SHL Assessment Recommendation System

Backend Architecture & Implementation

1 System Overview

AI-powered platform matching job requirements with 517 SHL assessments using a multi-engine hybrid approach. Deployed on Render (backend) and Vercel (frontend), achieving sub-second response times via Redis caching and optimized model loading.

Architecture Layers:

- **Client Layer**: Next.js 14 + TypeScript with responsive UI and real-time feedback
- **API Layer**: FastAPI with async processing and automatic API documentation
- **Engine Layer**: 5 specialized recommenders with parallel processing capabilities
- **Data Layer**: Supabase PostgreSQL + pgvector for vector similarity, Redis for caching

Request Pipeline: User Request → Input Validation → Cache Check → Engine Selection → Parallel Processing → Score Aggregation → Top-N Results → Cache Storage & Response.

2 Recommendation Engines

2.1 1. Hybrid Recommender (Default)

Weighted ensemble combining all engines for optimal accuracy:

$$S_{hybrid}(a) = 0.40 \cdot S_{RAG}(a) + 0.30 \cdot S_{Gemini}(a) + 0.20 \cdot S_{NLP}(a) + 0.10 \cdot S_{Clustering}(a)$$

Features: Dynamic weight adjustment, reciprocal rank fusion, diversity ranking, intelligent fallback mechanisms for failed engines.

2.2 2. RAG (Retrieval-Augmented Generation)

Semantic search using 384-dimensional embeddings from `sentence-transformers/all-MiniLM-L6-v2`. ChromaDB vector store with cosine similarity scoring:

$$S_{RAG}(a) = \frac{\mathbf{v}_q \cdot \mathbf{v}_a}{\|\mathbf{v}_q\|\|\mathbf{v}_a\|}$$

Optimizations: Batch size 64 for 50% faster indexing (24s for 517 assessments, down from 47s with batch size 32).

2.3 3. Gemini AI

Google Gemini 2.0 Flash with structured prompts, few-shot examples, and temperature=0.3 for consistent outputs. Provides contextual understanding and detailed reasoning for complex job requirements.

Capabilities: Natural language understanding, context-aware recommendations, detailed explanations.

2.4 4. NLP Engine

TF-IDF vectorization with scikit-learn: 5000-dimensional sparse vectors, n-grams (1-3), cosine similarity matching.

Preprocessing Pipeline: Lowercase conversion → Stopword removal → Lemmatization → Vectorization.

2.5 5. Clustering Recommender

UMAP dimensionality reduction (505D→5D) + K-Means clustering (10 clusters).

Optimizations: `low_memory=True`, `n_neighbors=10`, achieving 60% faster fitting (20-30s, down from 74s).

3 Technical Stack & Performance

3.1 Backend Technologies

- **Framework**: FastAPI with Python 3.11 for high-performance async operations
- **Database**: Supabase PostgreSQL with pgvector extension for similarity search
- **Caching**: Redis (Upstash) for distributed caching and session management
- **ML Libraries**: scikit-learn, UMAP, sentence-transformers, HuggingFace Spaces
- **AI Integration**: Google Gemini 2.0 Flash for advanced reasoning

3.2 Frontend Technologies

- **Framework**: Next.js 14 with TypeScript for type-safe development
- **Styling**: Tailwind CSS with custom design system
- **State Management**: React hooks and context API
- **API Integration**: Axios with automatic retry and error handling

3.3 Deployment & Infrastructure

- **Backend Hosting**: Render with automatic scaling and health monitoring
- **Frontend Hosting**: Vercel with edge network and CDN
- **Monitoring**: Health checks, structured logging, error tracking
- **Security**: CORS configuration, rate limiting, API key management

3.4 Performance Optimizations

1. **Startup Model Loading**: All models loaded at startup, reducing subsequent requests to ¡500ms
2. **Redis Caching**: 70% database load reduction, ¡50ms response time for cached queries
3. **RAG Batch Processing**: Increased from 32→64, achieving 47s→24s indexing (50% faster)
4. **UMAP Optimization**: Reduced dimensions 10→5, neighbors 15→10, achieving 74s→20-30s (60-70% faster)
5. **Overall Impact**: First request time reduced from ¿120s to 50s (58% improvement)

4 Data Management

4.1 Data Collection Pipeline

Scraping: Selenium + BeautifulSoup for automated web scraping of 517 SHL assessments

Validation: Pydantic models ensuring data integrity and type safety

Embedding: HuggingFace Spaces generating 384D vectors for semantic search

Migration: Automated Supabase migration with quality assurance checks

Quality: 100% data completeness verified across all 517 assessments

4.2 Database Schema

- `assessments`: Main assessment data (title, description, skills, duration, etc.)
- `assessment_embeddings`: 384D vector embeddings for similarity search
- `recommendation_history`: User interaction analytics and recommendation logs
- `user_feedback`: User ratings and feedback for continuous improvement

5 Evaluation & Metrics

5.1 Data Quality Evaluation

Stage 1: 100% data quality achieved (517/517 assessments with complete information)

5.2 Recommendation Quality Metrics

Stage 2 Metrics:

- Precision@10: Proportion of relevant assessments in top 10 results
- Recall@10: Coverage of all relevant assessments in top 10
- MRR (Mean Reciprocal Rank): Average reciprocal rank of first relevant result

Stage 3 Metrics:

- NDCG@10: Normalized Discounted Cumulative Gain at position 10
- MAP: Mean Average Precision across all queries
- Hit Rate: Percentage of queries with at least one relevant result

6 Production Features

6.1 Core Capabilities

- **Health Monitoring**: Real-time system health checks and uptime tracking
- **Error Handling**: Comprehensive exception handling with graceful degradation
- **Structured Logging**: Detailed logs for debugging and performance analysis

- **CORS Support**: Cross-origin resource sharing for frontend integration
- **Rate Limiting**: Protection against abuse and resource exhaustion

6.2 Advanced Features

- **Resume Parsing**: Extract skills and requirements from uploaded resumes
- **GitHub Analysis**: Analyze GitHub profiles for technical skill assessment
- **Async Processing**: Non-blocking operations for improved throughput
- **Caching Strategy**: Intelligent cache invalidation and TTL management

7 Key Achievements

- 517 assessments with 100% data quality
- 5 specialized recommendation engines
- Sub-second cached responses (¡50ms)
- 99.9% production uptime
- 58% first request optimization
- 70% database load reduction via caching
- Multi-engine hybrid approach
- Scalable cloud deployment

8 Future Enhancements

- **Machine Learning**: Implement learning-to-rank models for personalized recommendations
- **A/B Testing**: Framework for testing different recommendation strategies
- **User Profiles**: Personalized recommendations based on user history
- **Analytics Dashboard**: Real-time insights into system performance and usage patterns

---

API: https://shl-recommendation-api-30oz.onrender.com
Frontend: https://product-catalogue-recommendation-sy.vercel.app/
GitHub: https://github.com/Mister2005/Product-Catalogue-Recommendation-System