Enterprise Customer Onboarding Agent — Solution Design Document

February 2025 — StackAdapt Case Study Submission

1. Architecture Overview

The solution implements an **AI-powered automation agent** that orchestrates enterprise customer onboarding from deal closure through SaaS provisioning. Built with **LangGraph** for state machine orchestration and **FastAPI** for the REST interface, the agent integrates with multiple enterprise systems, validates business rules, assesses risks using LLM intelligence, and takes autonomous actions including tenant provisioning and task management.

1.1 System Integrations

The agent connects to enterprise systems via REST APIs. Field selection focused on business-critical data to minimize payload and optimize validation:

System	Objects & Key Fields
Salesforce CRM	`Account` (Id, Name, IsDeleted, Status, BillingCountry), `Opportunity` (StageName, Amount, CloseDate), `User` (IsActive, Email). [API Ref]
NetSuite ERP	`Invoice` (status, dueDate, amountRemaining, paymentStatus). Used for payment verification before provisioning. [API Ref]
CLM System	Contract status, signatories, effective dates, key terms. Mock implementation simulating DocuSign CLM.
Provisioning	Internal system for tenant creation + 14-task onboarding checklist with dependencies and due dates.

Field Selection Rationale: `Account.IsDeleted` validates account exists; `Opportunity.StageName="Closed Won"` confirms deal closure; `Invoice.status` identifies payment blockers. This minimizes API calls while capturing decision-critical data.

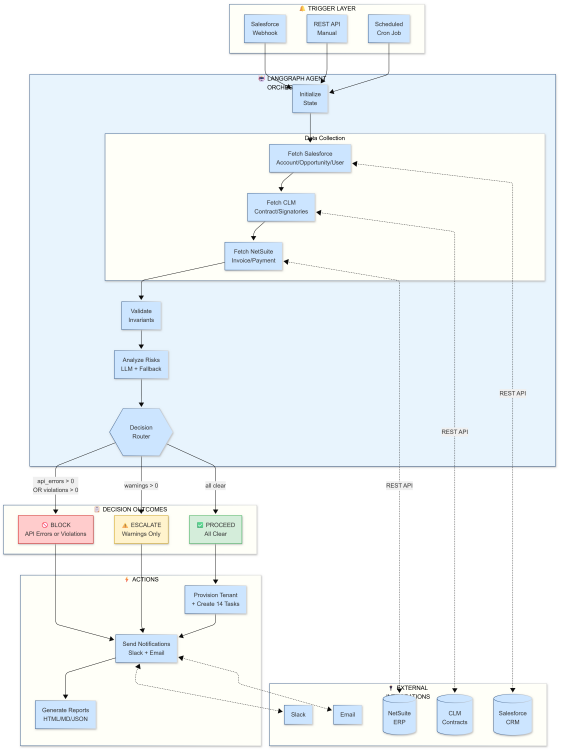

Figure 1: High-level architecture showing trigger sources, LangGraph orchestration, and integration layer.

2. AI Agent Application

The agent leverages **OpenAI GPT-4** for intelligent analysis with a deterministic rule-based fallback, ensuring the system operates even without LLM connectivity. This dual-mode approach balances intelligence with reliability.

2.1 LLM-Powered Intelligence

- **Risk Analysis**: Evaluates API errors, business rule violations, and warnings to generate risk levels (low/medium/high/critical) with business impact assessment and estimated resolution time.
- **Summary Generation**: Creates human-readable status reports for Customer Success teams, translating technical states into actionable insights.
- **Action Recommendations**: Produces prioritized remediation steps with owner assignment (CS, Finance, IT, Legal) based on issue type and urgency.
- **Error Interpretation**: Converts technical API error codes (e.g., `INVALID_SESSION_ID`, `INSUFFICIENT_ACCESS`) into plain-English explanations with specific resolution steps.

2.2 Autonomous Actions

Upon decision, the agent executes: **(1)** Auto-provisions tenant with tier-appropriate configuration; **(2)** Creates 14-task onboarding checklist with dependencies, owners, and due dates; **(3)** Sends Slack alerts to `#cs-onboarding` channel; **(4)** Emails welcome message with login credentials to customer; **(5)** Records all API failures with full context for audit.

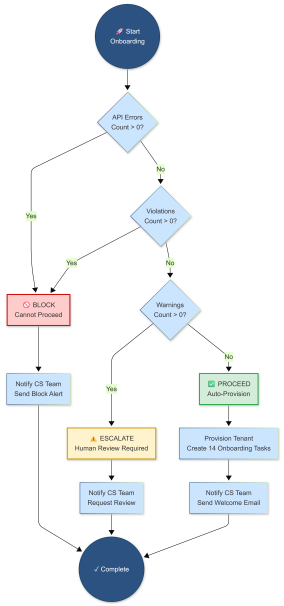

Figure 2: Decision logic: API Errors → BLOCK, Violations → BLOCK, Warnings → ESCALATE, Clear → PROCEED.

3. Orchestration & Event-Driven Flows

The agent supports multiple trigger mechanisms for different operational needs. LangGraph manages state transitions through a defined workflow with full observability via structured logging and optional LangSmith tracing.

Trigger	Source	Endpoint
Webhook	Salesforce Flow	`POST /webhook/onboarding`
Manual	CS Team	`POST /demo/run/{account_id}`
Batch	Scheduler/Cron	`POST /demo/run-all`
Task Update	CS Action	`PUT /demo/tasks/{id}/{task}`

State Machine Flow: Initialize → Fetch Salesforce → Fetch CLM → Fetch NetSuite → Validate Invariants → Analyze Risks (LLM) → Make Decision → [Provision if PROCEED] → Send Notifications → Generate Summary → Complete. Each node handles errors gracefully and updates state.

Error Simulation for Resilience Testing: The endpoint `/demo/enable-random-errors` injects configurable failures with adjustable rates:

- Authentication errors (HTTP 401) - expired tokens
- Validation errors (HTTP 400) - malformed data
- Rate limit errors (HTTP 429) - API throttling
- Server errors (HTTP 500) - backend failures

This enables chaos testing without modifying business logic, validating the agent's error handling and recovery capabilities.

4. Trade-offs, Assumptions & Considerations

4.1 Design Decisions

Decision	Rationale
API Errors → BLOCK	Data integrity over speed; cannot provision with incomplete data
Synchronous Processing	Simplicity for demo; production would use message queues (SQS/RabbitMQ)
Rule-based Fallback	Ensures availability without LLM; deterministic behavior for testing
14 Fixed Tasks	Predictable workflow; production would use configurable templates

4.2 Scalability & Security

Scalability Path: Current demo uses synchronous processing with in-memory state. Production deployment would include: message queues for async processing, Redis for distributed state, Kubernetes for horizontal scaling, and LLM response caching for cost optimization.

Security & Governance: OAuth 2.0 with token refresh simulation, structured audit trails with correlation IDs across all operations, PII masking in logs, role-based access control (RBAC) for API operations, and complete state snapshots for compliance.

5. Multi-Agent Collaboration via Model Context Protocol (MCP)

The architecture supports multi-agent collaboration through MCP, enabling specialized agents to communicate via standardized tool interfaces. Each agent is a domain expert with focused responsibilities.

Agent	Responsibility & MCP Tools
Coordinator	Orchestrates end-to-end workflow (`salesforce.*`, `provision.*`, `tasks.*`)
Contract Agent	Monitors signature status, sends reminders (`clm.get_contract`, `clm.send_reminder`)
Finance Agent	Tracks payments, initiates dunning (`netsuite.get_invoice`, `netsuite.send_dunning`)
Task Monitor	Detects overdue tasks, alerts CS (`tasks.get_overdue`, `notify.alert`)

Benefits: Separation of concerns (domain experts) • Reusable integrations (shared MCP servers) • Scalable complexity (add agents without modifying existing) • Cross-agent context sharing via tool responses.

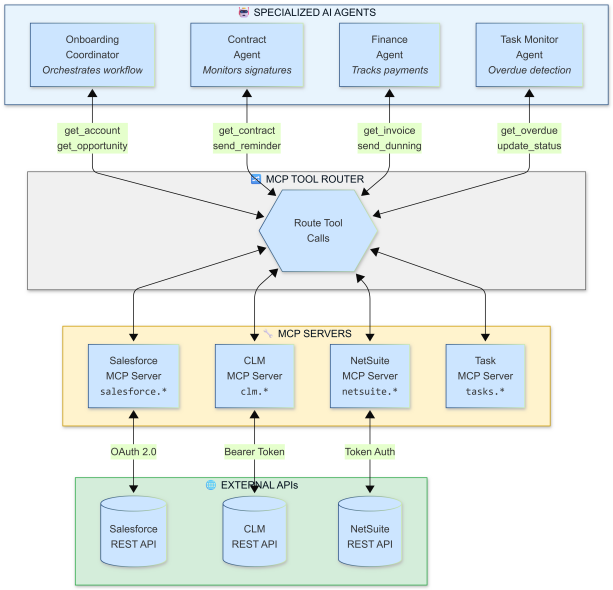

Figure 3: MCP architecture: Specialized agents collaborate through shared MCP servers wrapping external APIs.