AI Trader — Full Project Overview

1. Executive Context

AI Trader is a cloud-native, autonomous equities trading platform designed to ingest market data deterministically, generate probabilistic signals, apply explicit risk controls, and execute trades in a fully observable and auditable way. The system prioritizes **risk-first design, traceability, and operational discipline** over speculative AI behavior.

The goal is not to "beat the market" indiscriminately, but to build a **repeatable, governable trading engine** that can evolve safely over time.

---

2. Guiding Principles

- Deterministic data ingestion (replayable, auditable)
- Probabilistic signals (uncertainty-aware, not point forecasts)
- Explicit risk gates (fractional Kelly, exposure caps, drawdown halts)
- Shared pipelines for backtest and live trading
- Cloud-native operations with first-class observability

---

3. High-Level Architecture

The system is organized into **phased layers**, each with a clear responsibility:

1. **Data & Sessions** – multi-source market data ingestion, session tagging
2. **Probabilistic Core** – Kalman filtering, regime detection
3. **Strategies & Backtesting** – signal generation and historical evaluation
4. **Risk & Execution** – sizing, gating, broker routing
5. **Observability & Ops** – logs, metrics, traces, dashboards
6. **CI/CD & Automation** – build, deploy, scheduling

This separation prevents tight coupling and allows safe iteration.

---

4. Codebase Structure

The codebase is modular and import-safe, designed for long-term maintainability.

```
app/
 ├── api/              # FastAPI routes, health checks
 ├── dal/              # Market Data Abstraction Layer
```

```
├── probability/      # Kalman + regime pipeline
├── features/         # Indicators, MTF aggregation
├── scanners/         # Premarket & intraday scanners
├── services/         # Watchlist & orchestration services
├── strats/           # Strategy implementations
├── agent/            # Policy, sizing, risk, journaling
├── backtest/         # Backtest engine & broker simulator
├── execution/        # Live broker adapters
├── sessions/         # Market session calendar
├── observability/    # OTEL + logging helpers
├── monitoring/       # Streamlit dashboards
├── db/               # SQLAlchemy models & repositories
└── tests/            # Unit, integration, regression tests
```

Key design decisions: - Strategies never talk directly to vendors - Live and backtest share the same execution logic - Probabilistic signals are first-class data artifacts

---

5. Market Data & Probabilistic Core

Market Data Abstraction Layer (DAL)

The DAL unifies multiple vendors (Alpaca, Alpha Vantage, Finnhub, Yahoo fallback) behind a single interface.

Responsibilities: - Normalize OHLCV data into deterministic schemas - Handle HTTP + streaming ingestion - Perform automatic gap backfills - Cache results to Parquet (Blob Storage) - Optionally persist metadata to Postgres

Probabilistic Pipeline

- Kalman Filter produces smoothed price, velocity, and uncertainty
- Regime classifiers (HMM/GARCH planned) label volatility states
- Outputs feed strategy selection and risk sizing

This avoids overconfident signals and supports adaptive risk management.

---

6. Strategies, Risk & Execution

Strategies

- Breakout strategy: live and end-to-end wired
- Momentum / Mean-Reversion: scaffolded, pending parameter sweeps

Risk Management (Core Design)

- Fractional Kelly sizing (spec complete)
- Hard exposure caps (symbol + portfolio)
- Max drawdown halts
- Buying power and PDT guards
- Session-aware throttling

Execution

- Broker: Alpaca (paper/live)
- Order types:
- Limit / bracket orders in PRE/AFT
- Marketable limits during RTH
- OMS cache persisted to Postgres

Risk gates sit **before** any broker call.

---

7. Infrastructure (Azure)

Core Components

Layer	Technology
Compute	Azure App Service (Linux container)
Database	Azure PostgreSQL Flexible Server
Storage	Azure Blob Storage
Secrets	Azure Key Vault
Observability	OpenTelemetry + App Insights
CI/CD	GitHub Actions

Security & Identity

- Managed Identity everywhere possible
- No static credentials in code or repos
- Key Vault as the single source of truth

Database Design

Postgres schemas are domain-separated: - market - trading - backtest - analytics - storage

Backups, PITR, and restore drills are documented and tested.

---

8. Observability & Operations

Observability is a first-class concern:

- Structured JSON logging
- Distributed traces via OpenTelemetry
- Metrics for latency, errors, exposure, PnL
- Streamlit dashboards for live visibility
- Alerting for failures and risk breaches

Operational runbooks define: - Daily trading schedule - Data degradation behavior - Broker failure handling - Safe mode and recovery steps

---

9. CI/CD & Automation

Continuous Integration

- Linting (ruff)
- Formatting (black)
- Security scans (bandit, pip-audit)
- Pytest on every PR

Continuous Deployment

- GitHub Actions build containers
- Deploy to Azure App Service
- Slot-based promotion

Scheduling

- Webhook-driven jobs for:
- Premarket scans
- Watchlist refreshes
- End-of-day processing
- Model retraining

---

10. Project Status (Current)

Phase	Status
Bootstrap	Complete
Data & Sessions	Complete
Probabilistic Stack	~70%

Phase	Status
Strategy Coverage	~40%
Risk & Execution	~20%
Dashboard & Ops	~55%
CI/CD & Scheduling	~25%
Continuous Learning	~5%
Hardening	~5%

Primary focus areas: - Finish RiskManagementAgent - Wire execution loop fully - Enforce paper-trading discipline

---

11. Blunt Assessment

This is not a toy project.

Architecturally, the system is **sound, modern, and defensible**. The remaining risk is not technical—it is execution discipline and scope control.

If risk and execution are completed cleanly and paper trading is enforced rigorously, AI Trader is well-positioned to achieve its stated goal: a sustainable, autonomous trading engine with controlled downside and continuous learning potential.

Document purpose: single-source project context for future development, audits, or collaborators.