Project - High Level Design

On

Multi Agent Insurance System

Course Name: Agentic AI

Institution Name: Medi-caps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	MUSTAFA SHAIKH	EN22CS301624
2.	NAMAN MEHTA	EN22CS301629
3.	NAMAN RAGHUVANSHI	EN22CS301632
4.	NANDINI SHARMA	EN22CS301635
5.	NISHKARSH SHARMA	EN22CS301659

Group Name: Group 09D6

Project Number: AAI-20

Industry Mentor Name:

University Mentor Name: Prof. Nishant shrivastav

Academic Year:2026

Table of Contents

1.Introduction:

This High-Level Design (HLD) outlines a collaborative agent-based architecture designed for the insurance industry. It divides tasks between specialized Researcher and Writer agents. This setup automates complex policy analysis and reporting with high accuracy and traceability. The system shifts traditional insurance workflows into an independent multi-agent environment. This method reduces the "context overflow" and decision fatigue that often arise in large AI models when handling complex policy language.

1.1 Scope of the Document

This document defines:

- System architecture and component design
- Agent collaboration model
- Communication protocols
- Data design and persistence
- API contracts
- Security and performance considerations

Technologies covered:

- CrewAI (Agent orchestration)
- LangChain (Tooling & LLM integration)
- Ollama (Local LLM inference)
- MongoDB (State persistence)

1.2 Intended Audience

- AI Engineers
- Solution Architects
- Backend Engineers
- Technical Stakeholders in Insurance

Assumes familiarity with:

- Retrieval-Augmented Generation (RAG)
- Agentic AI workflows
- RESTful API design

1.3 System Overview

The system consists of two primary agents:

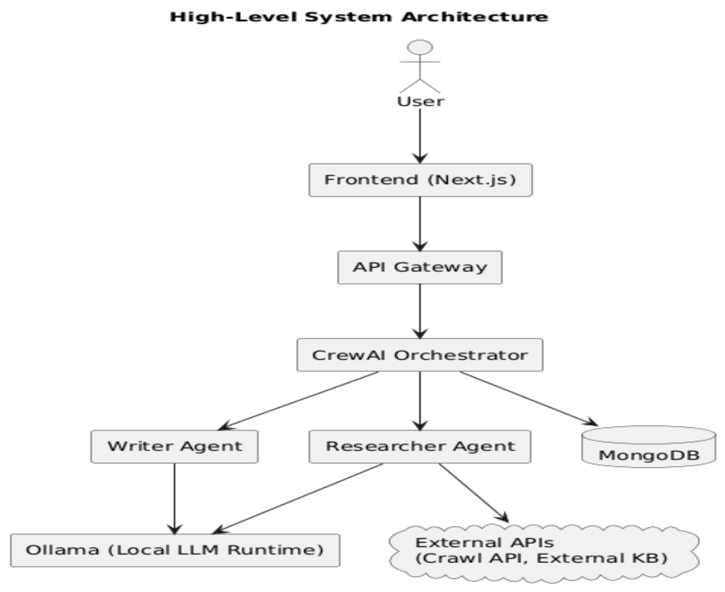
Researcher Agent

- Extracts policy clauses
- Queries internal knowledge base
- Uses external APIs (e.g., Crawl API)
- Produces structured JSON evidence

Writer Agent

- Consumes Researcher output
- Synthesizes structured findings
- Generates compliant insurance documentation

Orchestration is handled by **CrewAI** using sequential task execution and automatic context passing.


High-Level System Architecture
User
Frontend (Next.js)
API Gateway
CrewAI Orchestrator
Writer Agent
Researcher Agent
MongoDB
Ollama (Local LLM Runtime)
External APIs (Crawl API, External KB)

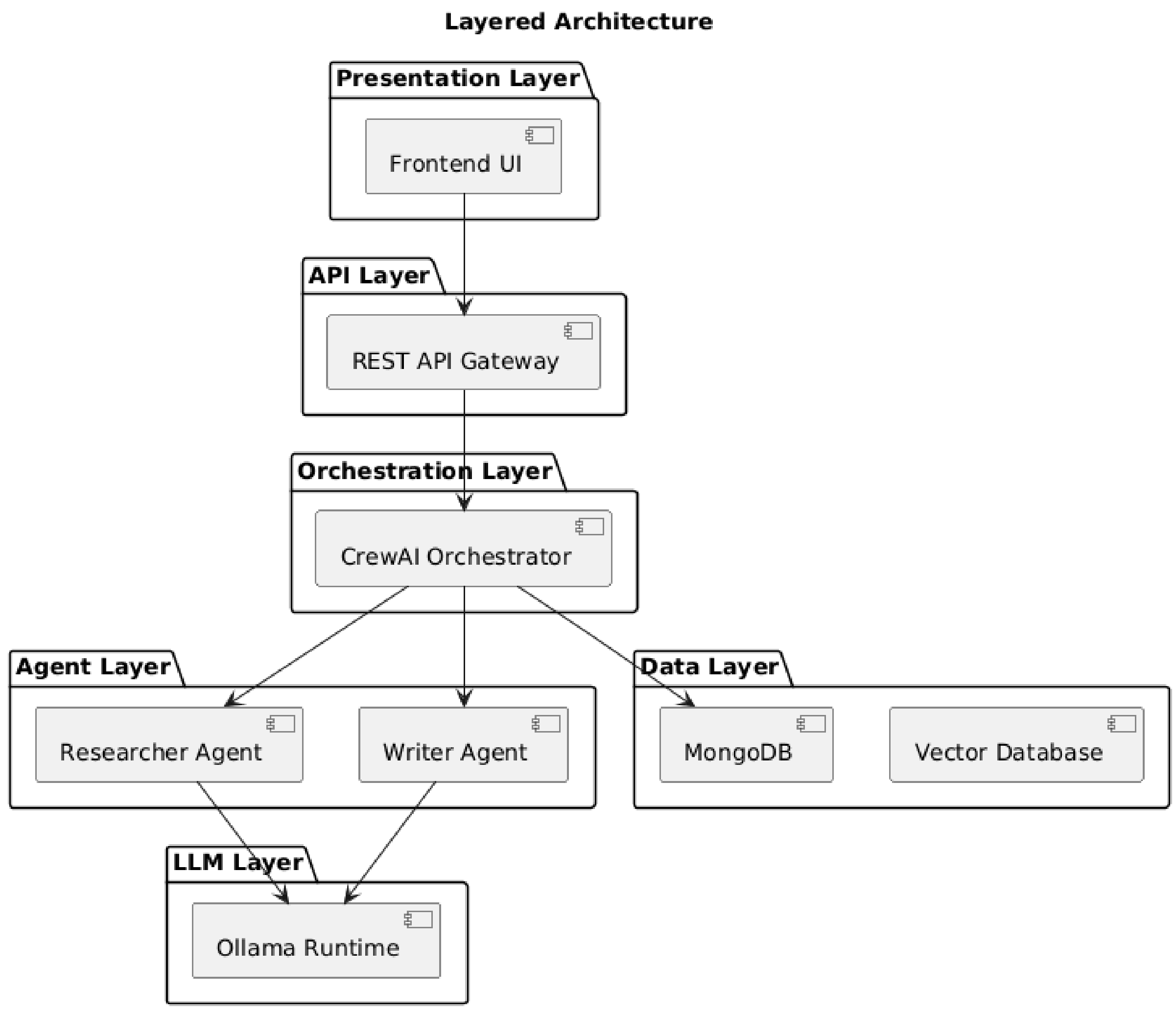
2. System Design

2.1 Application Design

The system follows a modular "Crew" pattern where each agent is an independent unit with:

- Role
- Goal
- Backstory
- Tools

Layer	Technology	Purpose
Orchestration	CrewAI	Task sequencing & collaboration
Framework	LangChain	Tool & LLM integration
LLM Runtime	Ollama	Local inference for privacy
Persistence	MongoDB	Workflow & result storage



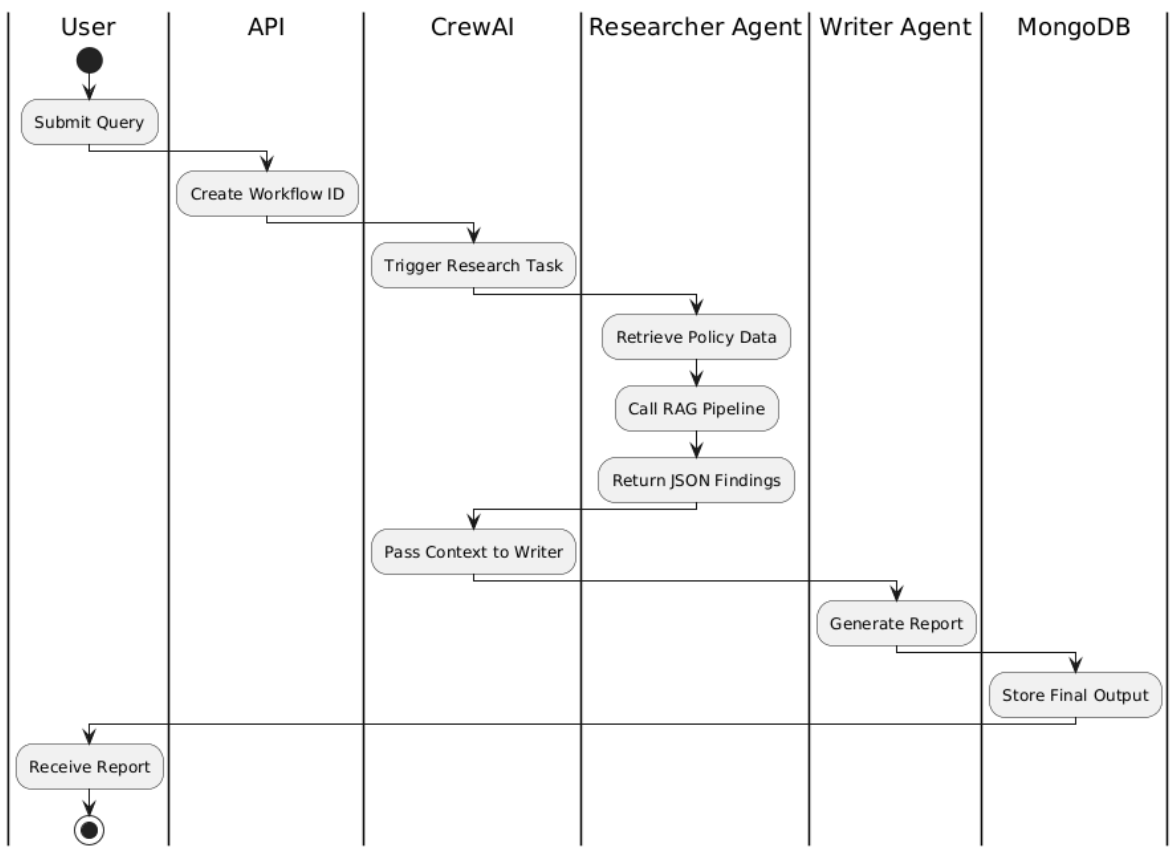
2.2 Process Flow

Step-by-Step Flow

1. **User Trigger**
    - User submits query via frontend.
    - API creates workflow ID.

2. **Research Phase**
    - Researcher retrieves relevant documents.
    - RAG pipeline performs semantic retrieval.
    - External API calls (if required).
    - Generates structured JSON findings.
3. **Context Transfer**
    - CrewAI automatically passes structured context.
    - No raw reasoning is transferred.
    - Prevents hallucination propagation.
4. **Synthesis Phase**
    - Writer generates formatted report.
    - Applies compliance rules.
    - Validates clarity and structure.
5. **Completion**
    - Final output stored in MongoDB.
    - Status updated to "Completed".
    - Response returned to user.

Sequential Workflow



2.3 Information Flow

The system uses structured contracts between agents.

Input

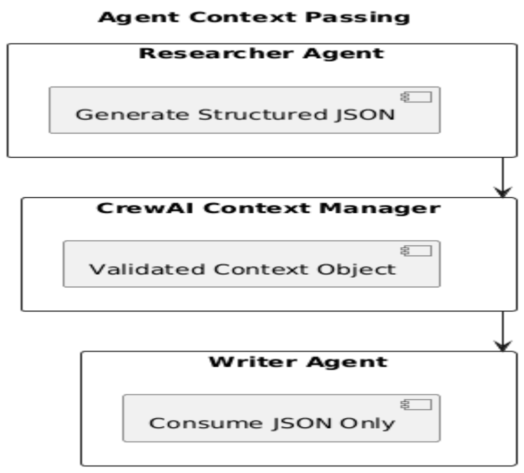
- Natural language query
- Metadata (policy ID, user role, jurisdiction)

Internal Context

- Managed by CrewAI
- Stored in temporary execution memory
- Contains validated structured objects only

Researcher Output JSON Example:

```
{
  "policy_id": "123",
  "coverage_summary": "...",
  "sources": ["internal_kb", "external_api"],
  "confidence_score": 0.92
}
```



2.4 Components Design

2.4.1 Researcher Agent

Purpose: Fact extraction and grounding

Responsibilities:

- Query vector database
- Perform similarity search
- Retrieve structured evidence
- Validate data consistency

Tools:

- Policy Coverage Search Tool (RAG-based)
- Crawl API Connector
- Data Validation Module

Output:

Structured JSON evidence with source traceability.

2.4.2 Writer Agent

Purpose: Structured reporting and compliance formatting

Responsibilities:

- Interpret research payload
- Generate clear coverage summary
- Apply compliance formatting rules
- Perform quality checks

Tools:

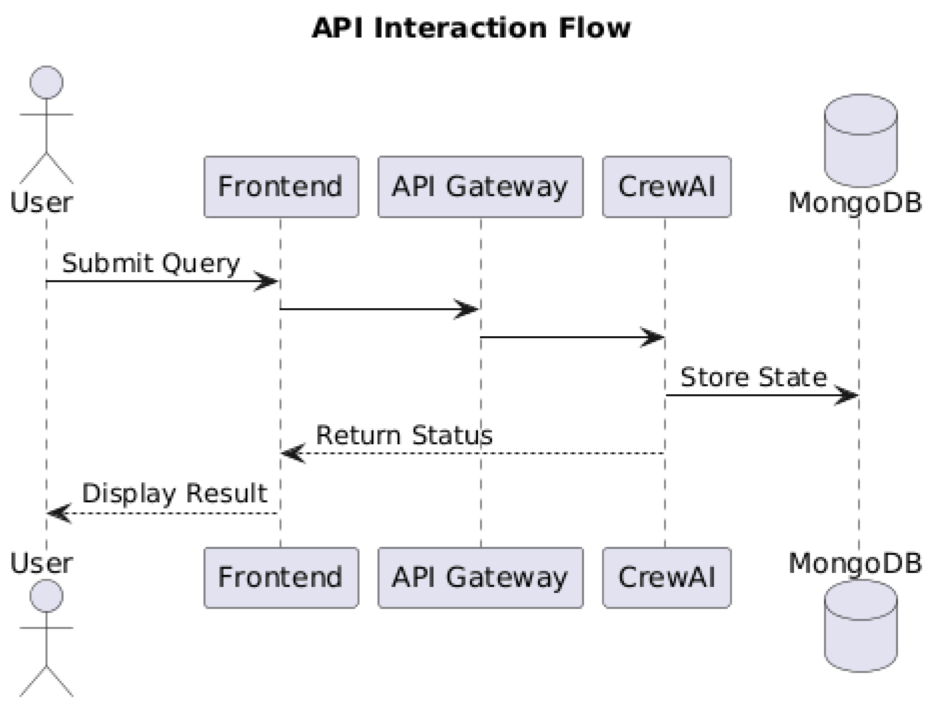
- Data Synthesis Engine
- Output Formatting Tool
- Compliance Validation Tool

2.5 Key Design Considerations

Agent Specialization: Isolating research from writing reduces the likelihood of hallucinations in the final report.

Context Passing: Using CrewAI's context parameter makes sure that the Writer gets accurate facts directly from the Researcher's output.

Local LLM (Ollama): Allows processing of sensitive policy data without exposure to public cloud LLMs; this ensures strict PII protection.



3. Data Design

3.1 Data Model

Three core collections:

1. workflow_states

- job_id
- user_id
- status
- timestamps

2. research_data

- raw clauses
- source URLs
- similarity scores

3. final_outputs

- formatted report
- PDF path
- quality metrics

3.2 Data Access Mechanism

The system uses a Semantic Retrieval-Augmented Generation model.

Document Ingestion

1. Recursive chunking (512–1024 tokens)
2. Embedding generation
3. Vector storage
4. Metadata tagging

Query Execution

1. Query embedding
2. Similarity search (Top-K)
3. Context retrieval

4. Agent reasoning

3.3 Data Retention Policies

The system adheres to regulatory requirements:

- Policy Records: Term + 3 years (P&C)
- Life/Annuity: In-force + 3 years
- Audit Logs: 5 years

- Cybersecurity Events: 5 years

Data encryption is applied at rest and in transit.

4. Interfaces

- **Agent-to-Agent**: Automatic context-sharing via CrewAI.
- **Agent-to-System**: REST API calls to external services like the Crawl API.

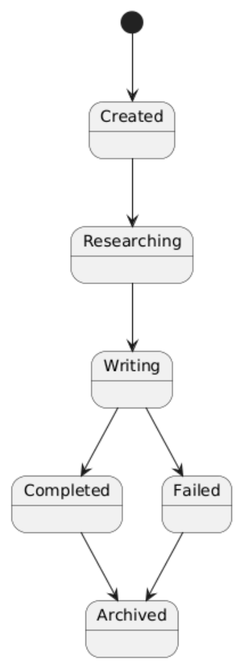
5. State and Session Management

- Short-term memory handled by CrewAI
- Long-term persistence in MongoDB Atlas
- Custom State Manager logs transitions
- Resume capability after failure

Workflow States:

- Created
- Researching
- Writing
- Completed
- Failed
- Archived

Workflow State Machine



6. Caching

To reduce latency and cost:

Semantic Caching

- Detects rephrased queries
- Similarity threshold ≈ 0.90

TTL-Based Caching

- Policy clauses → 30 days
- Regulatory laws → 7 days
- Claim status → 60 second

7. Non-Functional Requirements

7.1 Security Aspects

PII detection and masking before LLM processing

Zero Trust authentication

IAM integration (e.g., Amazon Cognito)

Local inference via Ollama (no public cloud LLM)

7.2 Performance Aspects

Circuit Breaker pattern

Exponential backoff (max 3 retries)

Distributed tracing (OpenTelemetry)

Centralized logging and metrics

8. References

1. Vyas | Medium, accessed on February 19, 2026, https://medium.com/@mukulvyasg/multi-agent-system-mas-architectures-a-comprehensive-guide-7387bb434d11
2. Introduction - CrewAI, accessed on February 19, 2026, https://docs.crewai.com/en/introduction
3. Multi-agent patterns in LlamaIndex | LlamaIndex Python ..., accessed on February 19, 2026, https://developers.llamaindex.ai/python/framework/understanding/agent/multi_agent/
4. The Complete Guide to Choosing an AI Agent Framework in 2025 - Langflow, accessed on February 19, 2026, https://www.langflow.org/blog/the-complete-guide-to-choosing-an-ai-agent-framework-in-2025
5. AI Agent Workflows for Insurance Claims Processing - Atlas ..., accessed on February 19, 2026, https://www.mongodb.com/docs/atlas/architecture/current/solutions-library/insurance-agentic-claims/
6. Semantic Caching and Memory Patterns for Vector Databases - Dataquest, accessed on February 19, 2026, https://www.dataquest.io/blog/semantic-caching-and-memory-patterns-for-vector-databases/