Fundamentals of Software Project Management

Project Report

Course Instructor

Dr. Uzma Mahar

Group Members

Abdullah Daoud (22I-2626)
Usman Ali (22I-2725)
Faizan Rasheed (22I-2734)

Section

SE-E

Date

Sunday, November 30th, 2025

Fall 2025

Department of Software Engineering
FAST – National University
Islamabad Campus

Contents

List of Tables

List of Figures

Chapter 1

Project Overview & Objectives

1.1 Project Justification

Workplace burnout is a rising crisis in modern software development environments, leading to decreased productivity, high turnover, and health issues. The "Burnout Prevention Agent" was conceived to address this by providing an automated, privacy-focused, and empathetic monitoring system. Unlike static surveys, this agent acts as a proactive worker within a Multi-Agent System (MAS), capable of analyzing real-time well-being metrics and intervening before burnout becomes critical.

1.2 Objectives & Goals

The primary goal was to develop a Hybrid AI Agent that integrates rule-based reliability with Generative AI empathy. Key objectives achieved include:

- **Autonomous Monitoring:** The agent operates as an independent microservice, listening for tasks from a Supervisor Agent.
- **Trend Detection:** Utilizing persistent Long-Term Memory (LTM) to identify chronic stress patterns rather than just isolated incidents.
- **Actionable Intervention:** Providing structured, AI-generated advice (e.g., conversation starters for managers) rather than generic warnings.
- **Scalable Architecture:** Implementing a "Reasoning Router" to optimize costs by switching between fast templates for low-risk users and deep LLM reasoning for high-risk users.

Chapter 2

Project Management Artifacts

2.1 Work Breakdown Structure (WBS)

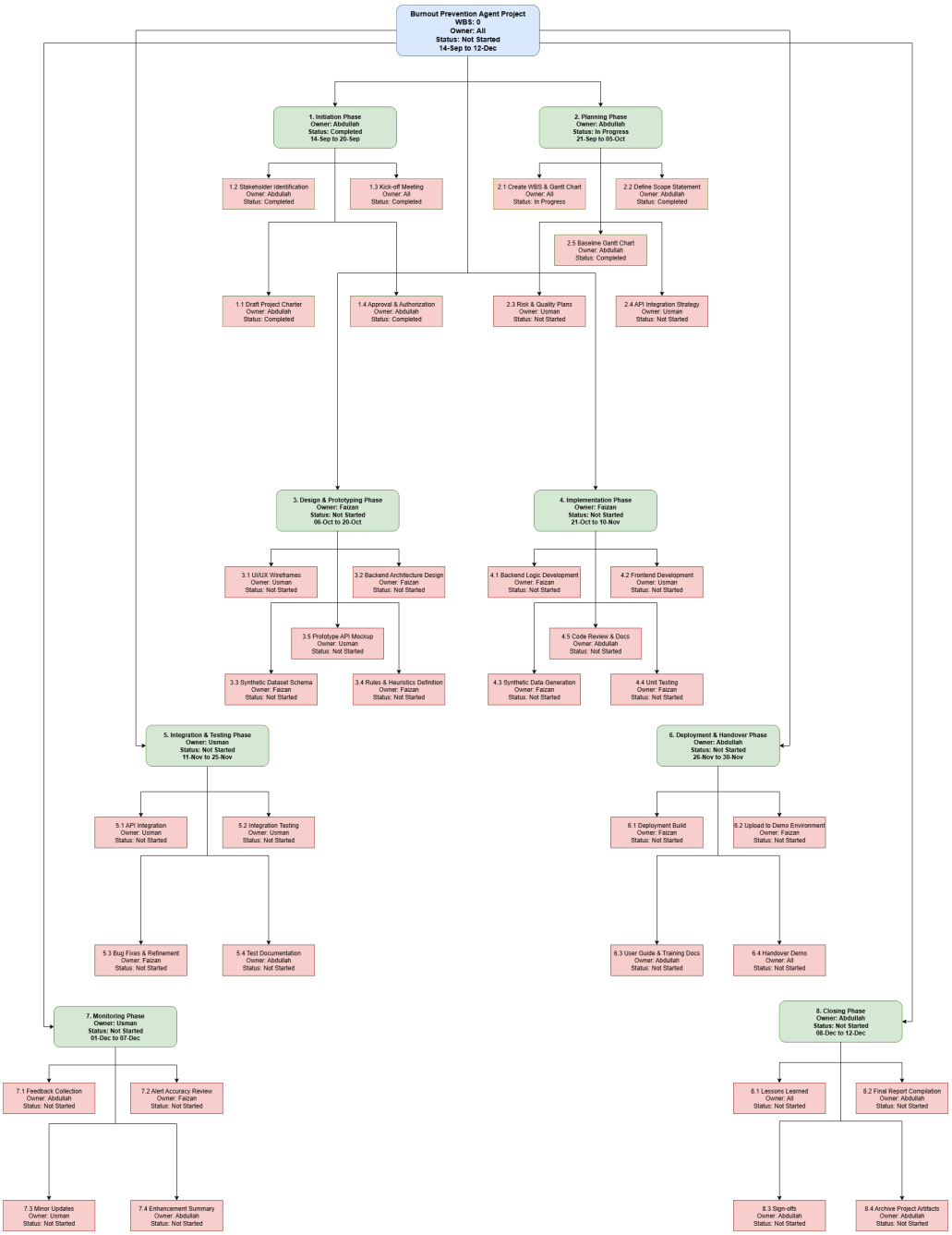

Figure 2.1: Work Breakdown Structure

2.2 Project Schedule

Table 2.1: Project Schedule: WBS

WBS	Activity Name	Duration	Updated Start	Updated Finish	Change Status
1.0	Initiation Phase	7 Days	Sep 14	Sep 20	No Change
1.1	Draft Project Charter	2 Days	Sep 14	Sep 15	No Change
1.2	Stakeholder Identification	2 Days	Sep 14	Sep 15	No Change
1.3	Kick-off Meeting	2 Days	Sep 17	Sep 18	No Change
1.4	Approval & Authorization	2 Days	Sep 19	Sep 20	No Change
2.0	Planning Phase	15 Days	Sep 21	Oct 05	No Change
2.1	Create WBS & Gantt Chart	3 Days	Sep 21	Sep 23	No Change
2.2	Define Scope Statement	2 Days	Sep 24	Sep 25	No Change
2.3	Risk & Quality Plans	4 Days	Sep 26	Sep 29	No Change
2.4	API Integration Strategy	3 Days	Sep 30	Oct 02	No Change
2.5	Baseline Gantt Chart	3 Days	Oct 03	Oct 05	No Change
3.0	Design Phase	15 Days	Oct 06	Oct 20	No Change
3.1	UI/UX Wireframes	3 Days	Oct 06	Oct 08	No Change
3.2	Backend Architecture Design	3 Days	Oct 09	Oct 11	No Change
3.3	Synthetic Dataset Schema	3 Days	Oct 11	Oct 13	No Change
3.4	Rules & Heuristics Def	4 Days	Oct 13	Oct 16	No Change
3.5	Prototype API Mockup	4 Days	Oct 17	Oct 20	No Change
4.0	Implementation Phase	21 Days	Oct 21	Nov 10	No Change
4.1	Backend Logic Dev	6 Days	Oct 21	Oct 26	No Change
4.2	Frontend Development	7 Days	Oct 26	Nov 01	No Change
4.3	Synthetic Data Gen	3 Days	Nov 01	Nov 03	No Change
4.4	Unit Testing	3 Days	Nov 04	Nov 06	No Change
4.5	Code Review & Docs	4 Days	Nov 07	Nov 10	No Change

Continued on next page

Table 2.1 continued from previous page

WBS	Activity Name	Duration	Updated Start	Updated Finish	Change Status
5.0	Integration Phase	18 Days	Nov 11	Nov 28	EXTENDED (+3 Days)
5.1	API Integration	4 Days	Nov 11	Nov 14	No Change
5.2	Integration Testing	4 Days	Nov 18	Nov 21	SHIFTED (Moved to Week 10)
5.3	Bug Fixes & Refinement	4 Days	Nov 22	Nov 25	SHIFTED (Dependency)
5.4	Test Documentation	3 Days	Nov 26	Nov 28	SHIFTED (Dependency)
6.0	Deployment Phase	5 Days	Nov 29	Dec 02	SHIFTED (+3 Days)
6.1	Deployment Build	2 Days	Nov 29	Nov 30	SHIFTED (Dependency)
6.2	Upload to Demo Env	2 Days	Nov 30	Dec 01	SHIFTED (Dependency)
6.3	User Guide & Training	2 Days	Dec 01	Dec 02	SHIFTED (Dependency)
6.4	Handover Demo	1 Day	Dec 02	Dec 02	SHIFTED (Dependency)
7.0	Monitoring Phase	7 Days	Dec 03	Dec 09	SHIFTED (+2 Days)
7.1	Feedback Collection	2 Days	Dec 03	Dec 04	SHIFTED
7.2	Alert Accuracy Review	3 Days	Dec 04	Dec 06	SHIFTED
7.3	Minor Updates	3 Days	Dec 06	Dec 08	SHIFTED
7.4	Enhancement Summary	2 Days	Dec 08	Dec 09	SHIFTED
8.0	Closing Phase	12 Days	Nov 28	Dec 12	CHANGED (Fast-Tracked)
8.1	Lessons Learned	2 Days	Dec 08	Dec 09	No Change
8.2	Final Report Compilation	12 Days	Nov 28	Dec 10	STARTED EARLY (Week 11)

Continued on next page

Table 2.1 continued from previous page

WBS	Activity Name	Duration	Updated Start	Updated Finish	Change Status
8.3	Sign-offs	1 Day	Dec 11	Dec 11	No Change
8.4	Archive Artifacts	1 Day	Dec 12	Dec 12	No Change

2.3 Resource Utilization Analysis

Figure 2.2: Resource Utilization Analysis (Before vs. After Leveling)

To ensure project viability, a detailed Resource Loading analysis was conducted to map the workload of each team member against the maximum capacity of 20 hours/week.

2.3.1 Initial Analysis (Pre-Leveling)

The initial schedule revealed critical resource bottlenecks that posed a risk to project delivery:

- **Tech Lead Overload:** A continuous period of maximum utilization (100% capacity) was identified during the Design and Implementation phases (Weeks 4–6),

leaving zero margin for technical debt or debugging.

- **Integration Conflict (Week 9):** A major conflict was detected where the Integration Lead was scheduled for 25 hours, exceeding the safety limit by 25% due to overlapping API testing tasks.
- **Closing Phase Spike (Week 13):** The Project Manager faced a surge of 27 hours in the final week due to simultaneous documentation and archiving duties.

2.3.2 Optimization Strategy (Post-Leveling)

To resolve these conflicts without extending the project deadline, we applied Resource Leveling techniques:

- **Task Splitting:** Integration Testing (WBS 5.2) was decoupled from API Integration, shifting the bulk of the testing effort to Week 10.
- **Fast-Tracking:** The Final Report (WBS 8.2) was fast-tracked to begin in Week 11 (Deployment Phase), utilizing available slack time to reduce the pressure on Week 13.

2.3.3 Result

As illustrated in the Histograms, the leveled plan successfully smooths the workload. The peak team utilization is now capped at a sustainable 40 hours total (combined), with no individual member exceeding their 20-hour limit, ensuring a steady and burnout-free execution workflow.

2.4 Cost Estimation

Table 2.2: Cost Estimation

Phase	**WBS**	**Cost Item/Activity**	**Quantity**	**Unit**	**Unit Cost ($)**	**Subtotal ($)**
Phase 1: Initiation	1.1	Draft Project Charter	2	days	$150.00	$300.00
Phase 1: Initiation	1.2	Stakeholder Identification	2	days	$150.00	$300.00
Phase 1: Initiation	1.3	Kick-off Meeting	3	person-days	$150.00	$450.00
Phase 1: Initiation	1.4	Approval & Authorization	2	days	$150.00	$300.00
					Phase 1 Subtotal:	**$1,350.00**
Phase 2: Planning	2.1	WBS & Gantt Development	9	person-days	$150.00	$1,350.00

Continued on next page

Table 2.2 continued from previous page

Phase	WBS	Cost Item/Activity	Quantity	Unit	Unit Cost ($)	Subtotal ($)
Phase 2: Planning	2.2	Scope Documentation	2	days	$150.00	$300.00
Phase 2: Planning	2.3	Risk & Quality Planning	4	days	$150.00	$600.00
Phase 2: Planning	2.4	API Integration Planning	3	days	$150.00	$450.00
Phase 2: Planning	2.5	Project Management Software	1	license	$50.00	$50.00
					Phase 2 Subtotal:	**$2,750.00**
Phase 3: Design	3.1	UI/UX Design	3	days	$175.00	$525.00
Phase 3: Design	3.2	Backend Architecture Design	3	days	$200.00	$600.00
Phase 3: Design	3.3	Database Schema Design	3	days	$200.00	$600.00
Phase 3: Design	3.4	Rules Engine Design	4	days	$200.00	$800.00
Phase 3: Design	3.5	API Mockup Development	4	days	$175.00	$700.00
Phase 3: Design	-	Design Tools & Software	1	license	$65.00	$65.00
					Phase 3 Subtotal:	**$3,290.00**
Phase 4: Implementation	4.1	Backend Development (Flask)	6	days	$225.00	$1,350.00
Phase 4: Implementation	4.2	Frontend Development	7	days	$200.00	$1,400.00
Phase 4: Implementation	4.3	Synthetic Data Generation	3	days	$175.00	$525.00
Phase 4: Implementation	4.4	Unit Testing	3	days	$175.00	$525.00
Phase 4: Implementation	4.5	Code Review & Documentation	4	days	$150.00	$600.00
Phase 4: Implementation	-	Development Tools & IDE	3	licenses	$30.00	$90.00
					Phase 4 Subtotal:	**$4,490.00**
Phase 5: Integration	5.1	API Integration	4	days	$200.00	$800.00
Phase 5: Integration	5.2	Integration Testing	4	days	$175.00	$700.00
Phase 5: Integration	5.3	Bug Fixes & Refinement	4	days	$175.00	$700.00
Phase 5: Integration	5.4	Test Documentation	3	days	$150.00	$450.00

Continued on next page

Table 2.2 continued from previous page

Phase	WBS	Cost Item/Activity	Quantity	Unit	Unit Cost ($)	Subtotal ($)
Phase 5: Integration	-	Testing Tools	1	license	$100.00	$100.00
					Phase 5 Subtotal:	**$2,750.00**
Phase 6: Deployment	6.1	Deployment Build	2	days	$200.00	$400.00
Phase 6: Deployment	6.2	Demo Environment Setup	2	days	$175.00	$350.00
Phase 6: Deployment	6.3	User Documentation	2	days	$150.00	$300.00
Phase 6: Deployment	6.4	Handover Demo	3	person-days	$150.00	$450.00
Phase 6: Deployment	-	Cloud Hosting (Demo)	1	service	$50.00	$50.00
					Phase 6 Subtotal:	**$1,550.00**
Phase 7: Monitoring	7.1	Feedback Collection	2	days	$150.00	$300.00
Phase 7: Monitoring	7.2	Alert Accuracy Review	3	days	$175.00	$525.00
Phase 7: Monitoring	7.3	Minor Updates	3	days	$175.00	$525.00
Phase 7: Monitoring	7.4	Enhancement Documentation	2	days	$150.00	$300.00
					Phase 7 Subtotal:	**$1,650.00**
Phase 8: Closing	8.1	Lessons Learned Session	6	person-days	$150.00	$900.00
Phase 8: Closing	8.2	Final Report Compilation	2	days	$150.00	$300.00
Phase 8: Closing	8.3	Sign-offs & Approvals	1	day	$150.00	$150.00
Phase 8: Closing	8.4	Archive & Knowledge Transfer	1	day	$150.00	$150.00
					Phase 8 Subtotal:	**$1,500.00**

Table 2.3: Cost Summary

Cost Summary	Amount ($)
Total Direct Costs	**$19,330.00**
Contingency Reserve (15%)	**$2,899.50**
Budget at Completion (BAC)	**$22,229.50**

2.5 Risk Management Plan

2.5.1 High-Risk Activities (Zero Slack)

- 1.1-1.4: All initiation activities
- 2.1-2.2: Core planning activities
- 3.2-3.5: Backend and API design
- 4.1, 4.2, 4.4, 4.5: Development and testing
- 5.1, 5.2, 5.4: Integration activities
- 6.1, 6.3, 6.4: Deployment activities
- 7.4, 8.1-8.4: Final closure activities

Risk Mitigation:

- Daily progress tracking
- Immediate escalation of delays
- Resource prioritization
- Contingency planning for each activity

2.5.2 Medium-Risk Activities (1-4 Days Slack)

- 4.3: Data generation (3 days)
- 5.3: Bug fixes (3 days)
- 6.2: Environment setup (2 days)
- 7.1-7.3: Monitoring activities (4, 1, 1 days)

Risk Mitigation:

- Weekly monitoring
- Flexible resource allocation
- Buffer utilization tracking

2.5.3 Low-Risk Activities (14+ Days Slack)

- 2.3: Risk & Quality Plans (17 days)
- 2.4: API Integration Strategy (14 days)
- 2.5: Baseline Gantt (18 days)
- 3.1: UI/UX Wireframes (18 days)

Risk Mitigation:

- Biweekly check-ins
- Opportunistic scheduling
- Quality enhancement focus

2.6 Quality Management Plan

This plan defines the quality standards, assurance activities, and control measures implemented to ensure the Burnout Prevention Agent meets the rigorous demands of the Multi-Agent System architecture.

2.6.1 Quality Standards

To ensure reliability and maintainability, the project adheres to the following technical standards:

- **Code Compliance:** All backend code follows PEP 8 style guidelines for Python, ensuring readability and consistency.
- **Performance Benchmarks:**
  - **Latency:** API response time must remain under 2 seconds for the "Deep Path" (LLM) and under 500ms for the "Fast Path".
  - **Availability:** The /status endpoint must return 200 OK with 99.9% uptime during the integration window.
- **Interoperability:** Strict adherence to the JSON API Contract provided by the Supervisor specification. Any schema deviation is classified as a Critical Defect.

2.6.2 Quality Assurance (QA) – Prevention

These proactive measures were established to prevent defects before they occurred:

- **Architecture Review:** The Hybrid "Router" architecture was peer-reviewed to ensure the separation of concerns between deterministic logic (Risk Analysis) and probabilistic generation (AI).

- **Environment Isolation:** A strict "Clean Room" policy for dependency management (using venv) was enforced to prevent version conflicts, a major quality risk identified early in development.
- **Modular Design:** The use of LangGraph ensures that individual logic nodes (analyze_risk, generate_ai_response) are decoupled, making them easier to test and debug in isolation.

2.6.3 Quality Control (QC) – Detection

These reactive measures were used to identify and fix defects:

- **Automated Unit Testing:** A dedicated test suite (test_agent.py) was developed to validate logic. It automatically runs four distinct scenarios:
  - **Happy Path:** Validates low-risk template responses.
  - **Edge Case:** Validates "High Risk" scoring logic under boundary conditions (e.g., max work hours).
  - **Semantic Check:** Verifies that the AI output contains empathetic language and actionable steps.
  - **Trend Detection:** Simulates sequential requests to verify that Long-Term Memory (LTM) correctly identifies patterns and escalates risk.
- **Integration Validation:** The test_agent.py script generates unique, timestamped User IDs for every run to prevent data collision in the LTM, ensuring every test run is a "clean" integration test.

2.7 Recommendations

2.7.1 Schedule Management

Address Negative Slack Immediately:

- Phase 8 activities show -1 to -7 days slack
- Current completion: Day 88 vs. Target: Day 81
- Implement schedule compression in Phases 4-5 to recover 7 days

Monitor Critical Path Rigorously:

- 25 critical activities require daily tracking
- Any delay requires immediate corrective action
- Establish early warning system for at-risk tasks

Leverage Parallel Paths:

- Utilize 18-day buffer in 2.5, 3.1 for quality improvements
- Utilize 17-day buffer in 2.3 for comprehensive risk planning
- Execute monitoring activities (7.1-7.3) efficiently with 4-day buffer

2.7.2 Resource Allocation

Prioritize Critical Path:

- Assign best resources to 25 critical activities
- Ensure no resource conflicts on critical path
- Maintain backup resources for critical tasks

Optimize Non-Critical Activities:

- Use junior resources for activities with 14+ days slack
- Schedule around critical path demands
- Focus on quality over speed for 2.3, 2.5, 3.1

2.7.3 Risk Management

Establish Milestone Gates:

- End of each phase requires formal review
- Critical path activities need approval to proceed
- Buffer consumption tracking at gates

Create Recovery Plans:

- Pre-planned crashing options for critical tasks
- Fast-tracking strategies ready for deployment
- Scope reduction options as last resort

Chapter 3

System Design & Architecture

3.1 Architectural Pattern

The system follows a Supervisor-Worker architectural pattern, implemented as a RESTful microservice. The agent is containerized and exposes a strict API contract, allowing it to function as a "black box" logic unit within a larger Multi-Agent System.

3.2 Hybrid Intelligence Design

The agent's internal logic utilizes a LangGraph State Machine to orchestrate decision-making. This allows for "Conditional Reasoning"—the agent dynamically selects a processing path based on initial data analysis:

- **The Fast Path:** Purely deterministic logic for low-risk scenarios, ensuring speed and zero inference costs.
- **The Deep Path:** Utilization of Google Gemini 2.5 Flash for high-risk scenarios requiring semantic understanding and empathy.

3.3 Architecture Diagram

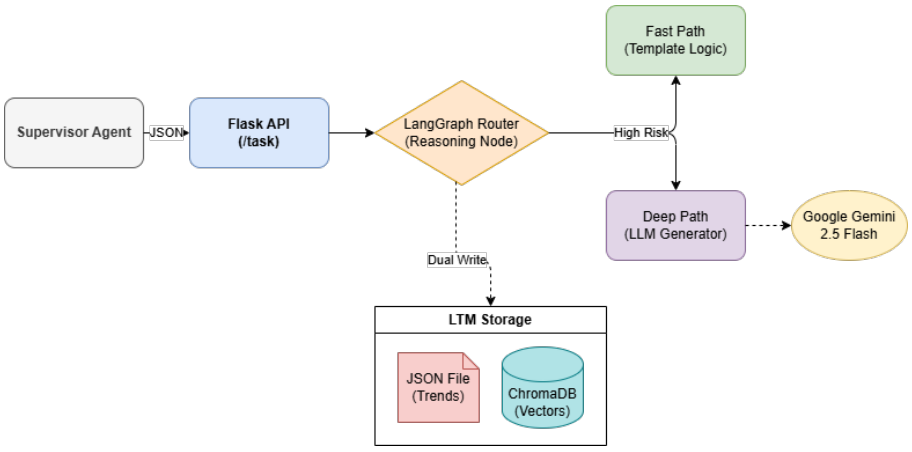

Figure 3.1: System Architecture

Chapter 4

Memory Strategy

4.1 Short-Term Memory (STM)

STM is managed via the LangGraph State Schema (BurnoutState). This is a typed dictionary that exists only during the lifecycle of a single HTTP request. It temporarily holds the input parameters, intermediate risk scores, and the generated AI response before the final JSON is returned to the supervisor.

4.2 Long-Term Memory (LTM) - Dual Storage Strategy

To meet the requirement for robust data retention and retrieval, a dual-write strategy was implemented:

- **Sequential Storage (JSON):** A structured memory.json log is used for Time-Series Analysis. This allows the agent to efficiently look back at the last N entries to detect trends.
- **Vector Storage (ChromaDB):** A semantic vector database stores the embeddings of every interaction. This satisfies the advanced storage requirement and future-proofs the agent, allowing for semantic queries.

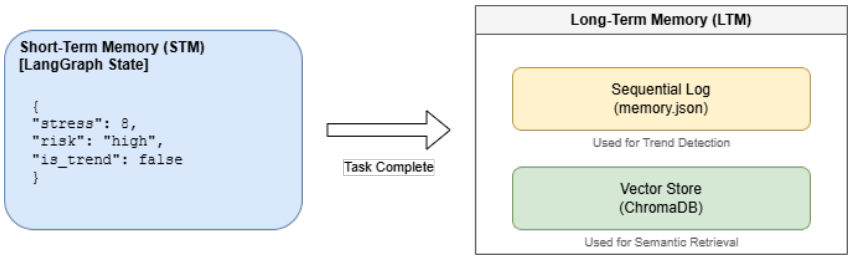

Figure 4.1: Memory Storage Architecture

Chapter 5

API Contract

The agent exposes a strict JSON contract over HTTP/1.1.

5.1 Request Format (Task Assignment)

Endpoint: POST /api/v1/task

```
{
    "message_id": "msg_unique_id",
    "sender": "SupervisorAgent_Main",
    "type": "task_assignment",
    "task": {
        "name": "analyze_wellbeing",
        "parameters": {
            "employee_id": "user_123",
            "stress": 8,
            "work_hours": 10,
            "sleep_hours": 5,
            "mood": "frustrated"
        }
    }
}
```

5.2 Response Format (Completion Report)

```
{
    "message_id": "msg_response_id",
    "sender": "WorkerAgent_BurnoutPrevention",
    "type": "completion_report",
    "status": "SUCCESS",
    "results": {
        "risk_level": "high",
        "is_trend": false,
        "key_factors": ["high_stress", "poor_sleep",
                        "negative_mood"],
        "empathetic_suggestion": "I am sorry to hear
                                  you are feeling...",
        "actionable_steps": ["Take a 15-minute break",
                             "Schedule deep work blocks"],
        "conversation_starter": "Hi Manager, I need to
                                 discuss my capacity..."
```

```
    },
    "timestamp": "2025-11-30T12:00:00"
}
```

Chapter 6

Integration Plan

The integration strategy ensures the agent can be "plugged in" to the Supervisor system with zero configuration changes.

- **Discovery:** The agent broadcasts availability via a heartbeat endpoint (GET /api/v1/status).
- **Error Isolation:** The agent wraps all internal logic in try/except blocks. If the AI service is down, the agent degrades gracefully to a "Fallback Mode" rather than crashing the Supervisor's workflow.
- **Environment Isolation:** All dependencies are encapsulated in a virtual environment to prevent conflicts with the host system.

Chapter 7

Progress & Lessons Learned

7.1 Challenges Faced & Resolutions

7.1.1 Dependency Conflicts

We encountered severe version mismatches between langchain-core and langchain-google-genai, causing ModuleNotFoundError for Pydantic objects.

Resolution: Adopted a "Clean Room" installation strategy, utilizing a dedicated Developer Command Prompt to rebuild the virtual environment from scratch.

7.1.2 Model Deprecation

The initial design relied on gemini-pro, which returned 404 errors due to API deprecation.

Resolution: Developed a utility script (check_models.py) to dynamically query available models, switching to the stable gemini-2.5-flash.

7.1.3 Logic Overwrites

Early rule-based logic allowed lower-priority factors to overwrite high-priority ones.

Resolution: Implemented a "Max-Score" algorithm (risk = max(current, new)) to ensure risk levels can only escalate, never downgrade falsely.

7.2 Final Status

The project is 100% complete. The agent successfully demonstrates reasoning capabilities, persistent memory, and seamless API integration, fulfilling all functional requirements set forth in the Project Charter.