AI Assistant Project – Technical Specification

Document Version: 1.0

Last Updated: June 26, 2025

Status: Ready for Implementation 🚀

# 1. Project Overview

## 1.1 Project Description

This project aims to build a full-stack AI assistant for task management, featuring a reactive agent, optional AI integration, and a modern, user-friendly interface.

## 1.2 Technology Stack

**• Frontend:** Vite + React  
**• Backend:** Express.js (Node.js)  
**• Styling:** Tailwind CSS  
**• AI Integration:** OpenAI API (optional)  
**• Deployment:** Vercel (frontend) + Render/Fly.io (backend)

## 1.3 Project Goals

• Create an intelligent task management assistant  
• Implement reactive decision-making logic  
• Provide natural language interactions  
• Deploy a production-ready application

# 2. Project Phases & Implementation Plan

## 🧱 Phase 1 – Initial Structure

**Objective:** Establish the foundational backend and frontend, ensuring basic integration.

**Duration:** 1-2 weeks

### Deliverables:

• Full-stack project setup (Vite + React for frontend, Express for backend)  
• Base API routes implementation  
• Simple React UI with basic functionality

### Technical Requirements:

#### Backend Setup

• Initialize Express.js server  
• Configure CORS and JSON middleware  
• Implement base API routes:  
 - POST /ask – Receive user queries  
 - POST /add – Add new tasks  
 - GET /list – List all tasks  
• Set up in-memory data storage (temporary)

#### Frontend Setup

• Initialize Vite + React project  
• Create basic components:  
 - Input field for user queries  
 - Task list display component  
 - Main app layout  
• Configure Axios or Fetch for API communication

### Acceptance Criteria:

☐ Backend server runs without errors  
☐ Frontend connects to backend successfully  
☐ Basic CRUD operations work  
☐ Simple UI displays tasks and accepts input

## 🧠 Phase 2 – Reactive Agent Logic

**Objective:** Implement core logic for task management and automated responses.

**Duration:** 2-3 weeks

### Data Model:

Task: {  
 id: string,  
 title: string,  
 description: string,  
 dueDate: Date,  
 priority: 'low' | 'medium' | 'high',  
 status: 'pending' | 'completed' | 'overdue',  
 createdAt: Date,  
 updatedAt: Date  
}

### Decision Logic Implementation:

• Overdue Detection: Identify and alert about overdue tasks  
• Priority Management: Sort and prioritize tasks  
• Status Updates: Automatically update task statuses  
• Smart Responses: Generate contextual responses based on task data

## 💬 Phase 3 – AI Integration (Optional)

**Objective:** Enhance responses using OpenAI or similar APIs.

**Duration:** 1-2 weeks

### AI Service Integration:

• Set up OpenAI API client  
• Implement secure API key management  
• Create AI service abstraction layer

### Enhanced Features:

• Smart Task Creation: Parse natural language to create tasks  
• Intelligent Responses: Use AI for more contextual replies  
• Fallback Logic: Provide helpful responses when no tasks match  
• Query Understanding: Better interpretation of user intents

## 🎨 Phase 4 – UI/UX & Deployment

**Objective:** Polish the interface and deploy the application.

**Duration:** 2-3 weeks

### UI/UX Improvements:

• Implement Tailwind CSS styling  
• Add subtle animations and transitions  
• Create responsive design for mobile devices  
• Improve accessibility (ARIA labels, keyboard navigation)

### Features Enhancement:

• Conversation History: Store and display chat history with timestamps  
• Loading States: Show loading indicators during API calls  
• Error Handling: User-friendly error messages  
• Dark/Light Mode: Theme switching capability

# 3. Technical Architecture

## 3.1 System Architecture

Frontend (React + Vite)  
 ↓ HTTP Requests  
Backend (Express.js)  
 ↓ API Calls  
External Services (OpenAI API)

## 3.2 API Design

### Endpoints:

• GET /health - Health check  
• POST /ask - Process user queries  
• POST /tasks - Create new task  
• GET /tasks - List all tasks  
• PUT /tasks/:id - Update task  
• DELETE /tasks/:id - Delete task  
• GET /conversations - Get conversation history

# 4. Development Guidelines

## 4.1 Code Quality Standards

• Linting: Use ESLint with recommended rules  
• Formatting: Use Prettier for consistent code formatting  
• Commits: Follow conventional commit message format  
• Comments: Add clear comments for complex logic  
• TypeScript: Consider migrating to TypeScript for better type safety

## 4.2 Testing Strategy

• Unit Tests: Test backend logic with Jest  
• Integration Tests: Test API endpoints  
• Frontend Tests: Use React Testing Library  
• E2E Tests: Consider Cypress for critical user flows

# 5. Security Considerations

## 5.1 API Security

• Input validation and sanitization  
• Rate limiting to prevent abuse  
• CORS configuration  
• Environment variable security

# 6. Future Enhancements

## 6.1 Short-term (Next 3 months)

• User authentication (JWT, OAuth)  
• Persistent database storage (PostgreSQL/MongoDB)  
• Mobile app development  
• Advanced task categorization

## 6.2 Long-term (6+ months)

• Multi-user support with teams  
• Advanced AI features (summarization, smart scheduling)  
• Integration with external calendars  
• Voice interface support  
• Analytics and reporting dashboard

# 7. Getting Started

## 7.1 Prerequisites

• Node.js (v18+)  
• npm or yarn  
• Git  
• Code editor (VS Code recommended)

## 7.2 Initial Setup Commands

# Backend setup  
mkdir ai-assistant-backend  
cd ai-assistant-backend  
npm init -y  
npm install express cors dotenv  
  
# Frontend setup  
npm create vite@latest ai-assistant-frontend -- --template react  
cd ai-assistant-frontend  
npm install axios