Basic Idea

This project is basically a personal life management tool with which the user will interact in natural language.

For example:

User says: I need to fix a bug in auth in avenue

Secretary: Created tasks “Auth bug in avenue”

User asks: What are all the pending tasks?

Secretary: Gives a list of all pending tasks

User says: I have a event on 16th October

Secretary: Saved that event

User asks: Am I free on 16th October

Secretary: No, you have a event on 16th October

Basically, this agent will store all the context of tasks and events

What exactly is happening?

1. **Conversation Engine (AI brain)**: Understands what you said and decides *what kind of task* it is — e.g., query, creation, update, summary, etc.
2. **Memory + Logic Layer (Backend brain):** Actually *does* the work — saves tasks, fetches events, retrieves memory, etc.
3. **Data Storage (Long-term brain):** Where all your facts live — tasks, events, notes, embeddings (memory).

Logical Flow

1. You send a message (frontend)

Example:

“Remind me to fix the login bug in Product 1 by Friday.”

Frontend (Next.js)

- Sends { message: "Remind me..." } → to your backend API (/api/chat).
- Shows your message in the chat UI.
- Waits for the assistant’s response (streaming or full).

2. Backend receives it and creates a “conversation state”

What the backend does:

- It collects recent chat history (maybe last 10 messages).
- Optionally fetches any relevant user memory (tasks/events).
- Combines that into a “prompt context”.

Why?
So the model has enough information to understand *your* current world.

3. Backend sends everything to the LLM (like GPT-4)

It sends a request that looks roughly like this:

```
[
  { role: "system", content: "You are a helpful secretary..." },
  { role: "user", content: "Remind me to fix the login bug in Product 1 by Friday." }
]
```

4. The LLM interprets the intent

Now the model decides:
→ Is this a **question**, **task creation**, **status update**, or **schedule query**?

Depending on that, it may:

- Ask for clarification (if it’s ambiguous), **or**
- Decide to **call a tool/function** (e.g. create_task).

5. If it calls a tool, backend executes it

Example tool call:

```
{
  "name": "create_task",
  "arguments": {
    "title": "Fix login bug in Product 1",
    "dueAt": "2025-10-10T17:00:00Z"
  }
}
```

Backend executes that:

- Inserts into your tasks table.
- Generates an **embedding** (vector) for that text → stores it in memory.
- Returns the new task info to the LLM.

6. LLM receives the result and replies naturally

It now formats the response:

> “Got it. I’ve added ‘Fix login bug in Product 1’ due Friday.”

Backend sends that back to frontend → chat updates.

7. Future queries use “memory”

Later you ask:

> “What’s pending this week?”

Backend flow:

1. Detect this is a **status query**.
2. Calls database: find tasks where status != done and due_at < end_of_week.
3. Optionally uses **vector search** for semantic matches (e.g., “finish product stuff”).

Sends retrieved items to the LLM:
```
{
  role: "system": "...",
  role: "user": "What's pending this week?",
  role: "assistant": "context: [task1, task2, task3]"
}
```

4.
5. LLM summarizes naturally:
   “You have 2 tasks due: fix the login bug in Product 1, and finalize the pricing doc for Project 2.”

8. Daily summary or proactive actions

You can have a cron job (or server trigger) that:

- Runs every morning.
- Queries DB for today’s schedule and overdue tasks.
- Uses the LLM to summarize:
  “Good morning! You have 2 meetings today and 1 overdue task from yesterday.”
- Sends that via chat, email, or notification.

Putting It All Together — Component View

1. Frontend (Chat UI / Task View)

- Sends user messages to /api/chat
- Displays messages from assistant
- Shows structured data (tasks, calendar, etc.)

2. Backend API (Logic Layer)

- Routes chat messages
- Calls OpenAI / LLM APIs
- Handles tool calls (create/list/update)
- Manages DB operations

- Stores embeddings for memory

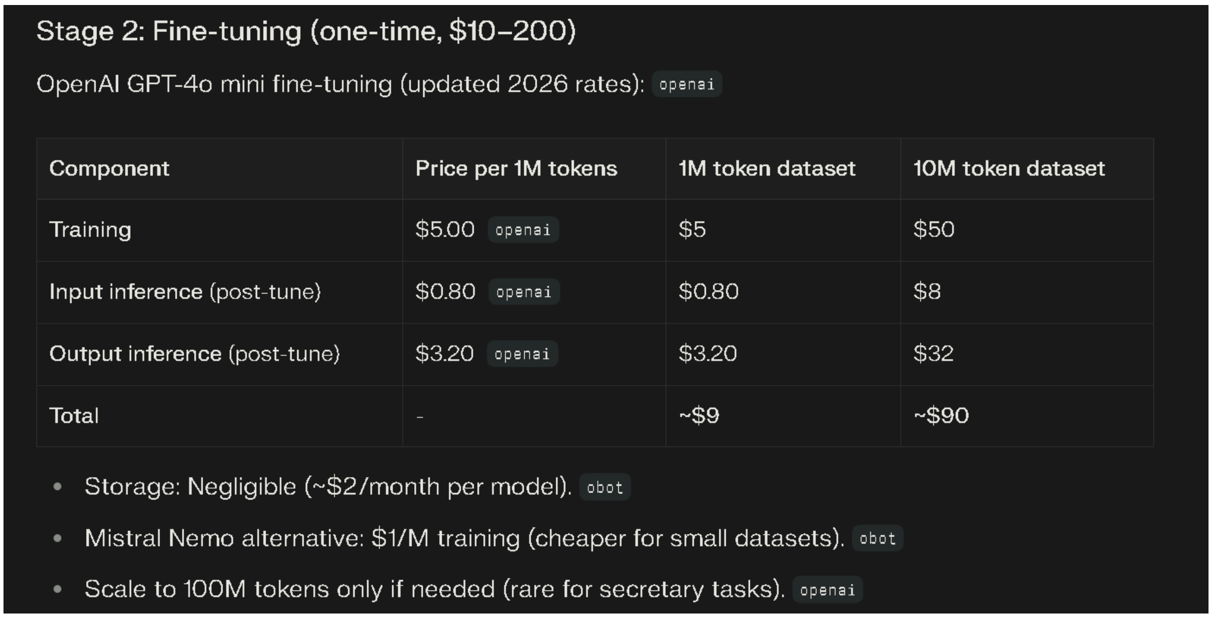
3. Database (Memory)

- Stores structured data (tasks, events)
- Stores vector embeddings for memory search
- Keeps chat history (optional)

4. LLM (Reasoning Engine)

- Interprets intent
- Decides when to call backend tools
- Generates human-friendly responses

Pricing



Stage 2: Fine-tuning (one-time, $10–200)

OpenAI GPT-4o mini fine-tuning (updated 2026 rates): `openai`

Component	Price per 1M tokens	1M token dataset	10M token dataset
Training	$5.00 `openai`	$5	$50
Input inference (post-tune)	$0.80 `openai`	$0.80	$8
Output inference (post-tune)	$3.20 `openai`	$3.20	$32
Total	-	~$9	~$90

- Storage: Negligible (~$2/month per model). `obot`
- Mistral Nemo alternative: $1/M training (cheaper for small datasets). `obot`
- Scale to 100M tokens only if needed (rare for secretary tasks). `openai`

Light usage (MVP / side project)

Say **100 interactions/day** (onboarding + some emails + FAQs):

- Daily tokens ≈ 40,000
- Monthly tokens ≈ 1.2M total

Monthly cost:

- Input: $0.15 \times 0.24 \approx 0.036$ USD
- Output: $0.60 \times 0.24 \approx 0.144$ USD
- **Total ≈ 0.18 USD/month** (around ₹15).[2] [1]

Moderate usage (small team / startup pilot)

Say **1,000 interactions/day**:

- Daily tokens ≈ 400,000
- Monthly tokens ≈ 12M
- Input: $0.15 \times 2.4 \approx 0.36$ USD
- Output: $0.60 \times 2.4 \approx 1.44$ USD
- **Total ≈ 1.80 USD/month** (≈ ₹150).[1] [2]

Heavy usage (10,000 interactions/day)

Still very cheap:

- Tokens/month ≈ 120M
- Input: $0.15 \times 24 \approx 3.60$ USD
- Output: $0.60 \times 24 \approx 14.40$ USD