Deepseek AI Chatbot Code Breakdown

Using Ollama & LangChain for Interactive Coding Assistance

Overview

INTERACTIVE AI CHATBOT BUILT WITH STREAMLIT

USES OLLAMA AS THE LLM BACKEND

PROVIDES DEBUGGING, DOCUMENTATION, AND CODE ASSISTANCE

Importing Required Libraries

- Streamlit for UI
- ChatOllama for LLM integration
- LangChain for structured prompt handling

Custom UI Styling

CSS to enhance readability & layout

Modifies background colors, input fields, and sidebar design

Sidebar Configuration

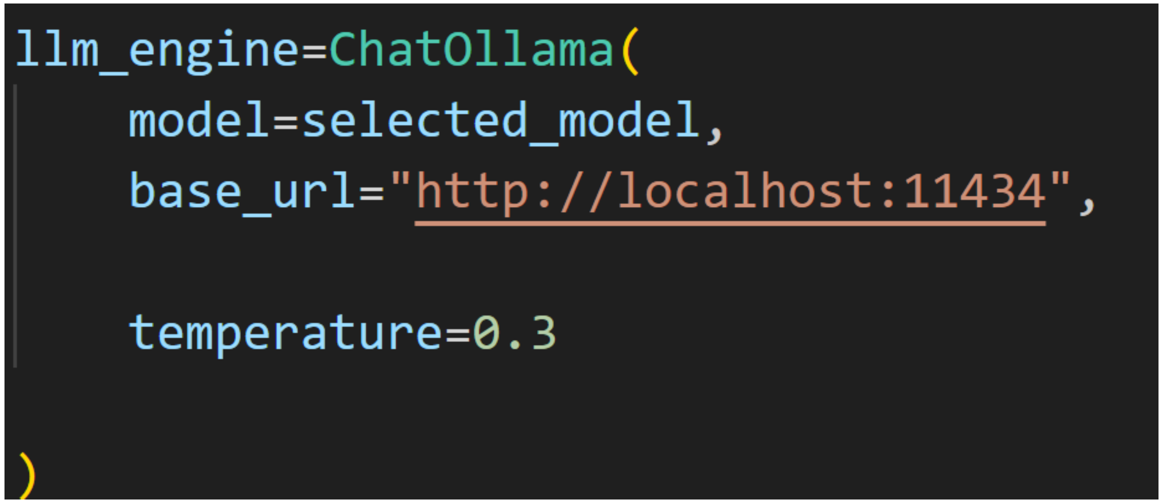
Model selection dropdown

Lists assistant capabilities

Provides credits to Ollama & LangChain

Initializing LLM Engine

- Uses ChatOllama with base URL
- Runs locally on port 11434
- Temperature set to 0.3 for controlled responses



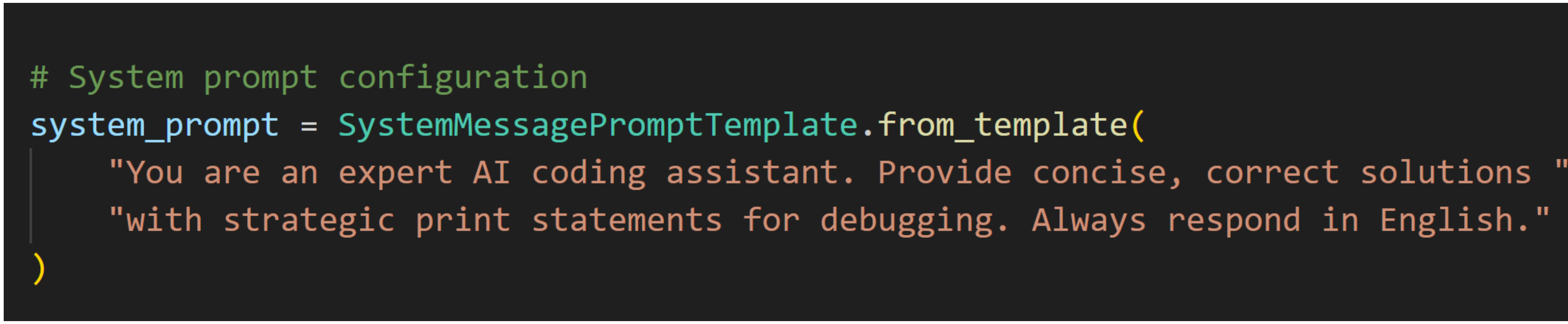
```
llm_engine=ChatOllama(
    model=selected_model,
    base_url="http://localhost:11434",

    temperature=0.3

)
```

System Prompt Configuration

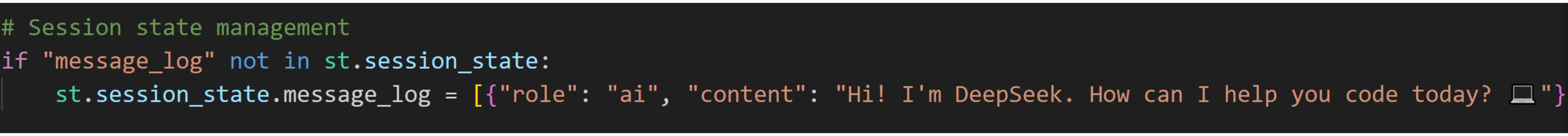
- Defines AI's role as a coding assistant
- Ensures concise, correct solutions
- Includes debugging strategies



```python
# System prompt configuration
system_prompt = SystemMessagePromptTemplate.from_template(
    "You are an expert AI coding assistant. Provide concise, correct solutions "
    "with strategic print statements for debugging. Always respond in English."
)
```

Chat Session Management

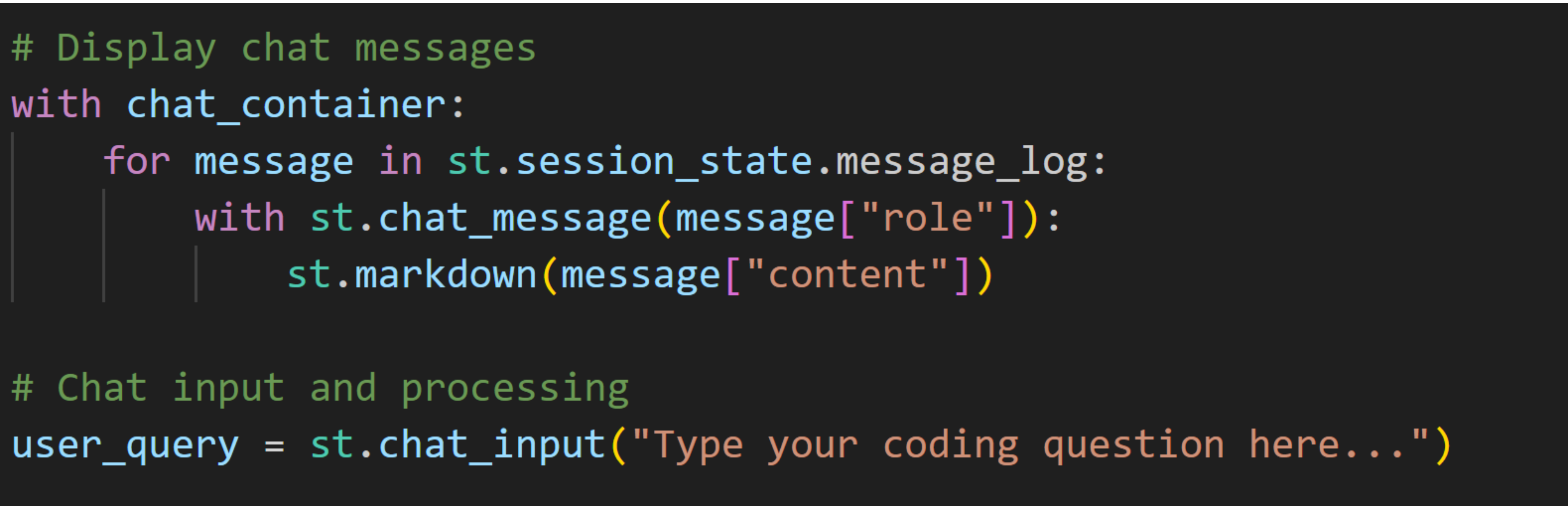
- Stores chat history using Streamlit session state
- Initializes with a welcome message from the AI



```python
# Session state management
if "message_log" not in st.session_state:
    st.session_state.message_log = [{"role": "ai", "content": "Hi! I'm DeepSeek. How can I help you code today? 💻"}
```

Chat Display & User Input

- Displays messages in a chat container
- User inputs queries through a text box

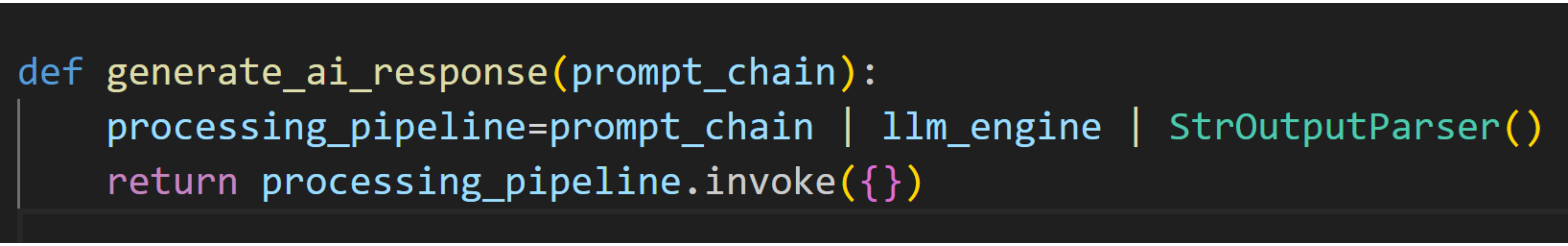


```python
# Display chat messages
with chat_container:
    for message in st.session_state.message_log:
        with st.chat_message(message["role"]):
            st.markdown(message["content"])

# Chat input and processing
user_query = st.chat_input("Type your coding question here...")
```

Generating AI Responses

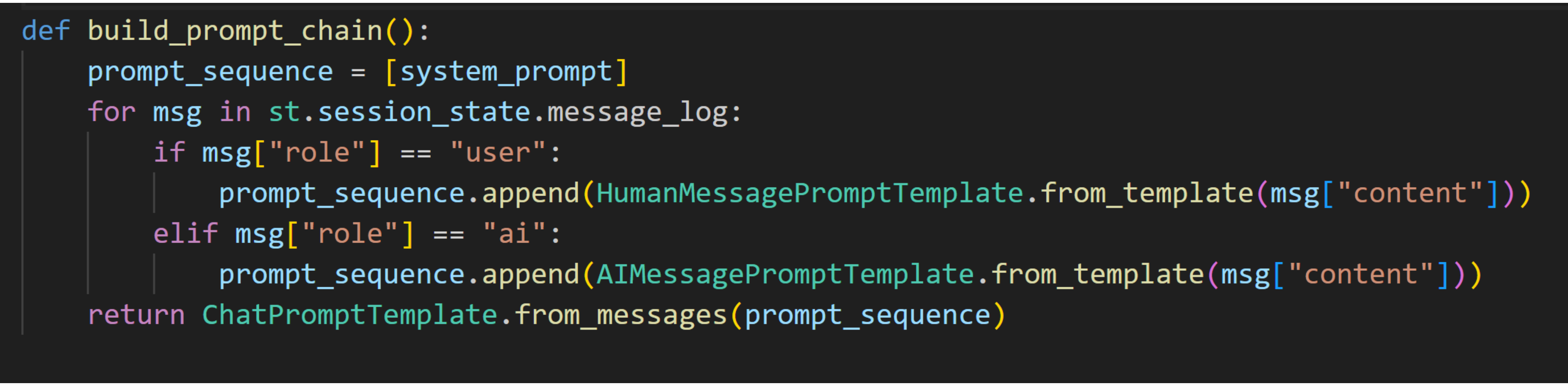
- Uses LangChain’s `|` operator for processing
- Chains the prompt, model, and output parser



```python
def generate_ai_response(prompt_chain):
    processing_pipeline=prompt_chain | llm_engine | StrOutputParser()
    return processing_pipeline.invoke({})
```

Constructing Chat Prompts

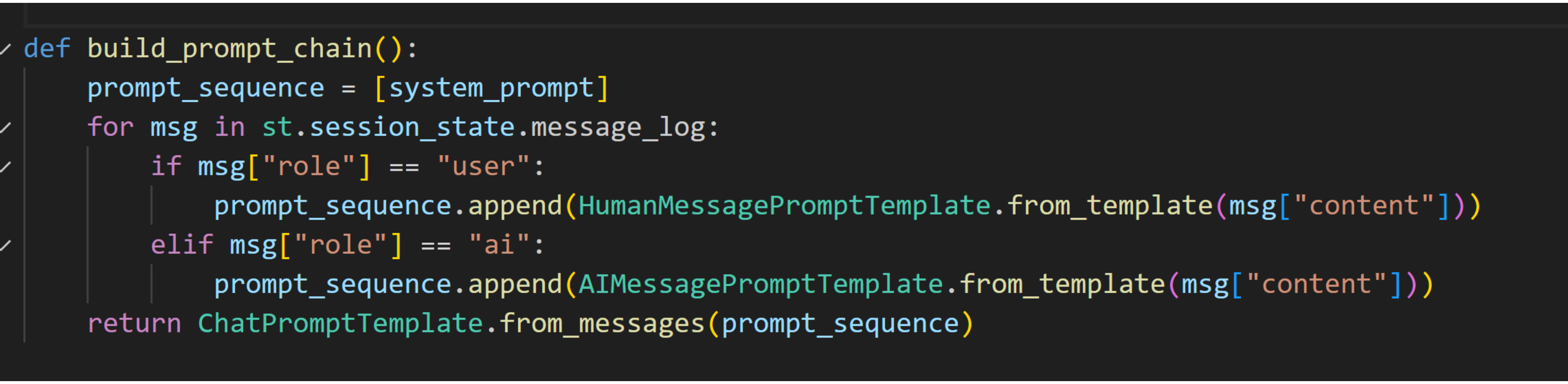
- Builds structured conversation history
- Includes system, user, and AI messages



```python
def build_prompt_chain():
    prompt_sequence = [system_prompt]
    for msg in st.session_state.message_log:
        if msg["role"] == "user":
            prompt_sequence.append(HumanMessagePromptTemplate.from_template(msg["content"]))
        elif msg["role"] == "ai":
            prompt_sequence.append(AIMessagePromptTemplate.from_template(msg["content"]))
    return ChatPromptTemplate.from_messages(prompt_sequence)
```

Processing User Queries

- Adds user messages to session state
- Generates AI response and updates the chat



```python
def build_prompt_chain():
    prompt_sequence = [system_prompt]
    for msg in st.session_state.message_log:
        if msg["role"] == "user":
            prompt_sequence.append(HumanMessagePromptTemplate.from_template(msg["content"]))
        elif msg["role"] == "ai":
            prompt_sequence.append(AIMessagePromptTemplate.from_template(msg["content"]))
    return ChatPromptTemplate.from_messages(prompt_sequence)
```

Key Features Summary

- Interactive coding assistant
- LLMpowered responses with debugging
- Persistent chat history
- Custom UI enhancements