Web App with Streamlit

We're going to build a **Virtual Pet Web App using Python and Streamlit**.
This project is beginner-friendly and a fun way to practice using session state, user input, and simple logic

import streamlit as st

- This imports the Streamlit library, which is used to build web apps using Python.

1️⃣ Sidebar Setup – Choosing Your Pet

On the left side of the app, we have a sidebar where the user can:

- Select their **pet type** – Dog, Cat, or Fox.
- And also **give a name** to their pet, like "Snowy".

```
# --- SIDEBAR: Choose Pet Type and Name ---
st.sidebar.markdown("## 🐾 Pet Setup")

# Pet type selector
pet_type = st.sidebar.selectbox("Choose your pet type:",
["Dog", "Cat", "Fox"])

```

st.sidebar:

- This sends content to the **sidebar** of the app (the vertical panel on the left side of the screen).

.markdown(...):

- This function displays **text using Markdown formatting**, which allows you to add headers, bold, italics, emojis, links, etc

means it's a **level 2 heading** (like a subheading).

Pet Setup is the title for that section.

.selectbox(...):

- Displays a dropdown menu where users can select one option.

"Choose your pet type:":

- This is the **label** shown above the dropdown menu.

["Dog", "Cat", "Fox"]:

- These are the **choices** the user can pick from.

pet_type =:

- The selected value is saved into the variable pet_type, so you can use it later in the app (for example, to show an emoji or pet info).

```
# Set emoji based on type
if pet_type == "Dog":
    pet_emoji = "🐶"
elif pet_type == "Cat":
    pet_emoji = "🐱"
else:
    pet_emoji = "🦊"

```

- Based on the selected pet, assigns an **emoji** to show in the app.

```
# Pet name input
pet_name = st.sidebar.text_input("Give your pet a name:",
value="Snowy")
```

.text_input(...)

- Creates a **text input box** in the **sidebar**.

"Give your pet a name:"

- This is the **label** shown above the input box (guides the user)

`pet_name =`

- The name typed by the user is saved in the variable `pet_name`.
- Lets the user **type a name** for their pet. Default name is `"Snowy"`.

2️⃣ Tracking Pet Stats with Session State

> 🔁 We use `st.session_state` to **remember the pet's stats**, even when the app refreshes.

```
if 'last_pet_name' not in st.session_state or
st.session_state.last_pet_name != pet_name:

    st.session_state.hunger = 5

    st.session_state.energy = 5

    st.session_state.happiness = 5

    st.session_state.last_pet_name = pet_name
```

- Checks if the pet's name is new or not stored yet.
- If the name is new, it resets all pet stats: hunger, energy, happiness to 5.

if 'last_pet_name' not in st.session_state

- Whether `last_pet_name` is **not already stored** in `st.session_state`.

- if it **is stored**, check if the **current pet name is different** from the previous one.

If **either is true**, it means:

- This is a **new session**, or the user has **changed the pet's name**.

St.session_state

- lets you **store values between interactions** (like button clicks or name changes). It acts like **memory** for your app.

```
# Initialize session state if not already done
if 'hunger' not in st.session_state:
    st.session_state.hunger = 5
    st.session_state.energy = 5
    st.session_state.happiness = 5
    st.session_state.pet_name = "Snowy 🐾"
```

- If the app is running for the **first time**, it initializes the session.
- Sets default values for the first run.
- This checks if the key `'hunger'` exists in `st.session_state`.
- If it **doesn't exist**, that means it's the **first time** the app is running for this session.
- So, the app proceeds to **set default values**.
- Without this `if` check, the values would get reset **every time the app reloads**, even on small changes. This way, your pet's data stays **persistent** across app actions until the user explicitly resets it (like by changing the name).
- Sets the **default name** of the pet to `"Snowy 🐾"` the first time.
  This value can be updated later when the user types a new name in the input box.

```
st.session_state.hunger = 5
st.session_state.energy = 5
st.session_state.happiness = 5
```

This means we can update the pet's **hunger, energy, and happiness** each time we interact with it.

3️⃣ Setting Up the Main App View

- Displays the app title and a message introducing the pet using the name and emoji.

```
# --- Title and layout ---

st.title("🐾 Virtual Pet")

st.subheader(f"Say hello to {pet_name} the {pet_type.lower()}
{pet_emoji}!")
```

- Displays the app title and a message introducing the pet using the name and emoji.
- `.title` sets the **main title** of your app.

creates a **subheading** with personalized content.

Uses an **f-string** to dynamically display the pet's:

- **Name** (`pet_name`)
- **Type** (`pet_type`, converted to lowercase)
- **Emoji** (`pet_emoji`)

4️⃣ How the Pet Feels – Mood Function

🧠 We use a function called `get_mood()` to decide how the pet is feeling, based on its current stats:

```
# --- Mood logic ---
def get_mood():
    if st.session_state.hunger >= 8:
        return "🍽️ Hungry"
    elif st.session_state.energy <= 3:
        return "😴 Tired"
    elif st.session_state.happiness >= 7:
        return "😊 Happy"
    else:
        return "😐 Okay"
```

This function decides the pet's **mood** based on:

- High hunger → hungry
- Low energy → tired
- High happiness → happy
- Otherwise → okay

- `def get_mood():` defines a new function.
- This function **returns a mood string** based on the pet's current values in `st.session_state`.
- Mood Decision Using `if-elif-else`

```
# --- Display pet mood as text ---
def show_pet_image(mood):
    st.markdown(f"### 🖼️ {pet_name} feels {mood.lower()} {pet_emoji}")
```

- Just shows a **text message** describing the pet's current mood.
- `show_pet_image(mood)` is a function to **display a simple sentence** showing your pet's mood..
- st.markdown(f"### 🖼️ {pet_name} feels {mood.lower()} {pet_emoji}") Displays a **Markdown-formatted subheading** (### = level 3 header).
- Uses an emoji 🖼️ to represent an image (though this is text, not a real image).
- **`{pet_name}`**: shows the pet's name.

- `{mood.lower()}`: converts the mood text to **lowercase** for a friendly sentence style.
- `{pet_emoji}`: adds the correct emoji for the pet (dog, cat, or fox).

5️⃣ What Can You Do With Your Pet?

Below that, we define **four actions** the user can take:

```
# --- Actions ---
def feed():
    st.session_state.hunger = max(st.session_state.hunger - 2, 0)
    st.session_state.happiness += 1
    st.success("🍖 Yum! Your pet enjoyed the meal.")

def play():
    st.session_state.happiness += 2
    st.session_state.energy = max(st.session_state.energy - 2, 0)
    st.session_state.hunger += 1
    st.success("🎾 That was fun!")

def sleep():
    st.session_state.energy += 3
    st.session_state.hunger += 1
    st.success("😴 Your pet had a nice nap.")

def talk():
    st.info(f"🗣 {pet_name} says: 'I feel {get_mood()}'")
```

Each action changes the pet's stats:

`feed()` ➡️ `lowers hunger, increases happiness`
Reduces hunger by 2 points (but never below 0 using max(..., 0)).
Increases happiness by 1.
Shows a **success message** .

play() ➡️ boosts happiness, reduces energy
Increases happiness by 2.
Decreases energy by 2 (never below 0).
Increases hunger by 1 (playing makes pets hungry).
Shows a **success message**

sleep() ➡️ restores energy
Increases energy by 3 (rested).
Increases hunger by 1 (they get hungry after sleep).
Shows a **success message**.

talk() ➡️ just shows how your pet feels
Calls your earlier get_mood() function.
Displays a message showing how your pet feels (e.g., "I feel happy").

- **Updates your pet's stats**
- **Triggers feedback messages**
- Uses st.session_state so values persist as users interact with the app

---

6️⃣ Showing the Pet's Status

📊 We then display how the pet is doing:

```
# --- Show status ---
mood = get_mood()
st.markdown("### 📃 Pet's Current Status")
st.write(f"**Hunger:** {st.session_state.hunger} 🍗")
st.write(f"**Energy:** {st.session_state.energy} ⚡")
st.write(f"**Happiness:** {st.session_state.happiness} 🎈")
```

```
st.write(f"**Mood:** {mood}")

# --- Show mood text ---
show_pet_image(mood)
```

- Displays the pet's current values for hunger, energy, happiness, and mood.
- Displays the pet's emoji and mood again in a visual text format.

  Calls the `get_mood()` function and saves the result (like `"😊 Happy"`) into the variable `mood`.

  Displays a **level 3 header** using Markdown.

  Uses the 🧾 **emoji** to represent a "status report".

  `st.write()` prints the current values from `st.session_state` for each pet attribute.

  `**...**` makes the text **bold** using Markdown formatting.

  Emojis add a fun visual indicator for each stat:

  Shows the mood returned by `get_mood()` with the emoji.

  Example: `**Mood:** 😊 Happy`

`show_pet_image(mood)`:

- Passes the current `mood` (like `"😊 Happy"`) to the function.

- Inside that function, this happens:

  st.markdown(f"### 🖼️ {pet_name} feels {mood.lower()} {pet_emoji}")

So the user sees a line like:

🖼️ Snowy feels 😊 happy 🐶

Displays a **personalized, mood-based sentence** with:

- Pet name

- Mood (in lowercase)

- Pet emoji

It helps the user understand what the pet needs next.

7️⃣ Buttons to Interact with the Pet

> ✅ Then we add **buttons** for each action, so users can click to feed, play, sleep, or talk.

```
# --- Buttons for interaction ---
st.markdown("### 💬 What would you like to do?")
col1, col2, col3, col4 = st.columns(4)

with col1:
    if st.button("🍽️ Feed"):
        feed()
with col2:
    if st.button("🎮 Play"):
        play()
with col3:
    if st.button("🛌 Sleep"):
        sleep()
with col4:
    if st.button("🗣️ Talk"):
        talk()
```

- Adds a **subheading** prompting the user to choose an action.

- Creates 4 columns for buttons in a row.
- Displays a **button** labeled `"🍽️ Feed"` in the first column.
- When clicked, it **calls the `feed()` function** to reduce hunger and increase happiness.
- `"🎮 Play"` button → Calls `play()` function to increase happiness, reduce energy, increase hunger.
- `"🛌 Sleep"` button → Calls `sleep()` function to increase energy and hunger.
- `"🗣️ Talk"` button → Calls `talk()` to show how your pet feels.
  The user can click any button, and the stats update immediately.

8️⃣ Restarting the Game

🔄 Lastly, we have a **Reset** button that clears all data and restarts the app fresh.

```
# --- Reset Button ---
if st.button("🔁 Restart Pet"):
    for key in list(st.session_state.keys()):
        del st.session_state[key]
    st.rerun()
```

Displays a button labeled "🔁 Restart Pet".

If the user clicks it, the following code inside the if block runs.

Loops through all keys in st.session_state (like hunger, energy, happiness, pet_name, etc.)

St.session_state **Stores** pet's current stats

del … **Deletes each key**, clearing all stored data (like wiping the memory).

st.rerun() **Restarts the Streamlit app** immediately.

This reloads the page and reinitializes all variables (like starting fresh).

🎉 Wrap-Up

And that's it! A fully working virtual pet app made with just Python and Streamlit!
You can now **customize it** further, maybe add pet images, more moods, or even sound effects.

FULL CODE

import streamlit as st

# --- SIDEBAR: Choose Pet Type and Name ---
st.sidebar.markdown("## 🐾 Pet Setup")

```
# Pet type selector
pet_type = st.sidebar.selectbox("Choose your pet type:", ["Dog", "Cat", "Fox"])

# Set emoji based on type
if pet_type == "Dog":
    pet_emoji = "🐶"
elif pet_type == "Cat":
    pet_emoji = "🐱"
else:
    pet_emoji = "🦊"

# Pet name input
pet_name = st.sidebar.text_input("Give your pet a name:", value="Snowy")

# Save the name to session state
if 'last_pet_name' not in st.session_state or st.session_state.last_pet_name != pet_name:
    st.session_state.hunger = 5
    st.session_state.energy = 5
    st.session_state.happiness = 5
    st.session_state.last_pet_name = pet_name

# Initialize session state if not already done
if 'hunger' not in st.session_state:
    st.session_state.hunger = 5
    st.session_state.energy = 5
    st.session_state.happiness = 5
    st.session_state.pet_name = "Snowy 🐾"

# --- Title and layout ---
st.title("🐾 Virtual Pet")
st.subheader(f"Say hello to {pet_name} the {pet_type.lower()} {pet_emoji}!")

# --- Mood logic ---
def get_mood():
    if st.session_state.hunger >= 8:
        return "🍽️ Hungry"
    elif st.session_state.energy <= 3:
        return "😴 Tired"
    elif st.session_state.happiness >= 7:
        return "😊 Happy"
    else:
        return "😐 Okay"

# --- Display pet mood as text ---
```

```
def show_pet_image(mood):
    st.markdown(f"### 🖼️ {pet_name} feels {mood.lower()} {pet_emoji}")

# --- Actions ---
def feed():
    st.session_state.hunger = max(st.session_state.hunger - 2, 0)
    st.session_state.happiness += 1
    st.success("🍖 Yum! Your pet enjoyed the meal.")

def play():
    st.session_state.happiness += 2
    st.session_state.energy = max(st.session_state.energy - 2, 0)
    st.session_state.hunger += 1
    st.success("🎾 That was fun!")

def sleep():
    st.session_state.energy += 3
    st.session_state.hunger += 1
    st.success("😴 Your pet had a nice nap.")

def talk():
    st.info(f"🗣️ {pet_name} says: 'I feel {get_mood()}'")

# --- Show status ---
mood = get_mood()
st.markdown("### 🧾 Pet's Current Status")
st.write(f"**Hunger:** {st.session_state.hunger} 🍗")
st.write(f"**Energy:** {st.session_state.energy} ⚡")
st.write(f"**Happiness:** {st.session_state.happiness} 🎈")
st.write(f"**Mood:** {mood}")

# --- Show mood text ---
show_pet_image(mood)

# --- Buttons for interaction ---
st.markdown("### 💬 What would you like to do?")
col1, col2, col3, col4 = st.columns(4)

with col1:
    if st.button("🍽️ Feed"):
        feed()
with col2:
    if st.button("🎮 Play"):
        play()
```

```
with col3:
    if st.button("🛌 Sleep"):
        sleep()
with col4:
    if st.button("🗣️ Talk"):
        talk()

# --- Reset Button ---
if st.button("🔁 Restart Pet"):
    for key in list(st.session_state.keys()):
        del st.session_state[key]
    st.rerun()
```