Technical Stack & Architectural Flow

Agentic AI Hiring Platform - Complete Technical Overview

TECHNICAL STACK

Backend Framework

- **FastAPI 0.109.0** - Modern, high-performance Python web framework
    - Async/await support for high concurrency
    - Automatic API documentation (Swagger UI & ReDoc)
    - Type validation with Pydantic
    - Fast request/response handling
- **Uvicorn 0.27.0** - ASGI server
    - Production-ready server
    - WebSocket support
    - HTTP/2 support

Database Layer

- **PostgreSQL** - Primary relational database
    - Robust ACID compliance
    - JSON support for complex data structures
    - Full-text search capabilities
- **SQLAlchemy 2.0.25** - ORM (Object-Relational Mapping)
    - Type-safe database operations
    - Connection pooling
    - Migration support
- **Alembic 1.13.1** - Database migrations
    - Version control for schemas
    - Automated migration scripts

AI & Machine Learning

- **HuggingFace Hub** - AI/ML model hosting
    - Access to transformer models
    - Inference API integration
    - Model: `sentence-transformers/all-MiniLM-L6-v2`
- **NumPy 1.26.3** - Numerical computing
    - Vector operations
    - Similarity calculations
    - Matrix operations

Document Processing

- **PyPDF2 3.0.1** - PDF text extraction
    - Resume parsing
    - Job description parsing
    - Multi-page document support

Web & API

- **Python Multipart 0.0.6** - File upload handling
    - Multipart form data processing
    - Large file support
- **Pydantic 2.5.3** - Data validation
    - Type checking
    - Schema validation
    - Automatic documentation
- **Python-dotenv 1.0.0** - Environment management
    - Configuration management
    - Secrets handling

Deployment & Infrastructure

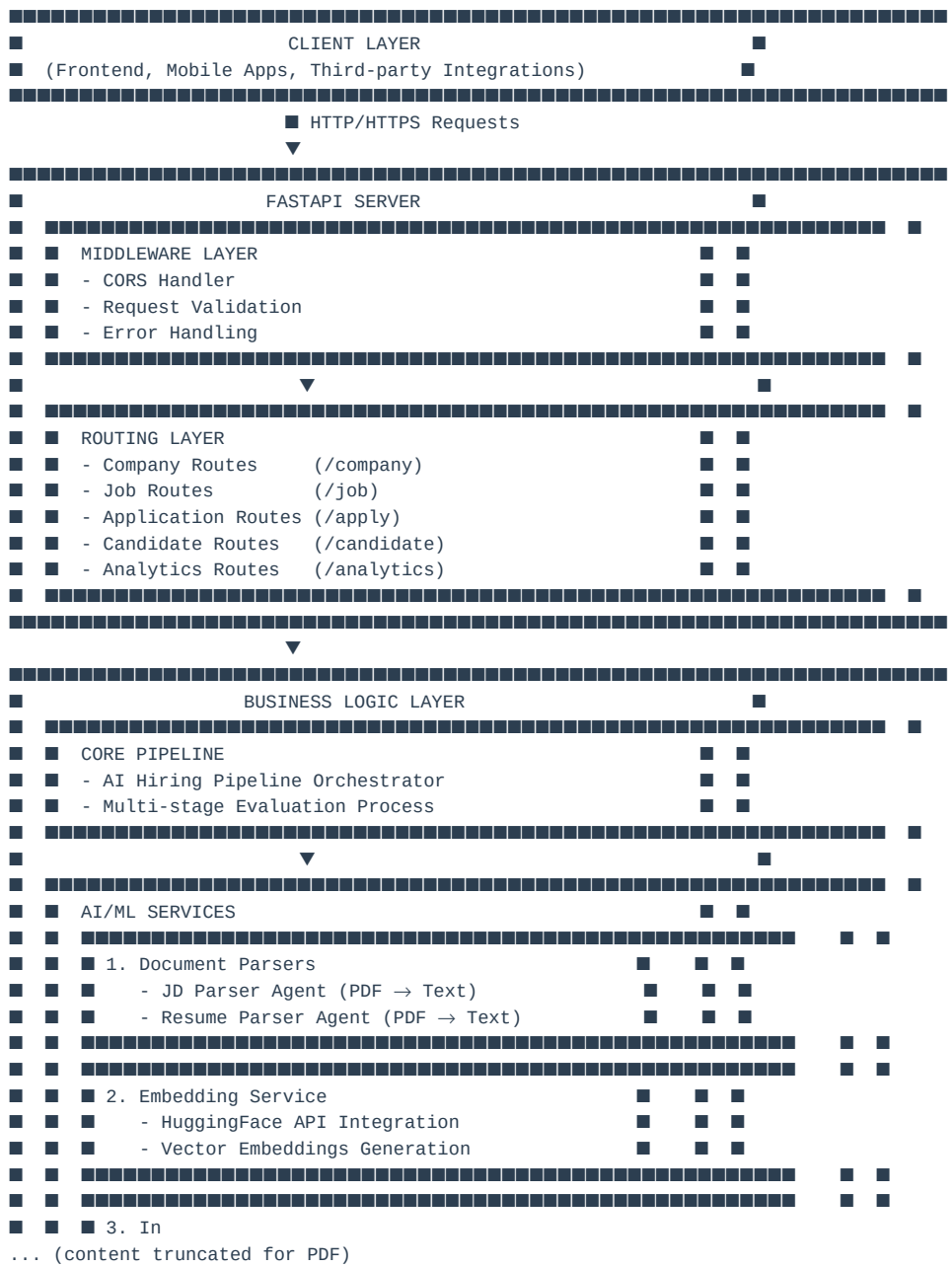
- **Docker** - Containerization
    - Consistent environments
    - Easy deployment
    - Scalability
- **Render.com** - Cloud hosting
    - Auto-deployment from Git
    - PostgreSQL hosting
    - Environment variables

Security & Performance

- **CORS Middleware** - Cross-Origin Resource Sharing
- **Connection Pooling** - Database optimization
- **Pre-ping Health Checks** - Connection reliability

ARCHITECTURAL FLOW

System Architecture Overview



... (content truncated for PDF)

DETAILED PROCESS FLOWS

1. Job Posting Flow

```
Recruiter Uploads JD PDF
        ↓
JD Parser Agent Extracts Text
        ↓
Text Cleaning & Normalization
        ↓
Embedding Service Generates Vector
        ↓
Inference Engine Extracts Skills
        ↓
Skills Classified (Technical/Soft)
        ↓
Required vs Nice-to-Have Detection
        ↓
Job Record Created in Database
        ↓
Job ID Returned to Client
```

Key Technologies:

- PyPDF2 for PDF parsing
- Text cleaning utilities
- HuggingFace embeddings
- AI skill extraction
- PostgreSQL storage

2. Application Submission & Evaluation Flow

```
Candidate Submits Application (Resume PDF + Details)
        ↓
═══════════════════════════════════════════════
■  STAGE 1: DOCUMENT PROCESSING           ■
═══════════════════════════════════════════════
Resume Parser Extracts Text
        ↓
Text Cleaning & Validation
        ↓
Email/Phone Extraction
        ↓
Candidate Record Created
        ↓
═══════════════════════════════════════════════
■  STAGE 2: FEATURE EXTRACTION            ■
═══════════════════════════════════════════════
Embedding Service Generates Resume Vector
        ↓
Inference Engine Extracts Skills
        ↓
Skills Normalized & Classified
        ↓
Experience Parsing & Validation
        ↓
═══════════════════════════════════════════════
■  STAGE 3: AI EVALUATION PIPELINE        ■
═══════════════════════════════════════════════
[3.1] Scoring Engine Computation
```

```
      - RFS: Cosine Similarity (JD vs Resume embeddings)
      - DCS: Weighted Skill Match (Technical skills only)
      - ELC: Experience Level Compatibility
      - Composite: Weighted Average
        ↓
[3.2] Fraud Detection Analysis
      - Compare with existing candidates
      - Resume similarity check (>90% threshold)
      - Email duplication check
      - Generate fraud report
        ↓
[3.3] Decision Service
      - Evaluate composite score against thresholds
      - Consider fraud flags
      - Generate decision (Selected/Rejected/Review)
      - Create decision reasoning
        ↓
[3.4] Explanation Generation
      - Identify strengths & weaknesses
      - Provide actionable recommendations
      - Generate detailed explanation
        ↓
[3.5] XAI Analysis (for transparency)
      - Break down decision factors
      - Assign contribution percentages
      - Generate confidence levels
        ↓
[3.6] Skill Gap Analysis
      - Identify matched vs missing skills
      - Separate required from nice-to-have gaps
      - Generate learning roadmap
      - Estimate closure time
      - Identify transferable skills
        ↓
[3.7] Skill Evidence Graph Generation
      - Create node-edge graph structure
      - Visual representation of skill relationships
        ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STAGE 4: RANKING & STORAGE             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Calculate Ranking Among Job Applicants
        ↓
Store Application with All Metadata
        ↓
Log Audit Trail
        ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  STAGE 5: RESPONSE GENERATION           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Return Complete Evaluation to Client
  - Application ID
  - All Scores (RFS, DCS, ELC, Composite)
  - Ranking Position
  - Decision & Reasoning
  - Explanation
  - Fraud Analysis
```

Processing Time: ~2-5 seconds per application

3. Candidate Master Endpoint Flow

```
Request: GET /candidate/{id}/master
        ↓
Database Query: Fetch Candidate
```

```
        ↓
Database Query: Fetch All Applications
        ↓
For Each Application:
  ■■ Fetch Job Details
  ■■ Fetch Company Details
  ■■ Fetch All Scores
  ■■ Fetch Decision & Reasoning
  ■■ Fetch Fraud Analysis
  ■■ Fetch Skill Match Data
        ↓
Calculate Summary Statistics
  ■■ Total Applications
  ■■ Selected/Rejected/Pending Counts
  ■■ Average Composite Score
  ■■ Best Application
        ↓
Compile Comprehensive Response
        ↓
Return JSON with Complete Data
```

4. Analytics & Reporting Flow

```
Request Analytics Endpoint
        ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  XAI Explanation                      ■
■  - Fetch application data             ■
■  - Calculate factor contributions     ■
■  - Generate confidence levels         ■
■  - Return transparent explanation     ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
        OR
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  Skill Gap Analysis                   ■
■  - Identify missing skills            ■
■  - Classify by priority               ■
■  - Generate learning roadmap          ■
■  - Calculate closure time             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
        OR
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  Company Dashboard                    ■
■  - Aggregate all company jobs         ■
■  - Calculate hiring metrics           ■
■  - Generate trends analysis           ■
■  - Return comprehensive dashboard     ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

SCORING ALGORITHMS

1. Role Fit Score (RFS)

```
RFS = cosine_similarity(JD_embedding, Resume_embedding)

Where:
- JD_embedding = 384-dimensional vector from sentence-transformers
- Resume_embedding = 384-dimensional vector from sentence-transformers
- Result range: 0.0 to 1.0 (0% to 100%)
```

2. Domain Competency Score (DCS)

```
WEIGHTED SKILL MATCHING:

Step 1: Classify JD skills
  - Required skills (must-have)
  - Nice-to-have skills (optional)

Step 2: Match candidate skills
  - Direct matches
  - Synonym matches
  - Related skill matches

Step 3: Calculate weighted score
  DCS = (Required_Match_Score × 0.7) + (NiceToHave_Match_Score × 0.3)

Where:
  Required_Match_Score = (Matched Required / Total Required) × 100
  NiceToHave_Match_Score = (Matched NiceToHave / Total NiceToHave) × 100

Note: Only technical skills counted, soft skills excluded
```

3. Experience Level Compatibility (ELC)

```
If candidate_exp >= required_exp:
    ELC = 1.0 (100% match)
Else if candidate_exp >= required_exp × 0.75:
    ELC = 0.8 (80% match)
Else if candidate_exp >= required_exp × 0.5:
    ELC = 0.6 (60% match)
Else:
    ELC = 0.3 (30% match - significant gap)

Additional factors:
  - Relevant experience bonus (+10%)
  - Over-qualification penalty (-5%)
```

4. Composite Score

```
Composite = (RFS × 0.25) + (DCS × 0.50) + (ELC × 0.25)

Weights explanation:
- 25% for semantic fit (RFS)
- 50% for skill matching (DCS) - most important
- 25% for experience (ELC)
```

5. Decision Thresholds

```
If fraud_flag == True:
    Decision = "Rejected - Fraud Detected"
Else if composite >= 0.75:
    Decision = "Selected"
Else if composite >= 0.60:
    Decision = "Review Required"
Else:
    Decision = "Rejected"
```

SECURITY FEATURES

1. **Environment Variables**
    - Sensitive data in .env files
    - No hardcoded credentials
    - DATABASE_URL encryption
2. **Database Security**
    - Connection pooling with pre-ping
    - SQL injection prevention (ORM)
    - ACID transactions
3. **API Security**
    - CORS configuration
    - Request validation
    - Error handling & logging
4. **Fraud Detection**
    - Resume similarity checks
    - Duplicate email detection
    - Suspicious pattern identification

DATA MODELS

Company

- id: Integer (PK)
- name: String
- description: Text
- created_at: DateTime

Job

- id: Integer (PK)
- company_id: Integer (FK)
- role: String
- location: String
- salary: String
- employment_type: String
- required_experience: Integer
- jd_text: Text
- jd_embedding: JSONB (384-dim vector)
- skills_extracted: JSONB
- created_at: DateTime

Candidate

- id: Integer (PK)
- name: String
- email: String (indexed)
- mobile: String
- linkedin: String
- github: String
- experience: Integer
- resume_text: Text
- resume_embedding: JSONB (384-dim vector)
- skills_extracted: JSONB
- created_at: DateTime

Application

- id: Integer (PK)
- job_id: Integer (FK)
- candidate_id: Integer (FK)
- rfs: Float (Role Fit Score)
- dcs: Float (Domain Competency Score)
- elc: Float (Experience Level Compatibility)
- composite_score: Float (indexed)
- rank: Integer (indexed)
- similarity_index: Float
- fraud_flag: Boolean (indexed)
- fraud_details: JSONB
- decision: String (indexed)
- decision_reason: Text
- explanation: JSONB
- skill_match: JSONB
- experience_details: JSONB
- status: String (indexed)

- created_at: DateTime (indexed)

AuditLog

- id: Integer (PK)
- entity_type: String (Job/Candidate/Application)
- entity_id: Integer
- action: String
- details: JSONB
- timestamp: DateTime

DEPLOYMENT ARCHITECTURE

Docker Container

- Base: Python 3.11-slim
- Port: 10000
- User: Non-root (appuser)
- Dependencies: Installed from requirements.txt

Render.com Configuration

- Auto-deploy from GitHub
- PostgreSQL database instance
- Environment variables management
- Health checks enabled
- Auto-scaling support

Environment Variables Required

```
DATABASE_URL=postgresql://...
HF_API_KEY=hf_...
SIMILARITY_THRESHOLD=0.90
ENVIRONMENT=production
LOG_LEVEL=INFO
```

PERFORMANCE CHARACTERISTICS

- **API Response Time**: 50-200ms (without AI processing)
- **Application Evaluation**: 2-5 seconds
- **Concurrent Requests**: 100+ (with proper scaling)
- **Database Queries**: Optimized with indexes
- **Embedding Generation**: ~500ms per document
- **Skill Extraction**: ~1-2 seconds per document

FUTURE ENHANCEMENTS (Optional Dependencies)

- **Redis**: Caching frequently accessed data
- **Celery**: Background task processing
- **JWT Authentication**: User authentication
- **Rate Limiting**: API request throttling
- **Sentry**: Error tracking & monitoring

API DOCUMENTATION

- **Swagger UI**: `/docs` endpoint
- **ReDoc**: `/redoc` endpoint
- **OpenAPI Schema**: Auto-generated
- **PDF Documentation**: Available in repository

This architecture provides a robust, scalable, and AI-powered hiring platform with transparent decision-making and comprehensive candidate evaluation.