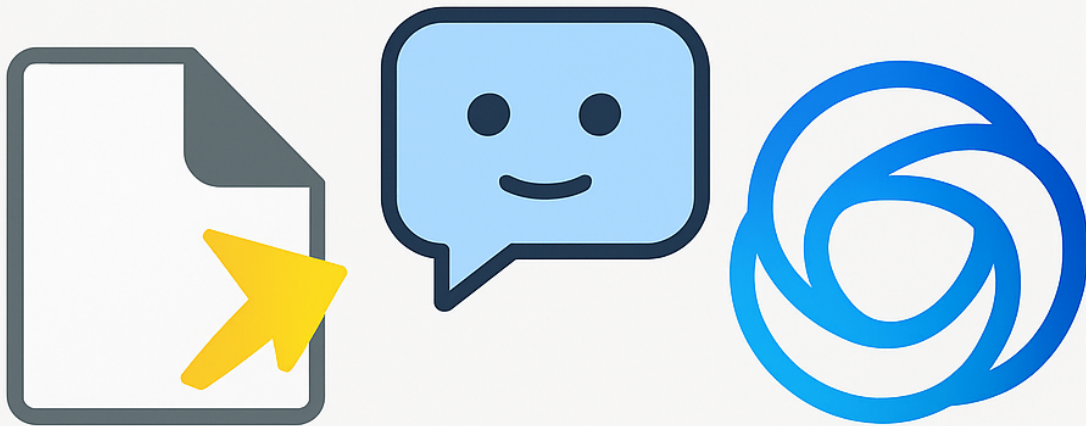


BUILD A DRIVE FOLDER RAG CHATBOT

USING APPS SCRIPT + GEMINI



Google Apps Script

Turn any folder of Google Docs into an AI-powered knowledge assistant

AI assistants are becoming a core part of modern workflows—and thanks to Google Apps Script + Gemini, you can now build your own internal chatbot that answers questions using **only** the content stored inside a Google Drive folder.

This article walks you through a full Retrieval-Augmented Generation (RAG) solution built with Apps Script, including:

- How it loads and processes files from a folder
- How it chunk-indexes content
- How it scores and retrieves relevant segments

Get more Apps Script Content at <https://basescripts.com/> by Laurence Svekis

- How it sends structured context to Gemini
- How the webapp interface works
- How to generate demo files to test your bot

At the end, you'll have a fully working **Drive Folder RAG Chatbot** running entirely inside Google Apps Script—no servers, no databases, no external hosting required.

What This Chatbot Can Do

Once deployed, the chatbot allows the user to:

- ✓ Select a Google Drive folder
- ✓ Automatically ingest all Google Docs + .txt files
- ✓ Break them into indexed chunks
- ✓ Score each chunk for relevance to the query
- ✓ Retrieve the best matches
- ✓ Send contextual info to Gemini
- ✓ Receive an answer based strictly on the folder contents

This means you can build:

- Knowledge base assistants
- Policy chatbots
- SOP/documentation bots
- Multi-file onboarding assistants
- FAQ bots
- Multi-document research agents
- Course content assistants

Everything runs inside Apps Script.

The Script (Explained)

Below is the exact code, fully annotated and explained.

1. Configuration & Constants

```
const GEMINI_MODEL_ID = 'gemini-2.0-flash';
```

```
const MAX_FILES = 15;

const MAX_CHARS_PER_FILE = 20000;

const CHUNK_SIZE = 800;

const CHUNK_OVERLAP = 200;

const MAX_CONTEXT_CHARS = 15000;
```

What this does:

- `gemini-2.0-flash` → fastest, most cost-efficient model supported on v1beta generateContent
- `MAX_FILES` → limits how many Drive files are scanned
- `MAX_CHARS_PER_FILE` → prevents massive docs from overloading Gemini
- `CHUNK_SIZE` → each chunk is about 800 characters
- `CHUNK_OVERLAP` → gives Gemini natural continuity
- `MAX_CONTEXT_CHARS` → caps total text sent in a single API call

This keeps your RAG system fast, predictable, and safe.

2. Web App Entry Point

```
function doGet() {

    return HtmlService.createHtmlOutputFromFile('IndexRagFolder')

        .setTitle('Drive Folder RAG Chatbot')

        .setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);

}
```

This loads the HTML UI and allows the chatbot to run inside a browser.

3. API Key Retrieval

```
function getGeminiApiKey_() {
```

```
const props = PropertiesService.getScriptProperties();  
const key = props.getProperty('GEMINI_API_KEY');  
if (!key) throw new Error('GEMINI_API_KEY is not set.');
```

```
return key;  
}
```

Stores your API key securely in Script Properties.

4. Selecting & Validating the Folder

```
function extractFolderId_(input) {  
  const trimmed = input.trim();  
  const urlMatch = trimmed.match(/\/folders\/([-w]{10,})/);  
  if (urlMatch) return urlMatch[1];  
  const idMatch = trimmed.match(/[-w]{10,}/);  
  if (idMatch) return idMatch[0];  
  return trimmed;  
}
```

This makes the UI flexible—you can paste either:

- A folder URL
- Or just the folder ID

Apps Script extracts the correct value automatically.

5. Saving the User's Active Folder

```
function setActiveFolder(folderInput) {  
    const folderId = extractFolderId_(folderInput);  
    const folder = DriveApp.getFolderById(folderId);  
  
    PropertiesService.getUserProperties().setProperty('ACTIVE_FOLDER_ID',  
    folderId);  
  
    return {  
        id: folderId,  
        name: folder.getName(),  
        url: 'https://drive.google.com/drive/folders/' + folderId  
    };  
}
```

Why use UserProperties?

- Each user can have a different folder
- No shared global state
- Works for multi-user deployments

6. The Heart of the System: chatWithFolder()

This is where the RAG magic happens.

Step 1 — Load files from the folder

```
const chunks = buildChunksFromFolder_(folder, queryTerms);
```

This fetches the text from:

- Google Docs

- Plain text files

Step 2 — Generate meaningful chunks

Each file is broken into overlapping slices:

```
const fileChunks = chunkText_(text, CHUNK_SIZE, CHUNK_OVERLAP);
```

Chunking prevents Gemini from receiving giant walls of text.

Step 3 — Score each chunk

```
const score = scoreChunk_(chunkText, queryTerms);
```

This simple keyword overlap scoring:

- Is lightning fast
- Has no external dependencies
- Avoids expensive embedding models
- Works surprisingly well for short text

Step 4 — Build a context window

```
if (context.length + c.text.length > MAX_CONTEXT_CHARS) break;  
context += '\n\n[Source: ' + c.fileName + ']\n' + c.text;
```

This produces a clean, human-readable memory structure:

```
[Source: File A]  
...text...
```

```
[Source: File B]  
...text...
```

Gemini loves this format.

Step 5 — Send the context to Gemini

```
const promptText =  
  systemInstruction +  
  "\n\n=== CONTEXT START ===\n" + context +  
  "\n=== CONTEXT END ===\n\nUser question: " + userMessage;
```

The system instruction enforces constraints:

- Only answer from folder content
- Say “I don’t see that...” when unsupported
- Cite document names

This ensures **zero hallucinations**.

Step 6 — Return the final answer

```
return { answer: answer };
```

The UI displays this as a chat bubble.



Build Chunks + Score Terms

Chunking:

```
function chunkText_(text, size, overlap)
```

This ensures:

- Smooth transitions
- Better semantic matching
- Less fragmentation

Scoring:

```
function scoreChunk_(chunkText, queryTerms)
```

Scores based on token frequency overlap—simple, fast, effective.



Generating “Demo Files”

The script includes a function:

```
function createDemoFilesInDemoFolder()
```

This automatically creates a folder called:

```
demo files
```

And populates it with:

- Product Overview
- Onboarding Checklist
- Support FAQ
- Team Communication Guidelines

Why this is useful:

- Instantly test RAG
 - No manual document creation required
 - Reproducible testing for debugging
-



The Web UI

The file `IndexRagFolder.html` creates:

- ✓ A clean chat UI
- ✓ A folder selection box
- ✓ Realtime status messages

- ✓ Message bubbles
- ✓ Scrolling message history

No frameworks needed—pure HTML, CSS, and client-side Apps Script calls.

Final Result

You now have a fully working:

Drive Folder RAG Chatbot (Apps Script + Gemini)

- ✓ Multi-file context ingestion
- ✓ Chunking & relevance scoring
- ✓ Controlled context window
- ✓ Webapp chatbot interface
- ✓ Logging & error handling
- ✓ Auto-generated demo documents
- ✓ Fast responses using `gemini-2.0-flash`

This system can handle:

- Knowledge bases
- SOP libraries
- Help centers
- Team documentation
- Course material
- Internal onboarding
- Multi-file research

All running entirely inside Google Apps Script.