

# An Empathetic Companion Chatbot for Alzheimer's Patients







Name: Nairit Das

---

## Overview

This is a **Streamlit app** for a chatbot called “**Memory Companion**”, designed to support individuals with memory challenges using **Google's Gemini API** (**gemini-1.5-pro**). It features a **friendly, empathetic assistant** that:

### Key Features:

-  **Memory-supportive UX:** The chatbot is styled for clarity and emotional comfort, using calm colors, rounded bubbles, and polite tone.
-  **Date Awareness:** It always knows today's date and can gently remind the user.
-  **Empathetic Messaging:** Detects memory-related phrases like “forgot” or “confused” and adds supportive language.
-  **Personalization:** Tracks names related to relationships like "wife", "doctor", or "son" to help the chatbot become more personal.
-  **Chat History:** Keeps a scrollable log of the conversation.
-  **Custom Styling:** Uses HTML/CSS in **st.markdown** for a beautiful, user-friendly chat layout.

This code creates a chatbot designed as a gentle companion for Alzheimer's patients. It uses Google's Generative AI model to generate empathetic responses and has a fun, visually appealing design. The whole experience is built to feel friendly and supportive!

## CODE

```
import streamlit as st

import google.generativeai as genai

from datetime import datetime

from streamlit.components.v1 import html


# Setup your Generative AI model

genai.configure(api_key="AlzaSyCeT_4AUUEzRY0D7HvOvhzkSSB9Ky-UaKg") # Replace
with your real API key

model = genai.GenerativeModel(

    model_name="gemini-1.5-pro",

    # safety_settings=[

        # {"category": "HARM_CATEGORY_HARASSMENT", "threshold": "BLOCK_NONE"},

        # {"category": "HARM_CATEGORY_HATE_SPEECH", "threshold": "BLOCK_NONE"},

        # {"category": "HARM_CATEGORY_SEXUALLY_EXPLICIT", "threshold":
"BLOCK_NONE"},

        # {"category": "HARM_CATEGORY_DANGEROUS_CONTENT", "threshold":
"BLOCK_NONE"},

    # ]

)


# Set up the main page configuration

today_date = datetime.now().strftime("%A, %B %d, %Y")

st.set_page_config(
```

```
page_title="Alzheimers Patient Chatbot🧠",  
page_icon="🧠",  
layout="wide"  
)
```

# Updated CSS with a new title color (bright teal)

```
st.markdown("""  
  
<style>  
  
  /* Page background with a soft gradient */  
  
  body {  
  
    background: linear-gradient(135deg, #FFFBFF, #C8E6C9);  
  
    font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;  
  
  }  
  
  /* Title styling with new color */  
  
  .app-title {  
  
    text-align: center;  
  
    font-size: 3rem;  
  
    font-weight: 700;  
  
    color: #008080;  
  
    margin-top: 1rem;  
  
  }  
  
  /* Chat section container */  
  
  .chat-box {
```

```
background-color: #FFFFFF;  
  
padding: 2rem;  
  
border-radius: 25px;  
  
box-shadow: 0px 8px 15px rgba(0,0,0,0.1);  
  
max-height: 70vh;  
  
overflow-y: auto;  
  
}
```

```
/* Chat bubbles for the user */
```

```
.chat-bubble.user {  
  
    background: #B3E5FC;  
  
    color: #01579B;  
  
    padding: 15px 20px;  
  
    border-radius: 20px 20px 0 20px;  
  
    margin: 1rem 0 1rem auto;  
  
    max-width: 65%;  
  
}
```

```
/* Chat bubbles for the bot */
```

```
.chat-bubble.bot {  
  
    background: #F1F8E9;  
  
    color: #33691E;  
  
    padding: 15px 20px;  
  
    border-radius: 20px 20px 20px 0;  
  
    margin: 1rem auto 1rem 0;
```

```
    max-width: 65%;
}

/* Input box styling */
.stTextInput > div > div > input {
    background-color: #FFF;
    color: #333;
    padding: 16px 20px;
    border-radius: 30px;
    border: 2px solid #B0BEC5;
    font-size: 1rem;
}

/* Water reminder box in the sidebar */
.water-reminder {
    background: #BBDEFB;
    border-radius: 15px;
    padding: 1.2rem;
    text-align: center;
    margin-top: 1rem;
    font-size: 1.1rem;
    font-weight: 600;
    color: #0D47A1;
    box-shadow: 0px 4px 10px rgba(0,0,0,0.1);
}
```

```

/* Scrollbar styling for chat area */

.chat-box::-webkit-scrollbar {

    width: 10px;

}

.chat-box::-webkit-scrollbar-track {

    background: #f0f0f0;

    border-radius: 10px;

}

.chat-box::-webkit-scrollbar-thumb {

    background: #ccc;

    border-radius: 10px;

}

</style>

"", unsafe_allow_html=True)

# Sidebar water reminder

with st.sidebar:

    st.markdown('<div class="water-reminder"> 💧 Stay Hydrated! Drink a glass of water
now.</div>', unsafe_allow_html=True)



    st.write("")

    st.markdown("<h4 style='text-align: center; color: #3E4E50;'>Alzheimers Patient
Chatbot 🧠</h4>", unsafe_allow_html=True)

    st.write("A gentle companion to help you remember the little things.")

```

# Title and Date on the main page

```
st.markdown(f'<div class="app-title">Alzheimers Patient Chatbotunsafe_allow_html=True)
```

```
st.markdown(f"""
```

```
    <div style="text-align:center; font-size: 1.3rem; margin-bottom: 2rem; color: #555;">
```

```
        Today's date: {today_date}
```

```
    </div>
```

```
""", unsafe_allow_html=True)
```

# Initialize session state variables if not already set

```
if "chat_history" not in st.session_state:
```

```
    st.session_state.chat_history = []
```

```
if "personal_details" not in st.session_state:
```

```
    st.session_state.personal_details = {}
```

# Set up the conversation context prompt

```
context_prompt = f"""
```

```
You are a gentle, patient companion for individuals with Alzheimers disease.
```

```
Today is {today_date}. Your responses should be:
```

1. EMPATHETIC: Respond kindly and reassuringly.
2. PERSONAL: Remember key details about the person's life.
3. CONCISE: Keep responses brief (1-2 sentences).
4. SUPPORTIVE: Avoid corrections or arguments, gently remind and care.

5. CURRENT: For any time questions, say "Today is {today\_date}.
6. STRAIGHT FORWARD: Always mention the term alzheimers as and when required."

Personal details to remember (if mentioned):

```
{st.session_state.personal_details}
```

```
"""
```

```
# Chat display container with delete options per message
```

```
with st.container():
```

```
    chat_placeholder = st.empty()
```

```
    with chat_placeholder.container():
```

```
        st.markdown('<div class="chat-box">', unsafe_allow_html=True)
```

```
# Use index to allow deletion of individual messages
```

```
for idx, (speaker, message) in enumerate(st.session_state.chat_history):
```

```
    cols = st.columns([0.9, 0.1])
```

```
    with cols[0]:
```

```
        if speaker == "You":
```

```
            st.markdown(f'<div class="chat-bubble user"><strong>You:</strong>  
{message}</div>', unsafe_allow_html=True)
```

```
        else:
```

```
            st.markdown(f'<div class="chat-bubble bot"><strong>Companion:</strong>  
{message}</div>', unsafe_allow_html=True)
```

```
    with cols[1]:
```



```
if st.button("Delete", key=f"delete_{idx}"):

    # Remove the message at index idx and re-run the script to update display

    st.session_state.chat_history.pop(idx)

    # Use experimental_rerun if available, or rely on natural re-run.

    try:

        st.experimental_rerun()

    except AttributeError:

        pass # If experimental_rerun is not available, it will update automatically.
```

```
st.markdown('</div>', unsafe_allow_html=True)
```

```
# User input area
```

```
with st.form("chat_input", clear_on_submit=True):
```

```
    user_input = st.text_input(

        "Your message:",

        placeholder="Talk to your companion...",

        key="input",

        label_visibility="collapsed"

    )
```

```
    submitted = st.form_submit_button("Send")
```

```
if submitted and user_input:
```

```
    # Append user message to chat history
```

```

st.session_state.chat_history.append(("You", user_input))

# If any relationship words are mentioned, extract names (simple heuristic)

relationships = ["husband", "wife", "son", "daughter", "child", "friend", "brother", "sister",
"partner", "doctor", "nurse", "caregiver"]

for term in relationships:

    if term in user_input.lower():

        name_parts = [word for word in user_input.replace(",", " ").split() if word[0].isupper()]

        if name_parts:

            st.session_state.personal_details[term] = name_parts[-1]

# Build conversation context with the latest messages

conversation_context = context_prompt + "\n\nRecent conversation:\n"

for speaker, message in st.session_state.chat_history[-6:]:

    conversation_context += f"{speaker}: {message}\n"

# Generate the companion's reply

try:

    response = model.generate_content(conversation_context)

    bot_reply = response.text.strip()

# Extra empathy for memory-related queries

memory_triggers = ["forgot", "confused", "remember", "who is", "what is", "where is",
"when did"]

```

```
if any(trigger in user_input.lower() for trigger in memory_triggers):
```

```
    bot_reply = f"I understand it can be hard to recall at times. {bot_reply}"
```

```
# For date/time questions, ensure consistency
```

```
if any(time_word in user_input.lower() for time_word in ["date", "day", "today", "time"]):
```

```
    bot_reply = f"Today is {today_date}."
```

```
except Exception as e:
```

```
    bot_reply = "I'm having a little trouble right now. Could you try asking again in a moment?"
```

```
# Append the bot's reply to the chat history
```

```
st.session_state.chat_history.append(("Companion", bot_reply))
```

```
try:
```

```
    st.experimental_rerun()
```

```
except AttributeError:
```

```
    pass
```

```
# Auto scroll to bottom (using JavaScript)
```

```
html("""
```

```
<script>
```

```
    window.onload = function() {
```

```
        var chatBox = document.querySelector('.chat-box');
```

```
        if(chatBox){ chatBox.scrollTop = chatBox.scrollHeight; }
    }

    window.addEventListener('load', function() {

        var chatBox = document.querySelector('.chat-box');

        if(chatBox){ chatBox.scrollTop = chatBox.scrollHeight; }

    });

</script>

""")
```

---

## Key Components Explained

### 1. Importing Libraries & Setup

- **Streamlit & Others:**  
The app uses Streamlit for the web interface, along with Google's generative AI for creating responses. Additionally, standard libraries like `datetime` help display the current date, and Streamlit's `html` component is used to insert custom JavaScript and CSS.
- **Generative AI Configuration:**  
The code configures the AI model (`gemini-1.5-pro`) using an API key. This key would normally be secured in an environment variable or other secret management system (something to consider for future improvements).

### 2. Page Design and Styling

- **Custom CSS Styling:**  
The CSS block adds a soft gradient background, modern typography, and colorful chat bubbles. This design creates a soothing and attractive interface for users.

- **Responsive Layout:**

With a wide layout and styled components, the app is designed to be both visually pleasing and mobile-friendly.

### 3. Sidebar with a Water Reminder

- **Encouraging Self-care:**

A light-hearted yet caring reminder is included in the sidebar (“Drink a glass of water now”) to promote good habits. This is great for Alzheimer’s patients who might benefit from gentle nudges to take care of themselves.

### 4. Session State Management

- **Chat History and Personal Details:**

The chatbot stores the conversation history and extracts any key personal details. It uses these details later to personalize conversations—crucial for a supportive, empathetic assistant.

### 5. Building the Conversation Context

- **Context Prompt:**

A pre-defined prompt is built into the chat engine to keep the chatbot’s responses empathetic, personal, concise, and supportive.

- **Key Instructions:** The assistant is reminded to always include the current date and ensure any references to “alzheimers” are clearly mentioned.
- **Personal Details:** Previous details mentioned in the conversation are appended, so the bot can remember “important stuff” like relationships if they come up.

### 6. Handling User Input & Generating Responses

- **User Input Collection:**

Users type in their messages in a clean input box. Once submitted, the text is appended to the session state chat history.

- **Generating AI Responses:**

The chat context (including the most recent messages) is sent to the AI model.

- **Error Handling:** If the AI call fails (maybe it's having a bad day), there's a gentle error message.
- **Triggering Empathy:** The code looks out for keywords (like "forgot" or "confused") and bumps up the empathy in the reply.
- **Time Reassurance:** When the user asks about the time or date, the response is set to always mention today's date!

- **Message Deletion Option:**

Each message on the chat can be deleted with a "Delete" button, allowing the user to clear out conversation history if needed.

## 7. Auto-Scrolling Mechanism

- **Smooth Chat Experience:**

With a bit of JavaScript injected via an HTML component, the chat automatically scrolls to the bottom. This way, new messages are always visible without manual scrolling.

---

## Step-by-Step Methodology

### 1. Initialization:

- **Import Libraries:** Load Streamlit, Google Generative AI library, and datetime.
- **Configure the AI Model:** Set up your API key and choose the model.

### 2. Design and Layout Setup:

- **Page Setup:** Configure page title, icon, and layout.
- **Inject CSS/JS:** Define custom styles for the title, chat bubbles, input boxes, and sidebar reminders.

### 3. Sidebar Setup:

- **Water Reminder:** Display a friendly water reminder.
- **App Description:** Provide the app's purpose using a brief description.

### 4. Main Content:

- **Title and Date:** Display the current date and a welcoming app title.
- **Session State:** Initialize chat history and personal details if not already set.

### 5. Conversation Context:

- **Context Prompt:** Create the base prompt that directs the AI's behavior.
- **Append Recent Conversation:** Include the last few chat messages to keep the context relevant.

### 6. Chat Interaction:

- **Display Messages:** Loop through the chat history to display user and bot messages.
- **Handle Message Deletion:** Allow deletion of individual messages using buttons tied to session state.

### 7. User Input and AI Response:

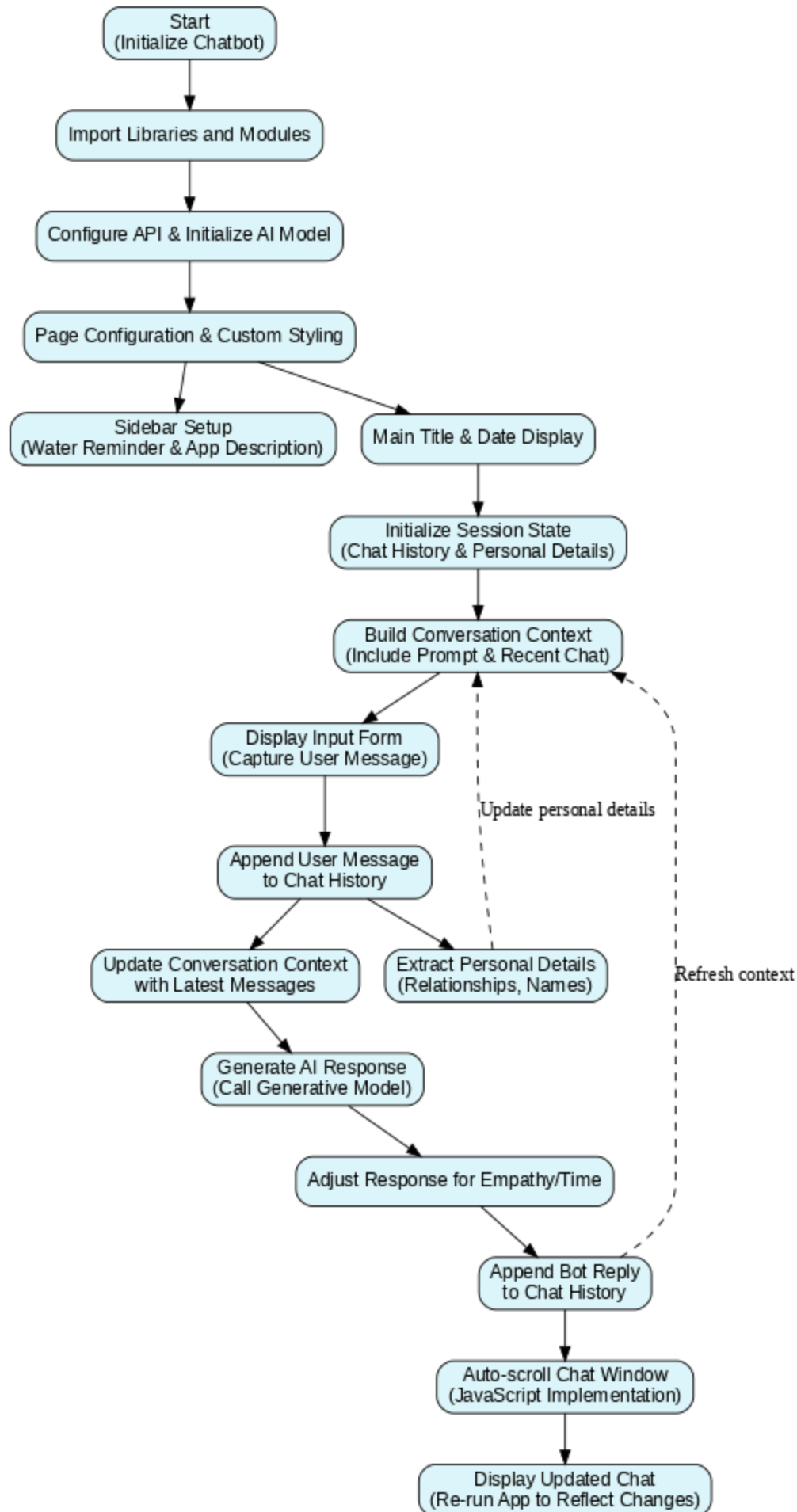
- **Collect User Message:** Get the input from the text box.
- **Update History:** Append the new message.
- **Extract Relationships:** Use simple heuristics (capitalized names) to remember personal details.
- **Generate Context and AI Response:** Formulate a context prompt with recent conversation snippets and get a response from the AI model.

- **Enhance Response for Empathy/Time Queries:** Adjust replies based on keywords for a caring tone.
- **Update Chat:** Append the response and refresh the display.

#### 8. **Scrolling:**

- **Auto-scroll Chat Box:** Use JavaScript to ensure the chat box scrolls down to the latest message.





---

## Future Scopes and Improvements

- **API Key Management:**  
Secure the API key (avoid hardcoding) by using environment variables or a secret management system.
  - **Enhanced Personalization:**  
Use better natural language processing (NLP) techniques to extract and manage personal details. This could also include sentiment analysis to adjust the bot's tone dynamically.
  - **Rich Interaction Elements:**
    - **Voice Input/Output:** Consider integrating speech-to-text and text-to-speech functionality for users who may have difficulties typing.
    - **Multimedia Support:** Add support for images or videos to make the interactions even richer.
  - **Accessibility Features:**  
Improve accessibility with features like larger text options, high-contrast themes, or even simplified language settings.
  - **More Detailed Memory Management:**  
Allow the chatbot to store and recall more detailed personal histories over multiple sessions while ensuring privacy and data protection.
  - **Error Handling and Robustness:**  
Enhance error messaging, perhaps even logging errors for further improvement, ensuring the bot remains responsive even during downtime.
  - **User Feedback Loop:**  
Implement feedback buttons so users can report if a response was especially helpful or if it needs tweaking. This could help fine-tune the model over time.
-

# Step By Step Explanation of the Code

Below is a casual yet detailed explanation of some of the most important code snippets in your Streamlit chatbot app for Alzheimer's patients. Each snippet is broken down with a friendly explanation of what it does and why it's key to the overall functionality.

---

## 1. Library Imports and Model Configuration

```
import streamlit as st
import google.generativeai as genai
from datetime import datetime
from streamlit.components.v1 import html
```

### What It Does:

This section imports the libraries needed for your application.

- **Streamlit:** Powers the web app UI.
  - **Google Generative AI:** Provides the generative model to create responses.
  - **Datetime:** Used to get and format the current date.
  - **HTML Component:** Enables embedding custom CSS and JavaScript for styling and functionality.
- 

## 2. API Key & Generative Model Setup

```
genai.configure(api_key="AIzaSyCeT_4AUEEzRY0D7HvOvhzkSSB9Ky-UaKg") # Replace with your
real API key
model = genai.GenerativeModel(
    model_name="gemini-1.5-pro",
)
```

### What It Does:

- **API Key Configuration:** Sets up the connection to Google's Generative AI service using your API key.

- **Model Initialization:** Chooses the specific model (here, `gemini-1.5-pro`) that will generate the chatbot responses.  
**Why It's Important:**  
This snippet is critical because it connects your app to the AI backend, making the chatbot's dynamic response generation possible.
- 

## 3. Page Configuration and Custom Styling

### Page Config Setup

```
today_date = datetime.now().strftime("%A, %B %d, %Y")
st.set_page_config(
    page_title="Alzheimers Patient Chatbot 🧠",
    page_icon="🧠",
    layout="wide"
)
```

#### What It Does:

- **Getting the Date:** Formats today's date in a friendly format.
- **Setting Page Config:** Customizes the app's title, icon, and layout to enhance the user experience.

### Injecting Custom CSS

```
st.markdown("""
<style>
  body {
    background: linear-gradient(135deg, #FFFBFF, #C8E6C9);
    font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;
  }
  .app-title {
    text-align: center;
    font-size: 3rem;
    font-weight: 700;
    color: #008080;
    margin-top: 1rem;
  }
  /* additional CSS for chat bubbles, input box, sidebar, etc. */
</style>

```

```
""", unsafe_allow_html=True)
```

#### What It Does:

- **Custom CSS Styling:** This block defines the visual style of your application, from the background gradient to the styling of chat bubbles and the input text field.

**Why It's Important:**

Visually, this code snippet makes the app inviting and comfortable for users, which is especially important for your target audience.

---

## 4. Sidebar and Main Title

### Sidebar Water Reminder

with st.sidebar:

```
    st.markdown('<div class="water-reminder"> 💧 Stay Hydrated! Drink a glass of water now.</div>',
unsafe_allow_html=True)
    st.markdown('<h4 style="text-align: center; color: #3E4E50;">Alzheimers Patient Chatbot 🧠</h4>',
unsafe_allow_html=True)
    st.write("A gentle companion to help you remember the little things.")
```

#### What It Does:

- **Friendly Reminder:** Encourages the user to stay hydrated—a simple way to remind users about self-care.
- **Sidebar Title and Description:** Reiterates the app's purpose in the sidebar, making navigation intuitive.

### Main Title and Date

```
st.markdown(f'<div class="app-title"> 🤖 Alzheimers Patient Chatbot 🧠</div>',
unsafe_allow_html=True)
st.markdown(f'""
<div style="text-align:center; font-size: 1.3rem; margin-bottom: 2rem; color: #555;">
    Today's date: {today_date}
</div>
""', unsafe_allow_html=True)
```

**What It Does:**

Displays a large, friendly title and the current date prominently on the main page, reinforcing context and ensuring the app feels up-to-date.

---

## 5. Session State Management for Chat History and Details

```
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "personal_details" not in st.session_state:
    st.session_state.personal_details = {}
```

**What It Does:**

- **Session State Initialization:** Ensures that conversation history and personal details are stored persistently across user interactions.

**Why It's Important:**

This makes sure that even as users interact and the page refreshes, the conversation remains intact and personal details can be remembered for a personalized experience.

---

## 6. Constructing the Conversation Context Prompt

```
context_prompt = f"""
```

You are a gentle, patient companion for individuals with Alzheimers disease.

Today is {today\_date}. Your responses should be:

1. EMPATHETIC: Respond kindly and reassuringly.
2. PERSONAL: Remember key details about the person's life.
3. CONCISE: Keep responses brief (1-2 sentences).
4. SUPPORTIVE: Avoid corrections or arguments, gently remind and care.
5. CURRENT: For any time questions, say "Today is {today\_date}."
6. STRAIGHT FORWARD: Always mention the term alzhimers as and when required."

Personal details to remember (if mentioned):

```
{st.session_state.personal_details}
"""
```

**What It Does:**

- **Building Context:** Sets out a clear prompt for the AI model that instructs it on the tone, style, and content to ensure that its responses are suitable for the target audience.

**Why It's Important:**

This context is essential to guide the AI in generating thoughtful and supportive responses, keeping the conversation consistent and appropriate for Alzheimer's patients.

---

## 7. Handling User Input and Chatbot Responses

### Capturing User Input and Appending to Chat History

```
with st.form("chat_input", clear_on_submit=True):
    user_input = st.text_input(
        "Your message:",
        placeholder="Talk to your companion...",
        key="input",
        label_visibility="collapsed"
    )
    submitted = st.form_submit_button("Send")

if submitted and user_input:
    st.session_state.chat_history.append(("You", user_input))
```

**What It Does:**

- **Input Form:** Displays a text box for the user to type their message.
- **Updating History:** Once the user hits “Send,” the message is added to the conversation history.

### Extracting Personal Details from Input

```
relationships = ["husband", "wife", "son", "daughter", "child", "friend", "brother", "sister", "partner",
                "doctor", "nurse", "caregiver"]
for term in relationships:
    if term in user_input.lower():
        name_parts = [word for word in user_input.replace(",", " ").split() if word[0].isupper()]
        if name_parts:
            st.session_state.personal_details[term] = name_parts[-1]
```

**What It Does:**

- **Heuristic Extraction:** Scans the input for keywords related to relationships.
- **Recording Details:** If it finds a match, it extracts a name (assuming it's capitalized) and stores it in `personal_details`.

**Why It's Important:**

Helps the chatbot personalize further interactions by remembering who's who—important for building a rapport with the user.

## Generating the AI Response

```
conversation_context = context_prompt + "\n\nRecent conversation:\n"
for speaker, message in st.session_state.chat_history[-6:]:
    conversation_context += f"{speaker}: {message}\n"
```

try:

```
    response = model.generate_content(conversation_context)
    bot_reply = response.text.strip()
    # Special tweaks for memory and time keywords...
```

except Exception as e:

```
    bot_reply = "I'm having a little trouble right now. Could you try asking again in a moment?"
```

### What It Does:

- **Building Context:** Concatenates the base context with the most recent messages (up to six) to maintain relevance.
- **Calling the Model:** Sends this context to the AI model and fetches a response.
- **Error Handling and Adjustments:** If there's an error or if memory/time keywords are detected, the code adjusts the reply accordingly.

**Why It's Important:**

This is where the magic happens—the chatbot crafts its friendly response based on your guided context and conversation, ensuring every reply is empathetic and context-aware.

## 8. Auto-Scrolling the Chat Window

```
html("""
<script>
    window.onload = function() {
```



```

    var chatBox = document.querySelector('.chat-box');
    if(chatBox){ chatBox.scrollTop = chatBox.scrollHeight; }
  }
  window.addEventListener('load', function() {
    var chatBox = document.querySelector('.chat-box');
    if(chatBox){ chatBox.scrollTop = chatBox.scrollHeight; }
  });
</script>
""")

```

### What It Does:

- **JavaScript Execution:** This code ensures that whenever a new message is added, the chat window automatically scrolls down so users always see the latest message.

### Why It's Important:

This small snippet greatly improves the user experience, avoiding manual scrolling and keeping the conversation flow smooth.

## Screenshot

