ChatGPT Frontend Clone (Streamlit-Based)

1. How It Will Work

✓ The sidebar lists all existing chats with the ability to start a **new chat**, **rename**, or **delete** chats.

- Chat title with rename and delete controls
- Conversation history (messages by user and assistant)
- A text box for typing new messages
- A "Send" button to post the message

On sending a message:

- The user's input is saved to the active chat.
- A **demo reply** is automatically generated (since no model API is connected).
- The conversation history updates instantly using `st.rerun()`.

Chats are stored locally in session state and saved to a JSON file for persistence. This means conversations remain available even after restarting the app.

2. Elements

Streamlit Sidebar

- New Chat button
- Search box to filter chats by title or content
- List of existing chats (clickable, rename, delete options)

Chat Window

- Title field (editable)
- Conversation display (user and assistant messages with timestamps)
- Input box for writing a message
- Send button

Storage Mechanism

- Chats saved as JSON for persistence
- Unique IDs generated for each chat
- Titles auto-generated from the first message or user-provided

Message Flow

- User messages stored as `{"role": "user", "text": ...}`
- Assistant messages simulated as demo replies
- Timestamps attached to all messages

3. Problems Encountered

Widget State Error

Attempting to clear a Streamlit input box directly using st.session_state[key] = "" after it was instantiated caused
StreamlitAPIException: cannot be modified after instantiation.

Fix: Removed forced clearing and relied on reruns for resetting inputs.

Deprecated Functions

- Older tutorials used st.experimental_rerun(), but this was removed in recent Streamlit versions.
- Fix: Replaced with the new st.rerun() method.

NoneType Errors

- Occurred when trying to access chat properties before initialization.
- Fix: Ensured every new chat is created with messages = [], title, and timestamps.

UI Alignment Issues

- Buttons like "Delete chat" and "Rename" were misaligned.
- Fix: Placed them inside proper layout containers (st.columns, st.container).

Persistence Handling

- Reloading the app would initially wipe chats.
- Fix: Implemented JSON storage to save and reload chat data.

4. Features

Chat Management

- Create new chats
- Rename chats
- Delete chats
- Search chats by title or content

Conversation Handling

- Add user messages
- Simulated assistant replies (demo)
- Auto-generated chat titles from the first message

Persistence

Chats are stored in JSON, ensuring sessions are not lost on reload

Streamlit UI

Sidebar for navigation

Main area for conversation and actions

Responsive rerendering with st.rerun().

Code:

```
import streamlit as st
import uuid
import json
import os
from datetime import datetime

# ---------- Config ----------
STORAGE_FILE = "chats.json"
MAX_TITLE_LEN = 60
```

```
# ---------- Utilities ----------
def load_chats():
    if not os.path.exists(STORAGE_FILE):
        return {}
    try:
        with open(STORAGE_FILE, "r", encoding="utf-8") as f:
            return json.load(f)
    except Exception:
        return {}
```

```
def save_chats(chats):
    with open(STORAGE_FILE, "w", encoding="utf-8") as f:
        json.dump(chats, f, ensure_ascii=False, indent=2)
```

```
def make_new_chat():
    cid = str(uuid.uuid4())
    now = datetime.utcnow().isoformat()
    return cid, {
        "id": cid,
        "title": None,
        "created_at": now,
        "updated_at": now,
        "messages": []
    }
```

```
def derive_title_from_first_message(text):
    if not text or not text.strip():
        return None
    t = text.strip().replace("\n", " ")
    if len(t) > MAX_TITLE_LEN:
        t = t[:MAX_TITLE_LEN].rsplit(" ", 1)[0] + "..."
    return t
```

```
def safe_title(title, existing_titles):
    base = title if title else "New chat"
    candidate = base
    i = 1
    while candidate in existing_titles:
        i += 1
        candidate = f"{base} {i}"
    return candidate
```

```
def now_iso():
    return datetime.utcnow().isoformat()
```

```
def get_active_chat():
    cid = st.session_state.active_chat_id
    if cid and cid in st.session_state.chats:
        return st.session_state.chats[cid]
```

```
    return None

# ---------- Streamlit App ----------
st.set_page_config(page_title="ChatGPT-like Clone", layout="wide")

# CSS styling
st.markdown(
    """
    <style>
    .stApp { background: #ffffff; color: #111827; }
    .chat-bubble-user { background:#e6ffe6; padding:10px; border-radius:10px; margin:4px 0; }
    .chat-bubble-assistant { background:#f1f0f0; padding:10px; border-radius:10px; margin:4px 0; }
    .title-bar { font-size:20px; font-weight:600; margin-bottom:6px; }
    .muted { color:#6b7280; font-size:13px; }
    </style>
    """,
    unsafe_allow_html=True,
)

# Load persistent chats
if "chats" not in st.session_state:
    st.session_state.chats = load_chats()

if "active_chat_id" not in st.session_state:
    st.session_state.active_chat_id = None

if "search_query" not in st.session_state:
    st.session_state.search_query = ""

# Sidebar
with st.sidebar:
    st.markdown("## ChatGPT (Local clone)")
    if st.button("+ New chat"):
        cid, new_chat = make_new_chat()
        st.session_state.chats[cid] = new_chat
        st.session_state.active_chat_id = cid
        save_chats(st.session_state.chats)
        st.rerun()

    st.markdown("---")
    q = st.text_input("Search chats", value=st.session_state.search_query, placeholder="Search by title or content")
    st.session_state.search_query = q

    st.markdown("### Chats")
    chats_list = list(st.session_state.chats.values())
    chats_list.sort(key=lambda c: c.get("updated_at",""), reverse=True)

    def matches(chat, q):
        if not q:
            return True
        ql = q.lower()
        if chat.get("title") and ql in chat.get("title","").lower():
            return True
        for m in chat.get("messages", []):
            if ql in m.get("text","").lower():
                return True
        return False

    shown = [c for c in chats_list if matches(c, q)]

    if not shown:
        st.markdown("*No chats yet. Create one with **New chat**.*")
```

```
    else:
        for c in shown:
            display_title = c.get("title") or "New chat"
            if st.button(display_title, key=f"open_{c['id']}"):
                st.session_state.active_chat_id = c["id"]
                st.rerun()
```

```
    st.markdown("---")
    st.markdown("### Library")
    st.write("Chats are saved locally in `chats.json`.")
    st.markdown("---")
    st.markdown("Hi there! ChatGPT here!!")
```

```
# Main area
active = get_active_chat()
```

```
if active is None:
    st.markdown("<div class='title-bar'>What can I help with?</div>", unsafe_allow_html=True)
    st.markdown("<div class='muted'>Start a new chat from the sidebar.</div>", unsafe_allow_html=True)
else:
    # Header row: Title + Rename + Delete
    col1, col2, col3 = st.columns([6, 2, 2])
    with col1:
        current_title = active.get("title") or "Chat"
        new_title = st.text_input("Chat title", value=current_title, key=f"title_{active['id']}")
        if new_title.strip() != current_title:
            existing = [c.get("title") for c in st.session_state.chats.values() if c["id"] != active["id"]]
            st.session_state.chats[active["id"]]["title"] = safe_title(new_title.strip(), existing)
            st.session_state.chats[active["id"]]["updated_at"] = now_iso()
            save_chats(st.session_state.chats)
    with col2:
        if st.button("Rename"):
            st.success("Title updated.")
    with col3:
        if st.button("  Delete chat"):
            del st.session_state.chats[active["id"]]
            save_chats(st.session_state.chats)
            st.session_state.active_chat_id = None
            st.rerun()
```

```
    st.markdown("---")
    # Messages
    for m in active.get("messages", []):
        if m["role"] == "user":
            st.markdown(f"<div class='chat-bubble-user'><b>You:</b> {m['text']}</div>", unsafe_allow_html=True)
        else:
            st.markdown(f"<div class='chat-bubble-assistant'><b>Assistant:</b> {m['text']}</div>",
unsafe_allow_html=True)
```

```
    st.markdown("---")
    # Input
user_input = st.text_area(
    "Message",
    key=f"input_{active['id']}",
    height=100,
    placeholder="Type a message..."
)
```

```
if st.button("Send"):
    text = user_input.strip()
    if text:
```

```
        # Save user message
        active["messages"].append({"role": "user", "text": text, "ts": now_iso()})
```

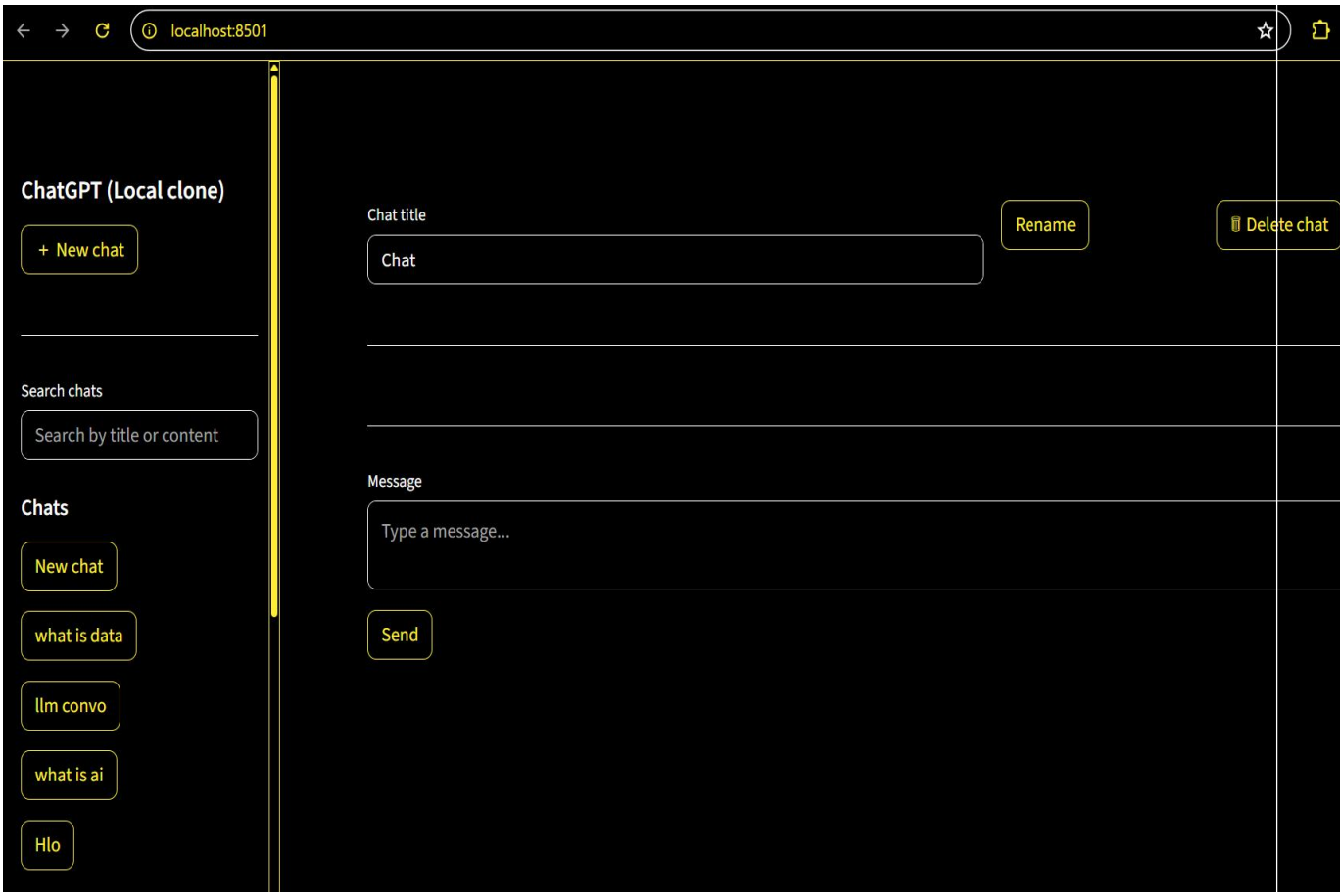
```
        # Set title if missing
        if not active.get("title"):
            t = derive_title_from_first_message(text)
            if t:
                existing = [
                    c.get("title")
                    for c in st.session_state.chats.values()
                    if c["id"] != active["id"]
                ]
                active["title"] = safe_title(t, existing)
```

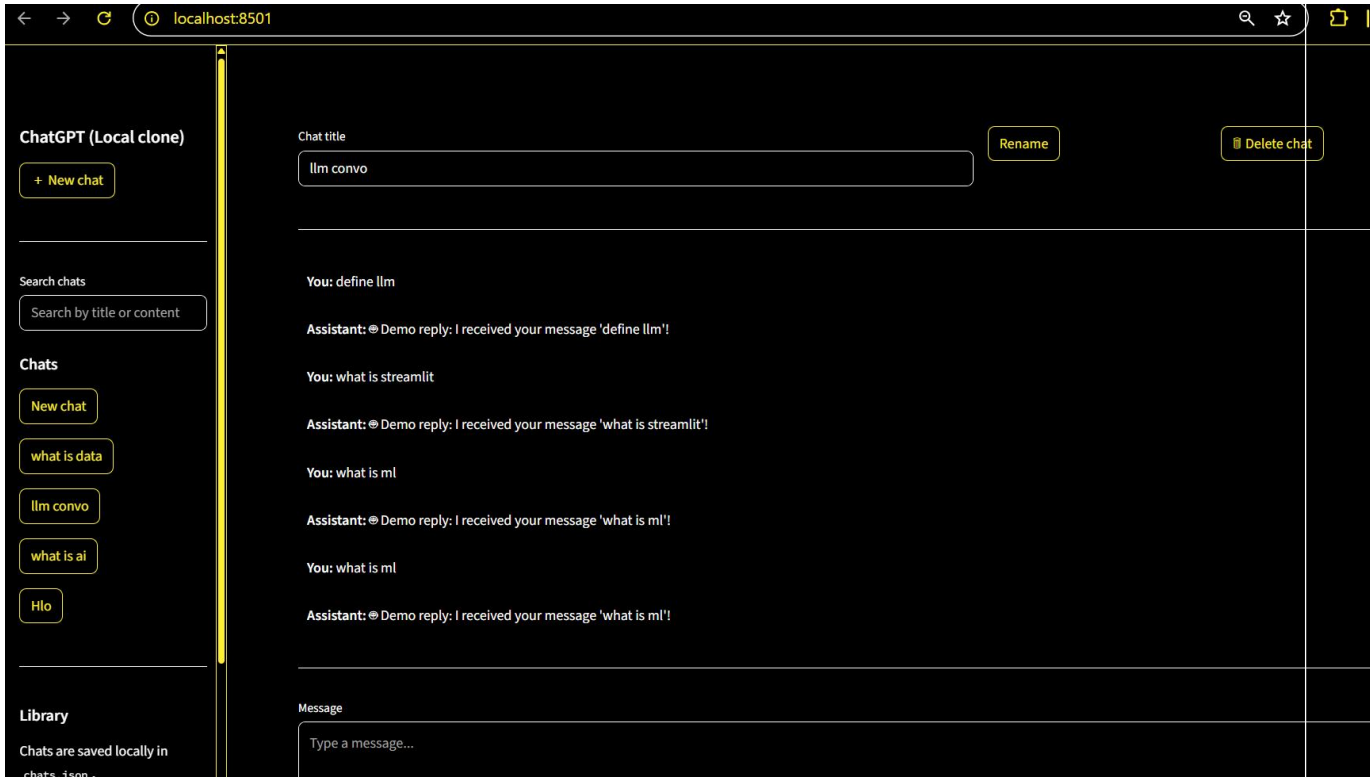
```
        # Demo assistant reply
        reply = f"  Demo reply: I received your message '{text}'!"
        active["messages"].append({"role": "assistant", "text": reply, "ts": now_iso()})
```

```
        # Update timestamp + save
        active["updated_at"] = now_iso()
        save_chats(st.session_state.chats)
```

```
        # Just rerun (no manual clearing needed)
        st.rerun()
```

Output




localhost:8501
ChatGPT (Local clone)
+ New chat
Search chats
Search by title or content
Chats
New chat
what is data
llm convo
what is ai
Hlo
Library
Chats are saved locally in
Chat title
llm convo
Rename
Delete chat
You: define llm
Assistant: Demo reply: I received your message 'define llm'!
You: what is streamlit
Assistant: Demo reply: I received your message 'what is streamlit'!
You: what is ml
Assistant: Demo reply: I received your message 'what is ml'!
You: what is ml
Assistant: Demo reply: I received your message 'what is ml'!
Message
Type a message...