

HR Q&A POC - Complete Execution Guide

System Architecture Overview

This is a comprehensive HR Q&A system with the following components:

- **Database:** MongoDB Atlas (10 collections covering 360° employee data)
- **Search:** Azure AI Search (semantic + vector + hybrid search)
- **AI:** Azure OpenAI GPT-4 (intent detection + response generation)
- **API:** FastAPI (REST endpoints with comprehensive functionality)
- **CLI:** Rich command-line interface for management
- **ETL:** Automated pipeline for Excel → MongoDB → Search Index

Prerequisites

1. Required Services

- **MongoDB Atlas** account and cluster
- **Azure AI Search** service
- **Azure OpenAI** service with GPT-4 deployment
- Python 3.8+ environment

2. Required Python Packages

```
bash
```

```
pip install -r requirements.txt
```

Expected packages:

- fastapi
- motor (MongoDB async driver)
- azure-search-documents
- azure-identity
- openai
- pandas
- uvicorn
- click

- pydantic
- python-dotenv

Configuration Setup

Step 1: Create Environment File

Create a `.env` file in the project root:

```
env

# MongoDB Configuration
MONGODB_CONNECTION_STRING=mongodb+srv://username:password@cluster.mongodb.net/?retryWrites=true
MONGODB_DATABASE=hr_qna_poc

# Azure Search Configuration
AZURE_SEARCH_SERVICE_NAME=your-search-service
AZURE_SEARCH_ENDPOINT=https://your-search-service.search.windows.net
AZURE_SEARCH_API_KEY=your-search-api-key

# Azure OpenAI Configuration
AZURE_OPENAI_ENDPOINT=https://your-openai-resource.openai.azure.com/
AZURE_OPENAI_API_KEY=your-openai-api-key
AZURE_OPENAI_CHAT_DEPLOYMENT=gpt-4o-11m
AZURE_OPENAI_EMBEDDING_DEPLOYMENT=text-embedding-ada-002
AZURE_OPENAI_API_VERSION=2023-05-15
```

Step 2: Verify Configuration

```
bash

python src/core/config.py
```

Execution Steps

Phase 1: Database Setup

1.1 Create MongoDB Collections

```
bash

python scripts/setup_collections.py --action create_all
```

This creates 10 collections with proper schemas:

- personal_info, employment, learning, experience, performance
- engagement, compensation, attendance, attrition, project_history

1.2 Verify Collections

```
bash  
  
python scripts/setup_collections.py --action validate_schema
```

Phase 2: Data Processing (ETL Pipeline)

2.1 Prepare Your Data

Place your HR Excel file in `data/input/` directory. Expected format:

- Multiple sheets corresponding to different data aspects
- Each sheet should have an `employee_id` column for joining

2.2 Run ETL Pipeline

```
bash  
  
python src/processing/etl_pipeline.py
```

Or use the CLI:

```
bash  
  
python -m src.cli.interface etl run data/input/hr_data.xlsx
```

2.3 Verify Data Load

```
bash  
  
python inspect_mongodb.py
```

Phase 3: Search Index Setup

3.1 Create Search Index

```
bash
```

```
python scripts/create_search_indexes.py --action create --index hr-employees-fixed
```

3.2 Index Employee Data

```
bash
```

```
python src/search/indexer.py
```

3.3 Generate Embeddings (Optional)

```
bash
```

```
python src/search/embeddings.py
```

3.4 Verify Search Index

```
bash
```

```
python scripts/create_search_indexes.py --action validate --index hr-employees-fixed
```

Phase 4: Testing & Validation

4.1 Test Query Engine

```
bash
```

```
python enhanced_query_engine_dry_run.py
```

4.2 CLI Testing

```
bash
```

```
python -m src.cli.interface query test
```

4.3 Interactive Testing

```
bash
```

```
python -m src.cli.interface query interactive
```

Example queries to test:

- "How many employees work in IT?"
- "Find developers with AWS certification"
- "Show me employees in the Sales department"
- "Who are the top performers?"

Phase 5: API Server

5.1 Start API Server

```
bash  
  
python src/api/main.py
```

Or:

```
bash  
  
uvicorn src.api.main:app --host 0.0.0.0 --port 8000 --reload
```

5.2 Access API Documentation

- Interactive API docs: <http://localhost:8000/docs>
- Alternative docs: <http://localhost:8000/redoc>
- Health check: <http://localhost:8000/health>

5.3 Test API Endpoints

```
bash  
  
curl -X POST "http://localhost:8000/query" \  
  -H "Content-Type: application/json" \  
  -d '{"query": "Find developers with Python skills", "top_k": 5}'
```

Management Commands

CLI Commands Overview

```
bash
```

```
# ETL Operations
```

```
python -m src.cli.interface etl run data/input/hr_data.xlsx
```

```
python -m src.cli.interface etl status
```

```
# Query Operations
```

```
python -m src.cli.interface query ask "Find developers"
```

```
python -m src.cli.interface query interactive
```

```
python -m src.cli.interface query test
```

```
# Data Operations
```

```
python -m src.cli.interface data stats
```

```
python -m src.cli.interface data employees --limit 10
```

```
python -m src.cli.interface data profile EMP0001
```

```
# Search Operations
```

```
python -m src.cli.interface search status
```

```
python -m src.cli.interface search reindex
```

```
python -m src.cli.interface search test "developer"
```

```
# Configuration
```

```
python -m src.cli.interface config show
```

```
python -m src.cli.interface config validate
```

Performance Testing

```
bash
```

```
python scripts/performance_test.py --test full_suite
```

```
python scripts/performance_test.py --test query_speed --iterations 100
```

```
python scripts/performance_test.py --test load_test --concurrent 10
```

Data Schema

The system handles 10 collections with comprehensive employee data:

Core Collections:

1. **personal_info**: Basic details (name, email, location, age, gender)
2. **employment**: Job details (department, role, work_mode, joining_date)
3. **learning**: Skills (certifications, courses, training hours)
4. **experience**: Background (total experience, company tenure, skills count)

5. **performance:** Ratings (performance rating, KPIs, awards)

Extended Collections:

6. **engagement:** Projects (current project, engagement score, feedback)

7. **compensation:** Financials (salary, bonus, total CTC) - *sensitive*

8. **attendance:** Leave data (leave balance, attendance %, patterns)

9. **attrition:** Risk factors (attrition risk, exit intent, retention plans)

10. **project_history:** Past projects and contributions

Query Capabilities

The system supports various query types:

Simple Queries:

- "How many employees work in IT?"
- "Find developers"
- "Show me managers in Sales"

Complex Analytics:

- "Compare average performance between IT and Sales"
- "Top 10 highest performers"
- "Employees with high attrition risk"
- "Who needs training based on low learning hours?"

Advanced Filtering:

- "Remote employees with AWS certification and 5+ years experience"
- "High performers with low engagement scores"
- "Employees in IT with PMP certification"

Troubleshooting

Common Issues:

1. MongoDB Connection Failed

- Check connection string in .env file
- Verify network access to MongoDB Atlas

- Ensure correct database name

2. Azure Search Not Working

- Verify Azure Search service is running
- Check API key permissions
- Ensure index exists: `python scripts/create_search_indexes.py --action list`

3. OpenAI API Errors

- Verify API key and endpoint
- Check deployment names match configuration
- Ensure sufficient quota

4. No Query Results

- Verify data was loaded: `python -m src.cli.interface data stats`
- Check search index: `python -m src.cli.interface search status`
- Try reindexing: `python -m src.cli.interface search reindex`

Debug Commands:

```
bash

# Check system status
python -m src.cli.interface system-info

# Validate configuration
python -m src.cli.interface config validate

# Test connections
python src/core/config.py
python inspect_mongodb.py
```



Performance Optimization

For Large Datasets:

1. **Batch Processing:** ETL pipeline processes in batches
2. **Indexed Searches:** MongoDB and Azure Search indexes
3. **Caching:** Query result caching (configurable)
4. **Parallel Processing:** Multi-threaded ETL operations

Monitoring:


```
bash
```

```
# Performance testing
```

```
python scripts/performance_test.py --test full_suite
```

```
# Resource monitoring
```







```
python -m src.cli.interface system-info --format json
```

Security Considerations

1. **Sensitive Data:** Compensation data marked as sensitive
2. **API Security:** Authentication can be enabled in production
3. **Environment Variables:** All credentials in .env file
4. **Data Privacy:** GDPR compliance features available

Success Validation

After setup, you should be able to:

1.  Load HR data from Excel → MongoDB (10 collections)
2.  Query using natural language via CLI or API
3.  Get intelligent responses for various employee questions
4.  Access REST API at <http://localhost:8000/docs>
5.  Use advanced search capabilities (semantic, vector, hybrid)
6.  Generate analytics and insights

Support

If you encounter issues:

1. Check logs in `logs/` directory
2. Use debug mode: `python -m src.cli.interface --debug`
3. Validate each component step by step
4. Check the troubleshooting section above

The system is designed to be robust with multiple fallback mechanisms and comprehensive error handling.