Tally Agentic Workflow - Documentation

Table of Contents

Project Overview

Tally Agentic Workflow is an AI-powered assistant that intelligently interacts with Tally ERP, providing users with natural language access to financial reports, automated summaries, and visual analytics. The system uses a multi-agent architecture with a ReAct-based supervisor to route queries intelligently between different specialized agents.

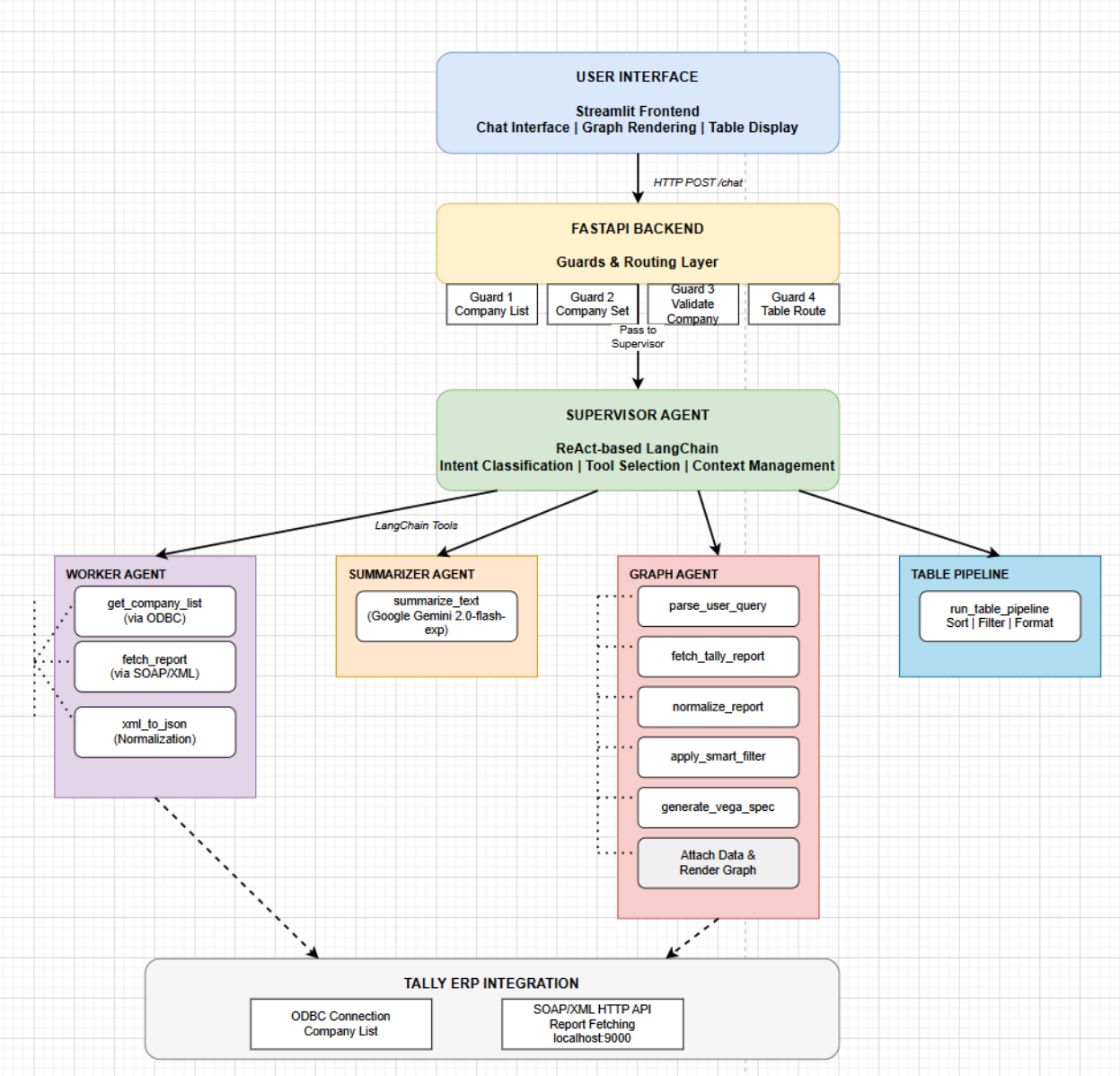
Key Capabilities:

- **Natural Language Queries**: Ask questions in plain English about Tally reports.
- **Intelligent Routing**: Automatic detection of query intent (Value, Table, Graph, Summary).
- **Multi-Report Support**: Balance Sheet, P&L, Stock Summary, Day Book, Bills Payable and Receivable.
- **Visual Analytics**: Automatic graph generation with Vega-Lite specifications.
- **Smart Company Management**: Seamless company selection with multiple input patterns.

Architecture & Flow

The Tally Agentic Workflow follows a layered, modular architecture designed to clearly separate concerns between user interaction, intelligent decision-making, data retrieval, and visualization. At the core of the architecture lies an intelligent Supervisor Agent, which acts as the central decision-maker.

System Architecture Flowchart(draw.io)



Tech Stack

Frontend Layer:

Technology	Purpose	Version
Streamlit	Interactive web UI with chat interface	Latest
Vega-Lite	Declarative visualization grammar	v5

Backend Layer:

Technology	Purpose	Version
FastAPI	RESTful API server with async support	Latest
Uvicorn	ASGI web server	Latest

Agent Framework:

Technology	Purpose	Version
LangChain	Agent orchestration and tool management	Latest
ReAct Pattern	Reasoning + Acting loop for decision making	-
ConversationBufferMemory	Chat history and context retention	-

AI/LLM Layer:

Technology	Purpose	Model
Google Gemini	Natural language understanding and summarization	gemini-2.0-flash-exp

Tally Integration:

Technology	Purpose	Protocol
PyODBC	Company list retrieval	ODBC
Requests	Report data fetching	SOAP/XML over HTTP
Tally ODBC Driver	Database connectivity	Native
Tally HTTP API	Report export	Port 9000

System Components

1. **Supervisor Agent**

   File: `SupervisorAgent.py`

   Technology**: LangChain ReAct Agent + Gemini LLM**

   Core Responsibilities:

- **Intent Classification**: Determines if query is VALUE, TABLE, GRAPH, or SUMMARY.
- **Report Type Inference**: Identifies which Tally report to use (Balance Sheet, P&L, Stock Summary, Day Book, Bills Payable, Bills Receivable).
- **Tool Orchestration**: Selects and coordinates appropriate tools.
- **Error Handling**: Manages failures and provides fallbacks.
- **Company Context**: Maintains active company state.

Intent Classification Logic:

User Query Example	Classified Intent	Action Taken
"What is my costliest stock item?"	VALUE	fetch_report + summarize
"Show top 5 items"	TABLE	run_table_pipeline
"Compare my particulars based on their Debit Amount "	GRAPH	graph_insights tool
"Summarize my balance sheet"	SUMMARY	fetch_report + summarize

Available Tools:

1. **fetch_companies** - Retrieve list of companies from Tally.
2. **list_companies_text** - Human-readable formatted company list.
3. **fetch_report** - Get raw XML report data.
4. **summarize_report** - Generate natural language summary.
5. **graph_insights** - Create visual analytics with Vega-Lite.
6. **json_normalizer** – Normalize raw Tally XML/JSON reports into clean, structured JSON by handling numeric parsing, and chart-ready data formatting.

---

2. Worker Agent

File: `agents.py`
Technology: Custom Python class with LangChain tool wrappers

Core Responsibilities:

- **Company List Fetching**: Connects via ODBC to retrieve available companies
- **Report Fetching**: Sends SOAP requests to Tally HTTP API
- **Connection Management**: Handles Tally connectivity and timeouts
- **Retry Logic**: Implements automatic retry on network failures

Key Integration Details:

ODBC Connection (Company List):

- Uses Tally ODBC Driver
- Queries Tally's internal database
- Returns list of company names

SOAP Connection (Reports):

- HTTP POST to `http://localhost:9000`
- XML envelope with report specifications
- Supports dynamic date ranges and filters
- Returns structured XML data

---

3. Summarizer Agent

File: `agents.py`
Technology: Gemini API

Core Responsibilities:

- Convert raw XML/JSON reports into concise summaries.
- Extract key financial insights.
- Handle large reports efficiently (5000+ lines).
- Provide context-aware explanations.

---

4. Graph Agent

File: `graph_agent.py`
Technology: Custom Python + Vega-Lite + Optional LLM filtering

Core Responsibilities:

- Parse natural language graph requests
- Infer report type and chart type (bar/pie/line)
- Normalize Tally XML data into chart-ready JSON
- Generate Vega-Lite specifications

- Apply intelligent filtering for "only X, Y, Z" queries

Smart Filtering Capabilities:

User Input	Filtering Action
"Show full balance sheet"	Return all rows
"Show only Assets and Liabilities"	Extract explicit fields → filter
"Plot only top 3 assets"	Use LLM to identify relevant items
"Balance sheet" (no filter)	Return all rows (default)

Supported Configurations:

- **Reports**: Balance Sheet, Profit & Loss, Stock Summary, Day Book, Bills Payable, Bills Receivable.
- **Chart Types**: Bar Chart (default), Pie Chart, Line Chart.
- **Filtering**: Keyword-based, LLM-assisted, Explicit field matching.

---

5. Table Pipeline

File: `graph_agent.py`
Technology: Direct data pipeline (no LLM)

Core Responsibilities:

- Handle "top N" queries efficiently.
- Sort and filter tabular data.
- Return structured JSON for frontend rendering.
- Generate brief summary using LLM.

Optimization:

- Bypasses agent overhead for simple queries.
- Direct data transformation.
- Faster response time for list-based requests.

---

6. FastAPI Backend

File: `api/main.py`
Technology: FastAPI + Uvicorn

Core Responsibilities:

- HTTP request handling and validation
- Early query routing (guardrails)
- Company selection management
- Response formatting (text/graph/markdown)
- Session state coordination

Guardrail System:

Guard	Trigger	Action
Guard 1	"list companies"	Return company list immediately
Guard 2	Company selection only	Set company, no agent call
Guard 3	No company set	Block query, prompt selection
Guard 4	Table request	Route to table pipeline
Guard 5	Other queries	Pass to supervisor agent

Company Selection Intelligence: Handles multiple input patterns:

- Number selection: `"1"`, `"2"`, `"company 3"`
- Full name: `"ABC Pvt. Ltd."`, `"XYZ Limited"`
- Keyword-based: `"use ABC"`, `"switch to XYZ"`
- Fuzzy matching: Partial name matches from available list

---

7. Streamlit Frontend

File: Streamlit_run.py
Technology: Streamlit + Vega-Lite

Core Responsibilities:

- Chat interface with message history
- Graph rendering with Vega-Lite

- Markdown table display
- Session state management
- Error handling and user feedback

Key Features:

- **Chat History**: Persistent conversation context
- **Multi-format Output**: Text, graphs, tables in single interface
- **Interactive Charts**: Hover tooltips, legends, responsive design
- **Debug View**: Expandable raw API response viewer
- **Mobile Responsive**: Adaptive layout for different screen sizes

---

Data Flow Examples

Example 1: Simple Value Query

User Query: "What is my costliest stock item?"

Flow:

```
1. User Input → Streamlit Frontend
2. HTTP POST /chat → FastAPI Backend
3. Guard Check: Company set?
4. Route to Supervisor Agent
5. Supervisor Classification: Intent = VALUE
6. Tool Selection: fetch_report("Stock Summary")
7. Worker Agent: SOAP request to Tally
8. Tally Response: XML with stock data
9. Tool Selection: summarize_report
10. Summarizer Agent: Gemini processes
11. Response: "Your costliest item is Widget A at ₹5000/unit"
12. Frontend: Display text summary
```

Example 2: Graph Generation Query

User Query: "Compare Capital Account, Assets, and Loans"

Flow:

```
1. User Input → Streamlit Frontend
2. HTTP POST /chat → FastAPI Backend
3. Guard Check: Company set?
4. Route to Supervisor Agent
5. Supervisor Classification: Intent = GRAPH
6. Tool Selection: graph_insights
7. Graph Agent:
```

```
   a. Infer report: Balance Sheet
   b. Extract fields: ["Capital Account", "Assets", "Loans"]
   c. Fetch report via Worker Agent
   d. Normalize XML → JSON rows
   e. Filter: Keep only 3 requested items
   f. Generate Vega-Lite spec (bar chart)
8. Response: JSON with vega_spec + summary
9. Frontend: Render interactive bar chart
```

Example 3: Table Request

User Query: "Show top 5 stock items by rate"

Flow:

```
1. User Input → Streamlit Frontend
2. HTTP POST /chat → FastAPI Backend
3. Guard Check: Detect "top N" pattern
4. Guard 4 Activates: Route to Table Pipeline (bypass agent)
5. Table Pipeline:
   a. Fetch Stock Summary report
   b. Parse and normalize data
   c. Sort by rate DESC
   d. Take top 5 rows
   e. Generate brief summary with LLM
6. Response: Markdown formatted table + summary
7. Frontend: Render table with styling
```

Example 4: Company Selection

User Query: "1" (after seeing company list)

Flow:

```
1. User Input → Streamlit Frontend
2. HTTP POST /chat → FastAPI Backend
3. Guard 2 Activates: Detect company selection intent
4. Extract company from query:
   - Pattern match: Number "1"
   - Fetch company list via ODBC
   - Map index → company name: "ABC Pvt. Ltd."
5. Set active company in Supervisor singleton
6. Response: "Company set to: ABC Pvt. Ltd."
7. Frontend: Display confirmation
8. No agent call made (optimization)
```

Key Features

1. Intelligent Intent Detection

The system automatically classifies user queries into four categories without requiring explicit commands:

- **VALUE**: Single-answer questions (e.g., "What is my capital account?")
- **TABLE**: List-based queries (e.g., "Show top 10 stock items")
- **GRAPH**: Comparative analytics (e.g., "Compare assets vs liabilities")
- **SUMMARY**: Overview requests (e.g., "Summarize my P&L")

2. Multi-Pattern Company Selection

Users can select companies in any natural format:

- By number: `"1"`, `"2"`, `"3"`
- By name: `"ABC Pvt. Ltd."`
- With keywords: `"use XYZ"`, `"switch to ABC"`
- Partial match: `"ABC"` (fuzzy matches from available list)

3. Smart Graph Filtering

The Graph Agent intelligently handles filtering requests:

- **Full report**: `"show balance sheet"` → all items
- **Explicit filtering**: `"only Assets and Liabilities"` → exactly 2 items
- **LLM-assisted**: `"only top assets"` → AI determines relevant items
- **Default behavior**: No filter keyword → show all

4. Report Type Auto-Detection

System infers correct report from query context:

- Keywords: "stock", "inventory" → Stock Summary
- Keywords: "assets", "liabilities" → Balance Sheet
- Keywords: "profit", "loss", "income" → Profit & Loss
- Keywords: "voucher", "debit", "credit" → Day Book

5. Guardrail System (Performance Optimization)

Early interception prevents unnecessary LLM calls:

- Company list requests → instant return (no agent)
- Company selection → direct state update (no agent)

- Table queries → direct pipeline (no agent)
- Only complex queries route to supervisor

6. Single Source of Truth

Centralized company state eliminates sync issues:

- All components access: `_SUPERVISOR_SINGLETON.active_company`
- No multiple state variables
- Prevents "wrong company" bugs

7. Dual Tally Integration

Two connection methods for optimal performance:

- **ODBC**: Fast company list retrieval (SQL-like queries)
- **SOAP**: Comprehensive report data (supports all report types)

Design Decisions & Rationale

Why ReAct Pattern?

- **Transparency**: Clear reasoning chain for debugging.
- **Control**: Explicit tool selection.
- **Reliability**: Predictable behaviour with fallback handling.

Why Early Guards?

- **Performance**: Avoid LLM overhead for simple queries.
- **Latency**: Company selection responds in <100ms.
- **Cost**: Reduces API calls to Gemini.

Why Separate Table Pipeline?

- **Speed**: Direct data pass-through (no agent reasoning)
- **Simplicity**: "Top N" queries don't need AI interpretation
- **Resource efficiency**: Lower LLM token usage

Why Centralized Company State?

- **Bug Prevention**: Eliminated sync issues between components.
- **Simplicity**: Single `get/set` interface.
- **Debugging**: One place to check company context.

Why Hybrid Filtering?

- **Robustness**: Keyword matching handles 80% of cases
- **Intelligence**: LLM fallback for ambiguous requests
- **User Experience**: Works with any phrasing style

Environment Configuration

Required environment variables in `.env` file.

```
TALLY_HTTP_HOST=http://localhost:9000
ODBC_CONNECTION_STRING=DRIVER={Tally ODBC Driver};SERVER=localhost;PORT=9000
GEMINI_API_KEY=”your_gemini_api_key_here”
GEMINI_MODEL=gemini-2.0-flash-exp
```

Deployment Instructions

Backend Server:

```
cd api
uvicorn main:app --host 0.0.0.0 --port 8000
```

Frontend Application:

```
streamlit run app.py --server.port 8501
```

Access URLs:

- Frontend: `http://localhost:8501`
- Backend API: `http://localhost:8000/chat`
- API Docs: `http://localhost:8000/docs`

System Capabilities Summary

Feature	Capability
Reports Supported	Balance Sheet, P&L, Stock Summary, Day Book, Bills Payable, Bills Receivable, Group Summary
Chart Types	Bar, Pie, Line
Query Types	Value, Table, Graph, Summary
Company Selection	Number, Name, Keyword
Response Formats	Text, Vega-Lite JSON/Graphs, Markdown Table
Tally Connections	ODBC (company list), SOAP(reports)
AI Models	gemini-2.0-flash-exp
Agent Pattern	ReAct (Reasoning + Acting)
Context Retention	Conversation Buffer Memory