Project Documentation: AI Driven Recruitment Backend

Generated: 2025-09-07

Project Documentation: AI Driven Recruitment Backend  
Filename: Project\_Documentation\_ai\_driven\_recruitment\_backend.docx  
Date: 2025-09-07

Table of Contents  
1. Executive Summary  
2. Functional Overview  
3. Architecture & Components  
4. File Inventory & Purpose  
5. Data Models (High-Level)  
6. Services & Pipeline Flow  
7. API Endpoints  
 7.1 Primary App (/api routers)  
 7.2 Background Pipeline Orchestrator  
 7.3 Pipeline Script API  
 7.4 Job Pipeline Utility API  
 7.5 Modular Demo Pipeline API  
 7.6 Root Application  
8. Environment Variables & Configuration  
9. External Integrations & Dependencies  
10. Local Development Setup  
11. Running the System (Scenarios)  
12. Deployment Notes (Docker)  
13. Security & Authentication Overview  
14. Notable Gaps / Observations  
15. Future Improvements  
16. Appendix: Key Pydantic Schemas  
17. Appendix: Resume & Cover Letter Flow

---------------------------------------------------------------------  
1. Executive Summary  
This backend combines a recruitment management API with an automated AI-driven job application pipeline (scrape → enrich → match → generate → send). Multiple FastAPI apps and a monolithic pipeline script coexist.

## 2. Functional Overview Domains: Recruitment (users, roles, jobs, applications, interviews, comms) and Automation (Apify, Apollo, OpenAI, SendGrid, enrichment, resume generation).

## 3. Architecture & Components Layers: API (multiple FastAPI apps), Core (config, DB, security), Models (SQLAlchemy), Pipeline Script, Routers for micro-stages, Auxiliary scripts, SQLite + optional PostgreSQL.

## 4. File Inventory & Purpose (Condensed) - main.py: Unified FastAPI app (mounts pipeline + background trigger). - job\_application\_pipeline.py: Full procedural pipeline + FastAPI endpoint. - app/main.py: API that exposes /api with modular routers. - app/routers/\*.py: scrape, match, enrich, generate, send, pipeline, status endpoints. - app/api/job\_pipeline\_api.py: Form/file-based granular pipeline endpoints. - app/api/pipeline\_api.py: Demo modular pipeline (mostly stubs). - app/core/config.py / database.py / security.py: Settings, engine mgmt, JWT & password hashing. - app/models/\*: Rich domain (User, Role, Permission, Job, Communication, etc.). - app/schemas/\*: Pydantic schemas mirroring domain + pipeline (job\_pipeline.py). - resume\_convertion.py / coverletter\_convertion.py: TXT→DOCX conversions. - apollotest.py: Apollo contact search helper. - Dockerfile: Minimal image (runs simple\_app.py). - docker-compose.yml: Single api service. - start.sh: Startup supervisor + watchdog. - requirements.txt: Dependencies. - users.json: Includes blacklist for filtering companies. - jobs.db / events.db: SQLite artifacts.

## 5. Data Models (High-Level) User/Role/Permission RBAC, Job (skills, departments, analytics fields), Communication (Message, Notification, CallLog), Application workflow (not all files shown but supported), Separate lightweight jobs table (pipeline).

## 6. Services & Pipeline Flow 1) Scrape (Apify) → 2) Persist JSON + SQLite → 3) Enrich (Apollo) → 4) Match (OpenAI analysis) → 5) Generate ATS resume & cover letter → 6) Send via SendGrid → 7) Track status.

## 7. API Endpoints (Representative) /api/scrape (POST) → scrape jobs /api/match (POST) → AI resume-job match /api/enrich (POST) → contact enrichment /api/generate (POST) → generate resume + cover letter /api/send (POST) → send application email /api/pipeline (POST) → launch background pipeline /api/status (GET) → list runs /api/status/{id} (GET) → run status /run-pipeline (POST) in job\_application\_pipeline.py /job\_pipeline\_api.py endpoints: scrape-jobs, save-jobs, enrich-contacts, match-jobs, generate-documents, send-application /pipeline\_api.py demo: scraping/jobs, resumes/upload, matching/match, generation/cover-letter, etc. /api/run-pipeline (POST) in main.py triggers external script.

## 8. Environment Variables APIFY\_TOKEN, OPENAI\_API\_KEY, SENDGRID\_API\_KEY, APOLLO\_API\_KEY, DATABASE\_URL, SECRET\_KEY, ADMIN\_EMAIL, ADMIN\_PASSWORD, Redis/SMTP optional.

## 9. External Integrations Apify (scraping), Apollo (contact enrichment), OpenAI (analysis & generation), SendGrid (email), pdfminer.six (PDF extraction), python-docx (DOCX conversion).

10. Local Development Setup  
python -m venv venv  
pip install -r requirements.txt  
Create .env (tokens + DATABASE\_URL)  
Initialize DB (init\_db)

11. Running Scenarios  
- Background pipeline: POST /api/pipeline then poll /api/status/{run\_id}  
- Manual sequence: scrape → match → generate → send  
- Script CLI: python job\_application\_pipeline.py --title ... --resume ... --dry-run

12. Deployment Notes  
Dockerfile runs fallback simple\_app.py (adjust CMD to uvicorn main:app). Add a process manager, persist pipeline run metadata, external DB for production.

13. Security & Authentication  
JWT + bcrypt; role/permission models present; pipeline endpoints currently unauthenticated—add protection. In-memory status store loses state on restart.

14. Notable Gaps  
Duplicate pipeline code, missing referenced routers (admin/auth), Docker entry mismatch, no tests, no persistence for run state, README conflicts, limited error resilience for OpenAI JSON parsing.

15. Future Improvements  
Unify pipeline service layer, add queue (Celery/RQ), persist run metadata, schema validate AI responses, test coverage, API auth for pipeline, rate limiting, structured logging & metrics.

16. Key Pydantic Schemas  
ScrapeRequest, MatchRequest, GenerateRequest, SendRequest, PipelineRequest, JobSearchParams, JobMatchingRequest, ResumeOptimizationRequest, CoverLetterRequest, EmailRequest.

17. Resume & Cover Letter Flow  
Extract text → OpenAI match analysis JSON → ATS optimization (emphasize match criteria, add missing keywords) → Cover letter generation → Save .txt + convert to .docx → Send email with attachments (or dry-run).