



Advanced Multi-Agent System Architecture

RAG + NLP + LLM + React + Flask



System Overview

Transform your project into a sophisticated multi-agent system with specialized AI agents handling different aspects of the insurance decision process.



Multi-Agent Architecture

Agent Hierarchy & Roles

mermaid

graph TD

```
A[User Query] --> B[Orchestrator Agent]
B --> C[Query Understanding Agent]
B --> D[Document Retrieval Agent]
B --> E[Business Rules Agent]
B --> F[Decision Making Agent]
B --> G[Quality Assurance Agent]
G --> H[Response Generation Agent]
H --> I[Audit Trail Agent]
I --> J[Final Response]

C --> K[Entity Extraction]
C --> L[Intent Classification]
D --> M[Semantic Search]
D --> N[Document Ranking]
E --> O[Rules Engine]
E --> P[Compliance Check]
F --> Q[Risk Assessment]
F --> R[Amount Calculation]
G --> S[Answer Validation]
G --> T[Confidence Scoring]
```



Specialized Agent Definitions

1. 📋 Orchestrator Agent (Master Controller)

Role: Coordinates all agents and manages workflow **Responsibilities:**

- Route queries to appropriate agents

- Manage agent communication
- Handle error recovery and fallbacks
- Monitor system performance

2. 🗣️ **Query Understanding Agent (NLP Specialist)**

Role: Advanced NLP processing and query comprehension **Responsibilities:**

- Named Entity Recognition (age, location, procedures, amounts)
- Intent classification (claim, coverage inquiry, policy check)
- Query expansion and context enhancement
- Sentiment analysis for urgency detection

3. 📖 **Document Retrieval Agent (RAG Specialist)**

Role: Intelligent document search and retrieval **Responsibilities:**

- Semantic search using vector embeddings
- Hybrid retrieval (dense + sparse)
- Document ranking and relevance scoring
- Context window optimization

4. ⚖️ **Business Rules Agent (Domain Expert)**

Role: Apply insurance business logic and regulations **Responsibilities:**

- Waiting period validation
- Age eligibility checks
- Coverage limit verification
- Exclusion clause analysis

5. 🎯 **Decision Making Agent (Core Reasoner)**

Role: Primary decision engine with LLM reasoning **Responsibilities:**

- Synthesize information from all agents
- Apply complex multi-criteria decision logic
- Calculate coverage amounts
- Generate preliminary decisions

6. 🔍 **Quality Assurance Agent (Validator)**

Role: Validate decisions and ensure quality **Responsibilities:**

- Cross-verify decisions against multiple criteria
- Confidence scoring and uncertainty detection
- Hallucination detection and correction
- Consistency checks across similar cases

7. 📝 **Response Generation Agent (Communicator)**

Role: Generate human-readable explanations **Responsibilities:**

- Create structured JSON responses
- Generate natural language explanations
- Format output for different audiences
- Ensure regulatory compliance in language

8. 📊 **Audit Trail Agent (Compliance Guardian)**

Role: Maintain detailed audit logs and compliance **Responsibilities:**

- Log all agent interactions and decisions
- Maintain regulatory audit trails
- Generate compliance reports
- Track performance metrics

🛠️ **Technology Stack**

Backend Architecture

```
python

# Multi-Agent Framework
- LangGraph for agent orchestration
- CrewAI for specialized agent roles
- LangChain for LLM integration
- Flask for REST API endpoints

# Core Technologies
- Python 3.10+
- FastAPI/Flask for APIs
- Redis for agent communication
- PostgreSQL for data persistence
- Celery for async processing
```

Frontend Architecture

```
javascript
```

```
// Modern React Stack
```

- React 18 with TypeScript
- Tailwind CSS for styling
- Zustand for state management
- React Query for API calls
- Socket.io for real-time updates
- Framer Motion for animations

AI/ML Stack

```
python
```

```
# LLM & NLP
```

- OpenAI GPT-4 / Claude / Llama
- spaCy for NLP processing
- Transformers for embeddings
- FAISS/Pinecone for vector storage
- Sentence-transformers for embeddings

Agent Communication Flow

Phase 1: Query Ingestion

```
json
```

```
{  
  "user_query": "46-year-old knee surgery in Pune, 3-month policy",  
  "session_id": "sess_123",  
  "timestamp": "2024-01-15T10:30:00Z"  
}
```

Phase 2: Agent Orchestration

```
python
```

```
# Orchestrator delegates to specialists
```

```
query_agent_result = await query_understanding_agent.process(user_query)  
retrieval_result = await document_retrieval_agent.search(expanded_query)  
rules_result = await business_rules_agent.evaluate(entities)
```

Phase 3: Decision Synthesis

```
python
```

```
# Decision agent combines all inputs
decision = await decision_making_agent.decide({
    "entities": query_agent_result,
    "context": retrieval_result,
    "rules_check": rules_result
})
```

Phase 4: Quality Validation

```
python

# QA agent validates decision
validated_decision = await qa_agent.validate(decision, context)
final_response = await response_agent.generate(validated_decision)
```

Frontend Agent Interface

Real-Time Agent Visualization

```
jsx
```

```
// Agent Activity Dashboard
const AgentDashboard = () => {
  return (
    <div className="grid grid-cols-4 gap-4 p-6">
      <AgentCard
        name="Query Understanding"
        status="processing"
        progress={75}
        result="Extracted: Age 46, Procedure: knee surgery"
      />
      <AgentCard
        name="Document Retrieval"
        status="complete"
        progress={100}
        result="Found 8 relevant policy clauses"
      />
      <AgentCard
        name="Business Rules"
        status="processing"
        progress={60}
        result="Checking waiting period..."
      />
      <AgentCard
        name="Decision Making"
        status="pending"
        progress={0}
        result="Awaiting inputs..."
      />
    </div>
  );
};
```

Interactive Decision Tree

```
jsx
```

```
const DecisionFlow = () => {
  return (
    <div className="decision-tree">
      <div className="agent-step completed">
        ✓ Query Understood: Age 46, Knee Surgery, 3mo Policy
      </div>
      <div className="agent-step processing">
        🔄 Checking Waiting Period: 3mo < 6mo required
      </div>
      <div className="agent-step pending">
        ⌚ Final Decision: Pending rule evaluation
      </div>
    </div>
  );
};
```

Flask API Architecture

Core API Endpoints

```
python
```

```

from flask import Flask, request, jsonify
from flask_socketio import SocketIO, emit
import asyncio

app = Flask(__name__)
socketio = SocketIO(app, cors_allowed_origins="*")

@app.route('/api/v1/process-query', methods=['POST'])
async def process_query():
    """Main endpoint for processing insurance queries"""
    data = request.get_json()

    # Initialize agent orchestrator
    orchestrator = InsuranceAgentOrchestrator()

    # Process with real-time updates
    result = await orchestrator.process_query_with_updates(
        query=data['query'],
        session_id=data['session_id'],
        callback=emit_agent_update
    )

    return jsonify(result)

@app.route('/api/v1/agents/status/<session_id>')
def get_agent_status(session_id):
    """Get current status of all agents"""
    status = agent_manager.get_session_status(session_id)
    return jsonify(status)

@socketio.on('connect')
def handle_connect():
    emit('connected', {'status': 'Connected to agent system'})

def emit_agent_update(agent_name, status, result):
    """Emit real-time agent updates"""
    socketio.emit('agent_update', {
        'agent': agent_name,
        'status': status,
        'result': result,
        'timestamp': datetime.now().isoformat()
    })

```

Agent Communication Layer

python


```

class AgentCommunicationHub:
    """Manages inter-agent communication"""

    def __init__(self):
        self.redis_client = redis.Redis()
        self.agent_registry = {}

    async def broadcast_to_agents(self, message, target_agents=None):
        """Broadcast message to specified agents"""
        for agent in target_agents or self.agent_registry.keys():
            await self.send_message(agent, message)

    async def collect_responses(self, agents, timeout=30):
        """Collect responses from multiple agents"""
        tasks = [agent.process_async() for agent in agents]
        results = await asyncio.gather(*tasks, timeout=timeout)
        return results

```



Advanced Features

1. Intelligent Agent Routing

```

python

class SmartOrchestrator:
    def route_query(self, query, context):
        # Simple query → Fast track with fewer agents
        if self.is_simple_query(query):
            return ["query_agent", "retrieval_agent", "decision_agent"]

        # Complex query → Full agent pipeline
        return ["query_agent", "retrieval_agent", "rules_agent",
                "decision_agent", "qa_agent", "response_agent"]

```

2. Agent Learning & Adaptation

```

python

class AdaptiveAgent:
    def learn_from_feedback(self, decision, outcome, feedback):
        """Agents learn from user feedback"""
        self.performance_tracker.record(decision, outcome)
        if feedback['rating'] < 3:
            self.retrain_component(decision.reasoning_path)

```

3. Multi-Modal Agent Capabilities

```
python

class MultiModalDocumentAgent:
    def process_document(self, file):
        if file.type == 'image':
            return self.ocr_agent.extract_text(file)
        elif file.type == 'pdf':
            return self.pdf_agent.extract_text(file)
        elif file.type == 'audio':
            return self.speech_agent.transcribe(file)
```

Agent Performance Monitoring

Real-Time Metrics Dashboard

```
python

@app.route('/api/v1/analytics/agents')
def agent_analytics():
    return {
        "agent_performance": {
            "query_agent": {"accuracy": 0.94, "avg_time": 1.2},
            "retrieval_agent": {"precision": 0.89, "recall": 0.92},
            "decision_agent": {"accuracy": 0.96, "confidence": 0.88}
        },
        "system_health": {
            "active_sessions": 45,
            "avg_response_time": 3.2,
            "error_rate": 0.02
        }
    }
```

College Project Advantages

Academic Excellence

- ✔ **Cutting-Edge Architecture:** Multi-agent systems are research-frontier
- ✔ **Multiple AI Techniques:** NLP + LLM + RAG + Agents
- ✔ **Real-World Complexity:** Industry-grade system design
- ✔ **Full-Stack Implementation:** Backend + Frontend + AI

Technical Innovation

✅ **Agent Specialization:** Each agent has distinct expertise ✅ **Collaborative AI:** Agents work together for better results ✅ **Real-Time Processing:** Live updates and streaming responses ✅ **Scalable**

Architecture: Can handle multiple simultaneous users

Demonstration Impact

✅ **Visual Agent Flow:** See each agent working in real-time ✅ **Explainable Decisions:** Trace exactly how decision was made ✅ **Interactive Interface:** Modern, responsive React frontend ✅ **Professional APIs:** RESTful services with documentation

🏆 Why This is EXCEPTIONAL for College

1. Research-Level Complexity 📊

- Multi-agent systems are graduate-level AI
- Combines 4+ major AI technologies seamlessly
- Addresses real business problem with production-quality solution

2. Industry Relevance 📁

- Microservices architecture (agents as services)
- Modern tech stack (React, Flask, AI agents)
- Scalable and maintainable codebase

3. Visual Impact ✨

- Watch agents collaborate in real-time
- Beautiful, modern UI with agent status indicators
- Interactive decision flows and explanations

4. Academic Rigor 🎓

- Extensive documentation and research paper potential
 - Performance metrics and evaluation framework
 - Comparative analysis of agent vs non-agent approaches
-

⚡ Implementation Timeline (10-12 weeks)

Week 1-2: Foundation

- Set up React + Flask architecture
- Implement basic agent framework

- Create agent communication layer

Week 3-4: Core Agents

- Query Understanding Agent (NLP)
- Document Retrieval Agent (RAG)
- Business Rules Agent (Logic)

Week 5-6: Decision Agents

- Decision Making Agent (LLM)
- Quality Assurance Agent (Validation)
- Response Generation Agent (Output)

Week 7-8: Frontend Integration

- Real-time agent visualization
- Interactive decision flows
- Performance dashboards

Week 9-10: Advanced Features

- Agent learning and adaptation
- Multi-modal capabilities
- Performance optimization

Week 11-12: Polish & Demo

- Comprehensive testing
- Documentation and presentation
- Demo video and deployment

Result: A cutting-edge multi-agent system that will absolutely wow your professors and demonstrate mastery of the latest AI technologies! 🚀