Enterprise Customer Onboarding Agent

Solution Design Document — February 2025 — StackAdapt Case Study

1. Architecture Overview

The solution implements an **AI-powered automation agent** using **LangGraph** for state machine orchestration and **FastAPI** for REST interface, automating customer onboarding from deal closure through SaaS provisioning.

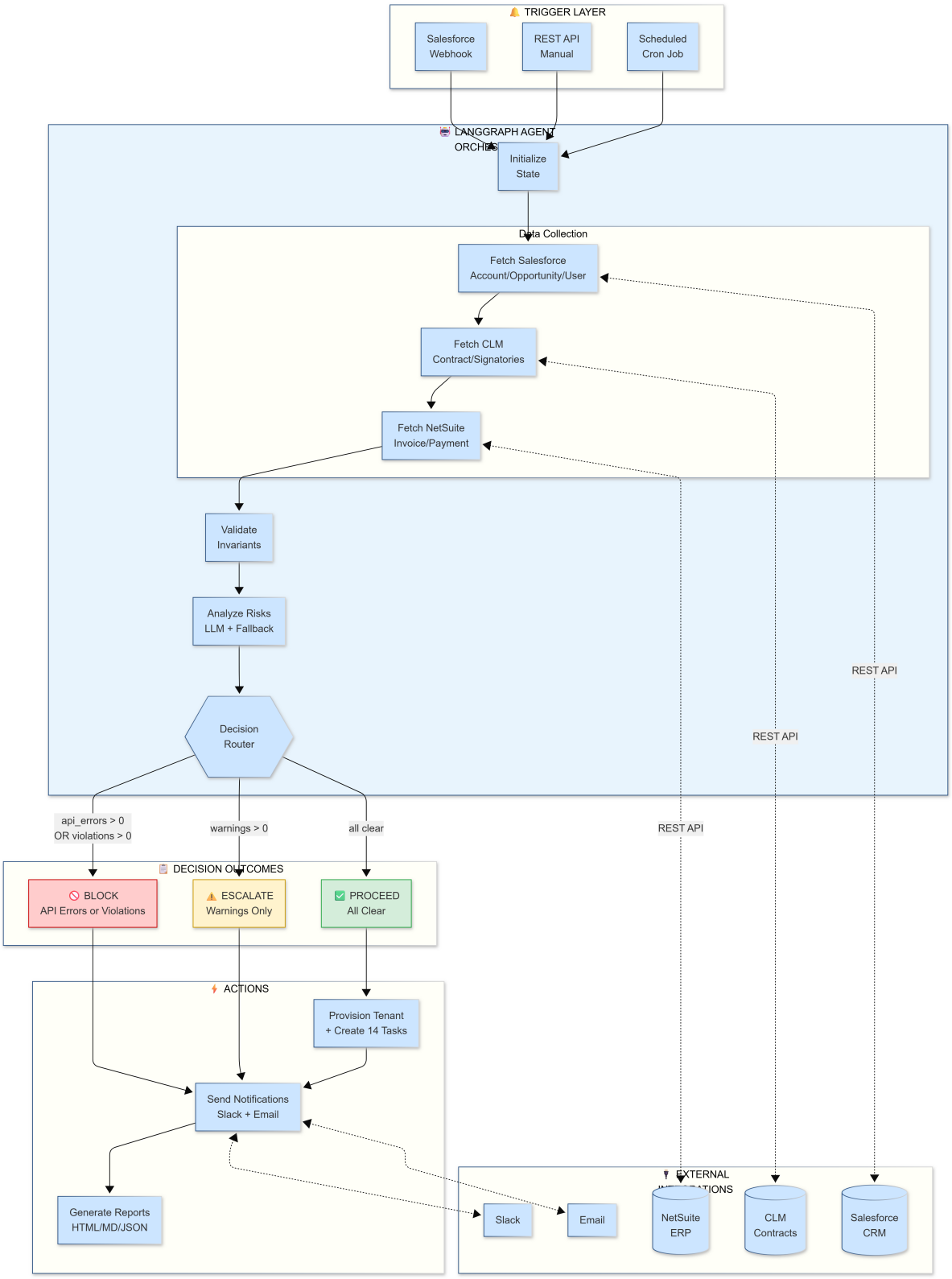

Figure 1: Architecture: Triggers → LangGraph Agent (Init→Fetch→Validate→Analyze→Decide) → Actions

1.1 System Integrations & Field Selection

System	Objects & Key Fields	Reference
Salesforce	`Account` (IsDeleted, Status), `Opportunity` (StageName, Amount), `User` (IsActive)	SF Objects
NetSuite	`Invoice` (status, dueDate, amountRemaining, paymentStatus)	NS REST API
CLM	Contract status, signatories, effective dates (Mock API)	Custom
Provisioning	Tenant creation + 14-task onboarding checklist with dependencies	Internal

Why these fields? Only business-critical fields: `Account.IsDeleted` for validity, `Opportunity.StageName="Closed Won"` for deal verification, `Invoice.status` for payment issues. Minimizes API payload and focuses on decision-relevant data.

2. AI Agent Application

The agent uses **OpenAI GPT-4** with a deterministic rule-based fallback ensuring availability without LLM connectivity.

LLM-Powered Intelligence:

- **Risk Analysis**: API errors, violations, warnings → risk level (low/medium/high/critical)
- **Summary Generation**: Human-readable reports for CS teams
- **Action Recommendations**: Prioritized steps with owner assignment
- **Error Interpretation**: `INVALID_SESSION_ID` → plain English

Autonomous Actions: Auto-provisions tenant • Creates 14-task checklist • Sends Slack + email • Records API failures → BLOCK

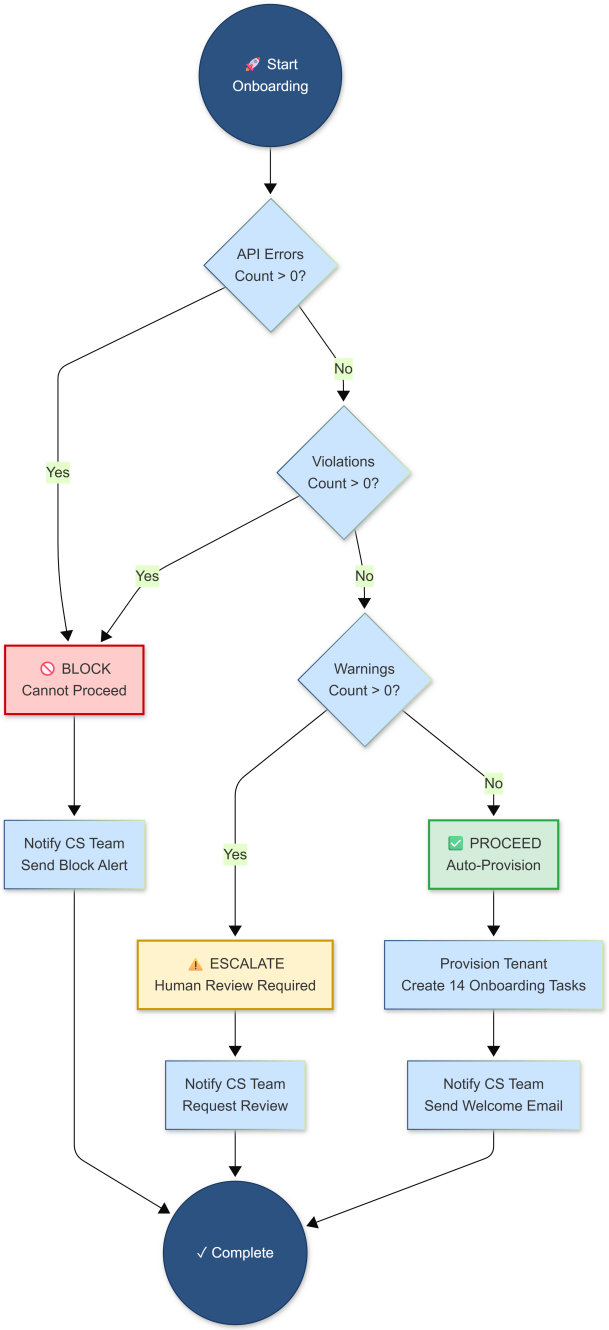

Figure 2: Decision Priority Flow

3. Orchestration & Event-Driven Flows

Trigger	Source	Use Case	Endpoint
Webhook	Salesforce	Opportunity → "Closed Won"	`POST /webhook/onboarding`
Manual	CS Team	On-demand for specific account	`POST /demo/run/{id}`
Batch	Scheduler	Overnight processing	`POST /demo/run-all`
Task	CS Action	Mark task complete	`PUT /demo/tasks/{id}/{task}`

State Machine: LangGraph manages: Init → Fetch(SF/CLM/NS) → Validate → Analyze → Decide → [Provision] → Notify → Complete. Conditional routing based on `api_errors`, `violations`, `warnings`. **Error Simulation**: `/demo/enable-random-errors` injects configurable failures (401/400/429/500) for resilience testing.

4. Trade-offs, Assumptions & Considerations

Decision	Rationale
API Errors → BLOCK	Data integrity over speed
Sync Processing	Simple for demo; prod uses queues
Rule-based Fallback	Works without LLM connectivity
14 Fixed Tasks	Predictable; prod uses templates

Scalability: Sync → SQS/RabbitMQ, Memory → Redis, Single → K8s, + LLM caching

Security: OAuth 2.0 with token refresh, audit trails with correlation IDs, PII masking in logs, RBAC for API operations

5. Multi-Agent Collaboration via MCP

The **Model Context Protocol (MCP)** enables specialized agents to collaborate through standardized tool interfaces:

Agent	Responsibility & Tools
Coordinator	Orchestrates workflow (`salesforce.*`, `provision.*`)
Contract	Monitors signatures (`clm.get_contract`)
Finance	Tracks payments (`netsuite.get_invoice`)
Task Monitor	Overdue detection (`tasks.get_overdue`)

Benefits: Domain expert agents • Shared MCP servers • Add agents without modifying existing • Cross-agent context via tool responses

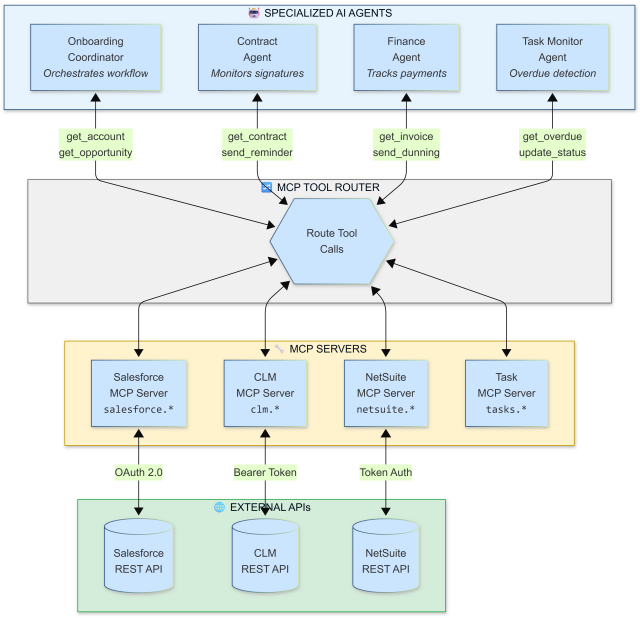

Figure 3: MCP: Agents → Router → Servers → APIs