# backend/main.py

# This file contains the main FastAPI application, including API endpoints,

# background tasks for document processing, and OpenAI integration.

import os

import shutil

import uuid

import json

from fastapi import FastAPI, File, UploadFile, HTTPException, BackgroundTasks, status

from fastapi.middleware.cors import CORSMiddleware

from fastapi.responses import FileResponse, JSONResponse

import aiofiles

import docx

from pydantic import BaseModel

import openai # Import the OpenAI library

# --- Application Setup ---

app = FastAPI(title="AI RFP Generator")

# --- OpenAI API Configuration ---

# IMPORTANT: Replace "YOUR\_OPENAI\_API\_KEY" with your actual OpenAI API key.

# For better security, use environment variables.

openai.api\_key = "YOUR\_OPENAI\_API\_KEY"

# CORS (Cross-Origin Resource Sharing) middleware

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Restrict in production

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# --- Directory Setup ---

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

UPLOAD\_DIR = os.path.join(BASE\_DIR, "uploads")

GENERATED\_DIR = os.path.join(BASE\_DIR, "generated")

PROMPTS\_DIR = os.path.join(BASE\_DIR, "prompts")

os.makedirs(UPLOAD\_DIR, exist\_ok=True)

os.makedirs(GENERATED\_DIR, exist\_ok=True)

os.makedirs(PROMPTS\_DIR, exist\_ok=True)

# --- In-memory "database" for tracking task status ---

tasks\_db = {}

class TaskStatus(BaseModel):

status: str

message: str

file\_path: str | None = None

# --- Prompt Loading Function ---

def load\_prompts(filename: str) -> dict:

"""Loads prompts from a JSON file in the prompts directory."""

try:

with open(os.path.join(PROMPTS\_DIR, filename), 'r') as f:

return json.load(f)

except FileNotFoundError:

# Fallback if file doesn't exist, though it should.

print(f"Warning: Prompt file '{filename}' not found. Using empty prompts.")

return {}

except json.JSONDecodeError:

print(f"Error: Could not decode JSON from '{filename}'.")

return {}

# --- OpenAI API Integration ---

async def call\_openai\_api\_for\_extraction(text: str, prompt: str) -> dict:

"""Calls the OpenAI API to extract structured JSON data from text."""

print(f"--- Calling OpenAI API for Extraction: {prompt[:30]}... ---")

try:

response = await openai.ChatCompletion.acreate(

model="gpt-4o",

messages=[

{"role": "system", "content": "You are an expert financial analyst extracting data from RFPs. Respond ONLY with valid JSON matching the user's requested schema."},

{"role": "user", "content": f"{prompt}\n\n--- RFP TEXT ---\n{text}"}

],

temperature=0.1,

response\_format={"type": "json\_object"}

)

return json.loads(response.choices[0].message['content'])

except Exception as e:

print(f"OpenAI API Error (Extraction): {e}")

return {"error": str(e)}

async def call\_openai\_api\_for\_generation(context\_summary: str, prompt: str) -> str:

"""Calls the OpenAI API to generate a section of the new RFP."""

print(f"--- Calling OpenAI API for Generation: {prompt[:30]}... ---")

try:

response = await openai.ChatCompletion.acreate(

model="gpt-4o",

messages=[

{"role": "system", "content": "You are a professional RFP document generator. Write the requested section clearly and formally based on the provided context."},

{"role": "user", "content": f"{prompt}\n\n--- CONTEXT FROM EXTRACTED RFPS ---\n{context\_summary}"}

],

temperature=0.5,

)

return response.choices[0].message['content']

except Exception as e:

print(f"OpenAI API Error (Generation): {e}")

return f"Error generating content: {e}"

# --- Document Processing Logic ---

def extract\_text\_from\_docx(filepath: str) -> str:

try:

doc = docx.Document(filepath)

return "\n".join([para.text for para in doc.paragraphs])

except Exception as e:

print(f"Error reading docx {filepath}: {e}")

return ""

def add\_content\_to\_doc(doc, content):

for line in content.split('\n'):

if line.startswith('## '):

doc.add\_heading(line.lstrip('## '), level=2)

elif line.startswith('# '):

doc.add\_heading(line.lstrip('# '), level=1)

elif line.strip():

doc.add\_paragraph(line)

async def process\_uploaded\_rfps(task\_id: str, filenames: list[str]):

try:

tasks\_db[task\_id] = TaskStatus(status="processing", message="Loading AI prompts...")

# Load the full set of prompts from external files

prompts\_extraction = load\_prompts("extraction\_prompts.json")

prompts\_generation = load\_prompts("generation\_prompts.json")

tasks\_db[task\_id] = TaskStatus(status="processing", message="Starting comprehensive RFP analysis...")

all\_extracted\_data = []

for idx, filename in enumerate(filenames):

tasks\_db[task\_id] = TaskStatus(status="processing", message=f"Reading file {idx+1}/{len(filenames)}: {filename}")

filepath = os.path.join(UPLOAD\_DIR, filename)

text = extract\_text\_from\_docx(filepath)

if not text: continue

doc\_extractions = {"original\_filename": filename}

for i, (key, prompt) in enumerate(prompts\_extraction.items()):

tasks\_db[task\_id] = TaskStatus(status="processing", message=f"AI Analysis on {filename} (Step {i+1}/{len(prompts\_extraction)} - {key})...")

extracted\_section = await call\_openai\_api\_for\_extraction(text, prompt)

doc\_extractions[key] = extracted\_section

all\_extracted\_data.append(doc\_extractions)

# --- Enhanced Knowledge Base Simulation and Context Generation ---

tasks\_db[task\_id] = TaskStatus(status="processing", message="Aggregating all extracted data...")

# Create a rich summary of all extracted data for the generation phase

synthesis\_context = {"documents\_analyzed": len(all\_extracted\_data)}

aggregated\_summaries = [doc.get("prompt\_1\_summary", {}).get("summary\_text") for doc in all\_extracted\_data]

synthesis\_context["aggregated\_summaries"] = [s for s in aggregated\_summaries if s]

aggregated\_questions = []

for doc in all\_extracted\_data:

reqs = doc.get("prompt\_5\_requirements", {}).get("categorized\_requirements", [])

if isinstance(reqs, list):

for category in reqs:

aggregated\_questions.extend(category.get("questions", []))

synthesis\_context["common\_questions"] = list(set(aggregated\_questions))

# Convert the rich context into a string for the AI

context\_summary\_for\_generation = json.dumps(synthesis\_context, indent=2)

tasks\_db[task\_id] = TaskStatus(status="processing", message="Synthesizing new RFP template...")

generated\_doc = docx.Document()

generated\_doc.add\_heading('Generated Request for Proposal (RFP)', level=0)

for i, (heading, prompt) in enumerate(prompts\_generation.items()):

tasks\_db[task\_id] = TaskStatus(status="processing", message=f"Generating section {i+1}/{len(prompts\_generation)}: {heading}...")

section\_content = await call\_openai\_api\_for\_generation(context\_summary\_for\_generation, prompt)

add\_content\_to\_doc(generated\_doc, section\_content)

output\_filename = f"Generated\_RFP\_{task\_id}.docx"

output\_filepath = os.path.join(GENERATED\_DIR, output\_filename)

generated\_doc.save(output\_filepath)

tasks\_db[task\_id] = TaskStatus(status="completed", message="RFP generated successfully!", file\_path=output\_filepath)

except Exception as e:

print(f"Error during background task {task\_id}: {e}")

tasks\_db[task\_id] = TaskStatus(status="error", message=str(e))

# --- API Endpoints ---

@app.post("/generate-rfp/")

async def generate\_rfp\_endpoint(background\_tasks: BackgroundTasks, files: list[UploadFile] = File(...)):

if openai.api\_key == "YOUR\_OPENAI\_API\_KEY":

raise HTTPException(status\_code=500, detail="OpenAI API key not configured on the backend.")

if len(files) > 7:

raise HTTPException(status\_code=400, detail="Please upload a maximum of 7 files.")

task\_id = str(uuid.uuid4())

filenames = []

for file in files:

if not file.filename.endswith(".docx"):

raise HTTPException(status\_code=400, detail=f"Unsupported file type: {file.filename}. Only .docx is supported.")

filepath = os.path.join(UPLOAD\_DIR, f"{task\_id}\_{file.filename}")

async with aiofiles.open(filepath, 'wb') as out\_file:

await out\_file.write(await file.read())

filenames.append(f"{task\_id}\_{file.filename}")

background\_tasks.add\_task(process\_uploaded\_rfps, task\_id, filenames)

return JSONResponse(status\_code=status.HTTP\_202\_ACCEPTED, content={"task\_id": task\_id, "message": "RFP generation started."})

@app.get("/tasks/{task\_id}/status")

async def get\_task\_status(task\_id: str):

task = tasks\_db.get(task\_id)

if not task:

raise HTTPException(status\_code=404, detail="Task not found")

return task

@app.get("/download/{task\_id}")

async def download\_file(task\_id: str):

task = tasks\_db.get(task\_id)

if not task or task.status != "completed":

raise HTTPException(status\_code=404, detail="File not ready or task failed.")

filepath = task.file\_path

return FileResponse(path=filepath, filename=os.path.basename(filepath), media\_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document')

# --- Static Files Setup ---

# Create prompt files if they don't exist

if not os.path.exists(os.path.join(PROMPTS\_DIR, "extraction\_prompts.json")):

with open(os.path.join(PROMPTS\_DIR, "extraction\_prompts.json"), 'w') as f:

# This now includes all 8 prompts

json.dump({

"prompt\_1\_summary": "Extract the executive summary or the primary overview section from the following RFP text. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section if identifiable (e.g., 'Executive Summary', 'Introduction'). If not found, set to 'Overview'. - `summary\_text`: (STRING) The full text of the executive summary or a synthesized overview of the RFP's main purpose. - `project\_scope\_overview`: (STRING) A brief description of the services or project being sought. - `overarching\_goals`: (ARRAY of STRING) A list of high-level goals or desired outcomes.",

"prompt\_2\_background": "From the following RFP text, extract information pertaining to the RFP issuer's (client's) background and history. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Organization Background'. - `firm\_name`: (STRING) The full legal name of the RFP issuing organization. - `establishment\_details`: (STRING) When the organization was established. - `mission\_and\_values`: (STRING) The organization's mission, vision, or core values as stated.",

"prompt\_3\_objectives": "From the following RFP text, extract the detailed objectives and specific goals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'RFP Objectives'. - `primary\_objectives\_list`: (ARRAY of STRING) A list of the main objectives of this specific RFP. - `quantitative\_goals`: (ARRAY of STRING) Any measurable or quantitative targets. - `qualitative\_goals`: (ARRAY of STRING) Any qualitative outcomes desired.",

"prompt\_4\_scope": "Extract the detailed Scope of Work (SOW) and specific deliverables. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Scope of Work'. - `scope\_summary`: (STRING) A comprehensive summary of the entire scope of work. - `key\_tasks\_responsibilities`: (ARRAY of STRING) A list of the main tasks and responsibilities. - `expected\_deliverables`: (ARRAY of STRING) A list of concrete outputs, reports, systems, or services expected.",

"prompt\_5\_requirements": "From the following RFP text, extract all specific requirements, detailed questions, and criteria that a prospective vendor must address in their proposal. Categorize these items logically. Output should be a JSON object with the following key: - `categorized\_requirements`: (ARRAY of OBJECTS) Each object represents a category and should have 'category\_name' (STRING) and 'questions' (ARRAY of STRING).",

"prompt\_6\_evaluation": "Extract the criteria and process by which the RFP issuing organization will evaluate the submitted proposals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Proposal Evaluation'. - `key\_criteria\_list`: (ARRAY of STRING) A list of the most important factors for evaluation. - `weighting\_factors`: (STRING, nullable) Any explicit weighting or scoring breakdown mentioned. - `evaluation\_stages`: (ARRAY of STRING) A sequence of steps in the evaluation process.",

"prompt\_7\_timelines": "Extract all important dates, deadlines, and specific instructions for submitting proposals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'RFP Timeline & Submission'. - `key\_dates`: (ARRAY of OBJECTS) Each object representing a key date with 'event' (STRING) and 'date' (STRING). - `submission\_method`: (STRING) How proposals should be submitted. - `contact\_person\_for\_queries`: (STRING, nullable) Name and contact details for questions.",

"prompt\_8\_commercials": "Extract all information related to the commercial aspects, pricing models, and fee structures. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Commercials and Fees'. - `pricing\_model\_requests`: (ARRAY of STRING) The types of pricing models requested. - `specific\_fee\_details\_requested`: (ARRAY of STRING) Any specific costs or charges they want itemized."

}, f, indent=4)

if not os.path.exists(os.path.join(PROMPTS\_DIR, "generation\_prompts.json")):

with open(os.path.join(PROMPTS\_DIR, "generation\_prompts.json"), 'w') as f:

json.dump({

"Executive Summary": "Prompt A: Generate a new RFP Executive Summary based on the provided context.",

"Requirements": "Prompt D: Generate a new RFP Requirements Section based on the provided context, focusing on the common questions listed.",

"Evaluation Criteria": "Prompt E: Generate a new RFP Evaluation Criteria section based on the context."

}, f, indent=4)

# This section serves the static files for the React app.

from fastapi.staticfiles import StaticFiles

STATIC\_DIR = os.path.join(BASE\_DIR, "static")

os.makedirs(STATIC\_DIR, exist\_ok=True)

if not os.path.exists(os.path.join(STATIC\_DIR, "index.html")):

with open(os.path.join(STATIC\_DIR, "index.html"), "w") as f:

f.write("<!DOCTYPE html><html><head><title>React App</title></head><body><div id='root'></div></body></html>")

app.mount("/", StaticFiles(directory=STATIC\_DIR, html=True), name="static")

# --- backend/prompts/extraction\_prompts.json ---

# {

# "prompt\_1\_summary": "Extract the executive summary or the primary overview section from the following RFP text. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section if identifiable (e.g., 'Executive Summary', 'Introduction'). If not found, set to 'Overview'. - `summary\_text`: (STRING) The full text of the executive summary or a synthesized overview of the RFP's main purpose. - `project\_scope\_overview`: (STRING) A brief description of the services or project being sought. - `overarching\_goals`: (ARRAY of STRING) A list of high-level goals or desired outcomes.",

# "prompt\_2\_background": "From the following RFP text, extract information pertaining to the RFP issuer's (client's) background and history. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Organization Background'. - `firm\_name`: (STRING) The full legal name of the RFP issuing organization. - `establishment\_details`: (STRING) When the organization was established. - `mission\_and\_values`: (STRING) The organization's mission, vision, or core values as stated.",

# "prompt\_3\_objectives": "From the following RFP text, extract the detailed objectives and specific goals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'RFP Objectives'. - `primary\_objectives\_list`: (ARRAY of STRING) A list of the main objectives of this specific RFP. - `quantitative\_goals`: (ARRAY of STRING) Any measurable or quantitative targets. - `qualitative\_goals`: (ARRAY of STRING) Any qualitative outcomes desired.",

# "prompt\_4\_scope": "Extract the detailed Scope of Work (SOW) and specific deliverables. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Scope of Work'. - `scope\_summary`: (STRING) A comprehensive summary of the entire scope of work. - `key\_tasks\_responsibilities`: (ARRAY of STRING) A list of the main tasks and responsibilities. - `expected\_deliverables`: (ARRAY of STRING) A list of concrete outputs, reports, systems, or services expected.",

# "prompt\_5\_requirements": "From the following RFP text, extract all specific requirements, detailed questions, and criteria that a prospective vendor must address in their proposal. Categorize these items logically. Output should be a JSON object with the following key: - `categorized\_requirements`: (ARRAY of OBJECTS) Each object represents a category and should have 'category\_name' (STRING) and 'questions' (ARRAY of STRING).",

# "prompt\_6\_evaluation": "Extract the criteria and process by which the RFP issuing organization will evaluate the submitted proposals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Proposal Evaluation'. - `key\_criteria\_list`: (ARRAY of STRING) A list of the most important factors for evaluation. - `weighting\_factors`: (STRING, nullable) Any explicit weighting or scoring breakdown mentioned. - `evaluation\_stages`: (ARRAY of STRING) A sequence of steps in the evaluation process.",

# "prompt\_7\_timelines": "Extract all important dates, deadlines, and specific instructions for submitting proposals. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'RFP Timeline & Submission'. - `key\_dates`: (ARRAY of OBJECTS) Each object representing a key date with 'event' (STRING) and 'date' (STRING). - `submission\_method`: (STRING) How proposals should be submitted. - `contact\_person\_for\_queries`: (STRING, nullable) Name and contact details for questions.",

# "prompt\_8\_commercials": "Extract all information related to the commercial aspects, pricing models, and fee structures. Output should be a JSON object with the following keys: - `section\_title`: (STRING) The exact title of the section. If not found, set to 'Commercials and Fees'. - `pricing\_model\_requests`: (ARRAY of STRING) The types of pricing models requested. - `specific\_fee\_details\_requested`: (ARRAY of STRING) Any specific costs or charges they want itemized."

# }

# --- backend/prompts/generation\_prompts.json ---

# {

# "Executive Summary": "Prompt A: Generate a new RFP Executive Summary based on the provided context.",

# "Requirements": "Prompt D: Generate a new RFP Requirements Section based on the provided context, focusing on the common questions listed.",

# "Evaluation Criteria": "Prompt E: Generate a new RFP Evaluation Criteria section based on the context."

# }

```react

// frontend/src/App.js

// This file contains the React frontend application.

// It uses Tailwind CSS for styling. Make sure you have it set up in your project.

// To run: npx create-react-app frontend; cd frontend; npm install axios; npm start

import React, { useState, useCallback, useEffect } from 'react';

import axios from 'axios';

import { useDropzone } from 'react-dropzone';

// --- Configuration ---

// Make sure this URL points to your FastAPI backend.

const API\_URL = '[http://127.0.0.1:8000](http://127.0.0.1:8000)';

function App() {

const [files, setFiles] = useState([]);

const [taskId, setTaskId] = useState(null);

const [taskStatus, setTaskStatus] = useState(null);

const [error, setError] = useState(null);

const [isPolling, setIsPolling] = useState(false);

// File dropzone setup

const onDrop = useCallback(acceptedFiles => {

// Limit to 7 files

const newFiles = [...files, ...acceptedFiles].slice(0, 7);

setFiles(newFiles);

}, [files]);

const { getRootProps, getInputProps, isDragActive } = useDropzone({

onDrop,

accept: {

'application/vnd.openxmlformats-officedocument.wordprocessingml.document': ['.docx'],

},

maxFiles: 7,

});

// Function to remove a file from the list

const removeFile = (fileName) => {

setFiles(files.filter(file => file.name !== fileName));

};

// Function to start the RFP generation process

const handleGenerate = async () => {

if (files.length === 0) {

setError("Please upload at least one RFP document.");

return;

}

if (files.length < 5) {

console.warn("Warning: For best results, upload 5-7 example RFPs.");

}

const formData = new FormData();

files.forEach(file => {

formData.append('files', file);

});

setError(null);

setTaskStatus({ status: 'uploading', message: 'Uploading files...' });

setIsPolling(true);

try {

const response = await axios.post(`${API\_URL}/generate-rfp/`, formData, {

headers: {

'Content-Type': 'multipart/form-data',

},

});

setTaskId(response.data.task\_id);

} catch (err) {

console.error(err);

setError(err.response?.data?.detail || "An unexpected error occurred during upload.");

setIsPolling(false);

setTaskStatus(null);

}

};

// Polling effect to check task status

useEffect(() => {

let interval;

if (isPolling && taskId) {

interval = setInterval(async () => {

try {

const response = await axios.get(`${API\_URL}/tasks/${taskId}/status`);

const statusData = response.data;

setTaskStatus(statusData);

if (statusData.status === 'completed' || statusData.status === 'error') {

setIsPolling(false);

clearInterval(interval);

}

} catch (err) {

console.error("Error polling for status:", err);

setError("Could not retrieve task status.");

setIsPolling(false);

clearInterval(interval);

}

}, 3000); // Poll every 3 seconds

}

return () => clearInterval(interval);

}, [isPolling, taskId]);

// Function to handle downloading the result

const handleDownload = () => {

window.open(`${API\_URL}/download/${taskId}`, '\_blank');

};

// Reset Application State

const handleReset = () => {

setFiles([]);

setTaskId(null);

setTaskStatus(null);

setError(null);

setIsPolling(false);

}

return (

<div className="bg-slate-900 text-white min-h-screen flex flex-col items-center justify-center font-sans p-4">

<div className="w-full max-w-3xl mx-auto bg-slate-800 rounded-2xl shadow-2xl p-8 space-y-8">

{/\* Header \*/}

<div className="text-center">

<h1 className="text-4xl font-bold text-cyan-400">AI RFP Generator</h1>

<p className="text-slate-400 mt-2">Upload 5-7 example RFPs (.docx) to generate a new, synthesized template.</p>

</div>

{/\* Main Content Area \*/}

{!taskStatus ? (

<>

{/\* Dropzone \*/}

<div {...getRootProps()} className={`border-4 border-dashed rounded-xl p-10 text-center cursor-pointer transition-colors ${isDragActive ? 'border-cyan-400 bg-slate-700' : 'border-slate-600 hover:border-cyan-500'}`}>

<input {...getInputProps()} />

<p className="text-lg">Drag & drop .docx files here, or click to select</p>

<p className="text-sm text-slate-500 mt-1">Maximum 7 files</p>

</div>

{/\* File List \*/}

{files.length > 0 && (

<div className="space-y-3">

<h3 className="font-semibold text-lg">Selected Files:</h3>

<ul className="space-y-2">

{files.map(file => (

<li key={file.name} className="flex justify-between items-center bg-slate-700 p-3 rounded-lg">

<span>{file.name}</span>

<button onClick={() => removeFile(file.name)} className="text-red-400 hover:text-red-300 font-bold">&times;</button>

</li>

))}

</ul>

</div>

)}

{/\* Action Button \*/}

<button

onClick={handleGenerate}

disabled={files.length === 0}

className="w-full bg-cyan-600 hover:bg-cyan-500 disabled:bg-slate-700 disabled:cursor-not-allowed text-white font-bold py-3 px-4 rounded-lg text-xl transition-all"

>

Generate RFP Template

</button>

</>

) : (

/\* Status/Result Display \*/

<div className="text-center p-6 bg-slate-700 rounded-lg">

<h2 className="text-2xl font-bold mb-4 text-cyan-400">{taskStatus.status.toUpperCase()}</h2>

<p className="text-lg mb-6">{taskStatus.message}</p>

{taskStatus.status === 'processing' && (

<div className="w-16 h-16 border-4 border-dashed rounded-full animate-spin border-cyan-400 mx-auto"></div>

)}

{taskStatus.status === 'completed' && (

<button onClick={handleDownload} className="w-full bg-green-600 hover:bg-green-500 text-white font-bold py-3 px-4 rounded-lg text-xl transition-all">

Download Generated RFP

</button>

)}

{taskStatus.status === 'error' && (

<div className="bg-red-900/50 p-4 rounded-lg text-red-300">

<strong>Error:</strong> {error || taskStatus.message}

</div>

)}

{(taskStatus.status === 'completed' || taskStatus.status === 'error') && (

<button onClick={handleReset} className="mt-4 w-full bg-slate-600 hover:bg-slate-500 text-white font-bold py-2 px-4 rounded-lg transition-all">

Start Over

</button>

)}

</div>

)}

{/\* Error Display \*/}

{error && !taskStatus && (

<div className="bg-red-900/50 p-4 rounded-lg text-red-300 text-center">

<strong>Error:</strong> {error}

</div>

)}

</div>

</div>

);

}

export default App;