SynthesisTalk

System Design Documentation

Youssef Hussein - 202202946
Youssef Abohendia - 202202699
Hussein Sabry - 202202653

1. System Architecture Description

SynthesisTalk is a modular, multi-tool AI research assistant built using a React frontend and a FastAPI backend. The system is architected to facilitate multi-turn, context-aware conversations, document analysis, tool-augmented reasoning (charts, export, web search), and seamless user experience.

Key Design Properties:

- **Loose coupling:** Each backend module (chat, memory, export, upload, websearch) is separate and interacts via explicit imports and function calls.
- **Session-aware:** Each chat is tied to a session_id, enabling per-user context, history, and notes.
- **Extensible tool integration:** New tools (summarization, advanced analytics) can be plugged in with minimal disruption.

2. Implementation Details for Reasoning Techniques

LLM Reasoning

- **Chain-of-Thought Prompting:** The system provides conversation history to the LLM, so answers can be contextually aware and build on previous user queries, improving reasoning coherence.
- **Action-based Routing:** Prompts are parsed for keywords/intent (e.g., "generate a bar chart", "search", "export as pdf"), and specific tool routines are invoked before or after LLM calls. This enables *ReAct-style* (Reason+Act) reasoning, where the system may call an external tool and compose the answer.

Workflow

1. **Intent Detection:**
   The backend parses incoming chat messages for tool-specific commands (e.g., "draw a chart", "take note", "export", "search").

2. **Tool Invocation:**
   If a tool is needed, the backend either:

   - Calls SerpAPI for web search queries

   - Invokes export functions for TXT/PDF

   - Stores or retrieves notes from memory

3. **LLM Synthesis:**
   For general questions or complex reasoning, the message and recent context are sent to the Groq LLM (Llama-3.3-70B-Versatile), which returns an answer.

   For websearch, we used the SerpAPI API key.

4. **Response Aggregation:**
   The backend merges tool outputs and LLM responses into a unified message for the frontend.

3. Tool Integration Approach

Plug-and-Play Tools

- **Memory Tool:** Session-based notes (memory.py), allowing users to store, view, and clear notes during a chat.

- **Web Search Tool:** API wrapper (websearch.py) for real-time fact-checking and research using SerpAPI.

- **Export Tool:** Export functions (export.py, export_pdf.py) allow saving the entire conversation in TXT or PDF.

How tools are integrated:

- Tool routines are imported into chat.py and invoked when a user query matches a known pattern or action.
- The main router (`@router.post("/chat")`) acts as a dispatcher, parsing the intent and routing to the appropriate tool handler.
- The design allows for adding new tools by creating new Python modules and registering their intent patterns in chat.py.

4. Challenges Encountered and Solutions

A. Robust Intent Detection

Challenge:
Distinguishing between general conversation and tool requests (e.g., chart vs. export vs. web search).

Solution:
A layered keyword- and intent-based parsing system in chat.py; fallback to LLM if no tool is detected. Systematic naming and session routing reduce ambiguity.

B. Session Management and Scalability

Challenge:
Retaining user context (history, notes) in-memory only works for development; risk of session loss.

Solution:
For prototype, in-memory storage suffices. For production, the design allows swapping memory.py for Redis/Postgres session storage with minimal changes.

D. Multi-format Output (Text & PDF)

Challenge:
Supporting exporting chat history in TXT and PDF.

Solution:

- Separate export.py and export_pdf.py modules handle formatting and static file serving.
- PDF export uses Python PDF libraries to embed text and images.

E. Real-time Tool Invocation without Blocking UI

Challenge:
Some operations (LLM calls, web search, chart generation) can be slow.

Solution:

- Frontend shows typing/loader indicators while waiting.
- Backend endpoints are non-blocking and provide clear error messages in case of failure

5. Evaluation of System Performance

Responsiveness

- **Chat and tool responses** are typically returned within 2–4 seconds, depending on LLM and tool call latency.
- **Chart and export features** are responsive, with the frontend immediately rendering returned images or download links.

Reliability

- Handles malformed requests gracefully (e.g., “not enough data to plot a graph”).
- Produces error or fallback messages for LLM/API failures, rather than crashing or hanging.

User Experience

- Users can upload files, ask for analysis, search the web, and export history in a unified chat interface.
- The system feels like a cohesive assistant rather than a collection of disconnected tools.

Extensibility

- New tools, features, or reasoning enhancements can be plugged into chat.py with little disruption.
- Modular design supports future DB integration, authentication, and advanced analytics.

Summary

SynthesisTalk demonstrates an extensible, modular, tool-enhanced research assistant, supporting interactive AI conversations with multi-modal output. Through careful design and layered reasoning, it blends language model intelligence with actionable tools and robust user experience.

Challenges (intent detection, incomplete data, session handling) were solved with a combination of modular code, fallback logic, and user-friendly feedback, resulting in a resilient and expandable system.