Phase III: Todo AI Chatbot

Basic Level Functionality

Objective: Create an AI-powered chatbot interface for managing todos through natural language using MCP (Model Context Protocol) server architecture and using Claude Code and Spec-Kit Plus.

Requirements

1. Implement conversational interface for all Basic Level features
2. Use OpenAI Agents SDK for AI logic
3. Build MCP server with Official MCP SDK that exposes task operations as tools
4. Stateless chat endpoint that persists conversation state to database
5. AI agents use MCP tools to manage tasks. The MCP tools will also be stateless and will store state in the database.

Technology Stack

Component	Technology
Frontend	OpenAI ChatKit
Backend	Python FastAPI
AI Framework	OpenAI Agents SDK
MCP Server	Official MCP SDK
ORM	SQLModel
Database	Neon Serverless PostgreSQL
Authentication	Better Auth

Architecture

```
┌────────────────┐     ┌─────────────────────────────────────────────┐
┌────────────────┐   │     │                FastAPI Server                │     │
│                │   │     │  ┌─────────────────────────────────────────┐  │     │
│   ChatKit UI   │──▶│  │          Chat Endpoint                 │  │     │
Neon DB        │
│   (Frontend)   │   │  │  POST /api/chat                        │  │     │
(PostgreSQL)     │
│                │   │  └──────────────┬──────────────────────────┘  │     │
│                │   │                 │                             │     │
- tasks          │
│                │   │                 ▼                             │     │
- conversations│
│                │   │  ┌─────────────────────────────────────────┐  │     │
- messages       │
│                │◀──│  │       OpenAI Agents SDK                │  │     │
│                │   │  │       (Agent + Runner)                 │  │     │
│                │   │  └──────────────┬──────────────────────────┘  │     │
│                │   │                 │                             │     │
│
```

```
│                  │     │                   ▼                       │     │
│                  │     │  ┌──────────────────────────────────────┐ │───►│
│                  │     │  │         MCP Server               │  │    │
│                  │     │  │ (MCP Tools for Task Operations)      │ │◄───│
│                  │     │  └──────────────────────────────────────┘ │     │
└──────────────────┘     └───────────────────────────────────────────┘
└──────────────────┘
```

Database Models

Model	Fields	Description
Task	user_id, id, title, description, completed, created_at, updated_at	Todo items
Conversation	user_id, id, created_at, updated_at	Chat session
Message	user_id, id, conversation_id, role (user/assistant), content, created_at	Chat history

Chat API Endpoint

Method	Endpoint	Description
POST	/api/{user_id}/chat	Send message & get AI response

Request

Field	Type	Required	Description
conversation_id	integer	No	Existing conversation ID (creates new if not provided)
message	string	Yes	User's natural language message

Response

Field	Type	Description
conversation_id	integer	The conversation ID
response	string	AI assistant's response
tool_calls	array	List of MCP tools invoked

MCP Tools Specification

The MCP server must expose the following tools for the AI agent:

Tool: add_task

Purpose	Create a new task
Parameters	user_id (string, required), title (string, required), description (string, optional)
Returns	task_id, status, title
Example Input	{“user_id”: “ziakhan”, "title": "Buy groceries", "description": "Milk, eggs, bread"}

Example Output	{"task_id": 5, "status": "created", "title": "Buy groceries"}

Tool: list_tasks

Purpose	Retrieve tasks from the list
Parameters	status (string, optional: "all", "pending", "completed")
Returns	Array of task objects
Example Input	{user_id (string, required), "status": "pending"}
Example Output	[{"id": 1, "title": "Buy groceries", "completed": false}, ...]

Tool: complete_task

Purpose	Mark a task as complete
Parameters	user_id (string, required), task_id (integer, required)
Returns	task_id, status, title
Example Input	{“user_id”: “ziakhan”, "task_id": 3}
Example Output	{"task_id": 3, "status": "completed", "title": "Call mom"}

Tool: delete_task

Purpose	Remove a task from the list
Parameters	user_id (string, required), task_id (integer, required)
Returns	task_id, status, title
Example Input	{“user_id”: “ziakhan”, "task_id": 2}
Example Output	{"task_id": 2, "status": "deleted", "title": "Old task"}

Tool: update_task

Purpose	Modify task title or description
Parameters	user_id (string, required), task_id (integer, required), title (string, optional), description (string, optional)
Returns	task_id, status, title
Example Input	{“user_id”: “ziakhan”, "task_id": 1, "title": "Buy groceries and fruits"}
Example Output	{"task_id": 1, "status": "updated", "title": "Buy groceries and fruits"}

Agent Behavior Specification

Behavior	Description
Task Creation	When user mentions adding/creating/remembering something, use add_task
Task Listing	When user asks to see/show/list tasks, use list_tasks with appropriate filter
Task Completion	When user says done/complete/finished, use complete_task
Task Deletion	When user says delete/remove/cancel, use delete_task
Task Update	When user says change/update/rename, use update_task
Confirmation	Always confirm actions with friendly response
Error Handling	Gracefully handle task not found and other errors

Conversation Flow (Stateless Request Cycle)

1. Receive user message
2. Fetch conversation history from database
3. Build message array for agent (history + new message)
4. Store user message in database
5. Run agent with MCP tools
6. Agent invokes appropriate MCP tool(s)
7. Store assistant response in database
8. Return response to client
9. Server holds NO state (ready for next request)

Natural Language Commands

The chatbot should understand and respond to:

User Says	Agent Should
"Add a task to buy groceries"	Call add_task with title "Buy groceries"
"Show me all my tasks"	Call list_tasks with status "all"
"What's pending?"	Call list_tasks with status "pending"
"Mark task 3 as complete"	Call complete_task with task_id 3
"Delete the meeting task"	Call list_tasks first, then delete_task
"Change task 1 to 'Call mom tonight'"	Call update_task with new title
"I need to remember to pay bills"	Call add_task with title "Pay bills"
"What have I completed?"	Call list_tasks with status "completed"

Deliverables

1. GitHub repository with:
- /frontend – ChatKit-based UI
- /backend – FastAPI + Agents SDK + MCP
- /specs – Specification files for agent and MCP tools
- Database migration scripts
- README with setup instructions

2. Working chatbot that can:
- Manage tasks through natural language via MCP tools
- Maintain conversation context via database (stateless server)
- Provide helpful responses with action confirmations
- Handle errors gracefully
- Resume conversations after server restart

Key Architecture Benefits

Aspect	Benefit
MCP Tools	Standardized interface for AI to interact with your app
Single Endpoint	Simpler API — AI handles routing to tools
Stateless Server	Scalable, resilient, horizontally scalable
Tool Composition	Agent can chain multiple tools in one turn

Key Stateless Architecture Benefits

- **Scalability:** Any server instance can handle any request
- **Resilience:** Server restarts don't lose conversation state
- **Horizontal scaling:** Load balancer can route to any backend
- **Testability:** Each request is independent and reproducible