**Junior Software Developer Intern working under Eric Shaw on Marshall Hayden’s team**

**🧠 Slide 1: LangChain & Chainlit Overview**

Hello everyone, I’m Bushra Khan, and I’m excited to share a chatbot project I’ve been working on using **LangChain** and **Chainlit**. These two frameworks are at the heart of many modern AI applications, especially those involving large language models.

**LangChain** is a backend framework that simplifies working with LLMs. It simplifies the complexity of prompt engineering, memory management, and model orchestration. In our project, LangChain connects to **Claude 3.5 Sonnet**, a powerful model hosted on AWS Bedrock. LangChain helps us structure conversations, maintain context, and stream responses efficiently.

1. **Modular Architecture**  
   LangChain allows you to build your chatbot as a pipeline of components—prompt templates, memory, tools, and chains. This modularity makes it easy to swap out parts (e.g., switching Claude 3.5 for another LLM or changing memory type) without rewriting the entire system.
2. **Enterprise-Grade Memory Management**  
   With ConversationBufferMemory, you can maintain context across turns, which is essential for financial conversations that often involve multi-step reasoning or referencing earlier inputs.
3. **Tool Integration**  
   LangChain supports integration with external tools like databases, APIs, and calculators. This opens the door to future enhancements like real-time financial data retrieval or transaction simulations.

**Chainlit** complements LangChain by providing a frontend interface. It’s designed specifically for LLM apps, offering a clean chat UI, session management, and interactive elements like side panels and buttons. It allows us to control the user experience while integrating seamlessly with LangChain’s backend logic.

1. **Rapid Prototyping with UI**  
   Chainlit provides a ready-to-use frontend for LangChain apps, which drastically reduces development time. You get a clean, interactive chat interface with minimal setup.
2. **Streaming Support**  
   Chainlit supports streaming responses, which improves user experience by reducing latency and making the chatbot feel more responsive—especially important in client-facing fintech tools.
3. **Custom Component Support**  
   You can add custom UI elements like buttons, forms, and side panels. This was key for your use case, where chat history is shown in the sidebar and user ID is manually entered.
4. **Easy Deployment**  
   Chainlit apps can be deployed within internal networks (like BMO’s), making it suitable for enterprise environments with strict security requirements.

Together, these frameworks enable rapid development of intelligent, responsive, and user-friendly chatbots.

* Security & Compliance Friendly: You can control data flow, integrate with internal auth systems (like Entra ID), and log interactions for auditability.
* Scalable & Maintainable: Modular design means you can iterate quickly and scale features as needed.
* User-Centric: The UI and memory features support a seamless user experience, which is crucial for banking professionals who need clarity and continuity.

**🧪 Slide 3: State-of-the-Art AI Techniques**

This project leverages several advanced AI techniques that make it both powerful and practical.

First, we use **Claude 3.5 Sonnet**, a cutting-edge model developed by Anthropic. It’s optimized for enterprise use cases, especially in domains like finance and law. Claude excels at reasoning, summarization, and multi-turn dialogue, making it ideal for banking applications.

We also use **streaming** to deliver responses token-by-token. This improves responsiveness and user experience, especially when dealing with longer or more complex queries.

**Prompt engineering** is another key technique. By loading onboarding instructions as a SystemMessage, we give the model a clear understanding of its role and expected behavior. This helps ensure that responses are relevant, accurate, and aligned with business goals.

Finally, **contextual memory** allows the model to remember what was said earlier in the conversation. This is crucial for maintaining coherence and avoiding repetition, especially in workflows that span multiple steps or involve detailed information.

**🧩 Slide 7: Closing**

To wrap up, this project demonstrates how modern AI frameworks like LangChain and Chainlit can be used to build intelligent, responsive, and secure chatbots tailored for enterprise use cases.

Whether it’s client onboarding, internal support, or compliance tracking, this architecture is flexible and scalable enough to meet the needs of many.

Thank you for your time, and I’m happy to take any questions!

**⚙️ Slide 2: Project Architecture & Code Flow**

“Let’s walk through the updated flowchart that powers our chatbot system. This architecture balances current infrastructure constraints with future scalability. and it’s built using LangChain, Chainlit, and Claude 3.5 via AWS Bedrock.”

“We begin with user identification. Since OAuth integration with Microsoft Entra ID is still pending, we’ve implemented a temporary workaround: users manually enter their first and last name. This acts as a lightweight user ID, allowing us to associate chat history with a specific individual. It’s simple, effective, and easy to swap out once secure authentication is available.”

“Next, the Chainlit app initializes the chat interface. At this stage, we query our PostgreSQL database to check for any existing conversations tied to the user ID. If chat history exists, it’s loaded into the side panel, giving users continuity and context from previous sessions—even if they’ve refreshed the page.”

“When a user sends a message, it’s passed to our LangChain pipeline. We’re using Claude 3.5 via AWS Bedrock as the LLM backend, and we’ve integrated ConversationBufferMemory to maintain session context. This ensures that the model understands the flow of the conversation and can respond intelligently to follow-ups.”

“The response is streamed back to the user in real time. Streaming improves the user experience by reducing perceived latency and making the interaction feel more dynamic and responsive.”

“Each interaction—both the user’s message and the model’s response—is saved to PostgreSQL, tagged with the user ID. This persistence layer is critical for maintaining continuity across sessions and enabling future analytics on user behavior and model performance.”

“Finally, once OAuth is available, we’ll replace the manual name entry with secure authentication via Microsoft Entra ID. This will allow us to tie chat history to verified corporate identities, enhancing both security and personalization.”

“Overall, this flow is modular, scalable, and designed to evolve. Whether we’re swapping out the database, upgrading the LLM, or integrating new authentication methods, the system is built to adapt without disrupting the user experience.”