

AiSO Food Corp — AI Operations Center

Enterprise Migration & Setup Manual

Version: 1.0 · **Date:** June 2026

Purpose: Everything your engineering team needs to stand this application up in your own infrastructure.

Table of Contents

1. [Architecture Overview](#)
 2. [Prerequisites](#)
 3. [Project Structure](#)
 4. [Environment Setup](#)
 5. [Database Setup](#)
 6. [LLM API Configuration](#)
 7. [Local Development](#)
 8. [Production Deployment](#)
 9. [Data Seeding](#)
 10. [Application Pages & Modules](#)
 11. [API Route Reference](#)
 12. [Key Libraries & Engines](#)
 13. [MCP Gateway — Integration Points](#)
 14. [Prisma Schema Overview](#)
 15. [Customization Guide](#)
 16. [LLM Provider Swap Guide](#)
 17. [Testing](#)
 18. [Documents Included](#)
 19. [Troubleshooting](#)
 20. [Security Considerations](#)
-

1. Architecture Overview

The AI Operations Center is a **Next.js 14** full-stack application with:

Browser (React 18 + Tailwind CSS + Recharts)

Next.js App Router (API Routes + SSR)

- └─ 61 API routes (87 HTTP methods)
- └─ 11 pages
- └─ 49 components

Business Logic Layer

- └─ Agent Swarm Orchestrators (MVP1-5)
- └─ Validity Engines (Trade Deal + Invoice)
- └─ MCP Gateway (6 stub integrations)
- └─ Policy Engines (Buyer, Approval)
- └─ Closed-**Loop** Learning Pipeline

PostgreSQL 14+ (Prisma ORM, 68 models)

LLM API (OpenAI-compatible endpoint)

Key design decisions:

- All agent reasoning is deterministic where possible (hash-based, not Math.random)
- Human-in-the-loop (HITL) dual-lane approval for high-value decisions
- SOX-compliant audit logging
- Every agent override feeds the closed-loop learning pipeline
- MCP Gateway stubs are designed for 1:1 replacement with real ERP adapters

2. Prerequisites

Requirement	Version	Notes
Node.js	20.x LTS or 22.x	Required for Next.js 14
Yarn	4.x (Berry)	Package manager (do NOT use npm)
PostgreSQL	14+	Any provider: RDS, Azure, Supabase, local
LLM API	Any OpenAI-compatible	See Section 6 for options
Git	2.x+	For version control

Hardware (Production):

- 2+ vCPUs, 4GB+ RAM for the Node.js server
- PostgreSQL with at least 1GB storage (grows with data)

3. Project Structure

aiso-food-corp-source-complete.tar.gz

```

└─ nextjs_space/
└─ app/
  └─ page.tsx
  └─ collection/page.tsx
  └─ invoices/page.tsx
  └─ buyer/page.tsx
  └─ demand/page.tsx
  └─ executive/page.tsx
  └─ agreement/page.tsx
  └─ escalations/page.tsx
  └─ my-work/page.tsx
  └─ workbench/page.tsx
  └─ reference/page.tsx
  └─ deduction/[id]/page.tsx
  └─ layout.tsx
  └─ globals.css
  └─ api/
    └─ deductions/
    └─ collections/
    └─ invoice-triage/
    └─ demand-sensing/
    └─ executive/
    └─ agent-swarm/
    └─ hitl-queue/
    └─ value-ledger/
    └─ writeback/
    └─ llm-chat/
    └─ search/
    └─ ... (see Section 11)
  └─ components/
    └─ ui/
    └─ layouts/
    └─ header.tsx
    └─ sidebar.tsx
    └─ executive-dashboard.tsx
    └─ roi-calculator.tsx
    └─ technical-reference.tsx
    └─ intake-dashboard.tsx
    └─ collection-dashboard.tsx
    └─ invoice-triage-dashboard.tsx
    └─ buyer-copilot.tsx
    └─ demand-sensing-dashboard.tsx
    └─ ... (49 total)
  └─ lib/
    └─ agent-swarm.ts
    └─ agent-swarm-mvp3.ts
    └─ agent-swarm-mvp4.ts
    └─ agent-swarm-mvp5.ts
    └─ mvp1-orchestrator.ts
    └─ mvp2-orchestrator.ts
    └─ mvp3-orchestrator.ts
    └─ mvp4-orchestrator.ts
    └─ mcp-gateway.ts
    └─ trade-deal-validator.ts
    └─ invoice-validity-engine.ts
    └─ remittance-parser.ts
    └─ reason-code-dictionary.ts
    └─ approval-policy.ts
    └─ buyer-policy-engine.ts
  └─ The Next.js application
  └─ App Router pages + API routes
  └─ Home (MVP1 Trade Deductions)
  └─ MVP2 Collections
  └─ MVP3 Invoice Triage
  └─ MVP4 Procurement Copilot
  └─ MVP5 Demand Sensing
  └─ Executive Dashboard + ROI Calculator
  └─ Agreement Monitoring
  └─ Escalation Inbox
  └─ Unified Worklist
  └─ Agent Workbench
  └─ Technical Reference
  └─ Deduction Detail
  └─ Root layout
  └─ Global styles + animations
  └─ 61 API route directories
  └─ React components
  └─ Shadcn/Radix primitives
  └─ Shell layouts
  └─ CIO ROI Calculator
  └─ Technical docs
  └─ MVP1
  └─ MVP2
  └─ MVP3
  └─ MVP4
  └─ MVP5
  └─ Business logic
  └─ MVP1 orchestrator
  └─ MVP3 orchestrator
  └─ MVP4 orchestrator
  └─ MVP5 orchestrator
  └─ Full MVP1 pipeline
  └─ Full MVP2 pipeline
  └─ Full MVP3 pipeline
  └─ Full MVP4 pipeline
  └─ MCP stub integrations
  └─ 7-point validity engine
  └─ Invoice 3-way match
  └─ Multi-format parser
  └─ Customer reason codes
  └─ SOX approval policy
  └─ Procurement policies

```

			schemas.ts	# Shared types + constants		
			persona-config.ts	# Role -based personas		
			persona-context.tsx	# React context		
			prisma.ts	# Prisma singleton		
			utils.ts	# Utilities		
			prisma/			
					schema.prisma	# 68 models, 1919 lines
			scripts/			
				safe-seed.ts	# Database seeder	
				public/	# Static assets	
				hooks/	# React hooks	
				types/	# Type declarations	
				.env	# Environment template	
				package.json	# Dependencies	
				yarn.lock	# Lock file	
				next.config.js	# Next.js config	
				tailwind.config.ts	# Tailwind config	
				tsconfig.json	# TypeScript config	
				postcss.config.js		
			docs/	# CIO Handoff Documents		
				INTEGRATION_GAP_REGISTER.md/.docx/.pdf		
				DEPLOYMENT_GUIDE.md/.docx/.pdf		
				API_REFERENCE.md/.docx/.pdf		
				ARCHITECTURE_DECISIONS.md/.docx/.pdf		
				SECURITY_POSTURE.md/.docx/.pdf		
				GLOSSARY.md/.docx/.pdf		
				COMPLETE_ROADMAP.md/.docx/.pdf		
				BUILD_LOG.md/.docx/.pdf		
				DEMO_SCRIPT.md/.docx/.pdf		
				COMPLETE_80_SLIDE_OUTLINE.md/.docx/.pdf		
				SLIDE_WEAVE.md/.docx/.pdf		

4. Environment Setup

Step 1: Extract the archive

```
mkdir aiso-operations-center
cd aiso-operations-center
tar xzf aiso-food-corp-source-complete.tar.gz
cd nextjs_space
```

Step 2: Configure environment variables

Create `.env` in the `nextjs_space/` directory:

```
# PostgreSQL connection string
DATABASE_URL="postgresql://USER:PASSWORD@HOST:5432/DB_NAME?schema=public"

# LLM API Key (see Section 6 for options)
ABACUSAI_API_KEY="your-api-key-here"

# Optional: Override for production URL
# NEXTAUTH_URL="https://your-domain.com"
```

That's it — only 2 required env vars.

Step 3: Install dependencies

```
# Ensure you're using Yarn 4+
corepack enable
yarn set version 4.13.0

# Install
yarn install
```

Step 4: Generate Prisma client

```
yarn prisma generate
```

Step 5: Push schema to database

```
# This creates all 68 tables
yarn prisma db push
```

Step 6: Start development server

```
yarn dev
# → http://localhost:3000
```

5. Database Setup

Option A: Local PostgreSQL

```
# macOS
brew install postgresql@14
brew services start postgresql@14
createdb aiso_ops_center

# .env
DATABASE_URL="postgresql://localhost:5432/aiso_ops_center?schema=public"
```

Option B: Docker

```
docker run -d \
  --name aiso-postgres \
  -e POSTGRES_DB=aiso_ops_center \
  -e POSTGRES_USER=aiso \
  -e POSTGRES_PASSWORD=secure_password \
  -p 5432:5432 \
  postgres:14-alpine

# .env
DATABASE_URL="postgresql://aiso:secure_password@localhost:5432/aiso_ops_center?
schema=public"
```

Option C: Managed (AWS RDS, Azure, Supabase, etc.)

Use the connection string from your provider. Ensure:

- SSL mode is configured if required
- The database user has CREATE TABLE permissions
- Connection pooling is configured (PgBouncer recommended for production)

Schema Management

```
# Push schema changes (development)
yarn prisma db push

# Generate migration files (production)
yarn prisma migrate dev --name descriptive_name
yarn prisma migrate deploy # production

# View database in browser
yarn prisma studio
```

6. LLM API Configuration

The application uses an **OpenAI-compatible API** for:

- Deduction classification (MVP1)
- Buyer Copilot chat (MVP4)
- Demand narrative generation (MVP5)

Current Implementation

The LLM calls are in these files:

- `lib/agent-swarm.ts` — MVP1 classification
- `lib/agent-swarm-mvp3.ts` — MVP3 invoice classification
- `lib/agent-swarm-mvp4.ts` — MVP4 procurement
- `lib/agent-swarm-mvp5.ts` — MVP5 demand sensing
- `app/api/llm-chat/route.ts` — Buyer Copilot streaming chat

They all use this pattern:

```
const response = await fetch('https://llmfoundry.straive.com/openai/v1/chat/comple-
tions', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${process.env.ABACUSAI_API_KEY}`,
  },
  body: JSON.stringify({
    model: 'gpt-5.4-mini', // or your preferred model
    messages: [...],
    temperature: 0.2,
  }),
});
```

Swap to Your LLM Provider

See Section 16 for a complete provider swap guide. The short version:

1. **OpenAI Direct:** Change URL to `https://api.openai.com/v1/chat/completions`, set `ABACUS-
AI_API_KEY` to your OpenAI key
2. **Azure OpenAI:** Change URL to your Azure endpoint
3. **Anthropic Claude:** Use the OpenAI-compatible proxy or modify fetch calls
4. **Self-hosted (vLLM, Ollama):** Point to your local endpoint

7. Local Development

```
cd nextjs_space

# Start dev server with hot reload
yarn dev

# Type checking
yarn tsc --noEmit

# Production build (test locally)
yarn build
yarn start

# Prisma Studio (database browser)
yarn prisma studio
```

Seed Sample Data

Once the app is running, hit these endpoints to populate sample data:

```
# Core data (customers, SKUs, suppliers, deductions)
curl -X POST http://localhost:3000/api/seed-data

# Extended data (invoices, PRs, demand forecasts)
curl -X POST http://localhost:3000/api/seed-extra

# OR visit http://localhost:3000 – many APIs auto-seed on first GET
```

8. Production Deployment

Option A: Docker

```
FROM node:20-alpine AS base

# Install dependencies
FROM base AS deps
WORKDIR /app
COPY package.json yarn.lock .yarnrc.yml ./
COPY .yarn ./.yarn
RUN corepack enable && yarn install --immutable

# Build
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN yarn prisma generate
RUN NEXT_OUTPUT_MODE=standalone yarn build

# Production
FROM base AS runner
WORKDIR /app
ENV NODE_ENV=production
COPY --from=builder /app/.next/standalone ./
COPY --from=builder /app/.next/static ./next/static
COPY --from=builder /app/public ./public

EXPOSE 3000
CMD ["node", "server.js"]
```

```
docker build -t aiso-ops-center .
docker run -p 3000:3000 \
  -e DATABASE_URL="postgresql://..." \
  -e ABACUSAI_API_KEY="..." \
  aiso-ops-center
```

Option B: Vercel

```
# Install Vercel CLI
npm i -g vercel

# Deploy
cd nextjs_space
vercel

# Set env vars in Vercel dashboard
# DATABASE_URL, ABACUSAI_API_KEY
```

Option C: AWS / Azure / GCP

Use the Docker approach above with your container orchestration platform (ECS, AKS, Cloud Run, etc.).

9. Data Seeding

The application auto-seeds many tables on first API call. For manual control:

Endpoint	Method	What It Seeds
/api/seed-data	POST	Customers, SKUs, Suppliers, Trade Deals, Invoices, Deductions
/api/seed-extra	POST	Extended data across all MVPs
/api/boundary-conditions	GET	Auto-seeds boundary condition defaults
/api/executive	GET	Auto-seeds baseline metrics
/api/escalations	GET	Auto-seeds escalation tickets
/api/hitl-queue	GET	Auto-seeds HITL queue items
/api/value-ledger	GET	Auto-seeds value ledger entries
/api/collections	GET	Auto-bootstraps collection accounts from customers

10. Application Pages & Modules

Route	Module	MVP	Description
/	Trade Deduction Intake	MVP1	Deduction classification, validity checking, triage
/collection	Collections Prioritization	MVP2	AR aging, collector worklist, communication scheduling
/invoices	Invoice Exception Triage	MVP3	3-way match, OCR validation, auto-resolution
/buyer	Procurement Copilot	MVP4	LLM-powered buyer chat, PR management, policy compliance
/demand	Demand Sensing	MVP5	Forecast vs. actuals, SKU×DC heatmap, planner copilot
/executive	Executive Dashboard	All	KPIs, ROI calculator, autonomy trends, live activity feed
/agreement	Agreement Monitoring	All	Classification & validity agreement rates, drift detection
/escalations	Escalation Inbox	All	Cross-MVP critical issue management
/my-work	Unified Worklist	All	Proposals, HITL queue, escalations, value ledger
/workbench	Agent Workbench	All	Agent swarm control, run replay, boundary conditions
/reference	Technical Reference	All	Architecture docs, data dictionary, flow diagrams, test coverage
/deduction/[id]	Deduction Detail	MVP1	Full deduction life-cycle, dispute track-

Route	Module	MVP	Description
			ing, cross-MVP sig-nals

11. API Route Reference

Full reference in `docs/API_REFERENCE.md` . Key categories:

Core Data

- `GET /api/deductions` — List deductions with related data
- `GET /api/collections` — Ranked collection accounts
- `GET /api/invoice-triage` — Invoice exception queue
- `GET /api/demand-sensing` — Forecast data
- `GET /api/executive` — Executive KPIs

Agent Operations

- `POST /api/agent-swarm/run` — Trigger agent pipeline
- `GET /api/agent-swarm/list` — List recent runs
- `GET /api/agent-swarm/[runId]` — Run detail + reasoning trace
- `POST /api/llm-chat` — Streaming buyer copilot chat

Governance

- `GET/PATCH /api/hitl-queue` — HITL approval queue
- `GET/PATCH /api/escalations` — Escalation management
- `GET/POST /api/value-ledger` — Financial value tracking
- `GET/POST /api/audit-log` — SOX audit trail
- `GET/PATCH/POST /api/boundary-conditions` — Autonomy thresholds
- `POST /api/writeback/submit|approve|execute` — 3-stage writeback

Evaluation

- `GET /api/agreement-rate` — Agreement-with-human metrics
 - `GET /api/mvp-metrics` — Per-MVP computed KPIs
 - `GET /api/test-runner` — Run E2E test suite
 - `GET /api/golden-cases` — Golden case management
-

12. Key Libraries & Engines

File	Lines	Purpose
<code>lib/trade-deal-validator.ts</code>	380	7-point trade deal validity engine
<code>lib/invoice-validity-engine.ts</code>	360	Invoice 3-way match engine
<code>lib/remittance-parser.ts</code>	400	Multi-format remittance parser (EDI, PDF, Email, Portal)
<code>lib/reason-code-dictionary.ts</code>	240	Customer-specific reason code mappings
<code>lib/mcp-gateway.ts</code>	~300	MCP stub integrations (6 systems)
<code>lib/agent-swarm.ts</code>	~400	MVP1 agent orchestration
<code>lib/agent-swarm-mvp3.ts</code>	~300	MVP3 agent orchestration
<code>lib/agent-swarm-mvp4.ts</code>	~300	MVP4 agent orchestration
<code>lib/agent-swarm-mvp5.ts</code>	~300	MVP5 agent orchestration
<code>lib/mvp1-orchestrator.ts</code>	~200	Full MVP1 pipeline (6 stages)
<code>lib/approval-policy.ts</code>	~150	SOX-compliant approval policy
<code>lib/buyer-policy-engine.ts</code>	~200	Procurement policy evaluation
<code>lib/cross-mvp-trust-score.ts</code>	~150	Weighted cross-MVP trust computation
<code>lib/schemas.ts</code>	~200	Shared types, constants, boundary defaults
<code>lib/persona-config.ts</code>	~100	Role-based persona definitions

13. MCP Gateway — Integration Points

The `lib/mcp-gateway.ts` file contains **stub implementations** for 6 external systems. Each stub is designed for 1:1 replacement with a real ERP adapter.

MCP System	Stub Tool	Real Target	Integration Notes
S4/HANA	s4_read_deduction	SAP S/4HANA OData API	Read deduction master data
AR-Cloud	ar_read_balance	SAP AR / Oracle AR	Read AR balances and aging
Source-to-Pay	s2p_read_contract	SAP Ariba / Coupa	Read contracts and pricing
DPP (Digital Product Passport)	dpp_read_sku	MDM / PIM system	Read SKU master data
TradePromo	tpm_read_deal	TPM system	Read trade deal/promotion data
Analytics	analytics_query	Data warehouse / BI	Run analytical queries

To replace a stub:

1. Find the tool function in `mcp-gateway.ts`
2. Replace the deterministic stub logic with your API call
3. Keep the same return type signature
4. The circuit breaker and rate limiter wrappers still apply

See `docs/INTEGRATION_GAP_REGISTER.md` for detailed effort estimates per system.

14. Prisma Schema Overview

The schema has **68 models** across 7 domains:

Core Domain (~15 models)

`Customer`, `SKU`, `Supplier`, `Invoice`, `InvoiceLineItem`, `TradeDeal`, `TradeDealSKU`, `CollectionAccount`, `CollectionInteraction`, `PurchaseRequisition`, `PRLineItem`, `DemandForecast`, `DemandSensor`

MVP1: Trade Deductions (~10 models)

`Deduction`, `ClassificationResult`, `ValidityCheck`, `RemittanceArtifact`, `TradeDealValidity`, `DeductionReasonCodeMap`, `GoldenCase`, `AgreementRateMetric`

MVP2: Collections (~3 models)

`Mvp2GoldenCase`, `Mvp2AgreementMetric`, `CollectionAccount` (shared)

MVP3: Invoice Triage (~5 models)

`DuplicateInvoiceLink`, `Mvp3GoldenCase`, `Mvp3AgreementMetric`, `InvoiceTriage`

MVP4: Procurement (~3 models)

`Mvp4GoldenCase` , `Mvp4AgreementMetric` , `BuyerCopilotConversation`

MVP5: Demand Sensing (~3 models)

`DemandForecast` , `DemandSensor` , `DemandSignal`

Governance (~20 models)

`ActionProposal` , `HITLQueueItem` , `EscalationTicket` , `ValueLedgerEntry` , `AuditLogEntry` , `AgentRun` , `OverrideReason` , `PromptVersion` , `PromptEvaluation` , `ABTestRun` , `PromptRefinement` , `BoundaryCondition` , `BaselineMetric` , `DriftMonitor` , `MCPToolCall` , `CostMeterEntry` , `ApprovalQueueItem`

15. Customization Guide

Branding

- **Company name:** Search for `Ai50 Food Corp` across the codebase
- **Colors:** Edit `tailwind.config.ts` and `globals.css` CSS variables
- **Logo:** Replace `public/favicon.svg` and `public/og-image.png`
- **Header:** `components/header.tsx`

Adding a New MVP

1. Create page: `app/your-mvp/page.tsx`
2. Create dashboard: `components/your-mvp-dashboard.tsx`
3. Create API route: `app/api/your-mvp/route.ts`
4. Create orchestrator: `lib/your-mvp-orchestrator.ts`
5. Add Prisma models to `schema.prisma`
6. Add to sidebar: `components/sidebar.tsx`
7. Add to persona config: `lib/persona-config.ts`
8. Add boundary conditions: `lib/schemas.ts`

Adjusting Autonomy Thresholds

All boundary conditions are defined in `lib/schemas.ts` → `BOUNDARY_DEFAULTS` . They're seeded to the database on first load and editable via the Workbench UI.

Persona Customization

Edit `lib/persona-config.ts` to add/modify roles (AR Manager, AP Manager, Buyer, Planner, Executive).

16. LLM Provider Swap Guide

All LLM calls go through a consistent pattern. To swap providers:

Option A: OpenAI Direct

```
# .env
ABACUSAI_API_KEY=sk-your-openai-key
```

Then find-and-replace in these files:

```
OLD: https://llmfoundry.straive.com/openai/v1/chat/completions
NEW: https://api.openai.com/v1/chat/completions

OLD: model: 'gpt-5.4-mini'
NEW: model: 'gpt-4o-mini' (or your preferred model)
```

Files to update:

- lib/agent-swarm.ts
- lib/agent-swarm-mvp3.ts
- lib/agent-swarm-mvp4.ts
- lib/agent-swarm-mvp5.ts
- app/api/llm-chat/route.ts

Option B: Azure OpenAI

```
# .env
AZURE_OPENAI_ENDPOINT=https://your-resource.openai.azure.com
AZURE_OPENAI_KEY=your-key
AZURE_OPENAI_DEPLOYMENT=your-deployment-name
```

Update fetch URLs to:

```
${process.env.AZURE_OPENAI_ENDPOINT}/openai/deployments/${
process.env.AZURE_OPENAI_DEPLOYMENT}/chat/completions?api-version=2024-02-01
```

Change auth header from `Authorization: Bearer` to `api-key: ${process.env.AZURE_OPENAI_KEY}`

Option C: Anthropic Claude (via OpenAI-compatible proxy)

Use a proxy like LiteLLM or the Anthropic OpenAI-compatible endpoint:

```
ABACUSAI_API_KEY=sk-ant-your-key
# Update URL to your proxy endpoint
```

Option D: Self-hosted (vLLM, Ollama)

```
# .env
ABACUSAI_API_KEY=not-needed
# Update URLs to http://localhost:8000/v1/chat/completions
```

17. Testing

Built-in Test Runner

```
# Run all tests (E2E + database integrity)
curl http://localhost:3000/api/test-runner

# E2E scenarios only (15 scenarios)
curl "http://localhost:3000/api/test-runner?type=e2e"

# Database integrity only (4 checks)
curl "http://localhost:3000/api/test-runner?type=database"
```

Test Coverage

- **15 E2E scenarios** across 4 categories: happy-path (5), edge-case (5), error-recovery (3), performance (2)
- **4 database integrity tests**: schema validation, relationship integrity, decimal precision, large dataset performance
- **Per-MVP eval harness**: Golden cases with agreement-with-human scoring

Type Checking

```
yarn tsc --noEmit
```

Production Build Test

```
yarn build
```

18. Documents Included

All in the `docs/` directory, each in `.md`, `.docx`, and `.pdf`:

Document	Purpose
INTEGRATION_GAP_REGISTER	Maps 6 MCP stubs to real ERP endpoints, effort estimates
DEPLOYMENT_GUIDE	Infrastructure requirements, Docker template
API_REFERENCE	All 87 endpoints documented
ARCHITECTURE_DECISIONS	8 ADRs explaining key design choices
SECURITY_POSTURE	Compliance gaps, SOX mapping, remediation priorities
GLOSSARY	58 terms + acronym reference
COMPLETE_ROADMAP	Full development roadmap across all waves
BUILD_LOG	Build history and decision log
DEMO_SCRIPT	Demo walkthrough script
COMPLETE_80_SLIDE_OUTLINE	Presentation outline
SLIDE_WEAVE	Slide content weave

19. Troubleshooting

“Cannot find module ‘@prisma/client’”

```
yarn prisma generate
```

“relation does not exist” or database errors

```
yarn prisma db push
```

LLM calls failing (401/403)

- Check `ABACUSAI_API_KEY` is set correctly in `.env`
- Verify the API endpoint URL matches your provider
- Check the model name exists for your provider

Build errors with `outputFileTracingRoot`

If not using the Abacus AI platform, you can safely remove the `experimental.outputFileTracingRoot` line from `next.config.js`.

Yarn version issues

```
corepack enable
yarn set version 4.13.0
```

Database connection issues

- Verify DATABASE_URL format: postgresql://USER:PASS@HOST:PORT/DB
- Ensure PostgreSQL is running and accessible
- Check firewall rules if using a remote database

20. Security Considerations

Current state (as built for POC — see docs/SECURITY_POSTURE.md for full analysis):

Area	Status	Action Needed
Authentication	✗ Not implemented	Add NextAuth.js or your IdP
RBAC	✗ Not implemented	Add role-based middleware
API signing	✗ Not implemented	Add HMAC signatures for MCP calls
PII handling	⚠ Partial	Add encryption-at-rest for sensitive fields
Audit logging	✓ Implemented	SOX-compliant audit trail exists
HITL governance	✓ Implemented	Dual-lane approval with override capture
Input validation	✓ Implemented	Zod schemas on API routes

See the Security Posture document for a prioritized remediation plan.

Quick Start Checklist

```
[ ] Extract archive
[ ] Install Node.js 20+ and Yarn 4+
[ ] Set up PostgreSQL database
[ ] Create .env with DATABASE_URL and ABACUSAI_API_KEY
[ ] yarn install
[ ] yarn prisma generate
[ ] yarn prisma db push
[ ] yarn dev
[ ] POST /api/seed-data to populate sample data
[ ] Verify at http://localhost:3000
[ ] (Optional) Replace LLM endpoint – see Section 16
[ ] (Optional) Replace MCP stubs – see Section 13
[ ] (Optional) Add authentication – see Section 20
```