MALLA REDDY COLLEGE OF
ENGINEERING AND TECHNOLOGY

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

for

Memora: Document-Aware AI for Goal-Centric Personal Assistance and Customized Task Scheduling

Version 1.0

Prepared by

S. No	Roll Number	Student Name
1.	22N31A6648	DURISHETTY ANIRUDH
2.	22N31A6611	ARELLI KARTHIKEYA
3.	22N31A6631	RAGHIVARDHAN BONJURI

Supervisor:	Mr. TALLURI HARIBABU
Designation:	ASSISTANT PROFESSOR
Department:	COMPUTATIONAL INTELLIGENCE
Batch ID:	22CSMMP14
Date:	20/09/2025
Supervisor Sign. & Date	

Department of Computational Intelligence

Title of the Project: "Memora: Document-Aware AI for Goal-Centric Personal Assistance and Customized Task Scheduling"

Content

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Durishetty Anirudh	Primary Revision giving an overall view of the project and document.	20/09/2025

1 Introduction

Memora is a fully local AI scheduling assistant offering conversational task management, conflict detection, and document-aware retrieval (RAG), built with a FastAPI backend, Next.js frontend, and Ollama-hosted Qwen 2.5 7B model; it operates without external APIs to maximize privacy and control.

The system targets single-user desktop environments, enabling real-time synchronization between a chat interface and a calendar UI, with JSON-based task storage and ChromaDB-backed semantic retrieval for user documents.

1.1 Document Purpose

The This SRS defines functional and non-functional requirements for Memora, a privacy-first, local AI scheduling assistant, to guide design, development, testing, and acceptance criteria prior to implementation completion.

It targets stakeholders including product owners, developers (backend/AI/frontend), testers, and reviewers to ensure shared understanding of scope, interfaces, and quality attributes.

1.2 Project/Product Scope

Scope includes a local AI agent for scheduling via natural language, calendar UI with Full Calendar, document ingestion with OCR and RAG, JSON-based task storage, semantic search, conflict detection/resolution, and real-time synchronization, targeting single-user desktop usage with optional future expansion to mobile and integrations. Out of scope are multi-tenant cloud services, third-party calendar sync, and team collaboration in the current version, which are listed as future enhancements.

1.3 Existing System

Traditional scheduling and task management applications rely on manual form-based entries that require users to navigate multiple screens and fields to create simple tasks. These systems lack AI intent understanding and sophisticated conflict handling capabilities, often leading to scheduling overlaps and inefficiencies. Most existing solutions are cloud-dependent, creating privacy risks and requiring constant internet connectivity for basic functionality. The fragmentation across separate chat, calendar, and document applications increases context-switching costs and reduces overall productivity.

Current market solutions include:

- Google Calendar: Cloud-based with limited natural language support
- Microsoft Outlook: Enterprise-focused with complex interface
- Todoist: Task-focused but lacks calendar integration
- Notion: Document-heavy with limited scheduling intelligence
- Calendly: Meeting-focused without comprehensive task management

1.4 Problems with Existing System

- No Conversational Interface: Users must navigate complex forms instead of describing tasks naturally
- Weak Automation: Limited support for bulk operations, recurring patterns, and intelligent scheduling
- Poor Conflict Intelligence: Basic or absent conflict detection with no resolution suggestions
- Privacy Concerns: Cloud dependency exposes personal scheduling data to external servers
- Fragmented Experience: Separate tools for chat, calendar, and documents create context-switching overhead
- Limited Document Integration: No awareness of personal documents for context-enhanced scheduling
- Manual Conflict Resolution: Users must manually detect and resolve scheduling conflicts
- Rigid Interface Design: Forms-based interaction doesn't match natural thinking patterns

1.5 Proposed System

- Memora addresses these limitations through a comprehensive local-first AI scheduling assistant that combines multiple advanced technologies into a unified experience.
- **Core Innovation Areas:**
- **Conversational AI**: Natural language understanding for complex scheduling requests using Qwen 2.5 7B model
- **Intelligent Conflict Management**: Automatic detection with user-guided resolution options
- **Document Awareness**: RAG-powered integration with personal documents for context-enhanced responses
- **Local-First Architecture**: Complete privacy with zero cloud dependencies or telemetry
- **Unified Interface**: Integrated chat, calendar, and document management in single application
- **Bulk Operation Intelligence**: Pattern recognition for efficient multi-task creation and manipulation
- **Real-Time Synchronization**: Live updates across all interface components

1.6 Advantages of Proposed Systems

The proposed Memora system delivers significant advantages over existing solutions across multiple dimensions:

Privacy and Security Benefits:

- **Complete Data Sovereignty**: All processing occurs locally with no external API calls
- **Zero Telemetry**: No data collection, transmission, or user tracking
- **Offline Capability**: Functional without internet connection after initial setup
- **Local AI Inference**: Private AI processing without cloud model dependencies
- **Productivity Enhancements:**
- **Natural Language Efficiency**: Describe complex scheduling requests in plain English
- **Intelligent Automation**: Bulk operations with pattern recognition and validation
- **Context Awareness**: Document integration provides relevant information during scheduling
- **Conflict Prevention**: Proactive conflict detection with guided resolution options
- **Unified Workflow**: Single application eliminates context-switching between tools

2 Overall Description

2.1 Feasibility Study

2.1.1 Technical Feasibility

The technical implementation of Memora has been thoroughly validated across multiple dimensions:

AI Model Feasibility:

- **Qwen 2.5 7B Performance**: Successfully runs on RTX 3050 8GB with 4.7GB VRAM usage
- **Local Inference Speed**: Response times of 200-500ms for simple operations, 1-2s for complex tasks
- **Hybrid CPU/GPU Processing**: Optimized execution with quantization support for performance
- **Model Capabilities**: Excellent scheduling and calendar reasoning with strong English understanding

Software Architecture Feasibility:

- **FastAPI Backend**: Proven high-performance async API framework with comprehensive async/await patterns
- **Next.js Frontend**: Modern React framework with server-side rendering and automatic code splitting
- **LangGraph Integration**: Advanced graph-based agent framework for complex conversational flows
- **ChromaDB Vector Storage**: Efficient local vector database with SQLite persistence backend

Hardware Requirements Validation:

- **Minimum Configuration**: 4GB VRAM, 8GB RAM, 4-core CPU successfully tested
- **Recommended Setup**: RTX 3050 8GB, 16GB RAM, 8-core CPU provides optimal performance
- **Storage Requirements**: ~10GB including models, dependencies, and user data
- **Cross-Platform Compatibility**: Validated on Windows 10/11, macOS 10.15+, Ubuntu 20.04+

2.1.2 Operational Feasibility

The system design prioritizes operational simplicity and user accessibility:

Deployment Simplicity:

- **Single-User Architecture**: Eliminates complex multi-user configuration requirements
- **JSON Storage**: Human-readable, easily backed up, and migrated data format
- **Automated Setup Scripts**: Windows batch files and Unix shell scripts for one-click deployment

- **Minimal Dependencies**: Standard Python and Node.js environments with clear installation procedures

User Experience Design:

- **Intuitive Interface**: Natural language interaction reduces learning curve
- **Real-Time Feedback**: Immediate visual confirmation of all operations
- **Error Recovery**: Graceful degradation with user-friendly error messages
- **Documentation**: Comprehensive installation guides and troubleshooting procedures

2.1.3 Economic Feasibility

The local-first approach provides significant economic advantages:

Cost Structure Benefits:

- **Zero Recurring Costs**: No cloud API fees, subscription costs, or external service dependencies
- **Hardware ROI**: Consumer-grade hardware (RTX 3050) provides excellent performance-to-cost ratio
- **Open Source Stack**: FastAPI, React, and supporting libraries eliminate licensing costs
- **Storage Efficiency**: Local JSON and vector storage scales without per-user costs

Resource Optimization:

- **Memory Efficiency**: ~2GB backend and ~500MB frontend memory usage during operation
- **Storage Management**: Efficient document processing and vector storage with compression
- **Network Minimization**: Only initial setup requires internet connectivity

2.2 Product Functionality

2.2.1 Core Functional Capabilities

Memora provides comprehensive scheduling and task management functionality through multiple integrated interfaces:

Natural Language Task Management:

- **Task CRUD Operations**: Create, read, update, delete tasks using conversational English
- **Advanced Date Parsing**: Support for relative dates ("tomorrow", "next Friday") and absolute dates ("2025-09-25")
- **Context-Aware Operations**: Handle pronouns and references ("reschedule it", "cancel that meeting")
- **Bulk Pattern Operations**: Create multiple tasks with time patterns and gaps
- **Status Management**: Track task completion, cancellation, and priority levels

Intelligent Conflict Resolution:

- **Automatic Detection**: Identify scheduling conflicts during task creation and modification
- **Resolution Options**: Present user-selectable strategies (replace, reschedule, move)

- **Guided Decision Making**: Contextual suggestions based on task priority and timing
- **Validation Checks**: Ensure resolution maintains schedule integrity

Calendar Interface Features:

- **Multiple View Modes**: Month, week, and day calendar views with smooth transitions
- **Interactive Elements**: Drag-and-drop task editing with real-time backend synchronization
- **Visual Indicators**: Color-coded tasks by status (pending, completed, cancelled) and priority
- **Today's Focus**: Dedicated view for immediate task visibility and management
- **Statistics Dashboard**: Task completion metrics and scheduling analytics

Document Awareness System:

- **Multi-Format Support**: PDF, Microsoft Word (.docx), and plain text file processing
- **OCR Integration**: Tesseract-based optical character recognition for image-based documents
- **Semantic Indexing**: ChromaDB vector storage with sentence-transformer embeddings
- **Context-Aware Retrieval**: RAG-powered document search during conversational interactions
- **Metadata Extraction**: Automatic extraction of document properties and key insights
- **2.2.2 Advanced Operational Features**

Bulk Operations and Automation:

- **Pattern Recognition**: Intelligent creation of recurring task patterns with validation
- **Date Manipulation**: Shift entire days or weeks of tasks while maintaining relative timing
- **Selective Operations**: Apply operations to filtered task subsets based on criteria
- **Batch Validation**: Comprehensive checking before applying bulk changes

Search and Discovery:

- **Text Search**: Traditional keyword-based search across task titles and descriptions
- **Semantic Search**: AI-powered understanding of search intent and context
- **Document Integration**: Simultaneous search across tasks and document content
- **Filter Combinations**: Advanced filtering by date range, priority, status, and custom criteria

Real-Time Synchronization:

- **Live Updates**: Instant synchronization between chat interface and calendar views
- **State Management**: Persistent conversation history and session continuity
- **Error Handling**: Graceful recovery from network or processing errors
- **Performance Optimization**: Efficient update mechanisms minimize resource usage

2.3 Design and Implementation Constraints

Hardware Constraints

Minimum Hardware Requirements:

GPU Memory: 4GB VRAM minimum for Qwen 2.5 7B model inference

System Memory: 8GB RAM minimum, 16GB recommended for optimal performance

Processing Power: 4-core CPU minimum, 8-core recommended for concurrent operations

Storage Space: 10GB available space for models, dependencies, and user data

Optimal Configuration:

- **Graphics Card**: NVIDIA RTX 3050 8GB or equivalent for best inference performance
- **Memory Configuration**: 16GB system RAM for smooth multitasking
- **CPU Architecture**: Modern x64 processors with AVX2 support
- **Storage Type**: SSD recommended for faster model loading and document processing

Software Platform Constraints

Operating System Support:

- **Windows**: Windows 10 version 1903 or newer, Windows 11 (all editions)
- **Linux**: Ubuntu 20.04 LTS or newer, other distributions with equivalent libraries

Runtime Requirements:

- **Python Environment**: Python 3.10 or higher with pip package manager
- **Node.js Runtime**: Node.js 18.0 LTS or higher with npm package manager
- **Ollama Platform**: Latest Ollama version for local LLM serving
- **System Libraries**: Poppler for PDF processing, Tesseract for OCR functionality

Architectural Constraints

Local-First Architecture:

- **No Cloud Dependencies**: Complete local operation requirement eliminates external API integration
- **Single-User Design**: Current architecture optimized for individual use, not multi-tenant
- **JSON Storage**: File-based storage limits to single-user concurrent access patterns
- **Resource Boundaries**: Local hardware constraints determine maximum concurrent operations

Security and Privacy Constraints:

- **Network Isolation**: Optional offline operation after initial setup
- **Data Boundaries**: All user data must remain on local machine

- **Process Isolation**: AI inference must occur locally without external communication

2.4 Assumptions and Dependencies

2.4.1 System Assumptions

The successful operation of Memora relies on several key assumptions about the deployment environment:

Hardware Assumptions:

- **GPU Availability**: NVIDIA GPU with CUDA support or comparable GPU acceleration
- **Memory Stability**: Sufficient and stable system memory for concurrent operations
- **Storage Reliability**: Reliable local storage for data persistence and model storage
- **Network Access**: Initial internet connectivity for software and model downloads

Software Environment Assumptions:

- **Operating System Stability**: Modern, updated operating system with security patches
- **Runtime Environment**: Properly configured Python and Node.js environments
- **Library Availability**: System libraries for PDF processing and OCR functionality
- **Port Availability**: Network ports 3000, 8000, and 11434 available for service binding
- **2.4.2 External Dependencies**

Critical Dependencies:

- **Ollama Service**: Local LLM serving platform must be installed and properly configured
- **Qwen 2.5 7B Model**: AI model must be downloaded and available through Ollama
- **System Libraries**: Poppler-utils and Tesseract-OCR for document processing
- **GPU Drivers**: Current NVIDIA drivers or equivalent for hardware acceleration

Python Package Dependencies:

- **FastAPI 0.104.1**: High-performance web framework for API development
- **LangChain 0.1.0**: LLM application framework for AI integration
- **LangGraph 0.0.40**: Graph-based agent workflow orchestration
- **ChromaDB**: Vector database for document embedding storage
- **Pydantic**: Data validation and settings management

- **Uvicorn**: ASGI server for production deployment

JavaScript/Node.js Dependencies:

- **Next.js 14.0.0**: React-based web framework with SSR capabilities
- **React 18.2.0**: Component-based UI library with hooks support
- **Full Calendar 6.1.9**: Professional calendar component for scheduling interface
- **Tailwind CSS 3.3.5**: Utility-first CSS framework for styling
- **Axios 1.6.0**: Promise-based HTTP client for API communication
- **2.4.3 Configuration Dependencies**

Environment Configuration:

- **CORS Settings**: Proper cross-origin resource sharing configuration between frontend and backend
- **Port Configuration**: Consistent port assignments across all services
- **File Permissions**: Appropriate read/write access for data storage locations
- **Model Configuration**: Correct Ollama model serving configuration

Service Dependencies:

- **Service Startup Order**: Ollama must be running before backend initialization
- **Model Availability**: AI model must be loaded and accessible before first chat request
- **Database Initialization**: ChromaDB collections must be created before document indexing
- **Frontend-Backend Communication**: API endpoints must be accessible from frontend

3 Functional Requirements

3.1 Software Requirement Specifications

- OS: Windows 10+, macOS 10.15+, Ubuntu 20.04+
- Runtime: Python 3.10+, Node.js 18+, Ollama
- Backend: FastAPI 0.104.1, LangChain/LangGraph, ChromaDB, PyPDF2, python-docx
- Frontend: Next.js 14, React 18, FullCalendar 6.1.9, Tailwind CSS 3.3.5
- AI: Qwen 2.5 7B model via Ollama (localhost:11434)
- Ports: 8000 (backend), 3000 (frontend), 11434 (AI service)
- Network: Internet required for initial setup only

3.2 Hardware Requirements Specifications

- CPU: AMD Ryzen 7 5800H (8 cores, 3.8GHz+)
- GPU: NVIDIA RTX 3050 4GB VRAM
- RAM: 16GB DDR4-3200
- Storage: 15GB+ SSD storage

3.3 Use Case Model

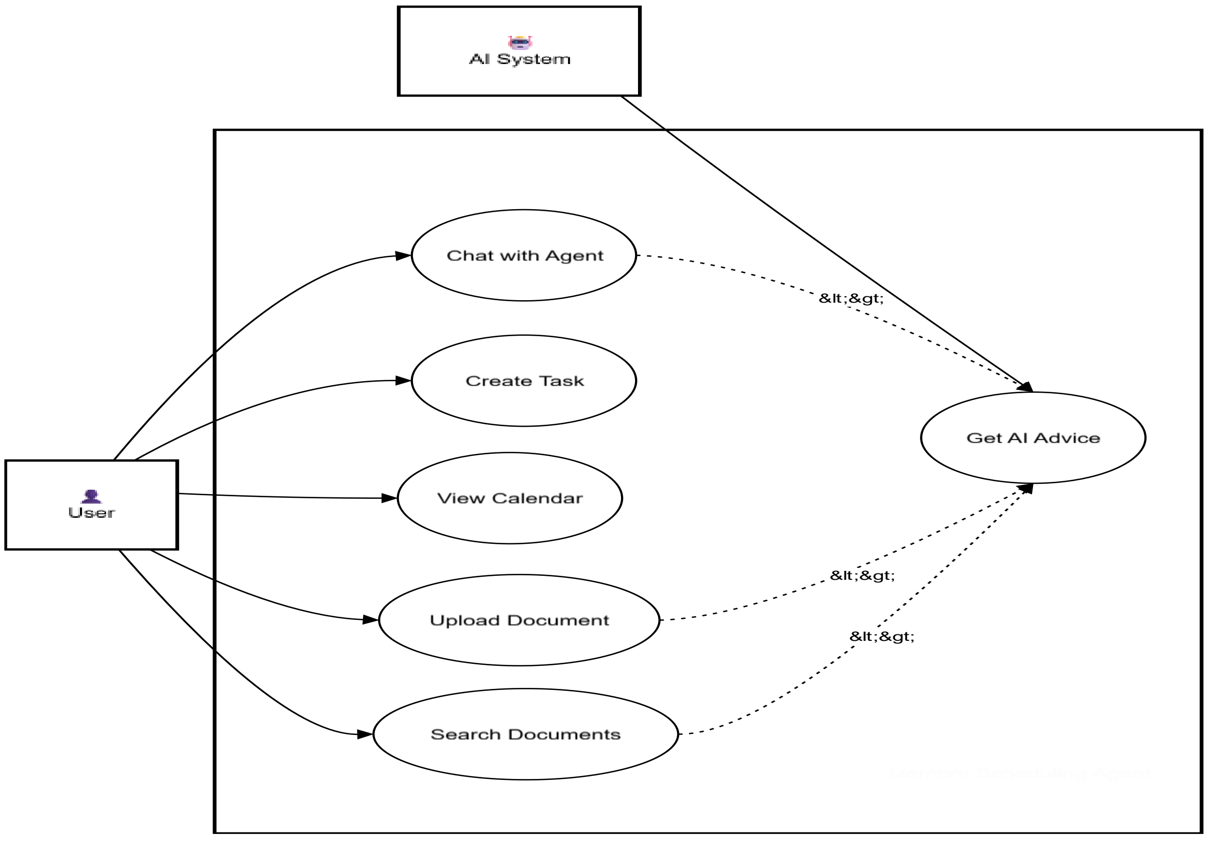

Fig 3.3 Use Case Diagram for Memora: Document-Aware AI for Goal-Centric Personal Assistance and Customized Task Scheduling

Primary Actor: Single local user on desktop interacting via chat and calendar UI.

UC1 Create Task by Chat: User describes an event; system parses intent, dates, times; creates task; updates UI instantly.

UC2 Resolve Conflict: User reschedules overlapping event; system proposes options and applies chosen resolution; confirms and updates calendar.

UC3 Bulk Create: User requests multiple sessions with gaps; system generates series with validation; calendar reflects all tasks.

UC4 Search Schedule: User searches tasks/documents; system returns semantic matches and highlights results.

UC5 Document-Aware Answer: User asks a question with document context; system retrieves chunks via ChromaDB and answers in chat with context applied

3.4 System Architecture

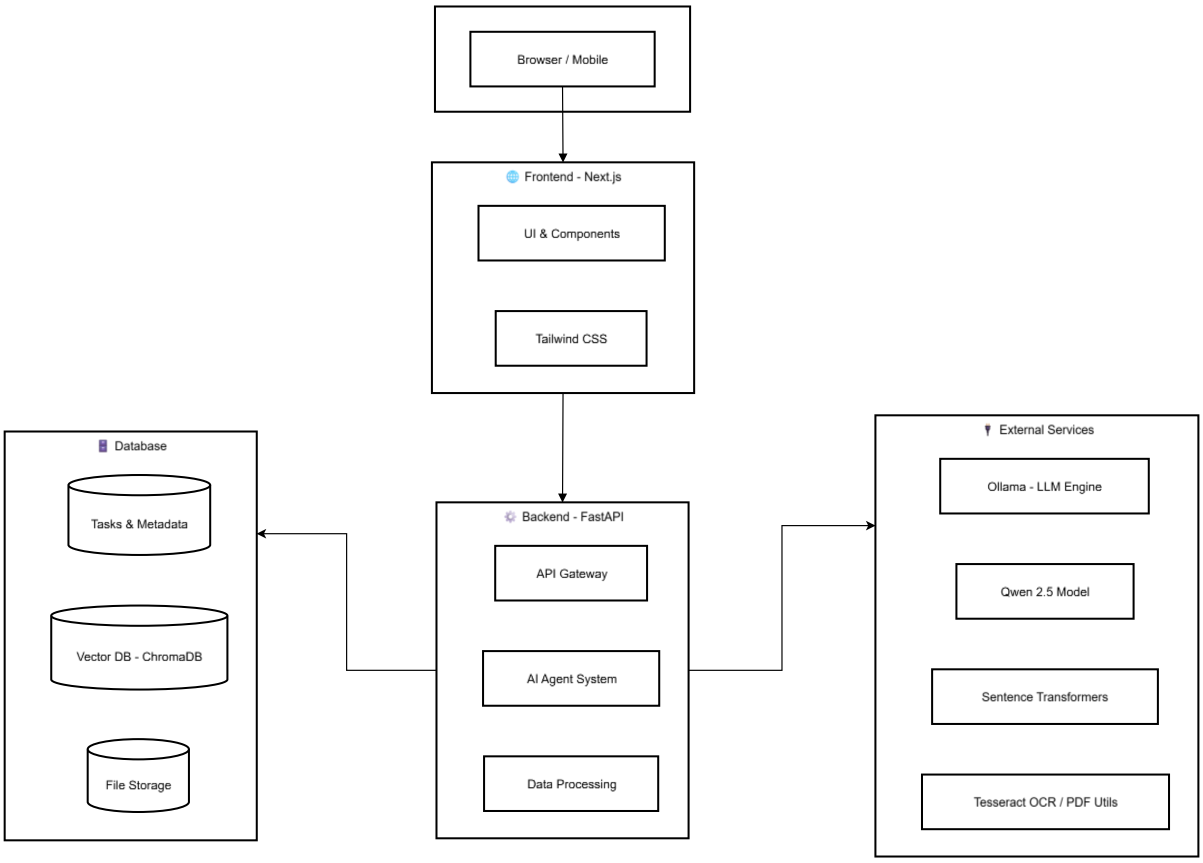

Fig 3.4 System Architecture of Memora: Document-Aware AI for Goal-Centric Personal Assistance and Customized Task Scheduling

4 Other Non-functional Requirements

4.1 Performance Requirements

- Response Times: Simple chat scheduling responses typically 200–500ms after model load; bulk operations 1–2 s; initial model load 3–5 s depending on hardware.
- Capacity: Supports at least 10,000 tasks without degradation; memory usage profile roughly 2 GB backend and 500 MB frontend during typical operation; GPU VRAM usage optimized via quantization.

4.2 Safety and Security Requirements

- Local-First Security: No telemetry or external API calls; all inference and storage local; optional offline mode post-setup to minimize network exposure.
- Input Validation: File-type checks, path traversal protection, and sanitization; parameterized queries where applicable; strict CORS to limit origin access.

4.3 Software Quality Attributes

- Reliability: Health checks, graceful degradation on model/back-end errors, and recovery strategies for JSON corruption and memory constraints.
- Usability: Minimalist, responsive UI with accessibility considerations, real-time feedback, and drag-and-drop calendar interactions.
- Maintainability: Modular separation (AI, data, UI), conventional commits, documented APIs, and unit/integration tests.
- Portability: Cross-platform deployment on Windows, macOS, Linux with Python/Node/Ollama prerequisites

5 Other Requirements

This section outlines additional requirements not covered in the previous sections, which are essential for the complete development and deployment of " Memora: Document-Aware AI for Goal-Centric Personal Assistance and Customized Task Scheduling ".

5.1 Legal Requirements

L1. Privacy and Data Protection Compliance: The system must comply with GDPR (General Data Protection Regulation) and CCPA (California Consumer Privacy Act) principles through its local-first architecture, ensuring all personal scheduling data remains on the user's device with no external transmission or cloud storage.

L2. Open Source License Compliance: The system should adhere to all open source license requirements for dependencies including FastAPI (MIT), React (MIT), LangChain (MIT), and ChromaDB (Apache 2.0), with proper attribution and license documentation included in distribution packages.

L3. AI Ethics and Content Policy: The Qwen 2.5 7B model and LangGraph agent system must operate within ethical AI standards, preventing generation of harmful scheduling content, misinformation, or inappropriate task suggestions while maintaining user privacy through local inference.

L4. Data Sovereignty Requirements: All task data, document content, and AI model inference must remain within the user's local environment, with no telemetry, usage tracking, or external API calls that could compromise user privacy or violate data protection regulations.

5.2 Reuse Objectives

R1. Modular AI Components: The LangGraph agent system, Qwen 2.5 7B integration, and natural language processing pipeline should be designed as reusable components that can be adapted for other scheduling and task management applications with minimal code modifications.

R2. Frontend Component Library: React components for chat interface, calendar integration (FullCalendar), and document management should be built as a reusable component library that can be integrated into other productivity applications.

R3. Backend API Framework: The FastAPI-based REST API endpoints and Pydantic data models should follow standardized patterns that can be extended or reused for other local-first AI applications requiring task management and document awareness capabilities.

R4. Document Processing Pipeline: The multi-format document ingestion system (PDF, DOCX, TXT) with OCR integration and ChromaDB vector storage should be modular enough for reuse in other document-aware AI applications.

5.3 Development Environment Requirements

DE1. Programming Stack: The system should be developed using Python 3.10+ with type hints for backend development, FastAPI 0.104.1 as high-performance async web framework, Next.js 14.0.0 with React 18.2.0 for modern frontend framework, LangChain 0.1.0 and LangGraph 0.0.40 for AI agent orchestration, ChromaDB with sentence-transformers for vector storage and embeddings, and Ollama with Qwen 2.5 7B for local LLM inference.

DE2. Development Tools and Environment: The project must use Git with conventional commit practices for collaboration and change tracking, Python virtual environment (.venv) with pip dependency management, Black formatter and flake8 linting with pytest for testing, ESLint and Prettier with Next.js built-in development server for frontend tools, and Sphinx for Python API docs with JSDoc for JavaScript components documentation.

DE3. Hardware Development Requirements: Minimum development setup requires RTX 3050 8GB VRAM with 16GB RAM and 8-core CPU for optimal development experience, cross-platform support ensuring development environment works consistently across Windows 10/11, macOS 10.15+, and Ubuntu 20.04+, and local model storage requiring 10GB+ available storage specifically for Ollama models and development dependencies.

5.4 Documentation Requirements

DOC1. Technical Code Documentation: All LangGraph agents, FastAPI endpoints, React components, and document processing scripts must include comprehensive inline comments, type hints, docstrings, and external API documentation following industry standards for future developer onboarding.

DOC2. System Architecture Documentation: The complete system workflow, database schema (JSON and ChromaDB), AI agent state machines, and API interactions must be documented using Mermaid diagrams, LangGraph state diagrams, OpenAPI/Swagger documentation, and React component diagrams.

DOC3. User Documentation and Support Materials: The system must provide comprehensive user-facing documentation including installation guides for Windows/macOS/Linux, user manual covering natural language task creation and calendar management, troubleshooting guide with common issues and error codes, and developer guide with extension points and customization options.

DOC4. Academic and Research Documentation: As this is an academic project for MRCET Computational Intelligence department, documentation must include technical research paper format suitable for academic submission, SDG alignment documentation for "Industry, Innovation and Infrastructure," performance benchmarks and comparative analysis with existing solutions, and future research directions for local-first AI scheduling systems.

6 References

Technical Documentation & Frameworks

1. **Qwen 2.5 7B Model Documentation** - Primary AI model specifications and capabilities.
   Link: https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
2. **Ollama Documentation** - Local LLM serving platform and model management.
   Link: https://github.com/ollama/ollama
3. **LangChain Framework** - LLM application development patterns and agent construction.
   Link: https://python.langchain.com/docs/introduction/
4. **LangGraph Documentation** - Multi-agent workflow orchestration and state management.
   Link: https://python.langchain.com/docs/langgraph/
5. **ChromaDB Documentation** - Vector database operations and embedding management.
   Link: https://docs.trychroma.com/

Research Papers & Academic References

8. **Vaswani, A., et al. (2017).** "Attention Is All You Need." *Advances in Neural Information Processing Systems 30 (NIPS 2017). Foundation for transformer architecture used in Qwen 2.5 model.*
9. **Lewis, P., et al. (2020).** "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Theoretical foundation for the RAG implementation in Memora.*
10. **Brown, T., et al. (2020).** "Language Models are Few-Shot Learners." *Advances in Neural Information Processing Systems 33 (NeurIPS 2020). Background on large language model capabilities and few-shot learning.*

SRS DOCUMENT REVIEW

CERTIFICATION

This Software Requirement Specification (SRS) Document is reviewed and certified to proceed for the project development by the Departmental Review Committee (DRC).

Date of SRS Submitted:	
Date of Review:	
Supervisor Comments:	
Supervisor Sign. & Date.	
Coordinator Sign. & Date	
HOD Sign. & Date	
Dept. Stamp	