# AI-Powered Email Assistant Chatbot: Architecture, Code Explanation, Libraries, and Implementation Thought Process

## Overview

This document explains the architecture, libraries used, code implementation, and the thought process behind the creation of an AI-powered Email Assistant Chatbot using Streamlit, Google's Generative AI, and a simple local NoSQL database (shelve). The purpose of this chatbot is to provide an intelligent assistant capable of interacting with users to manage emails in various ways, such as generating fake emails, querying past emails, and assisting with email-related tasks.

---

## Architecture

### High-Level Architecture

The architecture of the Email Assistant Chatbot is composed of several key components:

1. **Frontend (UI Layer)**: This is built using Streamlit, a Python library designed to create interactive web applications quickly. The Streamlit interface allows users to interact with the chatbot, generate fake emails, and view recent emails stored in a database.
2. **Backend (Processing Layer)**: The chatbot's responses are powered by Google's Generative AI model (Gemini Pro). This AI model interacts with the user input, processes the request, and generates meaningful responses for email management.
3. **Database (Storage Layer)**: A simple NoSQL storage system using the `shelve` library stores the generated fake emails. This database acts as a persistent layer to simulate real email data storage.
4. **External Libraries and APIs**:
    - **Faker**: Used to generate fake emails and related metadata.
    - **Google's Generative AI**: Provides natural language understanding and generation capabilities.
    - **Streamlit**: Facilitates creating an interactive user interface for seamless communication between the user and the AI system.

### System Components

1. **Streamlit Interface**: A web-based interface where users can input their queries, such as generating emails or querying the assistant about past emails.
2. **Generative AI Model (Google Gemini Pro)**: Powers the chatbot's natural language processing abilities, allowing it to generate coherent and intelligent responses related to email tasks.

3. **Local Email Storage**: The `shelve` library creates a persistent storage where generated emails are stored in the form of key-value pairs.

---

# Code Explanation

## Streamlit UI

The UI section of the code is designed using Streamlit's simple and effective syntax. It starts with:

```
st.title("✉ Email Assistant Chatbot")
```

This line sets the title for the web app. The following section contains buttons and inputs that allow the user to interact with the chatbot and the fake email generator:

```
if st.button("Generate Fake Emails"):
    message = store_emails_shelve(10)
    st.success(message)
```

When the "Generate Fake Emails" button is clicked, the `store_emails_shelve()` function is invoked to generate 10 fake emails using the `Faker` library, which are then stored in the `shelve` database.

Next, the code retrieves and displays recent emails stored in the database:

```
st.subheader("Recent Emails")
emails = get_emails()
for email in emails:
    st.write(email)
```

The `get_emails()` function fetches the first 5 stored emails from the `shelve` database and displays them in a user-friendly format.

## Email Assistant Chatbot Interaction

This part involves the chatbot where the user can input questions or tasks:

```
st.subheader("Chat with Email Assistant")
user_input = st.text_input("You:")
if st.button("Send"):
    if user_input:
        prompt = f"""
        You are an intelligent Email Assistant chatbot that understands and
helps with various email-related tasks.
        You can assist with:
        1) Sending emails
        2) Querying past emails
        3) Reading emails
        4) Deleting emails
        5) Forwarding emails
        6) Replying to emails
```

```
        Emails stored in the database:
        {get_emails()}

        Always provide clear and helpful responses regarding email
management.

        User Input: {user_input}
        """
        response = chat.send_message(prompt)
        st.text_area("Bot:", response.text, height=200)
```

- **User Input**: The user inputs their query or task in a text field.
- **Generative AI Interaction**: Once the "Send" button is pressed, the user's input is sent to the Google Generative AI model (Gemini Pro) as part of the prompt. The model generates a response, which is displayed in a text area.

The prompt sent to the AI model contains detailed instructions for the assistant on how to handle various email-related tasks, along with a list of the emails in the database. This ensures that the AI has contextual information about the email system.

## Email Storage and Retrieval

- **Storing Emails**: The `store_emails_shelve()` function uses the `Faker` library to generate fake email data (sender, receiver, subject, body, and timestamp). The generated data is stored in a `shelve` database, where each email is identified by a unique key (integer index).

```
def store_emails_shelve(n=10):
    fake = Faker()
    with shelve.open(db_path) as db:
        for i in range(n):
            email = {
                "sender": fake.email(),
                "receiver": fake.email(),
                "subject": fake.sentence(),
                "body": fake.paragraph(),
                "timestamp": str(fake.date_time_this_year())
            }
            db[str(i)] = email
    return f"Inserted {n} fake emails into the database."
```

- **Retrieving Emails**: The `get_emails()` function retrieves the first five emails from the `shelve` database and formats them for display.

```
def get_emails():
    emails = []
    with shelve.open(db_path) as db:
        for key in list(db.keys())[:5]:
            email = db[key]
            emails.append(f"From: {email['sender']}, Subject:
{email['subject']}")
    return emails
```

---

# Libraries and Tools Used

1. **Streamlit**: An open-source framework for creating web apps in Python. Streamlit allows quick creation of interactive front-end applications with minimal code. It is used here for building the UI and displaying the chat interface and generated email data.
2. **Google Generative AI**: This API allows the integration of advanced AI models like Gemini Pro, which enables intelligent and context-aware conversation generation. The AI model is responsible for interpreting the user's input and providing relevant email management responses.
3. **Faker**: A Python library used to generate fake data, which in this case is used to generate random email content, including senders, receivers, subjects, and email bodies.
4. **Shelve**: A Python library for persisting data between runs. It provides a simple dictionary-like interface to store key-value pairs on disk, which is used here for storing the generated emails.

---

# Thought Process Behind Implementation

1. **Email Management Simulation**: Since this is a simulation for an email assistant, using the `Faker` library allowed generating a realistic set of emails, mimicking real-world scenarios while testing the assistant's capabilities.
2. **Natural Language Processing**: By leveraging Google's Generative AI, the system is able to provide intelligent responses to user queries, making the assistant feel more natural and capable in handling various tasks related to email management.
3. **Simple Database Design**: The use of `shelve` for local storage was chosen because it is simple to implement and provides persistent data storage without requiring a complex database setup. This suffices for the scale of this project and simulates a real email database.
4. **User Interface Design**: The UI is designed to be intuitive and minimalistic. Streamlit's built-in features allow the rapid creation of an interface where users can interact with the system without much overhead.
5. **Extensibility**: This setup can be extended to include more features, such as email filtering, advanced querying, or sending real emails via SMTP, further enhancing the chatbot's capabilities.

---

# Conclusion

This project demonstrates how to build an AI-powered email assistant using simple tools and libraries. The architecture efficiently integrates frontend, backend, and storage components, while the Generative AI model powers natural, context-aware email management interactions. With this setup, the assistant can handle basic tasks like generating fake emails, querying emails, and responding to user requests effectively.