

lessoncode2

L2_App_HTML_file.py

This lesson extends the previous one by **allowing users to download AI-generated HTML files**. Instead of just previewing the HTML in a browser, this script **saves the HTML as a file** and provides a **download link**.

Code Explanation

1 Running the AI Model & Cleaning Responses

```
import gradio as gr
import subprocess
import re
import tempfile
import os
```

- **Newly Introduced:**

- `os` → Handles file paths for saving the HTML file.

```
def run_ollama_prompt(prompt, model):
    """Runs a prompt using Ollama CLI and returns the response."""
    command = ["ollama", "run", model, prompt]

    try:
        result = subprocess.run(command, capture_output=True, text=True, c
heck=True, encoding="utf-8")
        response = result.stdout.strip()
        # Remove <think>...</think> tags if present
        response = re.sub(r'<think>.*?</think>', '', response, flags=re.DOTAL
L).strip()
        return response
    except subprocess.CalledProcessError as e:
        return f"Error running Ollama: {e}"
```

Key Additions:

- Still removes `<think>` tags as in previous lessons.
- Handles errors safely without crashing.

2 Extracting & Saving HTML Code

```
def save_html_file(response):
    if "```html" in response.lower():
        match = re.search(r'```html\n(?:.*?)```', response, re.DOTALL)
        if match:
            html_content = match.group(1).strip()
        else:
            return None
    elif response.strip().lower().startswith("<!doctype html>") or "<html" in response.lower():
        html_content = response
    else:
        return None

    temp_file_path = os.path.join(tempfile.gettempdir(), "output.html")
    with open(temp_file_path, "w", encoding="utf-8") as temp_file:
        temp_file.write(html_content)

    return temp_file_path
```

Understanding This Part

- What Problem Does This Solve?
 - AI sometimes outputs HTML inside triple backticks (````html`).
 - Sometimes, it **directly returns** an HTML document.
 - This function **extracts the HTML correctly** in both cases.
- How Does It Work?
 1. Checks if the response contains ````html`
 - If yes, it extracts the HTML inside the triple backticks.
 2. Checks if the response starts with `<!DOCTYPE html>` or `<html>`
 - If yes, it treats it as a full HTML document.

3. Saves the extracted HTML in a **temporary file**.
 4. Returns the **file path** so Gradio can use it for downloads.
-

3 Connecting the Function to Gradio

```
def generate_html_download(response):  
    file_path = save_html_file(response)  
    if file_path:  
        return file_path  
    return None
```

- This function **returns the saved HTML file's path** for **download**.
-

4 Building the Web Interface

```
models = ["deepseek-r1:1.5b", "deepseek-r1:14b", "deepseek-r1:32b", "phi  
4:latest", "llama2:7b"]  
  
with gr.Blocks() as app:  
    gr.Markdown("# Ollama Chat App")  
  
    with gr.Row():  
        model_selector = gr.Dropdown(models, label="Select Model", value  
="deepseek-r1:1.5b")  
  
    with gr.Row():  
        prompt_input = gr.Textbox(label="Enter your prompt", placeholder="T  
ype something...")  
  
    with gr.Row():  
        submit_button = gr.Button("Generate Response")  
  
    response_output = gr.Textbox(label="Response", interactive=False)  
  
    with gr.Row():  
        html_download = gr.File(label="Download HTML File")  
        html_preview_button = gr.Button("Generate HTML File")
```

```
submit_button.click(fn=generate_response, inputs=[prompt_input, model
_selector], outputs=response_output)
html_preview_button.click(fn=generate_html_download, inputs=response
_output, outputs=html_download)

app.launch()
```

♦ **New Features Compared to Previous Lesson:**

- **Downloadable HTML file:**

- `gr.File(label="Download HTML File")` → Provides a download button.
- **The user can now save the HTML and use it later.**

- **HTML generation button:**

- `html_preview_button.click(fn=generate_html_download, inputs=response_output, outputs=html_download)`
 - Clicking this **creates an HTML file** and **offers it as a download**.
-