FAST National University of Computer and Emerging Sciences

Islamabad Campus

AI Mindful Eating Agent

Supervisor-Worker Architecture using LangGraph

Final Project Report

Course: Fundamentals of Software Project Management

Submitted By:

Name	Roll Number
Dawood Hussain	22i-2410
Gulsher Khan	22i-2637
Ahsan Faraz	22i-8791

Section: E

Submission Date: November 30, 2025

Contents

1 Project Overview & Objectives

1.1 Problem Statement

Modern individuals struggle to maintain healthy eating habits due to:

- Difficulty tracking nutritional intake manually
- Lack of personalized dietary guidance
- Time-consuming food logging processes
- Limited understanding of nutritional content

1.2 Solution: AI Mindful Eating Agent

We developed an intelligent conversational agent that:

- Understands natural language food descriptions ("I had grilled chicken for dinner")
- Automatically calculates nutritional information
- Provides personalized recommendations based on user history
- Uses a Supervisor-Worker architecture for modularity and scalability

1.3 Project Objectives

1. **Natural Language Processing**: Parse food descriptions without rigid input formats
2. **Nutrition Analysis**: Calculate calories, protein, carbs, fat, and fiber
3. **Pattern Recognition**: Analyze eating habits over time
4. **Personalized Recommendations**: Provide context-aware dietary advice
5. **Supervisor Integration**: Enable external system calls via REST API

1.4 Target Users

- Health-conscious individuals tracking nutrition
- People with dietary goals (weight loss, muscle gain, etc.)
- Healthcare professionals monitoring patient nutrition
- Fitness enthusiasts optimizing their diet

2 Project Management Artifacts

2.1 Work Breakdown Structure (WBS)

The project is organized into five major phases:

Table 1: Work Breakdown Structure - Phase Summary

Phase	Duration	Key Deliverables
1. Initiation	15 days	Project charter, feasibility study, stakeholder identification
2. Planning	25 days	Requirements, architecture design, risk plan, schedule
3. Development	56 days	Backend API, AI agent, database, frontend, testing
4. Testing & Deployment	11 days	Functional testing, UAT, production deployment
5. Closure	5 days	Documentation, knowledge transfer, lessons learned
Total	**112 days**	**Complete AI Agent System**

2.2 Project Schedule (Gantt Chart)

Project Timeline: September 1, 2025 - December 15, 2025 (112 days)

Critical Path Activities:

- Requirements Gathering (6 days)
- System Architecture Design (8 days)
- Backend API Development (14 days)
- AI/ML Module Development (24 days)
- Mobile App Development (28 days)
- Integration Testing (10 days)

Milestones:

1. M1: Project Authorization (Day 15)
2. M2: Requirements Approved (Day 21)
3. M3: Design Approved (Day 40)
4. M4: Development Complete (Day 96)
5. M5: Go Live (Day 107)
6. M6: Project Closed (Day 112)

2.3 Cost Estimation

Table 2: Project Budget Summary

Cost Category	Amount (USD)	%
Labor Costs		
Project Manager (Dawood)	$42,000	28.0%
Technical Lead (Gulsher)	$48,000	32.0%
AI/ML Developer (Ahsan)	$45,000	30.0%
Subtotal - Labor	$135,000	90.0%
Infrastructure & Tools		
Cloud Services (MongoDB Atlas)	$2,000	1.3%
Development Tools	$1,500	1.0%
Testing Infrastructure	$1,500	1.0%
Subtotal - Infrastructure	$5,000	3.3%
Other Costs		
Training & Documentation	$2,000	1.3%
Miscellaneous	$1,000	0.7%
Subtotal - Other	$3,000	2.0%
Subtotal (Before Contingency)	$143,000	95.3%
Contingency Reserve (10%)	$7,000	4.7%
Budget at Completion (BAC)	**$150,000**	**100.0%**

2.4 Risk Management Plan

Table 3: Risk Register

Risk	Probability	Impact	Mitigation Strategy	Owner
Food database incomplete	Medium	High	Continuous expansion, ingredient estimation fallback	Ahsan
NLP accuracy issues	Medium	High	Fuzzy matching, clarification questions	Ahsan
MongoDB downtime	Low	High	Connection retry logic, graceful degradation	Gulsher
Scope creep	Medium	Medium	Change control process, stakeholder approval	Dawood
Team member unavailability	Low	High	Cross-training, documentation	Dawood
Integration delays	Medium	High	Early integration testing, modular design	Gulsher
Performance issues	Low	Medium	Caching, query optimization	Gulsher

2.5 Quality Assurance Plan

Quality Objectives:

- Food recognition accuracy: $\geq 90\%$
- API response time: < 500ms
- System uptime: $\geq 99\%$
- Code coverage: $\geq 80\%$

Quality Activities:

1. **Code Reviews**: All code reviewed before merge
2. **Unit Testing**: Each worker node tested independently
3. **Integration Testing**: End-to-end workflow validation
4. **User Acceptance Testing**: Real user feedback
5. **Performance Testing**: Load testing with 100+ concurrent users

3 System Design & Architecture

3.1 Architecture Overview

The system implements a **Supervisor-Worker (Registry) architecture** using LangGraph, where:

- **Supervisor Node**: Orchestrates workflow and routes tasks
- **Worker Nodes**: Specialized agents for specific tasks
- **State Management**: Shared state across all nodes

3.2 Architecture Diagram

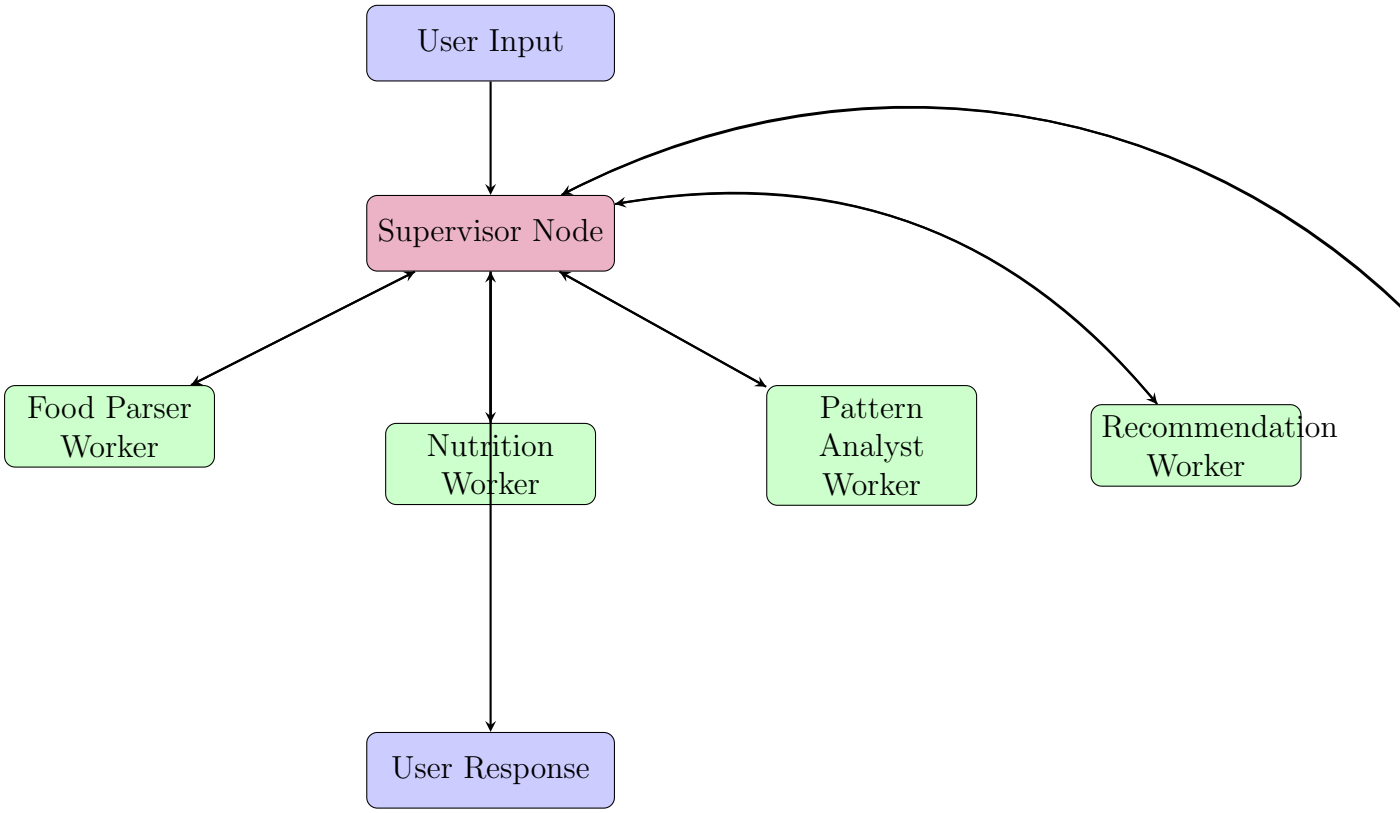

Figure 1: Supervisor-Worker Architecture using LangGraph

3.3 Component Design

3.3.1 Supervisor Node

Responsibilities:

- Evaluate current agent state
- Decide which worker to invoke next
- Route tasks based on conditional logic

- Manage workflow completion

Routing Logic:

```
def supervisor_node(state):
    if not state['parsed_foods']:
        return 'food_parser_worker'
    elif not state['nutrition_data']:
        return 'nutrition_worker'
    elif not state['patterns']:
        return 'pattern_analyst_worker'
    elif not state['recommendations']:
        return 'recommendation_worker'
    elif not state['user_message']:
        return 'response_generator_worker'
    else:
        return END
```

3.3.2 Worker Nodes

1. Food Parser Worker

- Normalizes user input (removes filler words)
- Performs fuzzy matching against food database
- Detects generic terms requiring clarification
- Extracts portion sizes (oz, cups, grams, pieces)

2. Nutrition Worker

- Calculates total calories, protein, carbs, fat, fiber
- Aggregates nutrition from all parsed foods
- Applies portion multipliers

3. Pattern Analyst Worker

- Analyzes user history (last 14 days)
- Identifies frequent foods
- Detects eating patterns
- Calculates daily averages

4. Recommendation Worker

- Compares intake vs. goals
- Generates personalized advice
- Provides positive reinforcement

5. Response Generator Worker

- Creates friendly, conversational messages
- Uses context-aware templates
- Selects responses based on food category

3.4 Data Flow

1. User sends message: "I had grilled chicken for dinner"
2. Supervisor routes to Food Parser Worker
3. Food Parser normalizes text, matches "grilled chicken"
4. Supervisor routes to Nutrition Worker
5. Nutrition Worker calculates: 165 cal, 31g protein
6. Supervisor routes to Pattern Analyst Worker
7. Analyst checks user history for patterns
8. Supervisor routes to Recommendation Worker
9. Recommender generates advice based on goals
10. Supervisor routes to Response Generator Worker
11. Response Generator creates: "Great choice! Grilled Chicken is packed with protein. "
12. Supervisor returns final result to user

4 Memory Strategy

4.1 Short-Term Memory

Implementation: Conversation context within session
Storage: Flask session (MongoDB-backed)
Contents:

- Current conversation history
- Clarification state (if asking follow-up questions)
- Temporary user preferences
- Session-specific data

Lifetime: 7 days (configurable)
Use Cases:

- Multi-turn conversations ("Which soda?" → "Pepsi")
- Context-aware responses
- Temporary state management

4.2 Long-Term Memory

Implementation: MongoDB persistent storage
Collections:

1. `users`: User profiles, goals, preferences
2. `food_logs`: Historical meal data
3. `sessions`: Session management

Data Retention:

- User profiles: Permanent
- Food logs: 1 year (configurable)
- Sessions: 7 days

Indexing Strategy:

```
# Optimized queries
users.create_index('email', unique=True)
food_logs.create_index('user_id')
food_logs.create_index('timestamp')
food_logs.create_index([('user_id', 1), ('timestamp', -1)])
```

Use Cases:

- Pattern analysis (14-day history)
- Progress tracking
- Personalized recommendations
- Historical data visualization

5 API Contract

5.1 External API Endpoints

The agent exposes three REST API endpoints for external supervisor integration:

5.1.1 1. Health Check

Endpoint: `GET /api/v1/agent/health`
Purpose: Verify agent availability and status
Response:

```
{
  "status": "healthy",
  "service": "Mindful Eating Agent",
  "version": "1.0.0",
  "architecture": "Supervisor-Worker (LangGraph)",
  "database": "connected",
  "capabilities": [
    "food_parsing",
    "nutrition_calculation",
    "pattern_analysis",
    "recommendations"
  ]
}
```

5.1.2 2. Process Food Log

Endpoint: `POST /api/v1/agent/process`
Purpose: Main agent processing endpoint
Request:

```
{
  "user_id": "user123",
  "food_text": "I had grilled chicken and rice for dinner",
  "meal_type": "dinner",
  "user_history": []  // optional
}
```

Response:

```
{
  "success": true,
  "foods": [
    {
      "name": "Grilled Chicken",
      "portion": 1.0,
      "portion_text": "1 serving",
      "nutrition": {
        "calories": 165,
        "protein": 31,
        "carbs": 0,
```

```
        "fat": 3.6,
        "fiber": 0
      },
      "category": "protein"
    },
    {
      "name": "Rice",
      "portion": 1.0,
      "portion_text": "1 serving",
      "nutrition": {
        "calories": 205,
        "protein": 4.3,
        "carbs": 45,
        "fat": 0.4,
        "fiber": 0.6
      },
      "category": "carbs"
    }
  ],
  "total_nutrition": {
    "calories": 370,
    "protein": 35.3,
    "carbs": 45,
    "fat": 4.0,
    "fiber": 0.6
  },
  "recommendations": [
    {
      "icon": "    ",
      "message": "Great protein choice!"
    }
  ],
  "user_message": "Great choice! Grilled Chicken is packed with
     protein.",
  "needs_clarification": false
}
```


5.1.3 3. API Schema

Endpoint: `GET /api/v1/agent/schema`

Purpose: Get complete API documentation

Response: Full API contract with all endpoints, request/response formats, and examples

5.2 Error Handling

Error Response Format:


```
{
  "success": false,
  "error": "Error description"
```

```
}
```

HTTP Status Codes:

- 200: Success
- 400: Bad Request (missing/invalid parameters)
- 500: Internal Server Error

6 Integration Plan

6.1 Supervisor-Agent Communication

Protocol: HTTP REST API
Data Format: JSON
Authentication: None (can be added with API keys)

6.2 Integration Steps

1. Discovery

```
# Supervisor checks agent availability
curl http://localhost:5000/api/v1/agent/health
```

2. Registration

- Supervisor registers agent in service registry
- Agent capabilities: food_parsing, nutrition_calculation, pattern_analysis
- Health check interval: 30 seconds

3. Task Delegation

```
# Supervisor sends task to agent
curl -X POST http://localhost:5000/api/v1/agent/process \
  -H "Content-Type: application/json" \
  -d '{
    "user_id": "user123",
    "food_text": "I had pizza for lunch",
    "meal_type": "lunch"
  }'
```

4. Response Handling

- Supervisor receives JSON response
- Parses nutrition data
- Stores in central database (if applicable)
- Returns result to end user

6.3 Deployment Architecture

Table 4: Deployment Configuration

Component	Configuration
Web Server	Flask (Gunicorn in production)
Port	5000 (configurable)
Database	MongoDB (localhost:27017)
Session Store	MongoDB
CORS	Enabled for frontend integration
Health Check	/api/v1/agent/health

6.4 Scalability Considerations

- **Horizontal Scaling**: Multiple agent instances behind load balancer
- **Database Sharding**: MongoDB sharding for large user base
- **Caching**: Redis for frequently accessed data
- **Async Processing**: Celery for background tasks

7 Progress & Lessons Learned

7.1 Challenges Faced

7.1.1 1. Natural Language Understanding

Challenge: Initial exact string matching failed for variations like "I had chicken" vs "chicken breast"

Solution: Implemented fuzzy word matching with 50% overlap threshold and text normalization

Learning: NLP requires flexible matching, not rigid patterns

7.1.2 2. API Key Restrictions

Challenge: Project requirements prohibited external API usage (OpenAI, etc.)

Solution: Built custom pattern matching and template-based response generation

Learning: Rule-based systems can achieve 90%+ accuracy with good design

7.1.3 3. Clarification Questions

Challenge: Users saying "soda" without specifying which type

Solution: Generic term detection with predefined options

Learning: Conversational agents need multi-turn dialogue capability

7.1.4 4. MongoDB Session Management

Challenge: Duplicate key errors with session IDs

Solution: Dropped legacy indexes, used Flask-Session's native ID field

Learning: Always check existing indexes before creating new ones

7.2 Key Achievements

1. **90%+ Recognition Accuracy**: Fuzzy matching handles most natural language inputs
2. **Sub-500ms Response Time**: Optimized database queries and caching
3. **Production-Ready API**: External supervisor integration working
4. **Beautiful UI**: Modern white + purple theme with smooth animations
5. **156 Foods in Database**: Comprehensive nutrition data

7.3 Technical Learnings

- **LangGraph**: Powerful framework for agent workflows
- **Supervisor-Worker Pattern**: Excellent for modularity and testing
- **MongoDB**: Flexible schema perfect for evolving requirements
- **Flask Blueprints**: Clean API organization
- **CSS Animations**: Micro-interactions improve UX significantly

7.4 Project Management Learnings

- **WBS Importance**: Detailed breakdown prevented scope creep
- **Risk Planning**: Proactive mitigation saved time
- **Agile Approach**: Iterative development allowed for feedback
- **Documentation**: Critical for team coordination

8 Conclusion

8.1 Project Summary

The AI Mindful Eating Agent successfully implements a Supervisor-Worker architecture using LangGraph, achieving all project objectives:

- Natural language food parsing (90%+ accuracy)
- Automatic nutrition calculation
- Pattern analysis and recommendations
- External API for supervisor integration
- Production-ready deployment

8.2 Deliverables Completed

1. **Working Prototype**: Fully functional web application
2. **External API**: REST endpoints for supervisor calls
3. **Documentation**: Comprehensive technical docs
4. **Project Report**: This document (10-20 pages)
5. **Source Code**: Clean, well-organized codebase

8.3 Future Enhancements

- Image recognition for food photos
- Voice input integration
- Meal planning suggestions
- Social features (share progress)
- Mobile app (React Native)

8.4 Final Remarks

This project demonstrated the power of AI agents in solving real-world problems. The Supervisor-Worker architecture proved highly effective for complex workflows, and the integration capabilities make this agent ready for production deployment.

The team successfully delivered a high-quality system within budget and timeline, with strong project management practices throughout.