwright credentials selectors	Playwright form filling cover letter generation browser extension fallback	2 retries per job Platform-specific selectors	Concurrent: 3 per platform	Playwright stealth cookies Chrome profiles MCP auto_apply
Discovery Sources	API keys JobSpy credits	JobSpy SerpApi LinkedIn RSS cached jobs	100-200 unique jobs per run Dedupe by URL hash	Rate limits per source	JobSpy API SerpApi Playwright RSS parsers
Scoring & RAG	LLM provider resume DB	4 criteria: location salary company recency match Weighted LLM scoring	Score 0-100 JSON output Match% Location% etc	Top 50% auto-apply threshold	RAG: job desc + resume chunks NVIDIA NIM Llama3.1
Data Model	Postgres only	run_batches job_posts job_scores applications queued_jobs logs_events	Primary keys: run_batch_id job_post_id	Retention: 90 days logs 365 days apps	native Postgres JSONB indexes

Name	Dependencies	Key Components	Key Specs	Limits/Constraints	Technologies
			application_id		
Core Architecture	All services	Postgres: 8 tables Supabase MCP Layer n8n Orchestration Playwright scraping LLM scoring	Single user 3 runs/week 100-200 jobs/run	Playwright 2GB RAM 4 cores	Postgres Supabase MCP FastAPI n8n Playwright JobSpy SerpApi NIM OpenRouter
Failure Modes Fallbacks	Postgres n8n cron	Scraping: cached jobs Scoring: default queued Apply: 2 retries then queue	Resume from checkpoint Stuck job cleanup >48h	Platform disable 24h on errors	n8n error branches Postgres checkpoints Playwright retries
Security Privacy ToS	User consent credentials	Encrypted credentials minimized LLM payloads rate limiting panic stop	Per-platform caps human-like timing ToS-aware behavior	Disable platforms on blocks	Postgres encryption Playwright cookies Vault
Observability Logging Notion	Postgres Notion API	run_batches job_scores applications queued_jobs logs_events	Structured JSON logs per event Metrics export to dashboards	Alert thresholds: error>10% captcha>20%	Postgres Prometheus Grafana Notion API Sentry
n8n Workflow Design	n8n MCP tools Postgres	9 main nodes: Trigger Discovery Normalization Scoring Decision Auto-Apply Queue Logging Summary	Linear workflow with error branches 100-200 jobs/run	Concurrency limits per platform	n8n Cron Postgres upsert Playwright MCP HTTP
MCP & Orchestration Layer	Postgres n8n LLM providers Playwright	MCP tools: discover_jobs score_job decide_route auto_apply queue_job log_event sync_notion	Stable tool interface for agents/workflows	3 runs/week max	n8n HTTP nodes Postgres JSON API