ABHIKARTA

Multi-Agent Orchestration System

Requirements & Design Specification

Version 1.0

October 2025

# 1. EXECUTIVE SUMMARY

Abhikarta is a sophisticated multi-agent orchestration and workflow management framework designed to coordinate multiple specialized AI agents for complex task execution. The system provides a comprehensive platform for planning, executing, monitoring, and managing AI-driven workflows with human oversight capabilities.

This document defines the complete requirements, architecture, and design specifications for the Abhikarta system, covering all eight core components and their interactions. The system leverages modern technologies including Flask for web framework, SQLite for data persistence, Kafka for messaging, and implements a user-friendly Bootstrap-based interface.

## 1.1 System Purpose

Abhikarta enables organizations to deploy and manage multiple AI agents working collaboratively on complex tasks that require coordination, specialized skills, and human oversight. The framework handles task decomposition, workflow planning, agent orchestration, and provides comprehensive monitoring and control capabilities.

## 1.2 Key Capabilities

* Dynamic workflow planning and execution using Directed Acyclic Graphs
* Multi-agent orchestration with capability-based task delegation
* UI-driven human-in-the-loop intervention and approval workflows
* Real-time agent monitoring and performance tracking
* Comprehensive job and session lifecycle management
* Error detection, handling, and recovery mechanisms
* Multi-channel notification system for workflow events
* Integration with external systems via Model Context Protocol

# 2. SYSTEM OVERVIEW

## 2.1 Technology Stack

**Backend Framework:** Flask (Python)

**Frontend Technologies:** Bootstrap 5.x, jQuery 3.x

**Database:** SQLite (primary), PostgreSQL (configurable)

**Message Queue:** Apache Kafka

**Authentication:** File-based (users.json) with cleartext comparison

**UI Constraints:** No modal dialogs, no DataTables library

## 2.2 Architecture Overview

Abhikarta follows a modular, layered architecture with clear separation between presentation, business logic, and data layers. The system uses an event-driven architecture for asynchronous communication between components via Kafka, while maintaining synchronous REST APIs for user interactions.

### 2.2.1 System Layers

* **Presentation Layer:** Bootstrap-based web UI with jQuery for client-side interactions
* **Application Layer:** Flask-based REST API endpoints and business logic services
* **Orchestration Layer:** Planner, DAG engine, and autonomous decision components
* **Integration Layer:** MCP integration, Kafka messaging, external system connectors
* **Data Layer:** SQLite database with abstraction for PostgreSQL migration

# 3. COMPONENT 1: PLANNER

## 3.1 Component Overview

The Planner is the strategic intelligence component responsible for translating high-level user requests into concrete, executable workflows. It performs task decomposition, dependency analysis, agent selection, and adaptive replanning based on runtime conditions.

## 3.2 Functional Requirements

### 3.2.1 Task Decomposition

* FR-PLN-001: System shall parse natural language requests and extract intent
* FR-PLN-002: System shall break complex goals into atomic subtasks
* FR-PLN-003: System shall identify task dependencies and constraints
* FR-PLN-004: System shall estimate resource requirements for each subtask

### 3.2.2 Workflow Planning

* FR-PLN-005: System shall generate DAG representations of workflows
* FR-PLN-006: System shall identify parallelization opportunities
* FR-PLN-007: System shall sequence tasks based on dependencies
* FR-PLN-008: System shall support both runtime DAG creation and pre-defined DAG invocation

### 3.2.3 Agent Selection

* FR-PLN-009: System shall maintain a registry of agent capabilities
* FR-PLN-010: System shall match tasks to agents based on required skills
* FR-PLN-011: System shall consider agent availability and load
* FR-PLN-012: System shall support multi-agent collaboration on complex tasks

### 3.2.4 Adaptive Replanning

* FR-PLN-013: System shall monitor task execution and detect deviations
* FR-PLN-014: System shall dynamically adjust workflows based on intermediate results
* FR-PLN-015: System shall handle task failures by creating recovery plans
* FR-PLN-016: System shall incorporate new information into active plans

## 3.3 Technical Specifications

### 3.3.1 Input Specifications

The Planner accepts the following inputs in JSON format:

* User request in natural language
* Pre-defined DAG templates (optional)
* MCP endpoint descriptions and capabilities
* Agent capability registry
* Execution context and constraints

### 3.3.2 Output Specifications

* Executable DAG structure with nodes and edges
* Task-to-agent assignments
* Estimated execution timeline
* Resource allocation plan
* Checkpoint definitions for human-in-the-loop reviews

## 3.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| plans | Stores workflow plans and their metadata |
| plan\_tasks | Individual tasks within a plan |
| plan\_dependencies | Task dependency relationships |
| agent\_capabilities | Registry of agent skills and capabilities |
| plan\_execution\_history | Historical plan executions for learning |

### 3.4.1 Plans Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| plan\_id | VARCHAR(36) | Primary key, UUID |
| user\_id | VARCHAR(50) | User who created the plan |
| request\_text | TEXT | Original user request |
| plan\_status | VARCHAR(20) | draft, active, completed, failed |
| dag\_definition | TEXT | JSON representation of DAG |
| created\_at | TIMESTAMP | Plan creation timestamp |
| updated\_at | TIMESTAMP | Last modification timestamp |
| estimated\_duration | INTEGER | Estimated seconds to complete |

### 3.4.2 Plan Tasks Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| task\_id | VARCHAR(36) | Primary key, UUID |
| plan\_id | VARCHAR(36) | Foreign key to plans |
| task\_name | VARCHAR(200) | Human-readable task name |
| task\_type | VARCHAR(50) | Type of task to execute |
| assigned\_agent | VARCHAR(100) | Agent identifier |
| task\_config | TEXT | JSON configuration for task |
| estimated\_duration | INTEGER | Estimated seconds |
| status | VARCHAR(20) | pending, running, completed, failed |
| priority | INTEGER | Execution priority (1-10) |

## 3.5 API Specifications

### 3.5.1 Create Plan API

**Endpoint:** POST /api/planner/create

**Description:** Creates a new execution plan from user request

**Request Body:**

{ "user\_id": "string", "request": "string", "context": {}, "use\_existing\_dag": "string (optional)" }

### 3.5.2 Get Plan API

**Endpoint:** GET /api/planner/plan/:plan\_id

**Description:** Retrieves plan details and current status

**Returns:** Complete plan object with all tasks and dependencies

# 4. COMPONENT 2: DIRECTED ACYCLIC GRAPH (DAG)

## 4.1 Component Overview

The DAG component provides the structural framework for workflow execution. It represents tasks as nodes and dependencies as directed edges, ensuring correct execution order while enabling parallel processing where possible. The DAG engine manages task states, handles checkpoints, and provides resilience mechanisms.

## 4.2 Functional Requirements

### 4.2.1 Graph Management

* FR-DAG-001: System shall create and store DAG definitions
* FR-DAG-002: System shall validate DAG structure for cycles
* FR-DAG-003: System shall support DAG versioning and templates
* FR-DAG-004: System shall allow dynamic DAG modification during execution

### 4.2.2 Node Management

* FR-DAG-005: System shall track node states (pending, running, completed, failed)
* FR-DAG-006: System shall store node execution results and outputs
* FR-DAG-007: System shall support node retry logic and failure policies
* FR-DAG-008: System shall enable checkpoint creation for long-running tasks

### 4.2.3 Execution Control

* FR-DAG-009: System shall determine task readiness based on dependencies
* FR-DAG-010: System shall identify and schedule parallel execution paths
* FR-DAG-011: System shall propagate results between dependent nodes
* FR-DAG-012: System shall support DAG pause, resume, and termination

## 4.3 Technical Specifications

### 4.3.1 DAG Structure

DAGs are represented as JSON documents with the following structure:

{ "dag\_id": "uuid", "nodes": [ { "node\_id": "uuid", "task\_config": {}, "dependencies": ["node\_id1", "node\_id2"] } ], "edges": [ { "from": "node\_id", "to": "node\_id", "condition": "optional" } ] }

## 4.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| dags | DAG definitions and metadata |
| dag\_nodes | Individual nodes within DAGs |
| dag\_edges | Dependency relationships between nodes |
| dag\_executions | Runtime execution instances |
| node\_executions | Individual node execution records |

### 4.4.1 DAGs Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| dag\_id | VARCHAR(36) | Primary key, UUID |
| dag\_name | VARCHAR(200) | Human-readable name |
| dag\_version | VARCHAR(20) | Version identifier |
| dag\_definition | TEXT | JSON DAG structure |
| created\_by | VARCHAR(50) | Creator user ID |
| is\_template | BOOLEAN | Template flag |
| created\_at | TIMESTAMP | Creation timestamp |
| updated\_at | TIMESTAMP | Last update timestamp |

### 4.4.2 Node Executions Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| execution\_id | VARCHAR(36) | Primary key, UUID |
| dag\_execution\_id | VARCHAR(36) | FK to dag\_executions |
| node\_id | VARCHAR(36) | FK to dag\_nodes |
| status | VARCHAR(20) | pending, running, completed, failed |
| start\_time | TIMESTAMP | Execution start |
| end\_time | TIMESTAMP | Execution end |
| result\_data | TEXT | JSON execution results |
| error\_message | TEXT | Error details if failed |
| retry\_count | INTEGER | Number of retries |

# 5. COMPONENT 3: AUTONOMOUS DECISION AND EXECUTION

## 5.1 Component Overview

The Autonomous Decision and Execution component serves as the runtime orchestration engine, managing task delegation, agent coordination, progress monitoring, and conflict resolution. The Orchestrator acts as the central control hub enforcing execution policies and maintaining workflow integrity.

## 5.2 Functional Requirements

### 5.2.1 Task Delegation

* FR-ORC-001: System shall assign tasks to agents based on capability matching
* FR-ORC-002: System shall balance load across available agents
* FR-ORC-003: System shall handle agent unavailability and reassignment
* FR-ORC-004: System shall maintain a task queue for pending assignments

### 5.2.2 Real-time Monitoring

* FR-ORC-005: System shall track progress of all active tasks
* FR-ORC-006: System shall detect task timeouts and stalled executions
* FR-ORC-007: System shall identify resource bottlenecks
* FR-ORC-008: System shall provide real-time status updates via WebSocket

### 5.2.3 Conflict Resolution

* FR-ORC-009: System shall detect conflicting agent outputs
* FR-ORC-010: System shall apply resolution rules to reconcile conflicts
* FR-ORC-011: System shall escalate unresolved conflicts to human reviewers
* FR-ORC-012: System shall maintain audit trail of conflict resolutions

## 5.3 Technical Specifications

### 5.3.1 Orchestrator Architecture

The Orchestrator implements a state machine pattern with the following components:

* Task Scheduler: Determines execution order and timing
* Agent Manager: Tracks agent status and capabilities
* Execution Monitor: Observes task progress and health
* Result Aggregator: Collects and synthesizes outputs
* Policy Engine: Enforces rules and constraints

## 5.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| agents | Agent registry with capabilities |
| agent\_status | Current agent availability and load |
| task\_assignments | Task-to-agent assignments |
| execution\_logs | Detailed execution event logs |
| conflict\_resolutions | Record of conflict resolution actions |

### 5.4.1 Agents Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| agent\_id | VARCHAR(100) | Primary key, unique agent ID |
| agent\_name | VARCHAR(200) | Human-readable name |
| agent\_type | VARCHAR(50) | Agent classification |
| capabilities | TEXT | JSON array of skills |
| max\_concurrent | INTEGER | Max parallel tasks |
| status | VARCHAR(20) | active, inactive, maintenance |
| registered\_at | TIMESTAMP | Registration timestamp |
| last\_heartbeat | TIMESTAMP | Last activity timestamp |

### 5.4.2 Task Assignments Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| assignment\_id | VARCHAR(36) | Primary key, UUID |
| task\_id | VARCHAR(36) | FK to plan\_tasks |
| agent\_id | VARCHAR(100) | FK to agents |
| assigned\_at | TIMESTAMP | Assignment timestamp |
| started\_at | TIMESTAMP | Execution start |
| completed\_at | TIMESTAMP | Execution completion |
| status | VARCHAR(20) | assigned, running, completed |
| result\_summary | TEXT | Brief result description |

# 6. COMPONENT 4: MCP INTEGRATION

## 6.1 Component Overview

The Model Context Protocol (MCP) Integration component provides standardized communication channels for agents to access external tools, data sources, and services. It ensures consistent data exchange formats, enforces access controls, and facilitates context sharing across the multi-agent system.

## 6.2 Functional Requirements

### 6.2.1 Standardized Communication

* FR-MCP-001: System shall provide uniform API interface for all MCP endpoints
* FR-MCP-002: System shall support JSON-based data exchange format
* FR-MCP-003: System shall validate data against MCP schema definitions
* FR-MCP-004: System shall handle version compatibility for MCP endpoints

### 6.2.2 Tool Access

* FR-MCP-005: System shall register and catalog available external tools
* FR-MCP-006: System shall enable agents to invoke tool operations
* FR-MCP-007: System shall handle authentication for external systems
* FR-MCP-008: System shall retry failed external calls with backoff

### 6.2.3 Context Sharing

* FR-MCP-009: System shall maintain shared context repository
* FR-MCP-010: System shall enable agents to read/write context data
* FR-MCP-011: System shall version context data for consistency
* FR-MCP-012: System shall support context isolation between workflows

### 6.2.4 Data Access Control

* FR-MCP-013: System shall enforce role-based access to MCP endpoints
* FR-MCP-014: System shall log all data access attempts
* FR-MCP-015: System shall support data masking for sensitive fields
* FR-MCP-016: System shall validate permissions before data operations

## 6.3 Technical Specifications

### 6.3.1 MCP Endpoint Configuration

MCP endpoints are configured via JSON files with the following structure:

{ "endpoint\_id": "uuid", "endpoint\_name": "string", "endpoint\_url": "string", "auth\_type": "none|basic|oauth", "capabilities": ["read", "write"], "schema\_version": "1.0", "retry\_policy": { "max\_retries": 3, "backoff": "exponential" } }

## 6.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| mcp\_endpoints | Registered MCP endpoint configurations |
| mcp\_access\_log | Access audit trail |
| shared\_context | Cross-agent context data |
| context\_versions | Context versioning history |

### 6.4.1 MCP Endpoints Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| endpoint\_id | VARCHAR(36) | Primary key, UUID |
| endpoint\_name | VARCHAR(200) | Endpoint name |
| endpoint\_url | VARCHAR(500) | Base URL |
| endpoint\_config | TEXT | JSON configuration |
| is\_active | BOOLEAN | Active status |
| created\_at | TIMESTAMP | Registration timestamp |
| last\_accessed | TIMESTAMP | Last usage timestamp |

### 6.4.2 Shared Context Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| context\_id | VARCHAR(36) | Primary key, UUID |
| workflow\_id | VARCHAR(36) | Associated workflow |
| context\_key | VARCHAR(200) | Context identifier |
| context\_data | TEXT | JSON context data |
| version | INTEGER | Version number |
| created\_by | VARCHAR(100) | Creating agent |
| created\_at | TIMESTAMP | Creation timestamp |
| expires\_at | TIMESTAMP | Expiration timestamp |

# 7. COMPONENT 5: CHAT AND AGENT RUN UTILITY

## 7.1 Component Overview

The Chat and Agent Run Utility provides the user-facing interface for interacting with the multi-agent system. It includes a conversational interface for natural language requests, session management, execution tracing, and real-time feedback on workflow progress.

## 7.2 Functional Requirements

### 7.2.1 Conversational Interface

* FR-CHT-001: System shall accept natural language requests
* FR-CHT-002: System shall maintain conversation history
* FR-CHT-003: System shall support multi-turn dialogues
* FR-CHT-004: System shall display formatted agent responses

### 7.2.2 Session Management

* FR-CHT-005: System shall create and track chat sessions
* FR-CHT-006: System shall associate sessions with user accounts
* FR-CHT-007: System shall support session resumption
* FR-CHT-008: System shall allow session deletion and archival

### 7.2.3 Execution Tracing

* FR-CHT-009: System shall log all agent actions with timestamps
* FR-CHT-010: System shall capture decision points and rationale
* FR-CHT-011: System shall provide trace visualization tools
* FR-CHT-012: System shall export traces in standard formats

### 7.2.4 Real-time Feedback

* FR-CHT-013: System shall display current workflow status
* FR-CHT-014: System shall show progress indicators for tasks
* FR-CHT-015: System shall notify users of important events
* FR-CHT-016: System shall update status via WebSocket connections

## 7.3 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| chat\_sessions | User chat sessions |
| chat\_messages | Individual messages in sessions |
| execution\_traces | Detailed execution event traces |
| run\_history | Historical workflow executions |

### 7.3.1 Chat Sessions Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| session\_id | VARCHAR(36) | Primary key, UUID |
| user\_id | VARCHAR(50) | User identifier |
| session\_title | VARCHAR(200) | Session name |
| created\_at | TIMESTAMP | Session start time |
| last\_activity | TIMESTAMP | Last message time |
| status | VARCHAR(20) | active, archived, deleted |
| context\_data | TEXT | JSON session context |

### 7.3.2 Chat Messages Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| message\_id | VARCHAR(36) | Primary key, UUID |
| session\_id | VARCHAR(36) | FK to chat\_sessions |
| sender\_type | VARCHAR(20) | user, agent, system |
| sender\_id | VARCHAR(100) | Sender identifier |
| message\_text | TEXT | Message content |
| message\_metadata | TEXT | JSON additional data |
| created\_at | TIMESTAMP | Message timestamp |

## 7.4 UI Specifications

### 7.4.1 Chat Interface Layout

* Left sidebar: Session list with search and filter
* Center panel: Message history with scrolling
* Bottom input: Text area with send button
* Right panel: Workflow status and agent activity

### 7.4.2 Chat Interface Requirements

* No modal dialogs - all interactions inline
* Simple HTML tables for data display (no DataTables library)
* Bootstrap responsive design for mobile compatibility
* jQuery for dynamic content updates

# 8. COMPONENT 6: USER MANAGEMENT

## 8.1 Component Overview

The User Management component handles authentication, authorization, and profile management. It implements a simple file-based authentication system using a users.json file with cleartext password comparison. The component enforces role-based access control throughout the application.

## 8.2 Functional Requirements

### 8.2.1 Authentication

* FR-USR-001: System shall authenticate users via users.json file
* FR-USR-002: System shall compare passwords in cleartext (no hashing)
* FR-USR-003: System shall create session tokens upon successful login
* FR-USR-004: System shall validate session tokens on each request

### 8.2.2 Authorization

* FR-USR-005: System shall support role-based access control (RBAC)
* FR-USR-006: System shall define roles: admin, operator, viewer
* FR-USR-007: System shall enforce permissions on API endpoints
* FR-USR-008: System shall restrict data access based on roles

### 8.2.3 Profile Management

* FR-USR-009: System shall allow users to view their profiles
* FR-USR-010: System shall support password changes
* FR-USR-011: System shall store user preferences
* FR-USR-012: Admins shall manage user accounts via UI

## 8.3 Technical Specifications

### 8.3.1 users.json File Format

{ "users": [ { "user\_id": "admin001", "username": "admin", "password": "admin123", "full\_name": "System Administrator", "email": "admin@Abhikarta.com", "role": "admin", "active": true, "created\_at": "2025-01-01T00:00:00Z" } ] }

### 8.3.2 Role Definitions

| **Role** | **Permissions** |
| --- | --- |
| admin | Full system access, user management, configuration |
| operator | Create workflows, monitor executions, approve HITL |
| viewer | Read-only access to workflows and results |

## 8.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| user\_sessions | Active user sessions and tokens |
| user\_activity | User activity audit log |
| user\_preferences | User settings and preferences |

### 8.4.1 User Sessions Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| session\_id | VARCHAR(36) | Primary key, UUID |
| user\_id | VARCHAR(50) | User identifier |
| session\_token | VARCHAR(100) | Authentication token |
| ip\_address | VARCHAR(45) | Client IP address |
| user\_agent | TEXT | Client user agent |
| created\_at | TIMESTAMP | Session creation |
| expires\_at | TIMESTAMP | Session expiration |
| last\_activity | TIMESTAMP | Last activity time |

## 8.5 API Specifications

### 8.5.1 Login API

**Endpoint:** POST /api/auth/login

**Description:** Authenticates user and creates session

**Request Body:**

{ "username": "string", "password": "string" }

# 9. COMPONENT 7: NOTIFICATION MANAGEMENT

## 9.1 Component Overview

The Notification Management component delivers alerts and updates to users based on workflow events. It supports event-based triggers, customizable subscriptions, and multi-channel delivery including email, in-app notifications, and system alerts.

## 9.2 Functional Requirements

### 9.2.1 Event-Based Alerts

* FR-NOT-001: System shall trigger notifications on workflow events
* FR-NOT-002: System shall send alerts for task completion
* FR-NOT-003: System shall notify on workflow failures
* FR-NOT-004: System shall alert users when HITL intervention required

### 9.2.2 Subscription Management

* FR-NOT-005: Users shall subscribe to specific event types
* FR-NOT-006: Users shall configure notification preferences
* FR-NOT-007: System shall support role-based default subscriptions
* FR-NOT-008: Users shall mute notifications for specific workflows

### 9.2.3 Multi-Channel Delivery

* FR-NOT-009: System shall deliver in-app notifications
* FR-NOT-010: System shall send email notifications
* FR-NOT-011: System shall support notification templates
* FR-NOT-012: System shall batch notifications to reduce volume

## 9.3 Technical Specifications

### 9.3.1 Notification Event Types

| **Event Type** | **Description** | **Default Recipients** |
| --- | --- | --- |
| workflow\_started | New workflow execution begun | Workflow creator |
| workflow\_completed | Workflow finished successfully | Workflow creator |
| workflow\_failed | Workflow encountered error | Creator, admins |
| hitl\_required | Human intervention needed | Operators, admins |
| task\_timeout | Task exceeded time limit | Workflow creator |
| agent\_error | Agent execution error | Admins |

## 9.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| notifications | Individual notification records |
| notification\_subscriptions | User subscription preferences |
| notification\_templates | Message templates |
| notification\_delivery\_log | Delivery status tracking |

### 9.4.1 Notifications Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| notification\_id | VARCHAR(36) | Primary key, UUID |
| user\_id | VARCHAR(50) | Recipient user ID |
| event\_type | VARCHAR(50) | Type of event |
| title | VARCHAR(200) | Notification title |
| message | TEXT | Notification content |
| link\_url | VARCHAR(500) | Associated URL |
| priority | VARCHAR(20) | low, medium, high, urgent |
| status | VARCHAR(20) | unread, read, archived |
| created\_at | TIMESTAMP | Creation timestamp |
| read\_at | TIMESTAMP | Read timestamp |

### 9.4.2 Notification Subscriptions Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| subscription\_id | VARCHAR(36) | Primary key, UUID |
| user\_id | VARCHAR(50) | User identifier |
| event\_type | VARCHAR(50) | Subscribed event type |
| channel | VARCHAR(20) | in\_app, email |
| is\_enabled | BOOLEAN | Subscription active |
| created\_at | TIMESTAMP | Subscription creation |

# 10. COMPONENT 8: UI-BASED HUMAN-IN-THE-LOOP

## 10.1 Component Overview

The UI-based Human-in-the-Loop (HITL) component provides structured interfaces for human oversight and intervention in automated workflows. It enables validation checkpoints, contextual review of agent decisions, approval workflows, and feedback collection to improve system performance.

## 10.2 Functional Requirements

### 10.2.1 Validation Checkpoints

* FR-HTL-001: System shall pause workflows at defined checkpoints
* FR-HTL-002: System shall present checkpoint data to reviewers
* FR-HTL-003: System shall track checkpoint response times
* FR-HTL-004: System shall support multiple checkpoint types

### 10.2.2 Contextual Review

* FR-HTL-005: System shall display workflow history to reviewers
* FR-HTL-006: System shall show agent reasoning and confidence
* FR-HTL-007: System shall provide relevant context data
* FR-HTL-008: System shall highlight critical decision points

### 10.2.3 Approval Interface

* FR-HTL-009: System shall provide approve/reject/modify options
* FR-HTL-010: System shall allow reviewers to provide corrections
* FR-HTL-011: System shall require review comments for rejections
* FR-HTL-012: System shall resume workflow after approval

### 10.2.4 Feedback Integration

* FR-HTL-013: System shall capture human corrections
* FR-HTL-014: System shall store feedback for model retraining
* FR-HTL-015: System shall analyze feedback patterns
* FR-HTL-016: System shall adjust confidence thresholds based on feedback

## 10.3 Technical Specifications

### 10.3.1 Checkpoint Types

| **Type** | **Description** | **Use Case** |
| --- | --- | --- |
| validation | Validate intermediate results | Data quality checks |
| approval | Approve before proceeding | High-risk decisions |
| review | Review for information | Audit requirements |
| correction | Provide corrected data | Error recovery |

## 10.4 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| hitl\_checkpoints | Checkpoint definitions in workflows |
| hitl\_reviews | Human review records |
| hitl\_feedback | Feedback and corrections |
| hitl\_queue | Pending review items |

### 10.4.1 HITL Checkpoints Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| checkpoint\_id | VARCHAR(36) | Primary key, UUID |
| workflow\_id | VARCHAR(36) | Associated workflow |
| checkpoint\_type | VARCHAR(20) | Type of checkpoint |
| checkpoint\_name | VARCHAR(200) | Checkpoint name |
| task\_id | VARCHAR(36) | Associated task |
| status | VARCHAR(20) | pending, in\_review, completed |
| assigned\_to | VARCHAR(50) | Reviewer user ID |
| created\_at | TIMESTAMP | Creation timestamp |
| due\_at | TIMESTAMP | Response deadline |

### 10.4.2 HITL Reviews Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| review\_id | VARCHAR(36) | Primary key, UUID |
| checkpoint\_id | VARCHAR(36) | FK to hitl\_checkpoints |
| reviewer\_id | VARCHAR(50) | Reviewing user |
| decision | VARCHAR(20) | approved, rejected, modified |
| review\_comments | TEXT | Reviewer comments |
| corrections | TEXT | JSON corrected data |
| review\_duration | INTEGER | Seconds to review |
| reviewed\_at | TIMESTAMP | Review timestamp |

## 10.5 UI Specifications

### 10.5.1 Review Queue Interface

* Display pending reviews in simple HTML table
* Show priority, workflow name, checkpoint type
* Provide inline action buttons (no modals)
* Filter by status, priority, assigned user

### 10.5.2 Review Detail Page

* Top section: Workflow context and history
* Middle section: Data requiring review
* Bottom section: Decision buttons and comment field
* All content displayed inline without modals

# 11. ERROR MANAGEMENT

## 11.1 Overview

Error Management is a cross-cutting concern that handles detection, logging, recovery, and notification of errors throughout the system. It provides centralized error handling, automatic retry mechanisms, and escalation workflows.

## 11.2 Functional Requirements

* FR-ERR-001: System shall detect and categorize errors by type
* FR-ERR-002: System shall log all errors with stack traces
* FR-ERR-003: System shall implement automatic retry for transient errors
* FR-ERR-004: System shall escalate critical errors to administrators
* FR-ERR-005: System shall provide error recovery workflows

## 11.3 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| error\_logs | Detailed error records |
| error\_recovery | Recovery action history |

### 11.3.1 Error Logs Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| error\_id | VARCHAR(36) | Primary key, UUID |
| error\_type | VARCHAR(50) | Error category |
| error\_severity | VARCHAR(20) | low, medium, high, critical |
| error\_message | TEXT | Error description |
| stack\_trace | TEXT | Full stack trace |
| component | VARCHAR(100) | Component where error occurred |
| workflow\_id | VARCHAR(36) | Associated workflow if any |
| user\_id | VARCHAR(50) | Associated user if any |
| created\_at | TIMESTAMP | Error timestamp |
| resolved\_at | TIMESTAMP | Resolution timestamp |

# 12. JOB AND SESSION TRACKING

## 12.1 Overview

Job and Session Tracking provides comprehensive monitoring of workflow executions and user sessions. It maintains detailed records of job lifecycle, resource utilization, and performance metrics.

## 12.2 Functional Requirements

* FR-JOB-001: System shall create job records for each workflow execution
* FR-JOB-002: System shall track job status throughout lifecycle
* FR-JOB-003: System shall record resource utilization per job
* FR-JOB-004: System shall calculate job performance metrics
* FR-JOB-005: System shall provide job history and search

## 12.3 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| jobs | Job execution records |
| job\_metrics | Performance and resource metrics |

### 12.3.1 Jobs Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| job\_id | VARCHAR(36) | Primary key, UUID |
| plan\_id | VARCHAR(36) | FK to plans |
| user\_id | VARCHAR(50) | Job creator |
| job\_status | VARCHAR(20) | queued, running, completed, failed |
| priority | INTEGER | Execution priority |
| started\_at | TIMESTAMP | Job start time |
| completed\_at | TIMESTAMP | Job completion time |
| duration\_seconds | INTEGER | Total execution time |
| result\_summary | TEXT | Job outcome summary |

# 13. AGENT MONITORING

## 13.1 Overview

Agent Monitoring provides real-time visibility into agent health, performance, and activity. It tracks agent metrics, detects anomalies, and provides alerting for agent-related issues.

## 13.2 Functional Requirements

* FR-MON-001: System shall collect agent health metrics
* FR-MON-002: System shall track agent performance statistics
* FR-MON-003: System shall detect agent failures and timeouts
* FR-MON-004: System shall provide agent activity dashboard
* FR-MON-005: System shall alert on agent anomalies

## 13.3 Database Schema

| **Table Name** | **Description** |
| --- | --- |
| agent\_metrics | Agent performance metrics |
| agent\_health\_checks | Health check results |

### 13.3.1 Agent Metrics Table

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| metric\_id | VARCHAR(36) | Primary key, UUID |
| agent\_id | VARCHAR(100) | FK to agents |
| metric\_type | VARCHAR(50) | Type of metric |
| metric\_value | FLOAT | Metric value |
| recorded\_at | TIMESTAMP | Metric timestamp |

# 14. SYSTEM INTEGRATION

## 14.1 Kafka Messaging Integration

Kafka serves as the primary messaging backbone for asynchronous communication between system components. The following topics are defined:

| **Topic Name** | **Description** | **Producers** | **Consumers** |
| --- | --- | --- | --- |
| workflow.events | Workflow lifecycle events | Planner, Orchestrator | Notification, UI |
| task.assignments | Task delegation events | Orchestrator | Agents |
| task.results | Task completion results | Agents | Orchestrator, DAG |
| agent.heartbeat | Agent health status | Agents | Agent Monitor |
| hitl.requests | Human review requests | Orchestrator | HITL UI |
| notifications | User notifications | All components | Notification Service |
| errors | Error events | All components | Error Manager |

## 14.2 Database Configuration

The system supports two database backends configured via environment variables:

DATABASE\_TYPE=sqlite # or postgresql DATABASE\_PATH=/path/to/Abhikarta.db # For PostgreSQL: DATABASE\_HOST=localhost DATABASE\_PORT=5432 DATABASE\_NAME=Abhikarta DATABASE\_USER=Abhikarta\_user DATABASE\_PASSWORD=password

## 14.3 External System Integration

External systems integrate via MCP endpoints. Integration patterns include:

* REST API integration for synchronous operations
* Webhook callbacks for asynchronous events
* File-based data exchange for batch operations
* Database views for direct data access (read-only)

# 15. NON-FUNCTIONAL REQUIREMENTS

## 15.1 Performance Requirements

* NFR-001: System shall support 100 concurrent users
* NFR-002: API response time shall be under 500ms for 95th percentile
* NFR-003: UI page load time shall be under 2 seconds
* NFR-004: System shall handle 50 simultaneous workflow executions

## 15.2 Scalability Requirements

* NFR-005: System shall support horizontal scaling of agent workers
* NFR-006: Database shall accommodate 1 million workflow executions
* NFR-007: Kafka topics shall handle 1000 messages per second

## 15.3 Reliability Requirements

* NFR-008: System uptime shall be 99.5 percent or greater
* NFR-009: Data shall be backed up daily
* NFR-010: System shall recover from failures within 5 minutes

## 15.4 Security Requirements

* NFR-011: All API endpoints shall require authentication
* NFR-012: Session tokens shall expire after 24 hours
* NFR-013: System shall log all user actions for audit
* NFR-014: Database connections shall use SSL in production

## 15.5 Usability Requirements

* NFR-015: UI shall be accessible on desktop and tablet devices
* NFR-016: System shall provide contextual help text
* NFR-017: Error messages shall be user-friendly and actionable

# 16. DEPLOYMENT ARCHITECTURE

## 16.1 Deployment Components

The Abhikarta system consists of the following deployable components:

| **Component** | **Type** | **Technology** | **Ports** |
| --- | --- | --- | --- |
| Web Server | Application Server | Flask + Gunicorn | 8000 |
| Database | Data Store | SQLite / PostgreSQL | 5432 |
| Message Queue | Event Bus | Kafka | 9092 |
| Agent Workers | Background Workers | Python | N/A |
| Static Files | Web Server | Nginx (optional) | 80, 443 |

## 16.2 Recommended Infrastructure

* Minimum 4 CPU cores, 8GB RAM for application server
* Minimum 2 CPU cores, 4GB RAM per agent worker
* 100GB storage for database and logs
* Network connectivity between all components

## 16.3 Configuration Management

System configuration is managed through environment variables and configuration files:

* config/app\_config.json - Application settings
* config/users.json - User authentication data
* config/mcp\_endpoints.json - MCP endpoint definitions
* .env - Environment variables for secrets

# 17. APPENDICES

## 17.1 Glossary

| **Term** | **Definition** |
| --- | --- |
| Agent | Autonomous software component performing specialized tasks |
| DAG | Directed Acyclic Graph representing workflow structure |
| HITL | Human-in-the-Loop, human oversight in automated workflows |
| MCP | Model Context Protocol for standardized data exchange |
| Orchestrator | Central component coordinating agent activities |
| Planner | Component responsible for workflow planning |
| Workflow | Sequence of tasks executed to achieve a goal |

## 17.2 Acronyms

| **Acronym** | **Full Form** |
| --- | --- |
| API | Application Programming Interface |
| DAG | Directed Acyclic Graph |
| HITL | Human-in-the-Loop |
| JSON | JavaScript Object Notation |
| MCP | Model Context Protocol |
| RBAC | Role-Based Access Control |
| REST | Representational State Transfer |
| UI | User Interface |
| UUID | Universally Unique Identifier |

## 17.3 References

1. Flask Documentation: https://flask.palletsprojects.com/
2. Bootstrap Documentation: https://getbootstrap.com/
3. Apache Kafka Documentation: https://kafka.apache.org/
4. SQLite Documentation: https://www.sqlite.org/
5. PostgreSQL Documentation: https://www.postgresql.org/

**--- END OF DOCUMENT ---**