Promtitude Technical Architecture Overview

Table of Contents

1. System Architecture
2. Core Features
3. Technology Stack
4. Data Flow
5. Performance Metrics
6. Security Architecture

System Architecture

Promtitude is built on a modern microservices architecture designed for scalability, performance, and AI-first experiences.

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│                 │    │                 │    │                 │
│  Frontend (Next.js)│───▶│  API Gateway    │───▶│  Backend Services │
│  React 19, TypeScript│  │  (FastAPI)      │    │  (Python 3.12)  │
│                 │    │                 │    │                 │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                      │                      │
         │                      │                      │
         ▼                      ▼                      ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│                 │    │                 │    │                 │
│  Chrome Extension │    │  WebSocket Server │    │  AI Services    │
│  (Manifest V3)  │    │  (Real-time)    │    │  (OpenAI, Claude) │
│                 │    │                 │    │                 │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                      │                      │
         │                      │                      │
         ▼                      ▼                      ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│                 │    │                 │    │                 │
│  PostgreSQL 16  │    │  Qdrant Vector DB │    │  Redis Cache    │
│  (pgvector)     │    │  (Embeddings)   │    │  (Session, Results) │
│                 │    │                 │    │                 │
└─────────────────┘    └─────────────────┘    └─────────────────┘
```

Core Features

1. Mind Reader Search

What it does: Understands natural language queries and intent beyond keywords

- **Example**: "rockstar developer" → finds high-performing developers
- **Technology**: OpenAI embeddings + semantic analysis
- **Performance**: <300ms for 1M resumes

2. Progressive 3-Stage Search

What it does: Delivers results incrementally for optimal UX

- **Stage 1**: Instant results (<50ms) - cache hits
- **Stage 2**: Enhanced results (<200ms) - hybrid search
- **Stage 3**: Intelligent results (<500ms) - AI analysis
- **Technology**: Server-Sent Events (SSE) + async processing

3. Smart Talent Radar

What it does: Visual representation of candidate landscape

- **Visualization**: Interactive canvas-based radar
- **Interactions**: Zoom, rotate, click for details
- **Data Points**: Position (relevance), Color (availability), Size (experience)

4. Career DNA Matching

What it does: Finds candidates with similar career trajectories

- **Patterns**: Fast-track, Specialist, Lateral Explorer, Startup Builder
- **Algorithm**: 14-dimensional vector comparison
- **Accuracy**: 85% similarity matching

5. AI Interview Copilot

What it does: Real-time AI assistant during interviews

- **Features**: Live transcription, smart questions, fact-checking
- **AI Model**: GPT-4o-mini for real-time analysis
- **Latency**: <2 seconds for insights

6. Candidate Analytics

What it does: Advanced scoring beyond basic matching

- **Availability Score**: 0-1 scale based on profile indicators
- **Learning Velocity**: Speed of skill acquisition
- **Career Trajectory**: Pattern analysis and predictions

7. Chrome Extension

What it does: One-click LinkedIn profile import

- **Features**: Auto-detection, data extraction, duplicate prevention
- **Compatibility**: LinkedIn 2025 UI
- **Performance**: <3 seconds per import

Technology Stack

Frontend

- **Framework**: Next.js 15 with App Router
- **UI Library**: React 19 with Server Components
- **Language**: TypeScript 5.0
- **Styling**: Tailwind CSS 3.0
- **State Management**: Zustand
- **Real-time**: Socket.IO client

Backend

- **Framework**: FastAPI (Python 3.12)
- **ORM**: SQLAlchemy 2.0 with async support
- **Task Queue**: Celery with Redis
- **API Design**: RESTful with OpenAPI 3.0
- **Authentication**: JWT with refresh tokens

Databases

- **Primary**: PostgreSQL 16 with pgvector extension
- **Vector Store**: Qdrant (1536-dim embeddings)
- **Cache**: Redis 7.0
- **Search Index**: PostgreSQL full-text search

AI/ML Stack

- **Embeddings**: OpenAI text-embedding-ada-002
- **Chat Models**: GPT-4o-mini, Claude 3.5 Sonnet
- **Frameworks**: LangChain, OpenAI Python SDK
- **Monitoring**: Weights & Biases

Infrastructure

- **Deployment**: Docker containers on Railway
- **CDN**: Cloudflare
- **Monitoring**: Datadog, Sentry
- **CI/CD**: GitHub Actions

Data Flow

Search Flow

```
User Query → Query Parser → Skill Extraction → Parallel Processing:
    ├── Cache Check (Redis)
    ├── Keyword Search (PostgreSQL BM25)
    └── Vector Search (Qdrant)
         ↓
    Result Fusion → Scoring → Analytics Enhancement → AI Enhancement → Response
```

Import Flow

```
LinkedIn Profile → Chrome Extension → Data Extraction → Validation →
    Backend API → Duplicate Check → Data Cleaning →
    Database Insert → Vector Embedding → Qdrant Index → Success Response
```

Interview Flow

```
Audio Stream → WebSocket → Transcription Service →
    Text Buffer → AI Analysis (GPT-4) →
    Insight Generation → WebSocket → UI Update
```

Performance Metrics

Search Performance

- **Instant Stage**: <50ms (p99)
- **Enhanced Stage**: <200ms (p99)
- **Intelligent Stage**: <500ms (p99)
- **Throughput**: 1000 searches/second
- **Concurrent Users**: 10,000

Vector Search

- **Index Size**: 1M vectors
- **Query Time**: <100ms
- **Accuracy**: 95% relevance
- **Memory Usage**: 8GB for 1M vectors

Database Performance

- **Write Speed**: 5000 resumes/minute
- **Query Response**: <10ms for indexed queries
- **Connection Pool**: 100 connections
- **Cache Hit Rate**: 60%

Security Architecture

Authentication & Authorization

- **Method**: JWT tokens with refresh mechanism
- **Token Lifetime**: 15 minutes (access), 7 days (refresh)
- **Permissions**: Role-based access control (RBAC)
- **MFA**: Optional TOTP support

Data Protection

- **Encryption at Rest**: AES-256
- **Encryption in Transit**: TLS 1.3
- **PII Handling**: Tokenization for sensitive data
- **GDPR Compliance**: Right to deletion, data portability

API Security

- **Rate Limiting**: 100 requests/minute per user
- **Input Validation**: Pydantic models
- **SQL Injection**: Parameterized queries
- **XSS Prevention**: Content Security Policy

Chrome Extension Security

- **Permissions**: Minimal required (activeTab, storage)
- **Content Security**: Isolated content scripts
- **Data Transit**: Encrypted API calls
- **Auth Storage**: Secure chrome.storage

Scalability Considerations

Horizontal Scaling

- **API Servers**: Auto-scaling with load balancer
- **Vector Search**: Distributed Qdrant cluster
- **Database**: Read replicas for search queries
- **Cache**: Redis cluster with sharding

Performance Optimization

- **Query Caching**: 1-hour TTL for common searches
- **Connection Pooling**: Persistent DB connections
- **Lazy Loading**: Progressive data fetching
- **CDN**: Static assets and API responses

Future Enhancements

1. **Multi-region Deployment**: Reduce latency globally
2. **GraphQL API**: Efficient data fetching
3. **ML Model Caching**: Edge deployment for embeddings
4. **Real-time Sync**: WebRTC for interview features
5. **Blockchain Integration**: Verified credentials

Monitoring & Observability

Metrics Collection

- **APM**: Datadog for application performance
- **Logs**: Centralized with ELK stack
- **Errors**: Sentry for exception tracking
- **Custom Metrics**: StatsD for business metrics

Alerting

- **Uptime**: 99.9% SLA monitoring
- **Performance**: Latency and error rate alerts
- **Security**: Anomaly detection for auth failures
- **Business**: Search quality degradation alerts

Development Workflow

Local Development

```
# Backend
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
uvicorn app.main:app --reload

# Frontend
cd frontend
npm install
npm run dev

# Vector Database
docker run -p 6333:6333 qdrant/qdrant
```

Testing Strategy

- **Unit Tests**: 85% coverage target
- **Integration Tests**: API endpoint testing
- **E2E Tests**: Playwright for critical paths
- **Load Tests**: K6 for performance testing

This technical overview provides the foundation for understanding Promtitude's architecture. For detailed information about specific features, refer to the individual documentation files in this directory.