* ChainLit – FrontEnd <https://docs.chainlit.io/get-started/overview>

Key Features:

Fast Prototyping: Build interactive UIs with just a few lines of Python.

LangChain Integration: Native support for LangChain chains and agents.

Chat Interface: Automatically generates a chat UI for your app.

Component Support: Add buttons, sliders, file uploads, and more.

Developer Tools: Logs, traces, and debugging tools for LLM workflows.

LangChain – Backend Logic <https://www.langchain.com/>

MCP (Model Context Protocol) <https://modelcontextprotocol.io/introduction>

* Flow Breakdown:
* User Input: The user types a message or uploads a file.
* Chainlit UI: Captures the input and sends it to the backend.
* LangChain Chain/Agent: Processes the input using logic, memory, and tools.
* Tools (Search, Calculator, etc.): Optional tools the agent can use to complete tasks.
* Memory: Stores and retrieves context from previous interactions.
* LLM Response: The final output generated by the language model.
* Back to Chainlit UI: The response is displayed to the user in the chat interface.

**LangChain**

LangChain is a modular framework for building applications powered by large language models (LLMs). It abstracts away the complexity of prompt engineering, memory management, and integration with external tools.

Key features used in this project:

* **LLM Wrappers**: We use ChatBedrock to interface with Claude 3.5 via AWS Bedrock.
* **Memory**: ConversationBufferMemory stores previous interactions in RAM, allowing the model to maintain context across multiple turns.
* **Prompt Management**: LangChain allows us to load and structure onboarding instructions as SystemMessage objects, guiding the model’s behavior.
* **Streaming Support**: LangChain supports token-by-token streaming, which improves responsiveness and user experience.

**Chainlit**

Chainlit is a frontend framework tailored for LLM-powered apps. It simplifies building interactive chat interfaces with:

* **Session Management**: Tracks user sessions using manually entered names (as a temporary workaround for OAuth).
* **UI Elements**: Displays chat history in a side panel using cl.Text and cl.Message.
* **Event Hooks**: @cl.on\_chat\_start, @cl.on\_message, and @cl.action\_callback allow us to control the chat lifecycle and trigger backend logic.

Together, LangChain and Chainlit form a full-stack solution for building intelligent, interactive chatbots.

**⚙️ 2. Project Architecture & Code Flow**

**Frontend (main 9.py)**

This script handles user interaction and UI rendering.

**Session Initialization**

* On chat start (@cl.on\_chat\_start), the user is prompted to enter their name.
* This name becomes the user\_id, stored in cl.user\_session.

**Chat History Retrieval**

* The function show\_recent\_conversations(user\_id) fetches the last 5 user/AI message pairs from PostgreSQL.
* These are summarized and displayed in the side panel using Chainlit's UI components.

**Message Handling**

* When a message is sent (@cl.on\_message):
  + The message is streamed from the backend using service.stream\_response().
  + The response is displayed in real time.
  + Both the user input and AI response are saved to the database using a ChatMessageDTO.

**Backend (langchain\_service 8.py)**

This script encapsulates the core logic for LLM interaction and database operations.

**Claude 3.5 Integration**

* Claude 3.5 Sonnet is accessed via AWS Bedrock using ChatBedrock.
* The model is initialized with parameters like temperature, top\_p, and max\_tokens to control creativity and response length.

**System Prompt Loading**

* Onboarding instructions are loaded from onboarding.txt, split into chunks using RecursiveCharacterTextSplitter, and joined into a single SystemMessage.

**Memory Management**

* ConversationBufferMemory stores recent interactions in RAM, allowing the model to reference previous exchanges during inference.

**Database Operations**

* PostgreSQL is used to persist chat history.
* Messages are saved with UUIDs and timestamps.
* The fetch\_recent\_conversations() method retrieves the latest exchanges for a given user.

**Streaming Responses**

* stream\_response(prompt) sends the full message context to Claude and yields chunks of the response.
* This enables real-time feedback in the UI while maintaining memory continuity.

**🧪 3. State-of-the-Art AI Techniques**

**Claude 3.5 Sonnet**

* Developed by Anthropic, Claude 3.5 is optimized for enterprise use cases.
* It excels in financial reasoning, document summarization, and multi-turn dialogue.
* It supports **streaming**, **contextual memory**, and **structured prompting**.

**Prompt Engineering**

* The onboarding instructions act as a system-level guide, shaping the model’s tone and behavior.
* This is crucial for domain-specific applications like banking, where accuracy and compliance matter.

**Memory & Context**

* LangChain’s memory module ensures that the model doesn’t “forget” previous messages.
* This is vital for coherent conversations, especially in workflows like client onboarding or loan processing.

**🧰 4. Tech Stack**

| **Layer** | **Technology** | **Purpose** |
| --- | --- | --- |
| LLM Backend | Claude 3.5 via AWS Bedrock | Natural language understanding and generation |
| Frameworks | LangChain, Chainlit | Backend logic and frontend UI |
| Memory | ConversationBufferMemory | Context retention across turns |
| UI | Chainlit | Chat interface and history panel |
| Database | PostgreSQL | Persistent storage of chat history |
| Deployment | BMO internal network | Secure enterprise hosting |
| Auth (future) | Microsoft Entra ID (OAuth) | Identity management and access control |

**💼 5. Relevance to Commercial Banking**

This chatbot architecture can be adapted to several high-impact use cases:

**Client Onboarding**

* Automate repetitive Q&A.
* Guide users through document submission.
* Summarize previous interactions for continuity.

**Loan Application Support**

* Answer eligibility questions.
* Provide real-time guidance on form completion.
* Maintain a history of interactions for audit purposes.

**Internal Knowledge Assistant**

* Help bankers retrieve policy documents, product details, or training materials.
* Reduce time spent searching internal portals.

**Compliance & Audit**

* Every message is timestamped and stored.
* Enables traceability and accountability in client interactions.

**🖥️ 6. Demo Talking Points**

Here’s how to walk through the demo:

1. **Start the Chat**
   * Show how the user enters their name.
   * Explain how this becomes the session ID.
2. **View History**
   * Click the side panel to show recent conversations.
   * Highlight how messages are grouped and summarized.
3. **Send a Message**
   * Type a question and show Claude streaming a response.
   * Point out the real-time feedback and contextual relevance.
4. **Database Logging**
   * Mention how each exchange is saved with UUIDs and timestamps.
   * Show how the backend uses ChatMessageDTO to structure data.
5. **Code Highlights**
   * stream\_response() for streaming LLM output.
   * ConversationBufferMemory for context retention.
   * SystemMessage for prompt engineering.