DataMigrate AI

Complete System Architecture

Technical Documentation

Version 2.0 - With Model Router & Fine-Tuning

Company:	OKO Investments
Author:	Alexander Garcia Angus
Date:	December 2025
Version:	2.0

Table of Contents

1. System Overview
2. High-Level Architecture
3. Frontend Layer (Vue.js 3)
4. Backend API Layer (Go)
5. AI Agents Layer (Python + LangGraph)
6. Model Router & AI Provider System
7. Fine-Tuning Data Pipeline
8. Guardian Agent Security
9. Database Layer (PostgreSQL)
10. Deployment Architecture
11. Data Flow Diagrams
12. Technology Stack Summary

1. System Overview

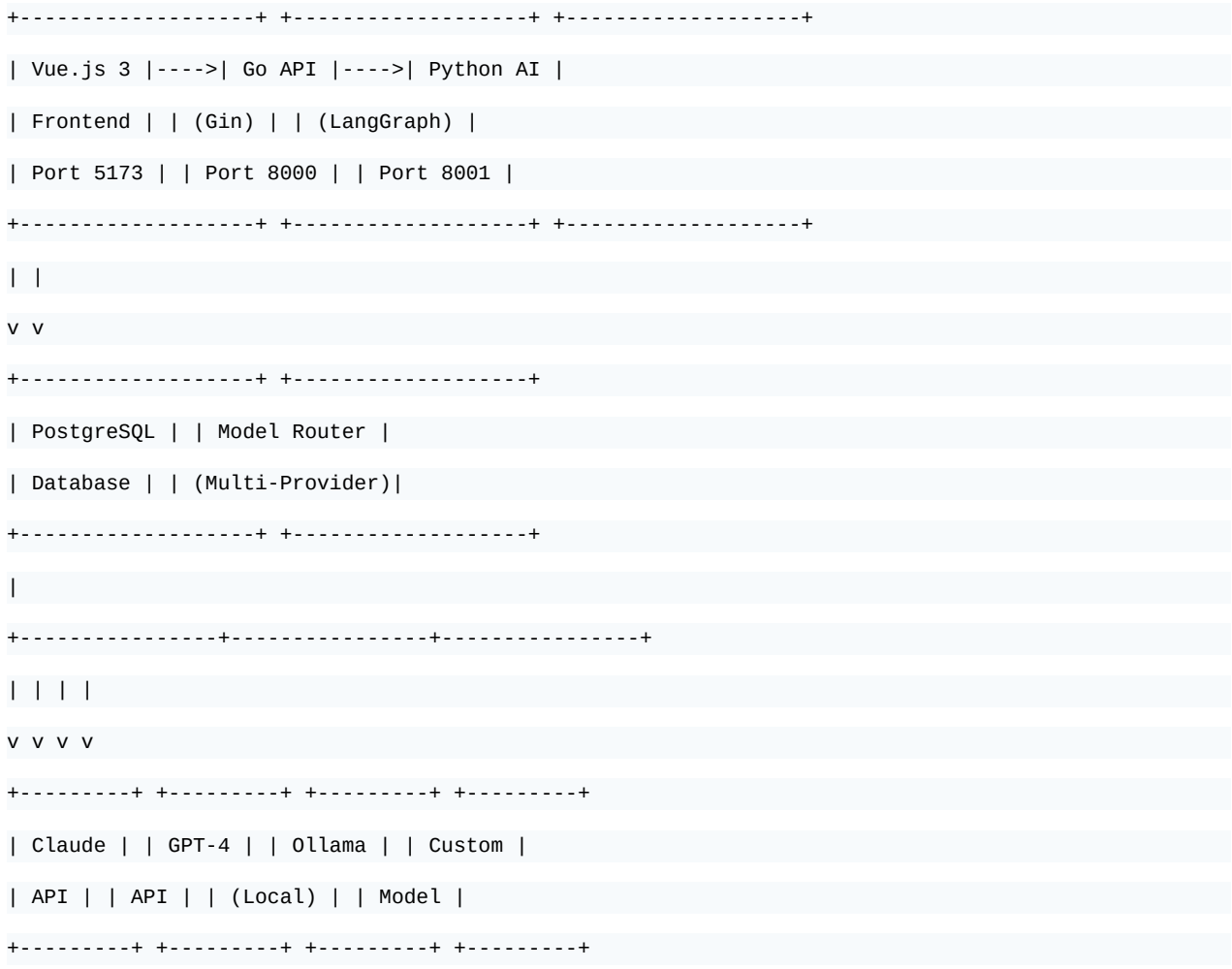
DataMigrate AI is an enterprise-grade platform that automates the migration of Microsoft SQL Server databases to dbt (data build tool). The system uses a multi-agent AI architecture orchestrated by LangGraph, with a Go backend for high-performance API handling and a Vue.js 3 frontend for user interaction.

Core Components:

- Frontend: Vue.js 3 + TypeScript + Pinia (Port 5173)
- Backend API: Go + Gin Framework (Port 8000)
- AI Service: Python + LangGraph + Model Router (Port 8001)
- Database: PostgreSQL 16
- Security: Guardian Agent (Python + Go implementations)
- AI Providers: Claude, GPT, Ollama, Custom Fine-tuned Models

2. High-Level Architecture

The system follows a microservices architecture with clear separation of concerns:



3. Frontend Layer (Vue.js 3)

The frontend provides a modern, responsive user interface built with Vue.js 3 and TypeScript.

Component	Technology	Purpose
Framework	Vue.js 3	Reactive UI with Composition API
Language	TypeScript	Type-safe development
State Management	Pinia	Centralized state stores
Routing	Vue Router 4	Client-side navigation
HTTP Client	Axios	API communication
Styling	Tailwind CSS	Utility-first CSS framework
Build Tool	Vite	Fast development server and bundler

Directory Structure:

- frontend/src/api/ - Axios API client and endpoints
- frontend/src/components/ - Reusable Vue components
- frontend/src/views/ - Page-level components
- frontend/src/stores/ - Pinia state management
- frontend/src/router/ - Vue Router configuration
- frontend/src/types/ - TypeScript interfaces

4. Backend API Layer (Go)

The Go backend provides a high-performance REST API using the Gin framework. It handles 95% of all requests (authentication, CRUD operations, data retrieval) while delegating AI-intensive tasks to the Python service.

Component	Technology	Purpose
Framework	Gin	High-performance HTTP router
ORM	GORM	PostgreSQL database access
Auth	JWT-Go	Token-based authentication
Security	Guardian Agent	Request validation and rate limiting
Config	Viper	Configuration management
Logging	Zap	Structured logging

API Endpoints:

- POST /api/v1/auth/register - User registration
- POST /api/v1/auth/login - User login (returns JWT)
- GET /api/v1/migrations - List user's migrations
- POST /api/v1/migrations - Create new migration (triggers AI)
- GET /api/v1/migrations/:id - Get migration details
- GET /api/v1/security/dashboard - Security metrics (admin)
- GET /api/v1/security/audit-logs - Audit trail (admin)

5. AI Agents Layer (Python + LangGraph)

The AI layer uses LangGraph to orchestrate a multi-agent workflow for database migration. Each agent specializes in a specific aspect of the migration process.

Agent	Responsibility	Output
Assessment Agent	Analyzes MSSQL schema complexity	Complexity score, risk factors
Planner Agent	Creates migration strategy	Ordered list of models to create
Executor Agent	Generates dbt model SQL	dbt .sql files
Tester Agent	Validates generated models	Test results, errors found
Rebuilder Agent	Fixes failed models	Corrected SQL
Optimizer Agent	Improves model performance	Optimized SQL, indexes
Guardian Agent	Security validation	Threat detection, audit logs

LangGraph Workflow:

1. START -> Assessment Agent (analyze schema)
2. Assessment -> Planner Agent (create migration plan)
3. Planner -> Executor Agent (generate dbt models)
4. Executor -> Tester Agent (validate models)
5. Tester -> Rebuilder Agent (if errors) OR Optimizer Agent (if success)
6. Rebuilder -> Tester Agent (retry validation)
7. Optimizer -> END (migration complete)

6. Model Router & AI Provider System

The Model Router provides a unified interface for multiple AI providers, enabling cost optimization, fallback chains, and future fine-tuning support. This is a key competitive advantage.

Supported Providers:

Provider	Models	Use Case	Cost
Anthropic	Claude Opus, Sonnet, Haiku	Complex reasoning, critical tasks	$$$
OpenAI	GPT-4o, GPT-4o-mini	General purpose, JSON generation	$$
Ollama	Llama 3, Mistral, CodeLlama	Simple tasks, on-premise	Free
Custom/vLLM	Fine-tuned models	Domain-specific SQL generation	Free

Smart Routing by Task Complexity:

Complexity	Default Model	Example Tasks
SIMPLE	Claude Haiku (cheap)	Basic SELECT queries, simple transformations
MEDIUM	GPT-4o-mini (balanced)	Standard migrations, moderate logic
COMPLEX	Claude Sonnet (quality)	Stored procedures, business logic
CRITICAL	Claude Opus (best)	Production migrations, complex validation

Fallback Chains:

If a model fails or is unavailable, the router automatically tries the next model in the chain:

- Claude Opus -> Claude Sonnet -> GPT-4o
- Claude Sonnet -> GPT-4o -> Claude Haiku
- Claude Haiku -> GPT-4o-mini -> Llama-3-8b
- Custom Model -> Claude Sonnet -> GPT-4o

7. Fine-Tuning Data Pipeline

The platform automatically collects high-quality migration examples for future model fine-tuning. This creates a competitive moat that grows stronger with each customer.

Data Collection Process:

1. Migration Execution - Agents process customer schema
2. Quality Scoring - Each output is scored (0-1) based on success
3. Filtering - Only high-quality examples (score > 0.8) are saved
4. Storage - Input/output pairs stored in JSONL format
5. Anonymization - Customer-specific data is removed/masked

Training Data Structure:

```
{
"messages": [
{"role": "system", "content": "You are a SQL migration expert..."},
{"role": "user", "content": "Convert this MSSQL: CREATE PROCEDURE..."},
{"role": "assistant", "content": "SELECT customer_id FROM..."}
]
}
```

ML Engineer Workflow:

1. Export Data - collector.export_for_training('jsonl')
2. Choose Base Model - Llama 3, Mistral, or SQLCoder
3. Fine-tune - Using LoRA or full fine-tuning
4. Evaluate - Test on held-out migration examples
5. Deploy - Via Ollama or vLLM
6. Register - router.register_custom_model('datamigrate-v1', config)

8. Guardian Agent Security

Enterprise-grade security layer that protects all AI operations from malicious input and abuse.

Feature	Implementation	Protection
Prompt Injection	25+ regex patterns	Blocks AI manipulation attempts
SQL Injection	OWASP patterns	Prevents malicious SQL generation
XSS Protection	HTML/JS detection	Blocks script injection
Rate Limiting	Sliding window	Prevents abuse (100 req/min)
Audit Logging	Full event capture	SOC 2/GDPR compliance
Multi-Tenant	Per-org policies	Isolated security rules

Dual Implementation:

Guardian Agent is implemented in both Python (for AI agents) and Go (for API middleware):

- Python: @protected_agent decorator wraps agent functions
- Go: Gin middleware validates all incoming API requests
- Both: Share same detection patterns and rate limit rules
- Both: Write to same PostgreSQL audit_logs table

9. Database Layer (PostgreSQL)

PostgreSQL 16 serves as the central data store for users, migrations, and security audit logs.

Table	Purpose	Key Fields
users	User accounts	id, email, password_hash, org_id
organizations	Multi-tenant orgs	id, name, plan_type
api_keys	API authentication	id, key_hash, user_id, scopes
migrations	Migration projects	id, user_id, status, config
migration_models	Generated dbt models	id, migration_id, sql, status
audit_logs	Security events	id, event_type, user_id, details
fine_tuning_data	Training examples	id, task_type, input, output, score

10. Deployment Architecture

Development Environment:

- Frontend: npm run dev (Vite, Port 5173)
- Backend: go run cmd/server/main.go (Port 8000)
- AI Service: uvicorn ai_service:app (Port 8001)
- Database: PostgreSQL local or Docker (Port 5432)

Production Environment (AWS):

- Frontend: S3 + CloudFront (static hosting)
- Backend: ECS Fargate (Go container)
- AI Service: ECS Fargate (Python container)
- Database: RDS PostgreSQL (Multi-AZ)
- Load Balancer: Application Load Balancer
- CDN: CloudFront for global distribution
- Secrets: AWS Secrets Manager
- Monitoring: CloudWatch + X-Ray

On-Premise Option:

- Docker Compose for all services
- Ollama for local AI inference (no API calls)
- PostgreSQL container
- Nginx reverse proxy
- No internet dependency for migrations

11. Data Flow Diagrams

User Login Flow:

1. User -> Frontend: Enter credentials
2. Frontend -> Go API: POST /auth/login
3. Go API -> PostgreSQL: Verify credentials
4. Go API -> Frontend: Return JWT token
5. Frontend: Store token in localStorage

Migration Creation Flow:

1. User -> Frontend: Submit migration config
2. Frontend -> Go API: POST /migrations (with JWT)
3. Go API -> PostgreSQL: Create migration record
4. Go API -> User: Return migration ID (30ms)
5. Go API -> Python AI: Async call to start agents
6. Python AI -> Model Router: Select appropriate model
7. Model Router -> AI Provider: Generate SQL
8. Python AI -> PostgreSQL: Save generated models
9. User -> Frontend: Poll for status updates

12. Technology Stack Summary

Layer	Technology	Version
Frontend	Vue.js + TypeScript	3.4+
State Management	Pinia	2.0+
Styling	Tailwind CSS	3.0+
Backend API	Go + Gin	1.21+
ORM	GORM	1.25+
AI Orchestration	LangGraph	0.0.40+
AI Framework	LangChain	0.1+
AI Providers	Claude, GPT, Ollama	Latest
Database	PostgreSQL	16
Security	Guardian Agent	Custom
Infrastructure	AWS CDK	2.0+
Container	Docker	24+