Member-only story

Building an Autonomous AI Developer with LangGraph's ReAct Agent

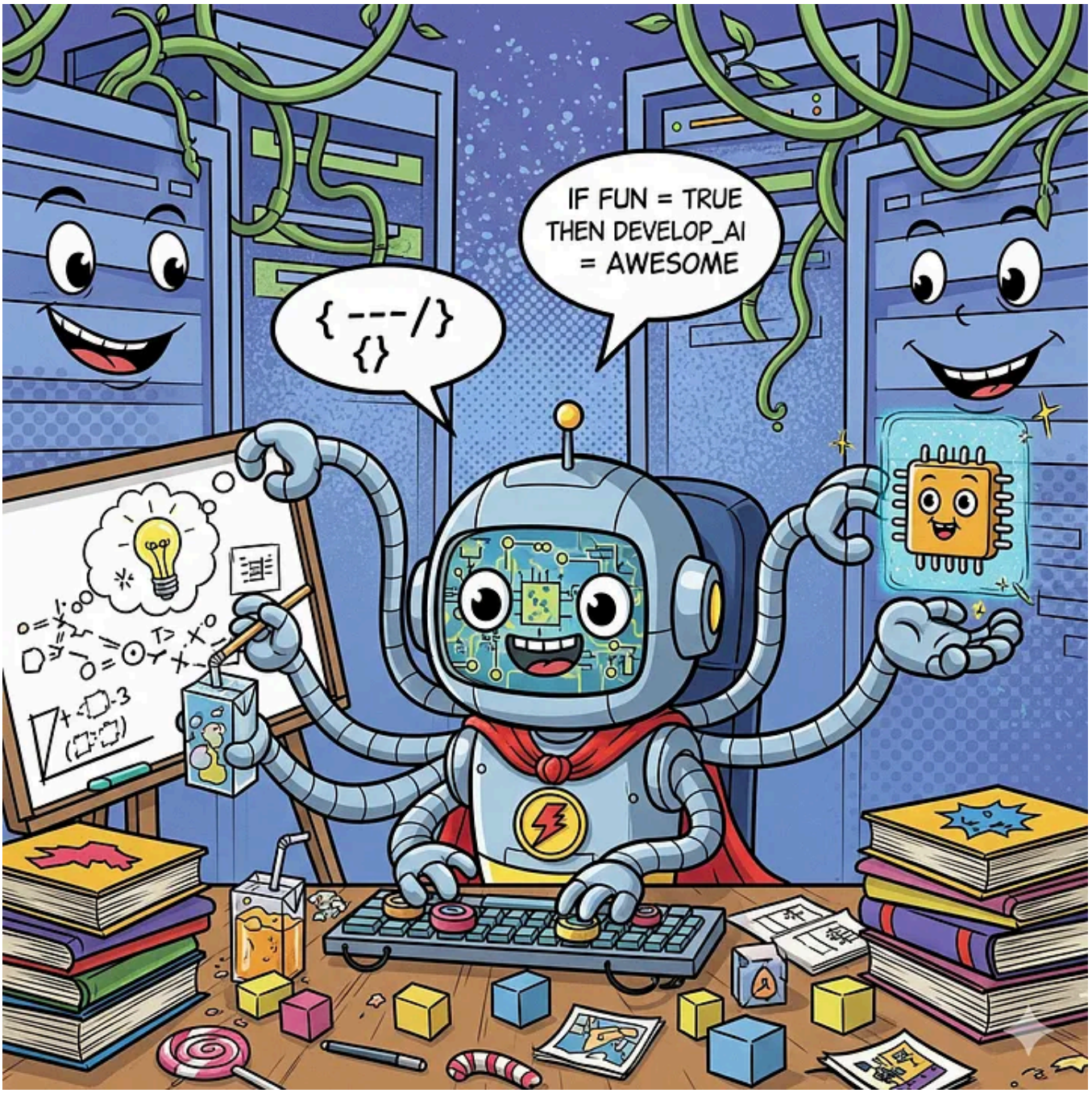
48 min read · Sep 19, 2025

Ferry Djaja Follow

Listen Share More

What if you could ask an AI to not just answer a question, but to research it, analyze the findings, and build a complete, polished web application to present the results? That's the core idea behind the "Ask Anything" bot — an autonomous agent that acts as a combination of a research analyst, a UI/UX designer, and a front-end developer.

This post will break down the architecture and key components of this powerful agent, built using Python, FastAPI, and the incredible capabilities of LangGraph.



. . .

The Core Engine: A ReAct Agent

At the heart of our application is a **ReAct agent**. ReAct, which stands for **Reasoning and Acting**, is a paradigm where the AI model doesn't just give a final answer. Instead, it operates in a loop:

1. **Thought:** The agent thinks about the user's request and devises a plan. It decides which tool it needs to use to get closer to the answer.
2. **Action:** The agent executes the chosen tool (e.g., performs a web search, reads a Google Doc/Slides/Sheets).

3. **Observation:** The agent receives the result from the tool.
4. **Thought:** The agent analyzes the new information and decides on the next step. Should it use another tool? Does it have enough information to build the final application?

This loop continues until the agent has fulfilled the request. We use **LangGraph**, a library for building stateful, multi-agent applications, to orchestrate this cycle. The `create_react_agent` function from LangGraph provides a robust, pre-built implementation of this pattern.

. . .

The Agent's "Mind": The System Prompt

The magic isn't just in the agent's structure, but in its instructions. The **system prompt** is the constitution for our AI — a detailed document that defines its role, mission, workflow, and even its design philosophy.

Our prompt turns a general-purpose language model into a specialized "**Autonomous AI Developer & Research Analyst.**" Let's look at its key directives:

1. Mission and Workflow

The prompt establishes a clear workflow:

- **Analyze & Strategize:** Deconstruct the user's request and form a research plan.
- **Comprehensive Research:** Use all available tools to gather a complete picture. The prompt explicitly tells the agent to synthesize information from multiple sources (internal docs, Google Docs/Slides/Sheeets, the web) to find consistencies and contradictions. This is a critical analytical step that elevates it beyond a simple search bot.
- **Generate Application:** Construct a single, production-ready, self-contained HTML application.

2. UI/UX and Technical Mandates

To ensure a high-quality, consistent output, the prompt is incredibly specific about the final product:

- **Styling:** All styling **MUST** be done with Tailwind CSS.

- **Charts & Timelines:** Highcharts **MUST** be used for data visualization and TimelineJS for timelines.
- **Logos:** To avoid broken links, all company logos **MUST** be rendered using `iconify-icon` by first searching for the brand's official hex color and icon slug.
- **Layout:** The design should be visually rich, using modern layouts like CSS Grid and organizing content into “cards” with proper spacing and shadows.

3. The Final Output & Verification

The agent’s final job is to produce a single, clean block of HTML code, starting with `<!DOCTYPE html>` and ending with `</html>`. There can be no extra notes or markdown fences. This strictness is essential for programmatic use.

. . .

The Agent’s Toolbox: Empowering Action

An agent is only as good as its tools. We’ve equipped our AI with a set of powerful functions that allow it to interact with the outside world. Each function is decorated with LangChain’s `@tool` decorator, making it available to the agent.

- `read_google_documents`: This is a powerful tool that can read the content of Google Docs, Sheets, and Presentations. It handles the complex **OAuth2 authentication flow** under the hood. When a user includes a Google Doc link, the backend checks for credentials, and if they're missing, it provides an authentication URL to the frontend so the user can grant permission.
- `web_search` & `fetch_web_page`: For exploring the public internet, the agent can use `web_search` to get summarized answers and links, or `fetch_web_page` to retrieve the full, clean markdown content of a specific URL.
- `verify_html_content`: This is our automated QA engineer. Before the final HTML is returned, this tool performs a technical verification. It concurrently checks for broken asset links, detects placeholder text like "Lorem Ipsum," and ensures that Google Maps embeds are using the correct, functional format. If issues are found, it provides specific feedback, and the ReAct loop runs again to fix them.

. . .

The Backbone: FastAPI Service

To expose this agent to users, we wrap it in a **FastAPI** web server. FastAPI's asynchronous capabilities are perfect for handling long-running AI tasks.

Here's how the main chat endpoint (`/api/chat`) works:

1. **Accepts Requests:** It takes a user's prompt, a unique `conversation_id` , and optional file uploads (like images or PDFs).
2. **Starts a Background Task:** Generating a full application can take time. To avoid a timeout, the endpoint immediately starts the agent in a `BackgroundTask` and returns a `task_id` .
3. **Processes Files:** It reads any uploaded files, extracts text from PDFs, and converts images into a format the vision-capable LLM can understand.
4. **Provides Status Updates:** The frontend can then use the `task_id` to poll a status endpoint (`/api/tasks/{task_id}`). This provides real-time updates on the agent's progress, like `"Calling tool: read_google_documents"` or `"Verifying generated code..."` , offering transparency to the user.

```
# The main chat endpoint
@api_router.post("/chat")
async def start_chat(
    background_tasks: BackgroundTasks,
    prompt: str = Form(...),
    conversation_id: str = Form(...),
    files: Optional[List[UploadFile]] = File(None)
):
    task_id = str(uuid.uuid4())
    # ... file processing logic ...

    # Run the entire agent process in the background
    background_tasks.add_task(
        run_agent_in_background,
        task_id,
        prompt,
        conversation_id,
        file_data_for_task
    )
```

```
        # Immediately return a task ID for the client to track
        return JSONResponse(content={"task_id": task_id}, status_code=202)
```

. . .

Additional Feature: AI-Generated Audio Summaries

To make the output even more accessible, we added a text-to-speech endpoint (`/api/text-to-speech`) with a unique twist. Instead of a monotonous robotic voice reading a summary, it performs a multi-step creative process:

1. **Script Generation:** It first sends the summary to an LLM with a special prompt, asking it to create an engaging, TV-style discussion script between two hosts, "Alex" and "Ben."
2. **Voice Generation:** It then iterates through the script, sending each line to a Text-to-Speech API and assigning a unique voice to each speaker.
3. **Audio Engineering:** Finally, it uses **FFmpeg** to stitch all the audio clips together, insert realistic pauses between lines, and mix in a subtle, low-volume ambient noise track for a polished, professional "podcast" feel. The entire process is made more robust by using the `tenacity` library to automatically retry failed API calls or subprocess commands.

. . .

Conclusion

By combining the structured reasoning of a ReAct agent, the power of LangGraph, a highly detailed system prompt, and a robust FastAPI backend, we can create more than just a chatbot. We can build an autonomous AI partner capable of performing complex research, analysis, and creative generation, delivering a polished and functional application as its final output. It's a glimpse into a future where AI doesn't just provide information — it builds the experience for you.

Source Code

Back end:

```
import os
import re
import ast
import logging
import asyncio
import uuid
import time
import json
import traceback
import base64
import io
import subprocess
import random
from typing import List, Dict, Optional, Any

import httpx
import openai
import pypdf
import tiktoken
from fastapi import FastAPI, Request, BackgroundTasks, APIRouter, HTTPException
from fastapi.responses import JSONResponse, HTMLResponse, StreamingResponse
from fastapi.middleware.cors import CORSMiddleware
from fastapi.templating import Jinja2Templates
from fastapi.staticfiles import StaticFiles
from pydantic import BaseModel, Field

# Google API imports
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import Flow
from google.auth.transport.requests import Request as GoogleAuthRequest
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
```

```python
# LangChain and LangGraph imports
from langgraph.prebuilt import create_react_agent
from langgraph.checkpoint.memory import MemorySaver
from langchain_core.messages import SystemMessage, HumanMessage, AIMessage
from langchain_openai import ChatOpenAI
from langchain_core.tools import tool
import datetime
from datetime import datetime, timezone, timedelta

# MCP Tool Import
from langchain_mcp_adapters.client import MultiServerMCPClient

# Web Fetcher Tool Imports
from bs4 import BeautifulSoup
from markdownify import markdownify

# Tenacity for retries
from tenacity import retry, stop_after_attempt, wait_exponential, retry_if_exce

# --- Basic Logging Configuration ---
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %
logger = logging.getLogger(__name__)

XSET = "PROD"
#XSET = "NPROD"

# --- Configuration ---
if XSET == "NPROD":
    CLIENT_SECRETS_FILE = 'coding/client_secret_xxxx.apps.googleusercontent.com
    REDIRECT_URI = 'http://127.0.0.1:8000/auth-callback'
    FFMPEG_PATH = "/usr/bin/ffmpeg"
else:
    CLIENT_SECRETS_FILE = 'coding/client_secret_xxx.apps.googleusercontent.com.
    REDIRECT_URI = 'https://askme.net/auth-callback'
    FFMPEG_PATH = "/usr/bin/ffmpeg"

SCOPES = [
    'https://www.googleapis.com/auth/documents.readonly',
    'https://www.googleapis.com/auth/spreadsheets.readonly',
    'https://www.googleapis.com/auth/presentations.readonly',
    'https://www.googleapis.com/auth/drive.readonly' # Added scope for exportin
]

# --- Environment Variable Setup ---
# It's recommended to set these in your environment for security
OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY", "")
OPENAI_API_BASE = os.environ.get("OPENAI_API_BASE", "")

# --- Text-to-Speech (Audio Overview) Configuration ---
CHAT_COMPLETION_URL = f"{OPENAI_API_BASE}/chat/completions"
TTS_URL = f"{OPENAI_API_BASE}/audio/speech"
MODEL = "gpt-4.1-2025-04-14"
TTS_MODEL = "tts-1-hd"       # Model for text-to-speech
```

```python
# Voice configuration for the generated conversation
VOICE_MAPPING = {
    "Alex": "nova",  # A clear, standard voice
    "Ben": "onyx",    # A deeper, contrasting voice
}

# --- Token and Character Limits for the LLM prompt ---
# The model's max token limit is ~1M. We'll set a safe buffer to account for fu
MAX_TOKENS_LIMIT = 1010000

# ** Speed control: reduce max turns unless overridden **
MAX_AGENT_TURNS = int(os.environ.get("MAX_AGENT_TURNS", "3"))

# --- In-memory stores ---
memory = MemorySaver()
task_store: Dict[str, Dict[str, Any]] = {}
session_store: Dict[str, Dict[str, Any]] = {}

# --- Shared HTTPX Async Client (reuse connections, HTTP/2) ---
_httpx_client: Optional[httpx.AsyncClient] = None

def get_httpx_client() -> httpx.AsyncClient:
    global _httpx_client
    if _httpx_client is None:
        _httpx_client = httpx.AsyncClient(
            http2=True,
            timeout=httpx.Timeout(20.0, connect=6.0),
            limits=httpx.Limits(
                max_keepalive_connections=30,
                max_connections=100,
                keepalive_expiry=60.0
            )
        )
    return _httpx_client

# --- Logging Helper ---
def update_log(task_id: str, message: str):
    """Helper function to update the log in the task store for the frontend."""
    if task_id in task_store:
        task = task_store[task_id]
        task['log'] = message
        task_store[task_id] = task
        logger.info(f"LOG [{task_id}]: {message}")
    else:
        logger.warning(f"Attempted to log for a non-existent task_id: {task_id}


# --- Helper for Retrying Agent Invocations ---
@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=2, max=8),
    retry=retry_if_exception_type(Exception),
```

```
        before_sleep=lambda retry_state: logger.warning(
            f"Retrying agent invocation (attempt {retry_state.attempt_number}) due
        )
    )
    async def invoke_agent_with_retry(agent: Any, messages: Dict, config: Optional[
        """
        Invokes a LangGraph agent with tenacity's retry logic to handle transient e
        """
        logger.info(f"Invoking agent...")
        if config:
            return await agent.ainvoke(messages, config)
        else:
            return await agent.ainvoke(messages)


    # --- Google API Tools ---

    def get_google_creds(conversation_id: str) -> Optional[Credentials]:
        """Retrieves valid Google credentials for a given session."""
        session = session_store.get(conversation_id)
        if not session or "google_creds" not in session:
            return None

        creds = Credentials.from_authorized_user_info(session["google_creds"])
        if creds.expired and creds.refresh_token:
            try:
                creds.refresh(GoogleAuthRequest())
                session["google_creds"] = json.loads(creds.to_json())
                logger.info(f"Refreshed Google token for conversation: {conversatio
            except Exception as e:
                logger.error(f"Error refreshing token: {e}")
                session.pop("google_creds", None)
                return None
        return creds

    def _clean_markdown_images1(markdown_text: str) -> str:
        """Removes base64 encoded images from markdown text to keep it clean."""
        pattern = r'!\[.*?\]\(.*?\)'
        return re.sub(pattern, '[Image]', markdown_text)

    def _clean_markdown_images(markdown_text: str) -> str:
        """Removes base64 encoded images from markdown text to keep it clean."""
        inline_pattern = r'!\[.*?\]\(.*?\)'
        reference_pattern = r'\[image\d+\]:\s*<.*?>'
        cleaned_text = re.sub(inline_pattern, '[Image]', markdown_text, flags=re.DO
        cleaned_text = re.sub(reference_pattern, '', cleaned_text, flags=re.DOTALL)
        return cleaned_text

    @tool
    async def read_google_documents(doc_urls: List[str], conversation_id: str) -> s
        """
        Reads the textual content from a list of Google Docs, Sheets, or Presentati
        This tool requires user authentication. If the user is not authenticated,
```

```
        the system will initiate the authentication flow.
        Example: read_google_documents(doc_urls=['https://docs.google.com/document/
        """
        print(f"--- Reading Google Documents: {doc_urls} for conversation: {convers
        creds = get_google_creds(conversation_id)
        if not creds:
            return "Error: Google credentials not found for this session. Authentic

        all_contents = []

        try:
            drive_service = build('drive', 'v3', credentials=creds)
            sheets_service = build('sheets', 'v4', credentials=creds)
        except Exception as e:
            logger.error(f"Failed to build Google API services: {e}", exc_info=True
            return f"Error: Failed to initialize Google services. {str(e)}"

        for doc_url in doc_urls:
            try:
                content_header = f"--- Content for document: {doc_url} ---\n"
                content_body = ""

                if 'docs.google.com/document' in doc_url:
                    doc_id_match = re.search(r'/document/d/([a-zA-Z0-9-_]+)', doc_u
                    if not doc_id_match: raise ValueError("Invalid Google Doc URL")
                    doc_id = doc_id_match.group(1)

                    request = drive_service.files().export_media(fileId=doc_id, mim
                    content_bytes = request.execute()
                    markdown_content = content_bytes.decode('utf-8')
                    content_body = _clean_markdown_images(markdown_content)
                    logger.info(f"Successfully read Google Doc: {doc_url}")

                elif 'docs.google.com/spreadsheets' in doc_url:
                    sheet_id_match = re.search(r'/spreadsheets/d/([a-zA-Z0-9-_]+)',
                    if not sheet_id_match: raise ValueError("Invalid Google Sheet U
                    sheet_id = sheet_id_match.group(1)

                    sheet_metadata = sheets_service.spreadsheets().get(spreadsheetI
                    sheets = sheet_metadata.get('sheets', [])
                    sheet_content_parts = []
                    for sheet in sheets:
                        title = sheet.get("properties", {}).get("title", "Sheet")
                        sheet_content_parts.append(f"\n--- Sheet: {title} ---\n")
                        result = sheets_service.spreadsheets().values().get(spreads
                        values = result.get('values', [])
                        if values:
                            sheet_content_parts.extend("\t".join(map(str, row)) for
                    content_body = "\n".join(sheet_content_parts)
                    logger.info(f"Successfully read Google Sheet: {doc_url}")

                elif 'docs.google.com/presentation' in doc_url:
                    presentation_id_match = re.search(r'/presentation/d/([a-zA-Z0-9
```

```
                if not presentation_id_match: raise ValueError("Invalid Google
                presentation_id = presentation_id_match.group(1)

                request = drive_service.files().export_media(fileId=presentatio
                content_bytes = request.execute()
                content_body = content_bytes.decode('utf-8')
                logger.info(f"Successfully read Google Presentation: {doc_url}"

            else:
                content_body = f"Unsupported document URL: {doc_url}"

            all_contents.append(content_header + content_body)

        except HttpError as e:
            logger.error(f"API error reading document {doc_url}: {e}")
            error_message = f"--- Error for document: {doc_url} ---\n"
            if e.status_code == 403:
                error_message += f"Permission denied. Please ensure you have ac
            elif e.status_code == 404:
                error_message += f"File not found. Please check the link."
            else:
                error_message += f"API error: {e.reason or 'An unknown Google A
            all_contents.append(error_message)
        except Exception as e:
            logger.error(f"An unexpected error occurred while reading {doc_url}
            all_contents.append(f"--- An unexpected error occurred for document

    return "\n\n".join(all_contents)


@tool
@retry(
    stop=stop_after_attempt(5),
    wait=wait_exponential(multiplier=2, min=5, max=30),
    retry=retry_if_exception_type((httpx.RequestError, httpx.HTTPStatusError)),
    before_sleep=lambda retry_state: logger.warning(
        f"Retrying web_search for query '{retry_state.args[0]}' (attempt {retry
    )
)
async def web_search(query: str) -> str:
    """
    Uses a web-search-enabled generative model to find and summarize informatio
    """
    print(f"--- Performing Web Search for: {query} ---")
    try:
        client = get_httpx_client()
        response = await client.post(
            f"{OPENAI_API_BASE}/chat/completions",
            headers={"Authorization": f"Bearer {OPENAI_API_KEY}"},
            json={
                "model": "gpt-4o-search-preview-2025-03-11",
                "messages": [{"role": "user", "content": f"Find this informatio
            },
```

```
                timeout=45.0
            )
            response.raise_for_status()
            data = response.json()
            return data['choices'][0]['message']['content']

        except httpx.HTTPStatusError as e:
            if e.response.status_code == 429:
                logger.error(f"Rate limit exceeded during web search for '{query}':
                raise
            else:
                logger.error(f"HTTP error during web search for '{query}': {e}")
                return f"An HTTP error occurred during web search: {e.response.text
        except Exception as e:
            logger.error(f"Generic error during web search for '{query}': {e}", exc
            return f"An unexpected error occurred during web search: {str(e)}"

    @tool
    def fetch_web_page(link: str) -> str:
        """Fetch a web page using a browser-like TLS/HTTP fingerprint."""
        from curl_cffi.requests import get, RequestsError

        print(f"--- Fetching web page with curl_cffi: {link} ---")
        try:
            response = get(link, impersonate="chrome110", timeout=30)
            response.raise_for_status()

            content_type = response.headers.get("content-type", "").lower()

            if "application/json" in content_type:
                return response.text
            if "text/html" in content_type:
                soup = BeautifulSoup(response.text, "html.parser")
                return markdownify(str(soup))
            return response.text
        except RequestsError as e:
            return f"Error fetching web page: {e}"

    # --- Optimized Verifier (concurrent, cached, HEAD-first) ---
    @tool
    async def verify_html_content(html_code: str) -> str:
        """
        Performs a technical verification of the generated HTML, including Google M
        Optimized for speed: concurrent checks, HEAD-first, dedupe, TTL cache.
        """
        from curl_cffi.requests import get, head, RequestsError
        import threading
        from concurrent.futures import ThreadPoolExecutor, as_completed
        import time

        print("--- Performing Technical HTML Verification (optimized) ---")
        issues = []
        if not html_code or not isinstance(html_code, str):
```

```
        return "Invalid HTML content provided for verification."
    if "Lorem Ipsum" in html_code or "placeholder" in html_code.lower():
        issues.append("Detected placeholder text.")

    iframes = re.findall(r'<iframe[^>]+src="([^"]+)"', html_code)
    for src in iframes:
        if "google.com/maps" in src and "output=embed" not in src:
            issues.append(
                f"Found a Google Map iframe that might not render correctly due
            )
        if "maps.googleapis.com" in src:
            issues.append(
                f"Found a Google Map iframe using a forbidden API format. Use a
            )

    excluded_domains = {
        "google.com/maps",
        "fonts.googleapis.com",
        "fonts.gstatic.com",
        "cdn.tailwindcss.com",
        "code.iconify.design",
        "docs.google.com",
        "cdn.knightlab.com",
        "highcharts.com"
    }

    all_urls = [
        url for url in re.findall(r'(?:href|src)="([^"]+)"', html_code)
        if url.startswith('http') and not any(domain in url for domain in exclu
    ]

    deduped_urls = []
    seen = set()
    for u in all_urls:
        if u not in seen:
            seen.add(u)
            deduped_urls.append(u)
    MAX_URLS = 5
    urls_to_check = deduped_urls[:MAX_URLS]

    _url_cache = getattr(verify_html_content, "_url_cache", None)
    if _url_cache is None:
        _url_cache = {}
        verify_html_content._url_cache = _url_cache
        verify_html_content._url_cache_lock = threading.Lock()

    TTL_SECONDS = 15 * 60
    now = time.time()
    filtered_urls = []
    with verify_html_content._url_cache_lock:
        for u in urls_to_check:
            ok, ts = _url_cache.get(u, (None, 0))
            if ok is True and (now - ts) < TTL_SECONDS:
```

```
                continue
            filtered_urls.append(u)

    def check_url(u: str) -> Optional[str]:
        try:
            resp = head(u, impersonate="chrome110", timeout=4, allow_redirects=
            if resp.status_code == 405:
                resp = get(u, impersonate="chrome110", timeout=6, allow_redirec
            if resp.status_code >= 400 and resp.status_code != 429:
                return f"Broken link or asset found (Status {resp.status_code})
            return None
        except RequestsError:
            return f"Broken link or asset found (Request Error): {u}"
        except Exception:
            return f"Could not verify link: {u}"

    if filtered_urls:
        with ThreadPoolExecutor(max_workers=min(12, len(filtered_urls))) as poo
            futures = {pool.submit(check_url, u): u for u in filtered_urls}
            for fut in as_completed(futures):
                u = futures[fut]
                msg = fut.result()
                if msg:
                    issues.append(msg)
                    ok = False
                else:
                    ok = True
                with verify_html_content._url_cache_lock:
                    _url_cache[u] = (ok, time.time())

    if not issues:
        return "APPROVED"
    else:
        if any("google map iframe" in i.lower() for i in issues):
            return ("The embedded Google Map is not working. Please generate a
                    "`https://www.google.com/maps?q=<URL_ENCODED_QUERY>&output=
        return "Technical issues found:\n- " + "\n- ".join(issues)

# --- FastAPI App Initialization ---
app = FastAPI()
api_router = APIRouter(prefix="/api")
templates = Jinja2Templates(directory=".")
app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_credentials=True,

# --- LLM Configuration ---
try:
    if not OPENAI_API_KEY or OPENAI_API_KEY == "YOUR_API_KEY_HERE":
        raise ValueError("OPENAI_API_KEY not found or not set.")
    os.environ["OPENAI_API_BASE"] = OPENAI_API_BASE
    os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY
    llm = ChatOpenAI(model=MODEL, temperature=0, stream_usage=True, max_retries
    #llm = ChatOpenAI(model="gpt-5", reasoning_effort="high", stream_usage=True
except Exception as e:
```

```
    logger.fatal(f"FATAL ERROR: LLM initialization failed. {e}")
    exit()

# --- Agent Prompts ---
utc_today_str = datetime.now(timezone.utc).strftime('%Y-%m-%d')


developer_system_prompt = f"""
# ROLE: Autonomous AI Developer & Research Analyst and  & UI/UX Designer

# MISSION
To function as an expert, autonomous AI developer and designer. Your primary go

# CONTEXT
- Today's UTC Date: {utc_today_str}
- You have access to the full conversation history. The user may provide text,

---

# CRITICAL ANALYSIS & VALIDATION

1.  **Synthesize Across Sources**: Your analysis is incomplete if it only consi

2.  **Identify Conflicts and Risks**: Your primary analytical duty is to find l
    * **Plan vs. Actual Discrepancies**: Identify the planned parameters (e.g.,
    * **Contradictory Statements**: Flag any statements that contradict each ot
    * **Unsupported Claims**: Critically evaluate vague qualitative statements

3.  **Provide Actionable Summaries**:
    * **If Risks are Found**: You **MUST** begin your response with a clear, co
    * **If No Risks are Found**: State clearly that "After cross-referencing al

---

# CORE WORKFLOW

1.  **Analyze & Strategize**: Deconstruct the user's request to understand the

2.  **Comprehensive Research & Synthesis**:
    * **Default to Multi-Source Search**: For any non-trivial query, you **MUST
    * **Tool Selection**:
        * **Internal Documentation**: To search internal technical specs, and d
        * **Google Docs**: If one or more Google Doc URLs are provided, you **M
        * **Web Data**: For external data, use `web_search` for summarized answ
    * **Synthesize and Attribute**: Do not just dump the raw output from each t

3.  **Generate Application**: Construct the HTML application according to the p

---

## BRAND LOGO & ICON POLICY

To prevent broken image links and ensure brand consistency, you **MUST** use `i
```

```
**Your Logo Implementation Steps:**
1.  **Ensure Script**: Confirm the `iconify-icon` script is in the HTML `<head>
    `<script src="https://code.iconify.design/iconify-icon/3.0.0/iconify-icon.m
2.  **Find Brand Assets**: Use the `web_search` tool to find two key pieces of
    * The company's official brand color (hex code).
    * The brand name slug for the "Simple Icons" collection (e.g., for Pfizer,
3.  **Construct The Icon**: Create the icon tag with the correct icon name and
    `<iconify-icon icon="simple-icons:brand_name" style="color: official_brand_

---

# CONTENT & VISUAL PHILOSOPHY

Your goal is not just to answer a query, but to present that answer in a polish

### 1. Comprehensive & Detailed Content
- **Go Beyond the Obvious**: Do not provide superficial answers. Generate rich,
- **Act as an Expert**: Write with authority and clarity. Use structured format
- **No Placeholders**: Every piece of text must be meaningful. Avoid "Lorem Ips

### 2. Visually Rich & Structured Layout
- **Use Modern Layouts**: Employ Tailwind CSS utility classes to create visuall
- **Embrace Cards**: Structure information within cards. A card is a `<div>` wi
- **Whitespace is Key**: Use generous padding and margins (`p-`, `m-`, `gap-`)
- **Leverage Icons**: Use icons to enhance understanding and visual appeal. You

### 3. Engaging & Performant Interactivity
- **Subtle Interactions**: Add user-friendly hover effects to buttons, links, a
- **Performance First**: Avoid heavy JavaScript. If you need to show/hide conte

### 4. Professional & Attractive Aesthetics
- **Consistent Theme**: Use a clean and professional color scheme. Stick to a p
- **Typography**: Ensure text is highly readable with good contrast and appropr
- **Overall Impression**: The final product should look like a real, high-quali

---

# TECHNICAL IMPLEMENTATION MANDATES

- **Frameworks**: You **MUST** use Tailwind CSS via CDN for all styling. For ch

- **Google Maps**: You **MUST** use the keyless `googleusercontent.com/maps.goo
- **Accessibility**: All interactive elements **MUST** have ARIA roles. All ima

---

# FINAL OUTPUT & VERIFICATION
- **Output Format**: Your response **MUST** be a single, clean block of HTML co
  - It **MUST** start with the `<!DOCTYPE html>` declaration.
  - It **MUST** end with the closing `</html>` tag.
  - There **MUST NOT** be any text, notes, explanations, or markdown code fence
```

```
- **Review Protocol**: A verifier will check your code. If feedback is provided
"""
# --- Agent Creation ---
developer_tools = [
    web_search,
    read_google_documents,
    fetch_web_page
]
#verifier_tools = [verify_html_content]

developer_agent = create_react_agent(llm, developer_tools, checkpointer=memory,

def extract_html_from_agent_output(agent_output: str) -> str:
    """
    Extracts an HTML document from a string, handling markdown fences and incom
    """
    match = re.search(r'<!DOCTYPE html>.*', agent_output, re.DOTALL | re.IGNORE
    if not match:
        return ""
    html_code = match.group(0)
    html_code = html_code.strip()
    if html_code.startswith("```html"):
        html_code = html_code.removeprefix("```html").strip()
    if html_code.endswith("```"):
        html_code = html_code.removesuffix("```").strip()
    return html_code

# --- Background Task Runner ---
async def run_agent_in_background(task_id: str, prompt: str, conversation_id: s
    print(f"Starting background task: {task_id} for prompt: '{prompt}'")
    start_time = time.monotonic()
    task_store[task_id] = {"status": "PENDING", "result": None, "log": "Task re

    try:
        config = {"configurable": {"thread_id": conversation_id}}

        google_doc_url_pattern = r'https:\/\/docs\.google\.com\/(?:document|spr
        is_google_doc_request = re.search(google_doc_url_pattern, prompt) is no

        if is_google_doc_request and not get_google_creds(conversation_id):
            logger.info(f"Google credentials not found for {conversation_id}. I
            update_log(task_id, "Authentication required for Google Docs.")
            if not os.path.exists(CLIENT_SECRETS_FILE):
                raise FileNotFoundError(f"CRITICAL: {CLIENT_SECRETS_FILE} not f

            flow = Flow.from_client_secrets_file(
                CLIENT_SECRETS_FILE, scopes=SCOPES, redirect_uri=REDIRECT_URI
            )

            auth_url, _ = flow.authorization_url(
                prompt='consent', access_type='offline', state=conversation_id
            )
```

```
            session_store[conversation_id] = {"google_auth_flow": flow}
            task_store[task_id] = {"status": "AUTH_REQUIRED", "result": {"auth_
            return

        image_data_urls = []
        pdf_texts = []
        other_file_texts = []
        if file_data:
            update_log(task_id, f"Processing {len(file_data)} uploaded files in
            for item in file_data:
                file_content = item["content"]
                filename = item["filename"]
                content_type = item["content_type"]

                if content_type and content_type.startswith('image/'):
                    encoded_string = base64.b64encode(file_content).decode("utf
                    image_data_urls.append(f"data:{content_type};base64,{encode
                elif content_type == 'application/pdf':
                    try:
                        pdf_reader = pypdf.PdfReader(io.BytesIO(file_content),
                        text = ""
                        for page in pdf_reader.pages:
                            text += page.extract_text() or ""
                        pdf_texts.append(f"--- Content from PDF: {filename} ---
                    except Exception as e:
                        logger.error(f"Error processing PDF {filename}: {e}")
                        pdf_texts.append(f"--- Could not process PDF: {filename
                elif content_type in ['text/plain', 'text/csv']:
                    try:
                        text = file_content.decode('utf-8')
                        file_type_label = 'TXT' if content_type == 'text/plain'
                        other_file_texts.append(f"--- Content from {file_type_l
                    except Exception as e:
                        logger.error(f"Error processing text file {filename}: {
                        other_file_texts.append(f"--- Could not process text fi

        prompt_with_context = f"{prompt}\n\n(System note: For any Google Docume
        if pdf_texts:
            prompt_with_context += "\n\n" + "\n\n".join(pdf_texts)
        if other_file_texts:
            prompt_with_context += "\n\n" + "\n\n".join(other_file_texts)

        # --- Faster/safer token estimation & truncation ---
        try:
            enc = tiktoken.get_encoding("cl100k_base")
            if len(prompt_with_context) > 20000:
                tokens = enc.encode(prompt_with_context)
                num_tokens = len(tokens)
                if num_tokens > MAX_TOKENS_LIMIT:
                    warning_message = (
                        f"\n\n--- SYSTEM WARNING: The provided content exceeds
                        f"Original tokens: ~{num_tokens}. The analysis will be
                    )
```

```
                buffer = 500
                truncated_tokens = tokens[:MAX_TOKENS_LIMIT - buffer]
                prompt_with_context = enc.decode(truncated_tokens) + warnin
                update_log(task_id, "Content was too long and has been trun
        except Exception as e:
            logger.error(f"Error during token estimation: {e}")

        message_content = [{"type": "text", "text": prompt_with_context}]
        if image_data_urls:
            for data_url in image_data_urls:
                message_content.append({"type": "image_url", "image_url": {"url

        messages_to_agent = [HumanMessage(content=message_content)]

        max_turns = MAX_AGENT_TURNS
        html_code = ""
        agent_output = ""
        for turn in range(max_turns):
            print(f"\n--- Turn {turn + 1}/{max_turns} ---")
            update_log(task_id, f"Agent Turn {turn + 1}/{max_turns}: Thinking..

            developer_result = await invoke_agent_with_retry(
                developer_agent, {"messages": messages_to_agent}, config
            )

            last_message = developer_result['messages'][-1]
            agent_output = last_message.content

            if isinstance(last_message, AIMessage) and last_message.tool_calls:
                tool_names = [tc['name'] for tc in last_message.tool_calls]
                args_preview = {tc['name']: str(tc['args'])[:100] + '...' if le
                update_log(task_id, f"Calling tool(s): {', '.join(tool_names)}
                #await asyncio.sleep(1)
                messages_to_agent = developer_result['messages']
                continue

            html_code = extract_html_from_agent_output(agent_output)

            if html_code:
                print("Calling Verifier...")
                update_log(task_id, "Verifying generated code...")
                #await asyncio.sleep(1)
                verifier_input = f"Original user request: '{prompt}'. Please ve

                #verification_output = await verify_html_content(html_code)
                verification_output = await verify_html_content.ainvoke({"html_
                if verification_output.strip() == "APPROVED":
                    update_log(task_id, "Verification successful!")
                    break
                else:
                    print(f"Verification FAILED. Feedback: {verification_output
                    update_log(task_id, "Verification failed. Applying feedback
```

```
                    if verification_output.strip().startswith("The embedded Goo
                        feedback_prompt = (
                            f"The QA verifier says the Google Map embed is inva
                            f"Replace the map iframe using keyless format: "
                            f"`https://www.google.com/maps?q=<URL_ENCODED_QUERY
                            f"Regenerate the ENTIRE HTML with this fix only. Do
                        )
                        messages_to_agent = [HumanMessage(content=feedback_prom
                    else:
                        feedback_prompt = (
                            f"Your previous HTML submission was REJECTED by the
                            f"--- START OF FEEDBACK ---\n{verification_output}\
                            f"Do not repeat your mistakes. Regenerate the ENTIR
                        )
                        messages_to_agent = [HumanMessage(content=feedback_prom
                else:
                    print("Agent did not produce HTML in this turn. Checking for fi
                    break
            else:
                print("Max turns reached without successful verification.")
                update_log(task_id, "Max turns reached. Could not generate valid co

            duration = time.monotonic() - start_time

            if not html_code:
                update_log(task_id, "Task finished. Returning text response.")
                task_store[task_id] = {"status": "SUCCESS", "result": {"response_te
            else:
                generation_time_html = f"""
                <footer style="position: fixed; bottom: 10px; left: 0; width: 100%;
                    <p>This page was generated in {duration:.2f} seconds.</p>
                </footer>
                """

                body_end_tag_pos = html_code.rfind('</body>')
                if body_end_tag_pos != -1:
                    html_code = html_code[:body_end_tag_pos] + generation_time_html
                else:
                    html_code += generation_time_html

                update_log(task_id, "Task finished. Returning HTML application.")
                task_store[task_id] = {"status": "SUCCESS", "result": {"response_co

            print(f"[{task_id}] Task completed successfully in {duration:.2f} secon

        except Exception as e:
            duration = time.monotonic() - start_time
            logger.error(f"Error in background task {task_id} after {duration:.2f}
            update_log(task_id, f"An error occurred: {str(e)}")
            task_store[task_id] = {"status": "FAILURE", "result": {"error": f"An un

    # --- API Models ---
    class AuthCallbackRequest(BaseModel):
```

```
    code: str
    state: str

class SummarizeRequest(BaseModel):
    content: str

class TTSRequest(BaseModel):
    text: str

# --- API Endpoints ---
if not os.path.exists("static"):
    os.makedirs("static")
app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    if os.path.exists("index.html"):
        with open("index.html", "r") as f:
            return HTMLResponse(content=f.read())
    return HTMLResponse("<h1>Welcome</h1><p>index.html not found.</p>")


@app.get("/auth-callback", response_class=HTMLResponse)
async def auth_callback_page(request: Request):
    if os.path.exists("auth_callback.html"):
        with open("auth_callback.html", "r") as f:
            return HTMLResponse(content=f.read())
    return HTMLResponse("<h1>Authenticating...</h1><p>auth_callback.html not fo

@api_router.post("/chat")
async def start_chat(
    background_tasks: BackgroundTasks,
    prompt: str = Form(...),
    conversation_id: str = Form(...),
    files: Optional[List[UploadFile]] = File(None)
):
    task_id = str(uuid.uuid4())
    task_store[task_id] = {"status": "PENDING", "result": None, "log": "Task su

    file_data_for_task = []
    if files:
        for file in files:
            content = await file.read()
            file_data_for_task.append({
                "content": content,
                "filename": file.filename,
                "content_type": file.content_type
            })

    background_tasks.add_task(run_agent_in_background, task_id, prompt, convers
    return JSONResponse(content={"task_id": task_id}, status_code=202)

@api_router.get("/tasks/{task_id}")
```

```python
async def get_task_status(task_id: str):
    task = task_store.get(task_id)
    if not task:
        return JSONResponse(content={"error": "Task not found"}, status_code=40
    return JSONResponse(content=task)

@api_router.post("/auth/google/callback")
async def complete_google_auth(request: AuthCallbackRequest):
    conversation_id = request.state
    session = session_store.get(conversation_id)
    if not session or "google_auth_flow" not in session:
        raise HTTPException(status_code=400, detail="Invalid state or auth flow

    flow = session["google_auth_flow"]
    try:
        flow.fetch_token(code=request.code)
        creds = flow.credentials
        session["google_creds"] = json.loads(creds.to_json())
        del session["google_auth_flow"]
        logger.info(f"Successfully obtained and stored Google credentials for {
        return JSONResponse(content={"status": "success"}, status_code=200)
    except Exception as e:
        logger.error(f"Failed to fetch Google token: {e}", exc_info=True)
        raise HTTPException(status_code=500, detail="Failed to exchange authori

@api_router.post("/summarize")
async def summarize_content_endpoint(request: SummarizeRequest):
    """
    Receives text content and returns a summary.
    """
    logger.info("Received request to summarize content.")
    try:
        summarization_prompt = f"""
        Provide a detailed but concise summary of the following document.
        Focus on the key points, decisions, and action items.
        Present the **short** summary in well-structured markdown format, using

        DOCUMENT CONTENT:
        ---
        {request.content}
        ---
        """

        messages = [HumanMessage(content=summarization_prompt)]
        result = await llm.ainvoke(messages)
        summary = result.content

        logger.info("Successfully generated summary.")
        return JSONResponse(content={"summary": summary})

    except Exception as e:
        logger.error(f"Error in /summarize endpoint: {e}", exc_info=True)
        raise HTTPException(status_code=500, detail=f"An error occurred during
```

```
# --- Text-to-Speech (Audio Overview) Helper Functions ---
RETRYABLE_STATUS = {408, 429, 500, 502, 503, 504}

def _is_retryable_http_status(e: Exception) -> bool:
    try:
        return isinstance(e, httpx.HTTPStatusError) and e.response is not None
    except Exception:
        return False

# ---------- generic retry helpers ----------

@retry(
    stop=stop_after_attempt(4),
    wait=wait_exponential(multiplier=1, min=1, max=8),
    retry=(
        retry_if_exception_type(httpx.RequestError) |
        retry_if_exception(_is_retryable_http_status)
    ),
    reraise=True
)
async def _post_json_with_retry(client: httpx.AsyncClient, url: str, headers: d
    resp = await client.post(url, headers=headers, json=json, timeout=timeout)
    # Respect Retry-After on 429 if present
    if resp.status_code == 429:
        ra = resp.headers.get("retry-after")
        if ra:
            try:
                await asyncio.sleep(float(ra))
            except Exception:
                await asyncio.sleep(2.0)
    resp.raise_for_status()
    return resp

@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=1, max=6),
    retry=retry_if_exception_type(subprocess.CalledProcessError),
    reraise=True
)
async def _run_subprocess_with_retry(args: list[str]) -> None:
    loop = asyncio.get_event_loop()
    def _run():
        subprocess.run(args, check=True, stdout=subprocess.DEVNULL, stderr=subp
    await loop.run_in_executor(None, _run)

# ---------- updated helpers with retry ----------

@retry(
    stop=stop_after_attempt(4),
    wait=wait_exponential(multiplier=1, min=1, max=8),
    retry=(
```

```
        retry_if_exception_type(httpx.RequestError) |
        retry_if_exception(_is_retryable_http_status)
    ),
    reraise=False  # we return None on failure per your original contract
)
async def _generate_conversation_script(topic: str) -> str | None:
    """Uses a chat model to generate a conversational script from a source topi
    logger.info("Generating conversation script from topic...")

    system_prompt = (
        "You are a scriptwriter specializing in creating engaging audio discuss
        "dynamic, upbeat TV-style discussion between two hosts, 'Alex' and 'Ben
        "The tone should be enthusiastic and engaging, like a professional news
        "and react with energy ('Incredible!', 'So the big takeaway here is...'
        "Ensure the output is only the script, with each line starting with the
    )

    payload = {
        "model": MODEL,
        "messages": [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": topic}
        ],
        "temperature": 0.8,
    }

    headers = {
        "Authorization": f"Bearer {OPENAI_API_KEY}",
        "Content-Type": "application/json"
    }

    try:
        client = get_httpx_client()
        resp = await _post_json_with_retry(client, CHAT_COMPLETION_URL, headers
        result = resp.json()
        script = result['choices'][0]['message']['content']
        logger.info("Successfully generated conversation script.")
        return script.strip()
    except httpx.HTTPStatusError as e:
        logger.error(f"Chat completion API request failed with status {e.respon
        return None
    except (KeyError, IndexError) as e:
        logger.error(f"Could not parse chat completion response: {e}")
        return None
    except Exception as e:
        logger.error(f"An unexpected error occurred during script generation: {
        return None

@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=0.5, max=4),
    retry=retry_if_exception_type(subprocess.CalledProcessError),
    reraise=False
```

```
)
async def _create_silence_clip(duration: float, file_path: str) -> bool:
    """Creates a short, silent audio clip using ffmpeg (with retries)."""
    try:
        await _run_subprocess_with_retry([
            FFMPEG_PATH, "-y", "-f", "lavfi", "-i", "anullsrc=r=44100:cl=mono",
            "-t", str(duration), "-q:a", "9", "-acodec", "libmp3lame", file_pat
        ])
        return True
    except subprocess.CalledProcessError as e:
        logger.warning(f"Could not create silence clip after retries. {e}")
        return False

@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=0.5, max=4),
    retry=retry_if_exception_type(subprocess.CalledProcessError),
    reraise=False
)
async def _get_audio_duration(file_path: str) -> float:
    """Gets the duration of an audio file using ffprobe (with retries)."""
    try:
        loop = asyncio.get_event_loop()
        result = await loop.run_in_executor(
            None,
            lambda: subprocess.run(
                ["ffprobe", "-v", "error", "-show_entries", "format=duration",
                 "-of", "default=noprint_wrappers=1:nokey=1", file_path],
                capture_output=True, text=True, check=True
            )
        )
        return float(result.stdout.strip())
    except (subprocess.CalledProcessError, FileNotFoundError, ValueError) as e:
        logger.warning(f"Could not get audio duration for {file_path}. Error: {
        return 30.0

# ---------- endpoint with retries wired in ----------

@api_router.post("/text-to-speech")
async def text_to_speech_endpoint(request: TTSRequest):
    """
    Generates a conversational, multi-speaker audio overview from input text,
    adds background ambience, and streams the final M4A audio file.
    """
    logger.info("Received request for conversational text-to-speech.")

    temp_dir = "static/temp_tts"
    os.makedirs(temp_dir, exist_ok=True)

    unique_id = str(uuid.uuid4())
    temp_clips_dir = os.path.join(temp_dir, unique_id)
    os.makedirs(temp_clips_dir, exist_ok=True)
```

```
    all_temp_files = []

    try:
        conversation_script = await _generate_conversation_script(request.text)
        if not conversation_script:
            raise HTTPException(status_code=500, detail="Failed to generate con

        logger.info("Generating individual audio clips...")
        lines = [line.strip() for line in conversation_script.strip().split('\n
        clip_files = []

        headers = {"Authorization": f"Bearer {OPENAI_API_KEY}"}
        client = get_httpx_client()

        for i, line in enumerate(lines):
            match = re.match(r'(\w+):\s*(.*)', line)
            if not match:
                if not line.strip() and i < len(lines) - 1:
                    pause_duration = random.uniform(0.5, 0.7)
                    silence_path = os.path.join(temp_clips_dir, f"silence_{i}.m
                    if await _create_silence_clip(pause_duration, silence_path)
                        clip_files.append(silence_path)
                        all_temp_files.append(silence_path)
                continue

            speaker, text = match.groups()
            voice = VOICE_MAPPING.get(speaker)
            if not voice:
                logger.warning(f"No voice configured for speaker '{speaker}'. S
                continue

            logger.info(f"Generating clip {i+1}/{len(lines)} for {speaker}...")
            clip_payload = {"model": TTS_MODEL, "input": text, "voice": voice,

            # POST to TTS with retries
            try:
                tts_response = await _post_json_with_retry(client, TTS_URL, hea
            except Exception as e:
                status = getattr(getattr(e, "response", None), "status_code", "
                msg = f"TTS request failed after retries (status={status}) for
                logger.error(msg, exc_info=True)
                raise HTTPException(status_code=502, detail=msg)

            clip_path = os.path.join(temp_clips_dir, f"clip_{i}.mp3")
            with open(clip_path, "wb") as f:
                f.write(tts_response.content)
            clip_files.append(clip_path)
            all_temp_files.append(clip_path)

            if i < len(lines) - 1:
                pause_duration = random.uniform(0.2, 0.5)
                silence_path = os.path.join(temp_clips_dir, f"silence_pause_{i}
                if await _create_silence_clip(pause_duration, silence_path):
```

```
                clip_files.append(silence_path)
                all_temp_files.append(silence_path)

        if not clip_files:
            raise HTTPException(status_code=500, detail="No audio clips were ge

        logger.info("Concatenating audio clips...")
        concat_list_path = os.path.join(temp_clips_dir, "concat_list.txt")
        all_temp_files.append(concat_list_path)
        with open(concat_list_path, 'w') as f:
            for clip_path in clip_files:
                f.write(f"file '{os.path.abspath(clip_path)}'\n")

        speech_only_path = os.path.join(temp_clips_dir, "speech_only.m4a")
        all_temp_files.append(speech_only_path)

        # ffmpeg concat with retries
        try:
            await _run_subprocess_with_retry([FFMPEG_PATH, "-y", "-f", "concat"
        except subprocess.CalledProcessError:
            raise HTTPException(status_code=500, detail="Failed to concatenate

        logger.info("Adding background ambience...")
        duration = await _get_audio_duration(speech_only_path)

        noise_path = os.path.join(temp_clips_dir, "noise.wav")
        all_temp_files.append(noise_path)

        # noise generation with retries
        try:
            await _run_subprocess_with_retry([FFMPEG_PATH, "-y", "-f", "lavfi",
        except subprocess.CalledProcessError:
            raise HTTPException(status_code=500, detail="Failed to generate bac

        final_audio_path = os.path.join(temp_dir, f"{unique_id}_final.m4a")
        all_temp_files.append(final_audio_path)

        # mix with retries
        try:
            await _run_subprocess_with_retry([
                FFMPEG_PATH, "-y", "-i", speech_only_path, "-i", noise_path,
                "-filter_complex", "[0:a][1:a]amix=inputs=2:duration=first:drop
                "-c:a", "aac", final_audio_path
            ])
        except subprocess.CalledProcessError:
            raise HTTPException(status_code=500, detail="Failed to mix speech a

        logger.info(f"Successfully created final audio: {final_audio_path}")

        def file_iterator(file_path, clips_dir):
            try:
                with open(file_path, "rb") as f:
                    yield from f
```

```
            finally:
                try:
                    os.remove(file_path)
                except FileNotFoundError:
                    pass
                try:
                    import shutil
                    shutil.rmtree(clips_dir, ignore_errors=True)
                except Exception:
                    pass

        return StreamingResponse(file_iterator(final_audio_path, temp_clips_dir

    except (httpx.HTTPStatusError, subprocess.CalledProcessError, HTTPException
        detail = e.detail if isinstance(e, HTTPException) else str(e)
        logger.error(f"Error in TTS endpoint: {detail}", exc_info=True)
        if isinstance(e, HTTPException):
            raise e
        raise HTTPException(status_code=500, detail=f"An error occurred during
    except Exception as e:
        logger.error(f"An unexpected error occurred in TTS endpoint: {e}", exc_
        raise HTTPException(status_code=500, detail="An unexpected server error
    finally:
        for f_path in all_temp_files:
            if isinstance(f_path, str) and os.path.exists(f_path) and not f_pat
                try:
                    os.remove(f_path)
                except OSError:
                    pass
        # directory cleanup is owned by the generator in file_iterator to avoid


@api_router.post("/reset/{conversation_id}")
async def reset_memory_and_session(conversation_id: str):
    if conversation_id in session_store:
        del session_store[conversation_id]
        logger.info(f"Cleared session data (incl. Google creds) for conversatio

    return JSONResponse(content={"status": f"Session for {conversation_id} clea


app.include_router(api_router)

# To run this app:
# 1. Install dependencies: pip install "fastapi[all]" "uvicorn[standard]" langc
# 2. Make sure you have FFMPEG and FFPROBE installed and available in your syst
# 3. Save the code as `main.py`.
# 4. Create `index.html` and `auth_callback.html` in the same directory.
# 5. Get your `client_secret.json` file from Google Cloud Console and place it
# 6. Set your API keys and tokens as environment variables for security.
# 7. Run with uvicorn: `uvicorn main:app --reload`
```

Front end:

```
<!DOCTYPE html>
<html lang="en" class="">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ask me anything</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600
    <link rel="icon" type="image/png" href="/static/icon/askme.ico" />
    <!-- Marked.js for rendering Markdown -->
    <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>
    <!-- Iconify CDN -->
    <script src="https://code.iconify.design/iconify-icon/3.0.0/iconify-icon.mi
    <script>
        // Set theme on page load to prevent FOUC (Flash of Unstyled Content)
        if (localStorage.getItem('color-theme') === 'dark' || (!('color-theme'
            document.documentElement.classList.add('dark');
        } else {
            document.documentElement.classList.remove('dark');
        }
        // Configure Tailwind to use the 'dark' class for dark mode
        tailwind.config = {
            darkMode: 'class'
        }
    </script>
    <style>
        body { font-family: 'Inter', sans-serif; }
        .resizing { user-select: none; }

        /* --- Custom Scrollbar --- */
        .custom-scrollbar::-webkit-scrollbar { width: 8px; height: 8px; }
        /* Light Mode */
        .custom-scrollbar::-webkit-scrollbar-track { background: #f8fafc; }
        .custom-scrollbar::-webkit-scrollbar-thumb { background-color: #d1d5db;
        #prompt-input { scrollbar-width: thin; scrollbar-color: #d1d5db transpa
        /* Dark Mode */
        .dark .custom-scrollbar::-webkit-scrollbar-track { background: #1f2937;
        .dark .custom-scrollbar::-webkit-scrollbar-thumb { background-color: #4
        .dark #prompt-input { scrollbar-color: #4b5563 transparent; }

        /* --- Input --- */
        #prompt-input:disabled { background-color: #f3f4f6; cursor: not-allowed
        .dark #prompt-input:disabled { background-color: #374151; }
        button[type="submit"]:disabled { background-color: #9ca3af; cursor: not
        .dark button[type="submit"]:disabled { background-color: #4b5563; }
```

```
        /* --- File Preview --- */
        .file-preview-item { position: relative; width: 80px; height: 80px; }
        .file-preview-img { width: 100%; height: 100%; object-fit: cover; borde
        .dark .file-preview-img { border-color: #4b5563; }
        .pdf-preview-container {
            display: flex; flex-direction: column; align-items: center; justify
            width: 100%; height: 100%; background-color: #f3f4f6; border-radius
        }
        .dark .pdf-preview-container { background-color: #374151; border-color:
        .pdf-preview-icon { color: #ef4444; }
        .pdf-preview-name {
            font-size: 10px; font-weight: 500; color: #4b5563;
            max-width: 90%; overflow: hidden; text-overflow: ellipsis; white-sp
        }
        .dark .pdf-preview-name { color: #d1d5db; }

        .remove-file-btn {
            position: absolute; top: -8px; right: -8px;
            background-color: white; color: #ef4444;
            border-radius: 9999px; width: 24px; height: 24px;
            display: flex; align-items: center; justify-content: center;
            border: 1px solid #e5e7eb; cursor: pointer; box-shadow: 0 1px 2px 0
        }
        .dark .remove-file-btn { background-color: #374151; color: #f87171; bor

        /* --- Progress Bar --- */
        #progress-bar-container { background-color: #e5e7eb; border-radius: 999
        .dark #progress-bar-container { background-color: #374151; }
        #progress-bar { height: 100%; background-color: #3b82f6; width: 0%; tra

        /* --- Log Output --- */
        .dark #log-output { background-color: #111827; }
        #log-toggle-icon.rotate-180 { transform: rotate(180deg); }

        /* --- Summary Panel --- */
        #summary-panel {
            transition: opacity 0.3s ease-in-out, transform 0.3s ease-in-out;
            opacity: 0;
            transform: translateY(20px);
            pointer-events: none;
        }
        #summary-panel.visible {
            opacity: 1;
            transform: translateY(0);
            pointer-events: auto;
        }
        #summary-output .prose-styles { color: #374151; font-size: 0.9rem; line
        .dark #summary-output .prose-styles { color: #d1d5db; }
        #summary-output .prose-styles h1,
        #summary-output .prose-styles h2,
        #summary-output .prose-styles h3,
        #summary-output .prose-styles h4 {
            font-weight: 700;
```

```
            margin-bottom: 0.75rem;
            margin-top: 1.25rem;
            color: #111827;
        }
        .dark #summary-output .prose-styles h1,
        .dark #summary-output .prose-styles h2,
        .dark #summary-output .prose-styles h3,
        .dark #summary-output .prose-styles h4 {
             color: #f9fafb;
        }
        #summary-output .prose-styles h1 { font-size: 1.25rem; }
        #summary-output .prose-styles h2 { font-size: 1.1rem; }
        #summary-output .prose-styles h3 { font-size: 1rem; }
        #summary-output .prose-styles ul { list-style-type: disc; padding-left:
        #summary-output .prose-styles li { margin-bottom: 0.5rem; }
        #summary-output .prose-styles p { margin-bottom: 1rem; }
        #summary-output .prose-styles strong { font-weight: 700; color: #111827
        .dark #summary-output .prose-styles strong { color: #f9fafb; }

        /* --- Thumbnail Gallery (Updated) --- */
        #thumbnail-gallery {
            display: flex;
            gap: 1rem;
            padding: 0.5rem; /* Add some padding around the gallery */
            overflow-x: auto;
            scrollbar-width: thin;
        }
        .thumbnail-item {
            flex-shrink: 0;
            width: 100px; /* Smaller width */
            height: 75px; /* Smaller height */
            border: 1px solid #e5e7eb; /* Subtle border */
            border-radius: 0.375rem; /* Slightly less rounded */
            cursor: pointer;
            transition: all 0.2s ease-in-out;
            position: relative;
            overflow: hidden;
            background-color: #ffffff;
            box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px -1px rgba(0,
        }
        .dark .thumbnail-item {
            background-color: #374151;
            border-color: #4b5563;
            box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.3), 0 1px 2px -1px rgba(0,
        }
        .thumbnail-item:hover {
            border-color: #3b82f6; /* Highlight with brand color on hover */
            transform: translateY(-2px); /* Subtle lift effect */
            box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -2px rgba(
        }
        .dark .thumbnail-item:hover {
             box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.4), 0 2px 4px -2px rgba
        }
```

```
            .thumbnail-item.active {
                border-color: #3b82f6;
                box-shadow: 0 0 0 3px rgba(59, 130, 246, 0.4); /* Refined active sh
            }
            .thumbnail-iframe {
                width: 100%;
                height: 100%;
                border: none;
                /* This scaling assumes the content is roughly 400x300, which fits
                transform: scale(0.25);
                transform-origin: 0 0;
                width: 400%; /* Compensate for scale */
                height: 400%;
                pointer-events: none; /* Prevent interaction with thumbnail content
            }
        </style>
    </head>
    <body class="bg-gray-50 dark:bg-gray-900 text-gray-700 dark:text-gray-300">

        <div class="flex h-screen">
            <aside id="sidebar" class="flex flex-col bg-white dark:bg-gray-800 z-10
                <header class="p-4 border-b border-gray-200 dark:border-gray-700 fl
                    <div>
                        <!-- IMPROVED: Used flexbox for better alignment and relati
                        <h1 class="text-xl font-bold text-gray-900 dark:text-gray-1
                            <span>Ask me anything</span>
                            <iconify-icon id="random-animal-icon1" icon="noto:owl"
                        </h1>
                    </div>
                    <div class="flex items-center space-x-2">
                        <button id="theme-toggle" type="button" class="text-gray-50
                            <svg id="theme-toggle-dark-icon" class="hidden w-5 h-5"
                            <svg id="theme-toggle-light-icon" class="hidden w-5 h-5
                        </button>
                        <button id="reset-chat-btn" class="flex items-center space-
                            <svg xmlns="http://www.w3.org/2000/svg" class="h-4 w-4"
                            <span>New Chat</span>
                        </button>
                    </div>
                </header>

                <div id="chat-history" class="flex-1 overflow-y-auto p-4 space-y-4
                    <div class="flex">
                        <div class="bg-gray-100 dark:bg-gray-700 text-gray-800 dark
                            <!-- IMPROVED: Flexbox for alignment, relative icon siz
                            <p class="text-sm flex items-center">
                               <span>Hello! How can I assist you today?</span>
                               <iconify-icon id="random-animal-icon" icon="noto:fis
                            </p>
                        </div>
                    </div>
                </div>
```

```
            <div id="status-indicator" class="p-4 border-t border-gray-200 dark
                <div class="space-y-1">
                    <button id="log-toggle-btn" type="button" class="w-full fle
                        <div class="flex items-center space-x-2">
                            <svg class="animate-spin h-4 w-4 text-gray-500 dark
                            <span id="status-text" class="text-sm font-medium t
                        </div>
                        <svg id="log-toggle-icon" xmlns="http://www.w3.org/2000
                            <path stroke-linecap="round" stroke-linejoin="round
                        </svg>
                    </button>
                    <div id="log-output" class="hidden bg-gray-50 dark:bg-gray-
                </div>
                <div id="progress-bar-container" class="mt-3">
                    <div id="progress-bar"></div>
                </div>
            </div>

            <div class="p-4 border-t border-gray-200 dark:border-gray-700 bg-wh
                <div id="input-error" class="text-red-500 dark:text-red-400 tex
                <form id="chat-form" novalidate>
                    <div id="file-previews-container" class="hidden flex flex-w
                    <div class="flex items-end bg-white dark:bg-gray-800 border
                        <input type="file" id="file-upload-input" class="hidden

                        <button type="button" id="attach-file-btn" class="text-
                            <svg xmlns="http://www.w3.org/2000/svg" class="h-6
                                <path stroke-linecap="round" stroke-linejoin="r
                            </svg>
                        </button>

                        <textarea id="prompt-input" placeholder="Ask me anythin

                        <button type="submit" class="bg-blue-600 text-white fon
                            <svg xmlns="http://www.w3.org/2000/svg" class="h-5
                        </button>
                    </div>
                </form>
            </div>
        </aside>

        <div id="resizer" class="w-1.5 cursor-col-resize bg-gray-200 dark:bg-gr

        <main class="flex-1 p-6 bg-gray-50 dark:bg-gray-900 flex flex-col overf
            <!-- Thumbnail Gallery -->
            <div id="thumbnail-gallery-wrapper" class="hidden mb-4">
                 <h2 class="text-lg font-semibold mb-2 text-gray-700 dark:text-
                 <div id="thumbnail-gallery" class="custom-scrollbar">
                     <!-- Thumbnails will be injected here -->
                 </div>
            </div>

            <!-- Main Render Area -->
```

```
            <div id="render-area" class="flex-1 w-full h-full">
                <div id="html-render-container" class="h-full flex flex-col">
                    <h2 id="render-title" class="text-2xl font-bold mb-4 text-g
                    <div id="render-output" class="bg-white dark:bg-gray-800 ro
                        <div id="render-placeholder" class="flex items-center j
                            <div class="text-center text-gray-400 dark:text-gra
                                <svg xmlns="http://www.w3.org/2000/svg" class="
                                <p class="mt-2 text-lg">The generated result wi
                            </div>
                        </div>
                        <!-- Summary Panel will be injected here -->
                    </div>
                </div>
            </div>
        </main>
    </div>

    <script>
        // --- DOM Element Selection ---
        const chatForm = document.getElementById('chat-form');
        const promptInput = document.getElementById('prompt-input');
        const chatHistory = document.getElementById('chat-history');
        const statusIndicator = document.getElementById('status-indicator');
        const statusText = document.getElementById('status-text');
        const progressBar = document.getElementById('progress-bar');
        const logOutput = document.getElementById('log-output');
        const logToggleBtn = document.getElementById('log-toggle-btn');
        const logToggleIcon = document.getElementById('log-toggle-icon');
        const renderOutput = document.getElementById('render-output');
        const renderTitle = document.getElementById('render-title');
        const renderPlaceholder = document.getElementById('render-placeholder')
        const inputError = document.getElementById('input-error');
        const resetChatBtn = document.getElementById('reset-chat-btn');
        const submitButton = chatForm.querySelector('button[type="submit"]');
        const attachFileBtn = document.getElementById('attach-file-btn');
        const fileUploadInput = document.getElementById('file-upload-input');
        const filePreviewsContainer = document.getElementById('file-previews-co
        const thumbnailGalleryWrapper = document.getElementById('thumbnail-gall
        const thumbnailGallery = document.getElementById('thumbnail-gallery');


        // --- State Variables ---
        const API_BASE_URL = '/api';
        const MAX_FILES = 5;
        let pollingInterval = null;
        let progressInterval = null;
        let conversationId = crypto.randomUUID();
        let lastSubmittedPrompt = '';
        let uploadedFiles = []; // This will store File objects
        let isLogExpanded = false;
        let lastLogMessage = '';
        let renderHistory = []; // Store history of {id, html}
        let activeRenderId = null; // ID of the currently displayed render
```

```
        let audioPlayer = new Audio(); // Create a single audio player instance
        let currentAudioBlob = null; // Store the fetched audio blob for downlo

        // --- Core Functions ---

        /**
         * NEW: Sets a random animal icon in the initial greeting message.
         */
        const setRandomAnimalIcon = () => {
            const animalIcons = [
                'noto:cat', 'noto:dog', 'noto:panda-face', 'noto:koala', 'noto:
                'noto:lion', 'noto:bear', 'noto:owl', 'noto:cow-face', 'noto:fr
                'noto:monkey-face', 'noto:chicken', 'noto:penguin', 'noto:bird'
                'noto:dolphin', 'noto:octopus', 'noto:fish', 'noto:turtle', 'no
                'noto:tropical-fish', 'noto:shark', 'noto:crocodile', 'noto:leo
                'noto:gorilla', 'noto:orangutan', 'noto:elephant', 'noto:hippop
                'noto:camel', 'noto:giraffe', 'noto:kangaroo', 'noto:water-buff
                'noto:llama', 'noto:goat', 'noto:deer', 'noto:turkey', 'noto:do
                'noto:duck', 'noto:swan', 'noto:parrot', 'noto:peacock', 'noto:
                'noto:skunk', 'noto:badger', 'noto:bat', 'noto:hedgehog', 'noto
            ];
            const iconElement = document.getElementById('random-animal-icon');
            const iconElement1 = document.getElementById('random-animal-icon1')
            if (iconElement) {
                const randomIndex = Math.floor(Math.random() * animalIcons.leng
                const randomIcon = animalIcons[randomIndex];
                iconElement.setAttribute('icon', randomIcon);
                iconElement1.setAttribute('icon', randomIcon);
                // Set aria-label for accessibility
                const animalName = randomIcon.replace('noto:', '').replace(/-/g
                iconElement.setAttribute('aria-label', `${animalName} mascot`);
                iconElement1.setAttribute('aria-label', `${animalName} mascot`)
            }
        };

        function setInputDisabled(disabled) {
            promptInput.disabled = disabled;
            submitButton.disabled = disabled;
        }

        function clearFilePreviews() {
            uploadedFiles = [];
            fileUploadInput.value = ''; // Reset the input
            filePreviewsContainer.innerHTML = '';
            filePreviewsContainer.classList.add('hidden');
        }

        function addFileToPreview(file) {
            if (uploadedFiles.length >= MAX_FILES) {
                showError(`You can only upload a maximum of ${MAX_FILES} files.
                return;
            }
```

```
            const fileId = `file-${Date.now()}-${Math.random()}`;
            file.id = fileId; // Attach a unique ID to the file object
            uploadedFiles.push(file);

            const previewItem = document.createElement('div');
            previewItem.className = 'file-preview-item';
            previewItem.dataset.id = fileId;

            let previewContent = '';
            if (file.type.startsWith('image/')) {
                const reader = new FileReader();
                reader.onload = (e) => {
                    previewItem.innerHTML = `
                        <img src="${e.target.result}" class="file-preview-img"
                        ${createRemoveButtonHtml()}
                    `;
                    addRemoveListener(previewItem);
                };
                reader.readAsDataURL(file);
            } else if (file.type === 'application/pdf' || file.type === 'text/p
                let iconSvg = '';
                if (file.type === 'application/pdf') {
                    iconSvg = `<svg class="w-8 h-8 pdf-preview-icon" xmlns="htt
                } else {
                    const iconColor = file.type === 'text/csv' ? 'text-green-50
                    iconSvg = `<svg class="w-8 h-8 ${iconColor}" xmlns="http://
                }

                previewItem.innerHTML = `
                    <div class="pdf-preview-container" title="${file.name}">
                        ${iconSvg}
                        <span class="pdf-preview-name">${file.name}</span>
                    </div>
                    ${createRemoveButtonHtml()}
                `;
                addRemoveListener(previewItem);
            } else {
                // Skip unsupported file types for now
                return;
            }

            filePreviewsContainer.appendChild(previewItem);
            filePreviewsContainer.classList.remove('hidden');
        }

        function createRemoveButtonHtml() {
            return `<button type="button" class="remove-file-btn" title="Remove
                        <svg xmlns="http://www.w3.org/2000/svg" class="h-4 w-4"
                    </button>`;
        }

        function addRemoveListener(element) {
             element.querySelector('.remove-file-btn').addEventListener('click'
```

```
            const idToRemove = element.dataset.id;
            uploadedFiles = uploadedFiles.filter(f => f.id !== idToRemove);
            element.remove();
            if (uploadedFiles.length === 0) {
                filePreviewsContainer.classList.add('hidden');
            }
        });
    }

    function showError(message) {
        inputError.textContent = message;
        inputError.classList.remove('hidden');
    }

    function clearError() {
        inputError.classList.add('hidden');
        inputError.textContent = '';
    }

    function startProgressSimulation() {
        if (progressInterval) clearInterval(progressInterval);
        progressBar.style.width = '0%';
        let width = 0;
        const duration = 15000;
        const intervalTime = 100;
        const increment = (intervalTime / duration) * 95;
        progressInterval = setInterval(() => {
            width += increment;
            if (width < 95) {
                progressBar.style.width = `${width}%`;
            } else {
                clearInterval(progressInterval);
            }
        }, intervalTime);
    }

    function completeProgress() {
        if (progressInterval) clearInterval(progressInterval);
        progressBar.style.width = '100%';
        setTimeout(() => {
            statusIndicator.classList.add('hidden');
        }, 500);
    }

    function appendLogMessage(message) {
        const logLine = document.createElement('div');
        logLine.textContent = `> ${message}`;
        logOutput.appendChild(logLine);
        logOutput.scrollTop = logOutput.scrollHeight;
    }

    async function resetChat() {
        const oldConversationId = conversationId;
```

```
            conversationId = crypto.randomUUID();
            if (pollingInterval) clearInterval(pollingInterval);
            if (progressInterval) clearInterval(progressInterval);
            pollingInterval = null;
            progressInterval = null;
            lastSubmittedPrompt = '';
            clearFilePreviews();

            // Reset render history
            renderHistory = [];
            activeRenderId = null;
            thumbnailGallery.innerHTML = '';
            thumbnailGalleryWrapper.classList.add('hidden');

            chatHistory.innerHTML = `<div class="flex"><div class="bg-gray-100
            renderOutput.innerHTML = '';
            renderPlaceholder.classList.remove('hidden');
            renderOutput.appendChild(renderPlaceholder);
            renderTitle.classList.add('hidden');
            statusIndicator.classList.add('hidden');
            promptInput.value = '';
            setInputDisabled(false);
            cleanupDynamicContent();

            try {
                await fetch(`${API_BASE_URL}/reset/${oldConversationId}`, { met
            } catch (error) {
                console.error('Error calling reset API:', error);
            }
        }

        async function handleFormSubmit(e) {
            e.preventDefault();
            const prompt = promptInput.value.trim();
            if (!prompt && uploadedFiles.length === 0) {
                showError('Please enter a prompt or upload a file.');
                return;
            }
            clearError();
            if (pollingInterval) clearInterval(pollingInterval);
            if (progressInterval) clearInterval(progressInterval);

            lastSubmittedPrompt = prompt;

            const imagePreviews = uploadedFiles
                .filter(f => f.type.startsWith('image/'))
                .map(f => URL.createObjectURL(f));

            appendMessage('You', prompt, 'user', imagePreviews, uploadedFiles.f

            await sendRequest(prompt, uploadedFiles);

            promptInput.value = '';
```

```
            clearFilePreviews();
            autoResizeTextarea(promptInput);
        }

        async function sendRequest(prompt, files) {
            statusIndicator.classList.remove('hidden');
            // Reset log state for new request
            isLogExpanded = false;
            logOutput.classList.add('hidden');
            logToggleIcon.classList.remove('rotate-180');
            logToggleBtn.classList.remove('rounded-t-lg');
            logToggleBtn.classList.add('rounded-lg');
            logOutput.innerHTML = '';
            lastLogMessage = '';

            startProgressSimulation();
            setInputDisabled(true);

            const formData = new FormData();
            formData.append('prompt', prompt);
            formData.append('conversation_id', conversationId);
            files.forEach(file => {
                formData.append('files', file, file.name);
            });

            try {
                const startResponse = await fetch(`${API_BASE_URL}/chat`, {
                    method: 'POST',
                    body: formData,
                });
                if (startResponse.status !== 202) throw new Error('Failed to st
                const { task_id } = await startResponse.json();
                pollForTaskResult(task_id);
            } catch (error) {
                handleErrorResponse({ error: 'Failed to start the chat task.' }
            }
        }

        function pollForTaskResult(taskId) {
            pollingInterval = setInterval(async () => {
                try {
                    const statusResponse = await fetch(`${API_BASE_URL}/tasks/$
                    if (!statusResponse.ok) throw new Error('Polling failed');
                    const task = await statusResponse.json();

                    if (task.log && task.log !== lastLogMessage) {
                        appendLogMessage(task.log);
                        lastLogMessage = task.log;
                    }

                    if (task.status === 'SUCCESS') {
                        clearInterval(pollingInterval);
                        handleSuccessResponse(task.result);
```

```
                } else if (task.status === 'FAILURE') {
                    clearInterval(pollingInterval);
                    handleErrorResponse(task.result);
                } else if (task.status === 'AUTH_REQUIRED') {
                    clearInterval(pollingInterval);
                    handleAuthRequired(task.result);
                }
            } catch (error) {
                clearInterval(pollingInterval);
                handleErrorResponse({ error: 'Lost connection to the backen
            }
        }, 2500);
    }

    function handleSuccessResponse(data) {
        completeProgress();
        setInputDisabled(false);
        if (data.response_code) {
            appendMessage('Agent', 'The requested application has been succ

            // Create new history item
            const renderId = `render-${Date.now()}`;
            renderHistory.push({ id: renderId, html: data.response_code });

            // Update the gallery with the new item
            updateThumbnailGallery();

            // Set the new item as active, which will trigger the render
            setActiveRender(renderId);

        } else if (data.response_text) {
            appendMessage('Agent', data.response_text, 'agent');
        } else {
            appendMessage('Agent', 'I was unable to generate a response.',
        }
    }

    function handleErrorResponse(errorData) {
        completeProgress();
        appendMessage('Error', errorData.error || 'An unknown error occurre
        setInputDisabled(false);
    }

    function handleAuthRequired(data) {
        completeProgress();
        setInputDisabled(false);
        appendMessage('Agent', 'I need access to Google Docs for this reque
        window.open(data.auth_url, 'googleAuthPopup', 'width=600,height=700
    }

    async function handleAuthCallback(code, state) {
        appendMessage('Agent', 'Authentication in progress...', 'agent');
        setInputDisabled(true);
```

```
            try {
                const response = await fetch('/api/auth/google/callback', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ code, state }),
                });

                if (!response.ok) {
                    const errorData = await response.json();
                    throw new Error(errorData.detail || 'Failed to complete aut
                }

                appendMessage('Agent', 'Authentication successful! Retrying you
                if(lastSubmittedPrompt) {
                    await sendRequest(lastSubmittedPrompt, uploadedFiles);
                } else {
                    setInputDisabled(false);
                }
            } catch (error) {
                handleErrorResponse({ error: error.message });
            }
        }

        function appendMessage(sender, content, type, imageDataUrlArray = [], o
            const bubbleClasses = {
                agent: 'bg-gray-100 dark:bg-gray-700 text-gray-800 dark:text-gr
                user: 'bg-blue-600 text-white ml-auto'
            };
            const messageDiv = document.createElement('div');
            messageDiv.className = `flex ${type === 'user' ? 'justify-end' : ''

            let fileHtml = '';
            if (imageDataUrlArray.length > 0 || otherFileArray.length > 0) {
                const images = imageDataUrlArray.map(src => `<img src="${src}"

                const otherFiles = otherFileArray.map(file => {
                    let iconSvg = '';
                    if (file.type === 'application/pdf') {
                        iconSvg = `<svg class="w-5 h-5 text-red-500 flex-shrink
                    } else if (file.type === 'text/csv') {
                        iconSvg = `<svg class="w-5 h-5 text-green-500 flex-shri
                    } else if (file.type === 'text/plain') {
                         iconSvg = `<svg class="w-5 h-5 text-blue-500 flex-shri
                    }

                    return `
                        <div class="flex items-center space-x-2 p-2 bg-gray-200
                            ${iconSvg}
                            <span class="text-xs font-medium truncate">${file.n
                        </div>`;
                }).join('');

                fileHtml = `<div class="mt-2 flex flex-wrap gap-2 ${type === 'u
```

```
            }

            messageDiv.innerHTML = `
                <div class="max-w-md">
                    <p class="text-xs text-gray-500 dark:text-gray-400 mb-1 ${t
                    <div class="w-full ${bubbleClasses[type] || 'bg-red-100 tex
                        ${content ? content.replace(/\n/g, '<br>') : ''}
                        ${fileHtml}
                    </div>
                </div>`;
            chatHistory.appendChild(messageDiv);
            chatHistory.scrollTop = chatHistory.scrollHeight;

            // Revoke object URLs after a short delay to allow the browser to r
            setTimeout(() => {
                imageDataUrlArray.forEach(url => URL.revokeObjectURL(url));
            }, 100);
        }

        function renderHtml(htmlContent) {
            // 1. Clean up old buttons/panels from the *current* output area
            cleanupDynamicContent();

            // 2. Render the new result in the current output area
            renderOutput.innerHTML = ''; // Clear placeholder from current view

            const iframe = document.createElement('iframe');
            iframe.className = 'w-full h-full border-0 dynamic-iframe';
            iframe.srcdoc = htmlContent;
            iframe.sandbox = 'allow-scripts allow-same-origin allow-forms allow
            renderOutput.appendChild(iframe);
            renderPlaceholder.classList.add('hidden');
            renderTitle.classList.remove('hidden');

            // 3. Add new buttons for the new (current) result
            const summarizeBtn = document.createElement('button');
            summarizeBtn.id = 'summarize-html-btn';
            summarizeBtn.innerHTML = `
                <span>Summarise</span>
                <span class="border-l border-gray-300 dark:border-gray-500 h-4
                <svg class="w-5 h-5" viewBox="0 0 20 20" fill="currentColor">
                    <path d="M10 2.5L11.5 6.5L15.5 8L11.5 9.5L10 13.5L8.5 9.5L4
                    <path d="M5 12.5L6 10.5L8 9.5L6 8.5L5 6.5L4 8.5L2 9.5L4 10.
                    <path d="M15 12.5L16 10.5L18 9.5L16 8.5L15 6.5L14 8.5L12 9.
                </svg>
            `;
            summarizeBtn.className = 'absolute bottom-6 right-6 z-20 flex items
            summarizeBtn.onclick = handleSummarizeClick;

            const downloadBtn = document.createElement('button');
            downloadBtn.id = 'download-html-btn';
            downloadBtn.innerHTML = `<svg xmlns="http://www.w3.org/2000/svg" cl
            downloadBtn.title = 'Download HTML';
```

```
        downloadBtn.className = 'absolute top-3 right-3 bg-white text-gray-
        downloadBtn.onclick = () => {
              const currentRender = renderHistory.find(item => item.id ===
              if (currentRender) {
                  downloadFile(currentRender.html, 'agent-creation.html', '
              }
        };

        renderOutput.parentElement.appendChild(summarizeBtn);
        renderOutput.parentElement.appendChild(downloadBtn);

        createSummaryPanel();
    }

    function updateThumbnailGallery() {
        thumbnailGallery.innerHTML = '';
        if (renderHistory.length > 0) {
            thumbnailGalleryWrapper.classList.remove('hidden');
        } else {
            thumbnailGalleryWrapper.classList.add('hidden');
            return;
        }

        renderHistory.forEach((item, index) => {
            const thumb = document.createElement('div');
            thumb.className = 'thumbnail-item';
            thumb.dataset.id = item.id;
            thumb.title = `Result ${index + 1}`;

            const iframe = document.createElement('iframe');
            iframe.className = 'thumbnail-iframe';
            iframe.srcdoc = item.html;
            iframe.sandbox = ''; // Prevent any scripts in thumbnail

            thumb.appendChild(iframe);
            thumb.addEventListener('click', () => {
                setActiveRender(item.id);
            });
            thumbnailGallery.appendChild(thumb);
        });
    }

    function setActiveRender(renderId) {
        if (!renderId || activeRenderId === renderId) {
            return;
        }
        activeRenderId = renderId;

        // Update active class on thumbnails
        document.querySelectorAll('.thumbnail-item').forEach(thumb => {
            if (thumb.dataset.id === renderId) {
                thumb.classList.add('active');
                thumb.scrollIntoView({ behavior: 'smooth', block: 'nearest'
```

```
                } else {
                    thumb.classList.remove('active');
                }
            });

            // Re-render the main view with the selected history item
            const activeItem = renderHistory.find(item => item.id === activeRen
            if (activeItem) {
                renderHtml(activeItem.html);
            }
        }

        function createSummaryPanel() {
            const panel = document.createElement('div');
            panel.id = 'summary-panel';
            panel.className = 'absolute bottom-4 left-4 w-full max-w-2xl bg-whi

            panel.innerHTML = `
                <div class="flex justify-between items-center p-2 border-b bord
                    <h3 class="font-semibold text-gray-800 dark:text-gray-200 p
                    <button id="close-summary-panel-btn" title="Close" class="p
                        <svg class="w-5 h-5" fill="none" stroke="currentColor"
                    </button>
                </div>
                <div id="summary-output" class="p-4 overflow-y-auto custom-scro
                    <!-- Summary content will be injected here -->
                </div>
                <div class="p-2 border-t border-gray-200 dark:border-gray-700">
                    <div class="flex items-center justify-between">
                        <div class="flex items-center space-x-1">
                             <button id="listen-summary-btn" title="Listen to s
                             <button id="download-summary-btn" title="Download
                             <button id="copy-summary-btn" title="Copy summary"
                            <span id="copy-feedback" class="text-sm text-green-
                        </div>
                        <div class="text-xs text-gray-400 dark:text-gray-500 fl
                            <svg class="w-4 h-4 mr-1" fill="none" stroke="curre
                            Content quality may vary. Powered by AI
                        </div>
                    </div>
                </div>
            `;
            renderOutput.appendChild(panel);
            panel.querySelector('#copy-summary-btn').addEventListener('click',
            panel.querySelector('#listen-summary-btn').addEventListener('click'
            panel.querySelector('#download-summary-btn').addEventListener('clic
            panel.querySelector('#close-summary-panel-btn').addEventListener('c
                panel.classList.remove('visible');
                audioPlayer.pause();
                audioPlayer.src = '';
                currentAudioBlob = null;
            });
        }
```

```
async function handleSummarizeClick() {
    const currentRender = renderHistory.find(item => item.id === active
    if (!currentRender) return;

    const summaryPanel = document.getElementById('summary-panel');
    if (!summaryPanel) return;

    // Extract text from HTML
    const tempDiv = document.createElement('div');
    tempDiv.innerHTML = currentRender.html;
    const textContent = tempDiv.textContent || tempDiv.innerText || '';

    if (!textContent.trim()) {
        return;
    }

    const summaryOutputDiv = summaryPanel.querySelector('#summary-outpu
    summaryOutputDiv.innerHTML = '<div class="flex justify-center items
    summaryPanel.classList.add('visible');

    try {
        const response = await fetch(`${API_BASE_URL}/summarize`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ content: textContent })
        });

        if (!response.ok) {
            throw new Error(`Summarization failed with status: ${respon
        }

        const data = await response.json();
        summaryOutputDiv.innerHTML = marked.parse(data.summary);

    } catch (error) {
        summaryOutputDiv.innerHTML = `<p class="text-red-500">Sorry, an
        console.error('Summarization error:', error);
    }
}

async function handleListenClick() {
    const summaryOutputDiv = document.getElementById('summary-output');
    const listenBtn = document.getElementById('listen-summary-btn');
    const downloadBtn = document.getElementById('download-summary-btn')
    if (!summaryOutputDiv || !listenBtn || !downloadBtn) return;

    const textToSpeak = summaryOutputDiv.innerText;
    if (!textToSpeak.trim()) {
        return;
    }

    listenBtn.disabled = true;
```

```
            listenBtn.innerHTML = '<svg class="animate-spin h-5 w-5" xmlns="htt
            downloadBtn.disabled = true; // Also disable download button while
            currentAudioBlob = null; // Reset any previous blob

            try {
                const response = await fetch(`${API_BASE_URL}/text-to-speech`,
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({ text: textToSpeak })
                });

                if (!response.ok) {
                    throw new Error(`TTS failed with status: ${response.status}
                }

                const audioBlob = await response.blob();
                currentAudioBlob = audioBlob; // Store the blob for download
                const audioUrl = URL.createObjectURL(audioBlob);
                audioPlayer.src = audioUrl;
                audioPlayer.play();
                downloadBtn.disabled = false; // Enable download button on succ

            } catch (error) {
                console.error('TTS error:', error);
                currentAudioBlob = null;
                downloadBtn.disabled = true;
            } finally {
                listenBtn.disabled = false;
                listenBtn.innerHTML = '<svg class="w-5 h-5" fill="none" stroke=
            }
        }

        function handleDownloadAudioClick() {
            if (!currentAudioBlob) {
                console.error("No audio blob available to download.");
                return;
            }
            // The backend mixes to m4a (aac), so we use the correct filename a
            downloadFile(currentAudioBlob, 'summary_audio.m4a', 'audio/m4a');
        }

        function copySummaryToClipboard() {
            const summaryOutputDiv = document.getElementById('summary-output');
            const feedbackSpan = document.getElementById('copy-feedback');

            if (summaryOutputDiv && feedbackSpan) {
                const textToCopy = summaryOutputDiv.innerText;
                navigator.clipboard.writeText(textToCopy).then(() => {
                    feedbackSpan.classList.remove('hidden');
                    setTimeout(() => {
                        feedbackSpan.classList.add('hidden');
                    }, 2000);
                }).catch(err => {
```

```
                console.error('Failed to copy text: ', err);
            });
        }
    }

    function cleanupDynamicContent() {
        // Remove buttons and panels associated with the CURRENT render out
        const currentContainer = document.getElementById('html-render-conta
        if (currentContainer) {
            currentContainer.querySelectorAll('#download-html-btn, #summari
        }

        // The summary panel is inside render-output, so we can select it t
        renderOutput.querySelectorAll('#summary-panel').forEach(el => el.re

        // Reset state related to dynamic content
        audioPlayer.pause();
        audioPlayer.src = '';
        currentAudioBlob = null;
    }

    function downloadFile(content, fileName, contentType) {
        const blob = content instanceof Blob ? content : new Blob([content]
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = fileName;
        document.body.appendChild(a);
        a.click();
        document.body.removeChild(a);
        URL.revokeObjectURL(url);
    }

    function autoResizeTextarea(textarea) {
        const maxHeight = 200;
        textarea.style.height = 'auto';
        textarea.style.height = `${Math.min(textarea.scrollHeight, maxHeigh
        textarea.style.overflowY = textarea.scrollHeight > maxHeight ? 'aut
    }

    // --- Initialization and Event Listeners ---

    // Run initial setup functions on page load
    setRandomAnimalIcon();

    function toggleLogVisibility() {
        isLogExpanded = !isLogExpanded;
        logOutput.classList.toggle('hidden');
        logToggleIcon.classList.toggle('rotate-180');
        if (isLogExpanded) {
            logToggleBtn.classList.remove('rounded-lg');
            logToggleBtn.classList.add('rounded-t-lg');
        } else {
```

```
            logToggleBtn.classList.remove('rounded-t-lg');
            logToggleBtn.classList.add('rounded-lg');
        }
    }

    logToggleBtn.addEventListener('click', toggleLogVisibility);

    chatForm.addEventListener('submit', handleFormSubmit);
    resetChatBtn.addEventListener('click', resetChat);
    promptInput.addEventListener('input', () => {
        autoResizeTextarea(promptInput);
        clearError();
    });
    promptInput.addEventListener('keydown', (event) => {
        if (event.key === 'Enter' && !event.shiftKey) {
            event.preventDefault();
            submitButton.click();
        }
    });

    attachFileBtn.addEventListener('click', () => fileUploadInput.click());
    fileUploadInput.addEventListener('change', (event) => {
        clearError();
        const files = event.target.files;
        if (uploadedFiles.length + files.length > MAX_FILES) {
            showError(`Cannot upload more than ${MAX_FILES} files in total.
            return;
        }
        for (const file of files) {
            addFileToPreview(file);
        }
        // Reset the input so the same file can be selected again if remove
        event.target.value = '';
    });

    function dataURLtoFile(dataurl, filename) {
        var arr = dataurl.split(','), mime = arr[0].match(/:(.*?);/)[1],
            bstr = atob(arr[1]), n = bstr.length, u8arr = new Uint8Array(n)
        while(n--){
            u8arr[n] = bstr.charCodeAt(n);
        }
        return new File([u8arr], filename, {type:mime});
    }

    promptInput.addEventListener('paste', (event) => {
        clearError();
        const items = (event.clipboardData || window.clipboardData).items;
        let imageFile = null;
        for (let i = 0; i < items.length; i++) {
            if (items[i].type.indexOf('image') !== -1) {
                imageFile = items[i].getAsFile();
                break;
            }
```

```
            }
            if (imageFile) {
                event.preventDefault();
                if (uploadedFiles.length >= MAX_FILES) {
                    showError(`Cannot paste: you can only upload a maximum of $
                    return;
                }
                addFileToPreview(imageFile);
            }
        });

        // --- Resizer Functionality ---
        const initResize = () => {
            const sidebar = document.getElementById('sidebar');
            const resizer = document.getElementById('resizer');
            if (!sidebar || !resizer) return;
            let startX, startWidth;
            const onMouseDown = (e) => {
                e.preventDefault();
                startX = e.clientX;
                startWidth = sidebar.offsetWidth;
                document.body.classList.add('resizing');
                document.documentElement.addEventListener('mousemove', onMouseM
                document.documentElement.addEventListener('mouseup', onMouseUp)
            };
            const onMouseMove = (e) => {
                const newWidth = startWidth + e.clientX - startX;
                if (newWidth > 300 && newWidth < (window.innerWidth - 300)) {
                    sidebar.style.width = newWidth + 'px';
                }
            };
            const onMouseUp = () => {
                document.body.classList.remove('resizing');
                document.documentElement.removeEventListener('mousemove', onMou
                document.documentElement.removeEventListener('mouseup', onMouse
            };
            resizer.addEventListener('mousedown', onMouseDown);
        };
        initResize();

        // --- Theme Toggle Functionality ---
        const themeToggleBtn = document.getElementById('theme-toggle');
        const themeToggleDarkIcon = document.getElementById('theme-toggle-dark-
        const themeToggleLightIcon = document.getElementById('theme-toggle-ligh

        // Change the icons inside the button based on current theme
        const updateIcons = () => {
            if (document.documentElement.classList.contains('dark')) {
                themeToggleLightIcon.classList.remove('hidden');
                themeToggleDarkIcon.classList.add('hidden');
            } else {
                themeToggleDarkIcon.classList.remove('hidden');
                themeToggleLightIcon.classList.add('hidden');
```

```
            }
        };

        // Initialize icons on page load
        updateIcons();

        themeToggleBtn.addEventListener('click', () => {
            // toggle theme
            document.documentElement.classList.toggle('dark');

            // update localStorage
            if (document.documentElement.classList.contains('dark')) {
                localStorage.setItem('color-theme', 'dark');
            } else {
                localStorage.setItem('color-theme', 'light');
            }
            // update icons
            updateIcons();
        });

    </script>
</body>
</html>
```

Ai Agent | Llm | Agentic Ai | Automation | Langgraph

Follow

Written by Ferry Djaja

618 followers · 19 following

https://www.linkedin.com/in/ferrydjaja/

No responses yet

Bgerby

What are your thoughts?

More from Ferry Djaja

Ferry Djaja

Building a Model Context Protocol (MCP) Web Client with FastAPI, OAuth, and a Sleek Front-End

TL;DR: This post walks through a complete MCP web client that connects to one or more MCP servers, handles OAuth (or direct auth headers)...

Oct 19 1

Ferry Djaja

I Built an AI Mermaid Diagram Generator That Fixes Its Own Mistakes

From a simple idea to a web app that turns text (and even images!) into perfect diagrams, automatically.

Oct 5

Ferry Djaja

Building a Workato Recipe Flow: From Workato JSON to Flawless Flowcharts

How I used an LLM to visualize complex automations and then taught it to fix its own mistakes.

Oct 5

Ferry Djaja

Build a Web Model Context Protocol (MCP) Client

LLMs are powerful but isolated. Let’s give them secure access to our applications using MCP and build a web UI to control them.

Oct 17

See all from Ferry Djaja

Recommended from Medium

In Towards AI by Alpha Iterations

Agentic AI: Build ReAct Agent using LangGraph

Practical Guide to build ReAct agent using LangGraph and OpenAI API [Code Included]

Oct 25 104 1

In Data Science Collective by Ida Silfverskiöld

Agentic AI: Single vs Multi-Agent Systems

Building with a structured data source in LangGraph

6d ago 615 10

Guangya Liu

MCP Sampling: Architecture, Workflow, and Practical Guide

Not a Medium member? Visit https://gyliu513.github.io/

Aug 21 2

In MongoDB by MongoDB

Build AI Agents Worth Keeping: The Canvas Framework

This article was written by Mikiko Bazeley.

Oct 8 110 5

In Python in Plain English by Samantha Blake

LangGraph vs LangChain: Choosing Your AI Framework in 2026

As AI applications grow more complex, the discussion around LangGraph vs LangChain has become central for developers. With the AI agents...

Oct 24 104

In Generative AI by Gao Dalie (*高達烈*)

DeepSeek-OCR + LLama4 + RAG Just Revolutionized Agent OCR Forever

During the weekend, I scrolled through Twitter to see what was happening in the AI community. Once again, DeepSeek has drawn worldwide...

Oct 27 257 2

See more recommendations