Real-Time Messaging App

This system implements a **real-time messaging application** that allows users to send and receive messages through a user interface powered by **Streamlit**, backed by an **SQLite** database for storing messages, and integrated with **Apache Kafka** for real-time message processing. The system uses Kafka's producer-consumer architecture to send and consume messages across different components, ensuring real-time updates between the producer (UI) and the consumer (database).

Kafka Integration in the System:

- **Kafka Topics**:
  - The system uses a Kafka topic named `sample-topic` where messages are sent and consumed. Each message consists of a `sender`, `recipient`, and `message` along with a timestamp.
- **Producer-Consumer Interaction**:
  - The **Kafka producer** is responsible for sending messages to the Kafka topic (`sample-topic`) whenever a user sends a new message via the UI. This happens in the `kafka_producer.py` file, where the `send_message` function sends the message to the topic.
  - The **Kafka consumer** listens to the Kafka topic and consumes messages in real-time. The consumer processes each message by extracting data from the Kafka message and stores it in the SQLite database using the `store_message` function in the `database.py` file. This ensures that every message sent by the producer is persisted in the database and can be retrieved later.
- **UI Interaction**:
  - The **Streamlit UI** allows users to enter their username and view/send messages. The `user_page.py` file handles the frontend logic, including:
    - Displaying messages retrieved from the SQLite database via the `get_messages_for_user` function.
    - Sending new messages via the Kafka producer when the user inputs a recipient and message and clicks the "Send Message" button.
    - Implementing real-time updates by periodically refreshing the list of messages every 5 seconds to display new messages as they are consumed from Kafka.

Summary of Workflow:

1. **Producer**: A user sends a message through the Streamlit UI. The message is sent to the Kafka topic (`sample-topic`) by the producer.
2. **Consumer**: The Kafka consumer listens for new messages on the Kafka topic and stores them in the SQLite database.
3. **UI**: The user interface periodically reloads the messages, showing new messages in real-time without requiring a manual refresh.

CODE

```
# database.py
import sqlite3

# Function to create a connection to the SQLite database
def create_connection():
    return sqlite3.connect('messages.db')

# Function to create a table for storing messages in the SQLite database
def create_table():
    try:
        conn = create_connection()
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS messages (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    sender TEXT,
                    recipient TEXT,
                    message TEXT,
                    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP)''')
        conn.commit()
        conn.close()
    except Exception as e:
        print(f"Error creating table: {e}")

# Function to store a message in the SQLite database
def store_message(sender, recipient, message):
    try:
        conn = create_connection()
        cursor = conn.cursor()
        cursor.execute("INSERT INTO messages (sender, recipient, message) VALUES (?, ?, ?)",
                       (sender, recipient, message))
        conn.commit()
        conn.close()
    except Exception as e:
        print(f"Error storing message: {e}")

# Function to retrieve messages for a given user, supporting pagination
def get_messages_for_user(username, page=1, page_size=10):
    try:
        conn = create_connection()
        cursor = conn.cursor()
        offset = (page - 1) * page_size
        cursor.execute('''SELECT sender, recipient, message, timestamp
                          FROM messages
                          WHERE recipient = ? OR sender = ?
                          ORDER BY timestamp DESC
                          LIMIT ? OFFSET ?''', (username, username, page_size, offset))
        messages = cursor.fetchall()
        conn.close()
        return messages
    except Exception as e:
        print(f"Error retrieving messages: {e}")
        return []

if __name__ == "__main__":
    create_table()
```

# kafka_consumer.py

```
from confluent_kafka import Consumer, KafkaException, KafkaError
import json
from database import store_message

# Kafka consumer configuration
conf = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'message_group',
    'auto.offset.reset': 'earliest'
}

consumer = Consumer(conf)

# Function to consume messages from the Kafka topic and store them in the database
def consume_messages():
    consumer.subscribe(['sample-topic'])

    try:
        while True:
            msg = consumer.poll(timeout=1.0)
            if msg is None:
                continue
            elif msg.error():
                if msg.error().code() == KafkaError._PARTITION_EOF:
                    print(f"End of partition reached {msg.topic()}/{msg.partition()}")
                else:
                    raise KafkaException(msg.error())
            else:
                data = json.loads(msg.value().decode('utf-8'))
                sender = data['sender']
                recipient = data['recipient']
                message = data['message']
                store_message(sender, recipient, message)
                print(f"Message stored: {data}")

    except KeyboardInterrupt:
        print("Consumer interrupted")
    finally:
        consumer.close()

if __name__ == "__main__":
    consume_messages()
```

# kafka_producer.py

```
from confluent_kafka import Producer
import json

# Kafka producer configuration
conf = {
    'bootstrap.servers': 'localhost:9092',
    'client.id': 'python-producer'
}

producer = Producer(conf)
```

```
# Function to send a message to the Kafka topic
def send_message(sender, recipient, message):
    data = {
        "sender": sender,
        "recipient": recipient,
        "message": message
    }
    producer.produce('sample-topic', value=json.dumps(data), callback=delivery_report)
    producer.flush()

# Callback function for message delivery confirmation
def delivery_report(err, msg):
    if err is not None:
        print(f"Message delivery failed: {err}")
    else:
        print(f"Message delivered to {msg.topic()} [{msg.partition()}]")

if __name__ == "__main__":
    send_message("user1", "user2", "Hello from user1!")
```

# user_page.py

```
import streamlit as st
from database import get_messages_for_user
from kafka_producer import send_message
import time

# Streamlit App
st.title("Real-Time Messaging App")
```

```
# Input username
username = st.text_input("Enter your username", key="username")

if username:
    st.write(f"Welcome, {username}!")

    # Session state setup
    if "page" not in st.session_state:
        st.session_state.page = 1
    if "messages" not in st.session_state:
        st.session_state.messages = []
    if "last_refresh" not in st.session_state:
        st.session_state.last_refresh = time.time()

    # Reload messages function
    def reload_messages():
        new_messages = get_messages_for_user(username, page=1, page_size=10)
        if new_messages:
            st.session_state.messages = new_messages

    # Periodic refresh
    if time.time() - st.session_state.last_refresh > 5:  # Refresh every 5 seconds
        reload_messages()
        st.session_state.last_refresh = time.time()

    # Display messages
    st.subheader("Messages")
    for msg in st.session_state.messages:
```

```
    st.write(f"{msg[0]} -> {msg[1]}: {msg[2]} at {msg[3]}")

# Pagination
if st.button("Load More Messages"):
    st.session_state.page += 1
    more_messages = get_messages_for_user(username, page=st.session_state.page, page_size=10)
    if more_messages:
        st.session_state.messages.extend(more_messages)
    else:
        st.warning("No more messages to load.")

# Send a new message
st.subheader("Send a Message")
recipient = st.text_input("Recipient", key="recipient")
message = st.text_area("Message", key="message")

if st.button("Send Message"):
    if recipient and message:
        send_message(username, recipient, message)
        st.success(f"Message sent to {recipient}!")
        reload_messages()  # Refresh the message list after sending
    else:
        st.error("Recipient and message cannot be empty.")
```