

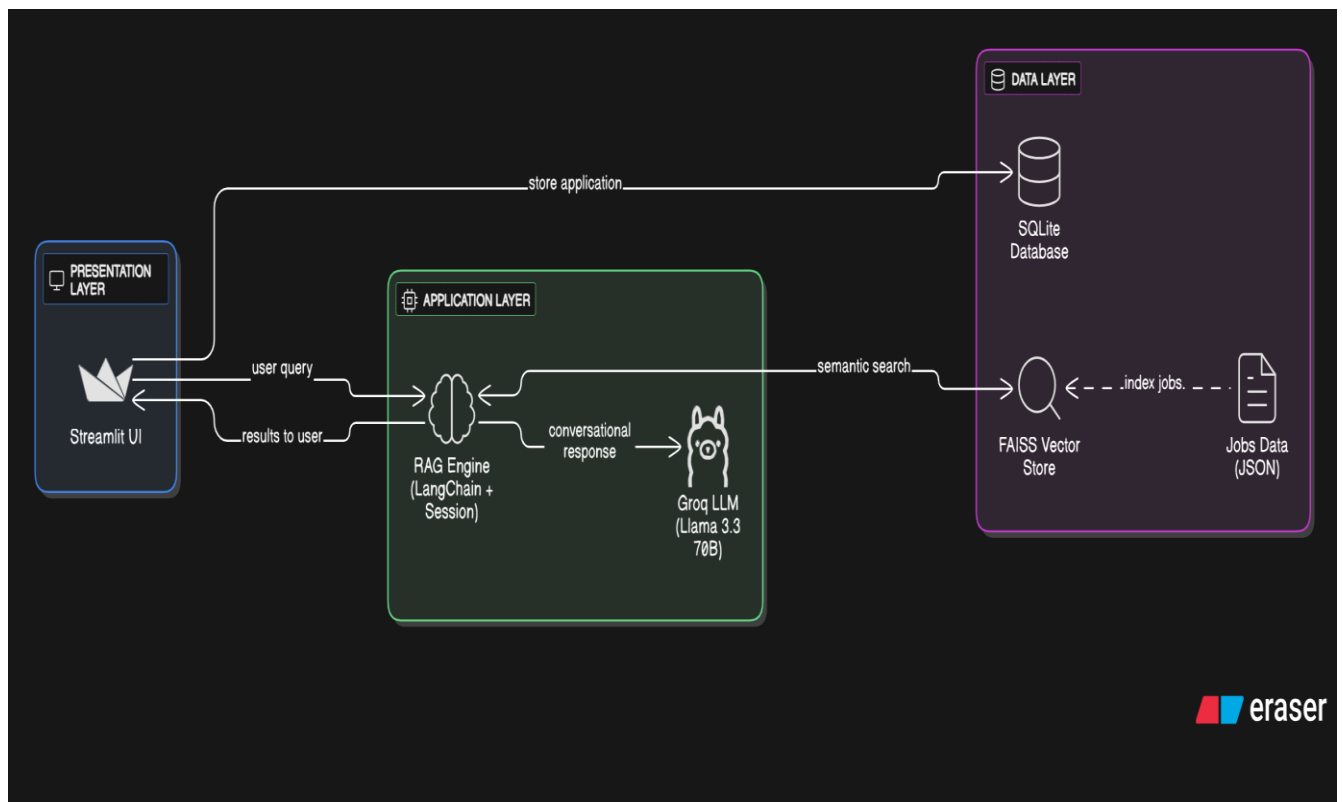
# System Design Document – SmartApply RAG Agent

## 1. Executive Summary

**SmartApply** is an AI-powered job application system that leverages Retrieval-Augmented Generation (RAG) to provide intelligent job matching and streamlined application processing. The system uses semantic search to match candidates with relevant positions and guides them through a conversational application process.

## 2. System Architecture

### 2.1 High-Level Architecture



### 2.2 Component Details

#### Presentation Layer (app.py)

- **Technology:** Streamlit
- **Responsibilities:**
  - User interface for job search
  - Application form handling

- Resume upload management
- Admin dashboard
- Session state management

## Application Layer (RAG Engine (rag\_engine.py))

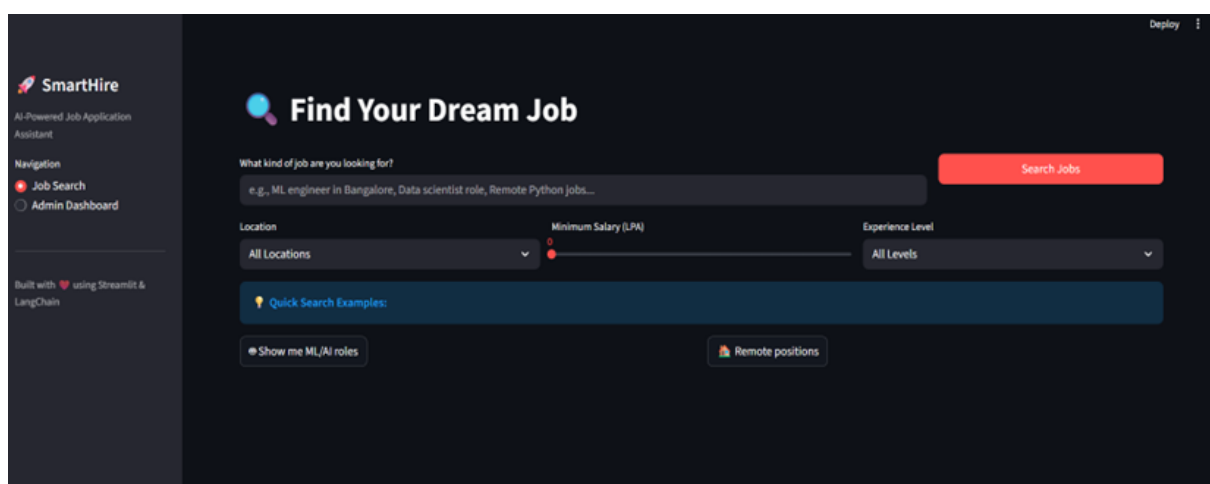
- **Core Components:**
  - **Embeddings:** HuggingFace sentence-transformers/all-MiniLM-L6-v2
  - **Vector Store:** FAISS with persistent indexing
  - **LLM:** Groq API (Llama 3.3 70B)
  - **Memory:** ConversationBufferMemory for context retention

## Database Layer (database.py)

- **Technology:** SQLite with SQLAlchemy ORM
- **Tables:**
  - applications: Core application data
  - screening\_responses: Q&A storage
  - resumes: Binary file storage

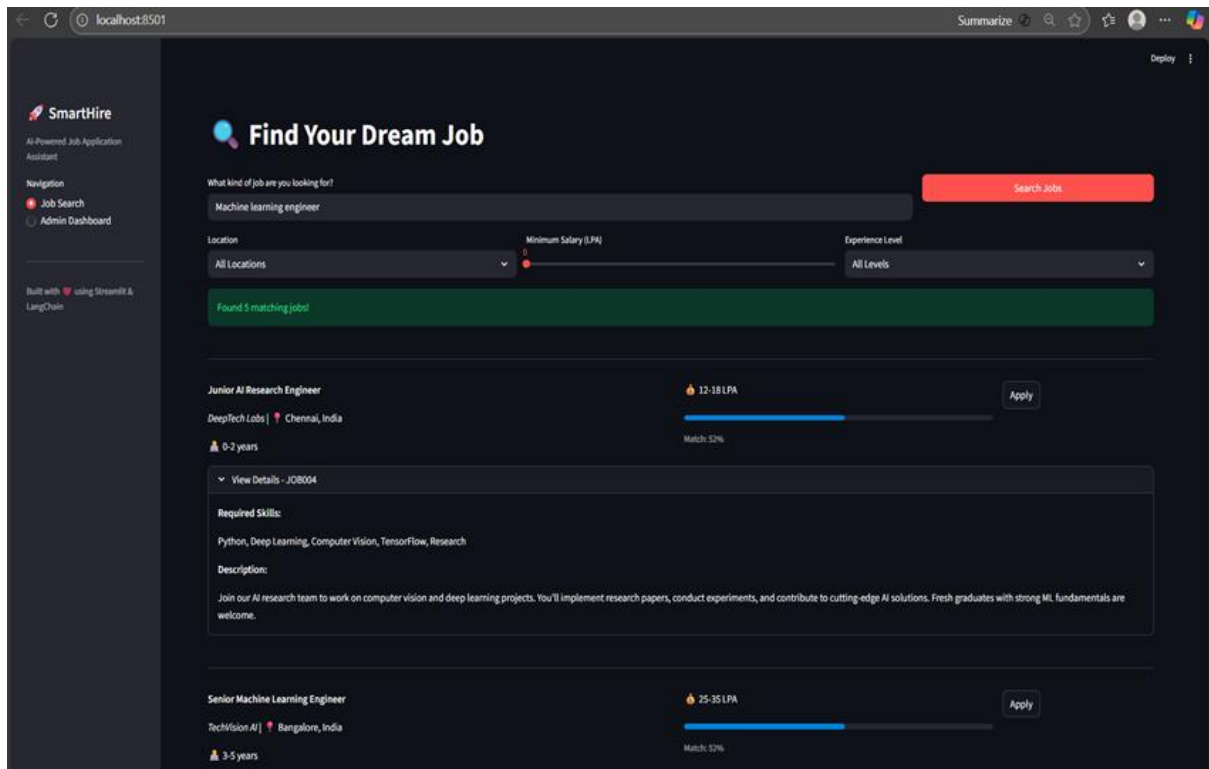
## 2.3 User Interface Examples

The system architecture translates to these key user interfaces that demonstrate the end-to-end workflow:



## Job Search Interface

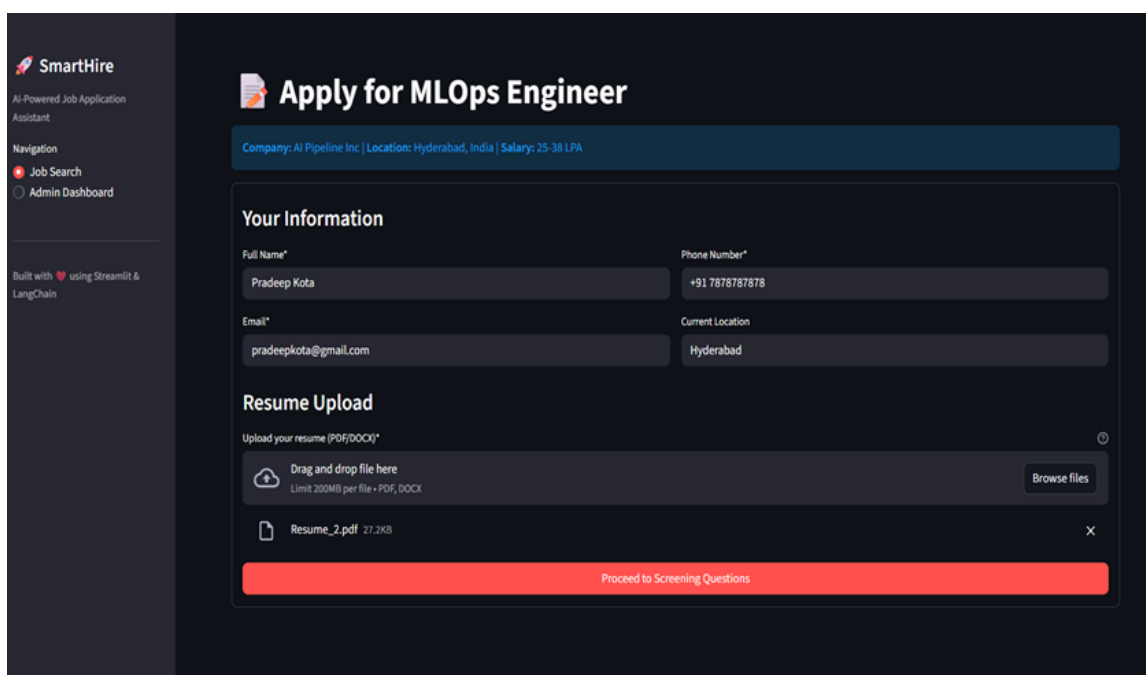
Implements semantic search with RAG-powered matching and relevance scoring



The screenshot shows the SmartHire Job Search Interface. The left sidebar contains the SmartHire logo, the role of 'AI-Powered Job Application Assistant', and navigation links for 'Job Search' and 'Admin Dashboard'. The main content area is titled 'Find Your Dream Job' and features a search bar with the text 'Machine learning engineer'. Below the search bar are filters for 'Location' (All Locations), 'Minimum Salary (LPA)' (0 to 12-18 LPA), and 'Experience Level' (All Levels). A green banner indicates 'Found 5 matching jobs'. Two job listings are displayed: 'Junior AI Research Engineer' at DeepTech Labs in Chennai, India, with a salary of 12-18 LPA and a match score of 52%; and 'Senior Machine Learning Engineer' at TechVision AI in Bangalore, India, with a salary of 25-35 LPA and a match score of 52%. Each listing includes a 'View Details' link and an 'Apply' button.

## Application Form

Shows multi-step application process with resume upload and candidate information collection



The screenshot shows the SmartHire Application Form for the 'MLOps Engineer' position. The left sidebar is identical to the Job Search Interface. The main content area is titled 'Apply for MLOps Engineer' and displays the company name 'AI Pipeline Inc', location 'Hyderabad, India', and salary '25-38 LPA'. The form is divided into two sections: 'Your Information' and 'Resume Upload'. The 'Your Information' section contains fields for 'Full Name\*' (Pradeep Kota), 'Phone Number\*' (+91 7878787878), 'Email\*' (pradeepkota@gmail.com), and 'Current Location' (Hyderabad). The 'Resume Upload' section includes a file upload area with a 'Drag and drop file here' instruction, a 'Browse files' button, and a list of uploaded files (Resume\_2.pdf, 27.2KB). A red button at the bottom of the form is labeled 'Proceed to Screening Questions'.

## Screening questions

Dynamic question-answering interface with job-specific screening and progress tracking

The screenshot shows the 'Screening Questions' interface for the SmartHire application. On the left is a dark sidebar with the SmartHire logo, the text 'AI-Powered Job Application Assistant', a 'Navigation' menu with 'Job Search' (active) and 'Admin Dashboard', and a footer mentioning 'Streamlit & LangChain'. The main content area has a dark background. At the top, it says 'Screening Questions' with a document icon. Below this, a blue box instructs the user to 'Please answer 3 questions for the ML Ops Engineer position'. A progress bar shows 'Question 1 of 3'. The current question is 'How do you handle model drift in production?'. Below the question, it says 'Your answer:' followed by a large text input field containing the text: 'Continuously monitor model performance using tools like Evidently AI and retrain or update the model when performance degrades beyond a defined threshold'. At the bottom right of the input field is a red button labeled 'Next +'.

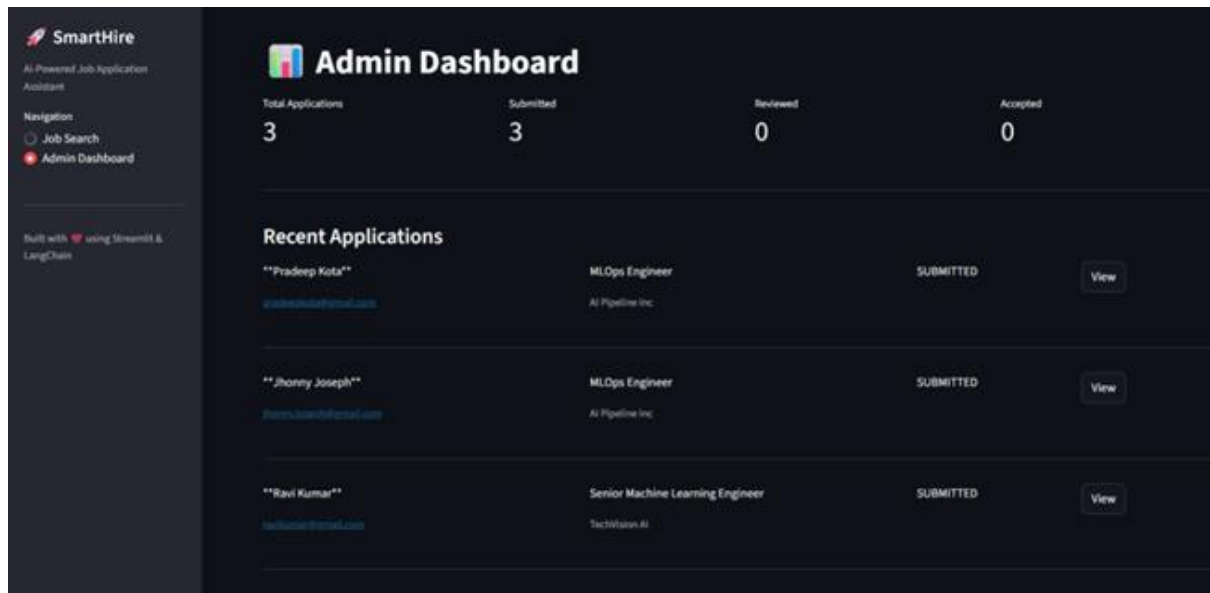
## Application Submission screen

Success confirmation interface displaying application ID and submission status

The screenshot shows the 'Application Submitted Successfully!' confirmation screen. The left sidebar is identical to the previous screen. The main content area has a dark background with a green banner at the top stating 'Application Submitted Successfully!'. Below the banner, the 'Application Details' are listed: 'Application ID: #3', 'Candidate: Pradeep Kota', 'Email: pradeepk@ml.com', and 'Status: Submitted'. A message follows: 'You will receive a confirmation email within 24 hours. The hiring team will review your application and contact you within 5 business days.' At the bottom left of the details section is a red button labeled 'Search for More Jobs'. The background of the main content area is decorated with several large, semi-transparent colored circles in shades of green, blue, orange, and red.

## Admin Dashboard

Provides application management, analytics, and candidate tracking capabilities



## 3. Data Flow

### 3.1 Job Search Flow

1. User Query → "ML engineer in Bangalore"
2. Query Embedding → Convert to vector representation
3. Semantic Search → FAISS similarity search
4. Retrieve Jobs → Top-k matching jobs
5. Display Results → Ranked by relevance score

### 3.2 Application Flow

1. Job Selection → User clicks "Apply"
2. Information Collection → Name, email, phone, location
3. Resume Upload → PDF/DOCX file processing
4. Screening Questions → Dynamic Q&A based on job
5. Data Persistence → Store in SQLite
6. Confirmation → Application ID generation

## 4. Key Features

### 4.1 Semantic Job Search

- Vector similarity matching using FAISS
- Relevance scoring (0-100% match)
- Filter support (location, salary, experience)

### 4.2 Intelligent Caching

- MD5 hash-based change detection for jobs.json
- Persistent FAISS index to avoid re-computation
- Automatic reindexing on data changes

### 4.3 Conversational Interface

- Natural language job queries
- Context-aware responses using conversation memory
- Multi-turn dialogue support

### 4.4 Application Management

- Complete application lifecycle tracking
- Status management (submitted/reviewed/accepted/rejected)
- Admin dashboard with statistics

## 5. Technical Specifications

### 5.1 Dependencies

- **LangChain:** Orchestration framework
- **FAISS:** Vector similarity search
- **Groq:** LLM inference
- **Streamlit:** Web interface
- **SQLAlchemy:** Database ORM
- **Sentence-Transformers:** Free embeddings

### 5.2 Data Models

#### Job Schema

```
{
```

```
"job_id": "string",  
"title": "string",  
"company": "string",  
"location": "string",  
"experience_required": "string",  
"salary_range": "string",  
"skills_required": ["array"],  
"screening_questions": ["array"]  
}
```

### **Application Schema**

```
applications (  
  id: INTEGER PRIMARY KEY,  
  job_id: VARCHAR,  
  candidate_name: VARCHAR,  
  candidate_email: VARCHAR,  
  status: VARCHAR,  
  submitted_at: DATETIME  
)
```

## **6. Performance Optimizations**

### **6.1 Embedding Caching**

- First run: ~30 seconds to create embeddings
- Subsequent runs: <2 seconds (cached loading)
- Hash-based invalidation for data updates

### **6.2 Resource Usage**

- Memory: ~200MB for vector store (20 jobs)
- Storage: ~10MB for FAISS index
- API Calls: Minimal (only for LLM responses)

## **7. Security Considerations**

- API keys stored in environment variables
- SQL injection prevention via ORM
- File upload validation (type and size checks)
- No PII exposed in logs

## **8. Scalability**

### **Current Limitations**

- Single-user session management
- Local file storage for resumes
- In-memory conversation history

### **Scaling Strategy**

- Move to PostgreSQL for production
- Implement Redis for session management
- Use cloud storage (S3) for resumes
- Deploy on cloud platforms (AWS/GCP)

## **9. Testing Approach**

### **Functional Testing**

- End-to-end application flow
- Job search accuracy validation
- Database persistence verification

### **Performance Testing**

- Embedding generation time
- Search response latency
- Concurrent user handling

## **10. Deployment**

### **Local Development**

`pip install -r requirements.txt`

`streamlit run app.py`

### **Production Deployment**



- Containerize with Docker
- Environment variable management
- HTTPS enforcement
- Rate limiting implementation

## **11. Future Enhancements**

- Multi-language support
- Resume parsing and skill extraction
- Email notifications
- Advanced analytics dashboard
- Integration with ATS systems
- Real-time job updates

## **12. Conclusion**

The SmartApply RAG Agent successfully demonstrates an end-to-end AI-powered recruitment system with semantic search, conversational interaction, and complete application management. The architecture is modular, scalable, and optimized for performance while maintaining simplicity.