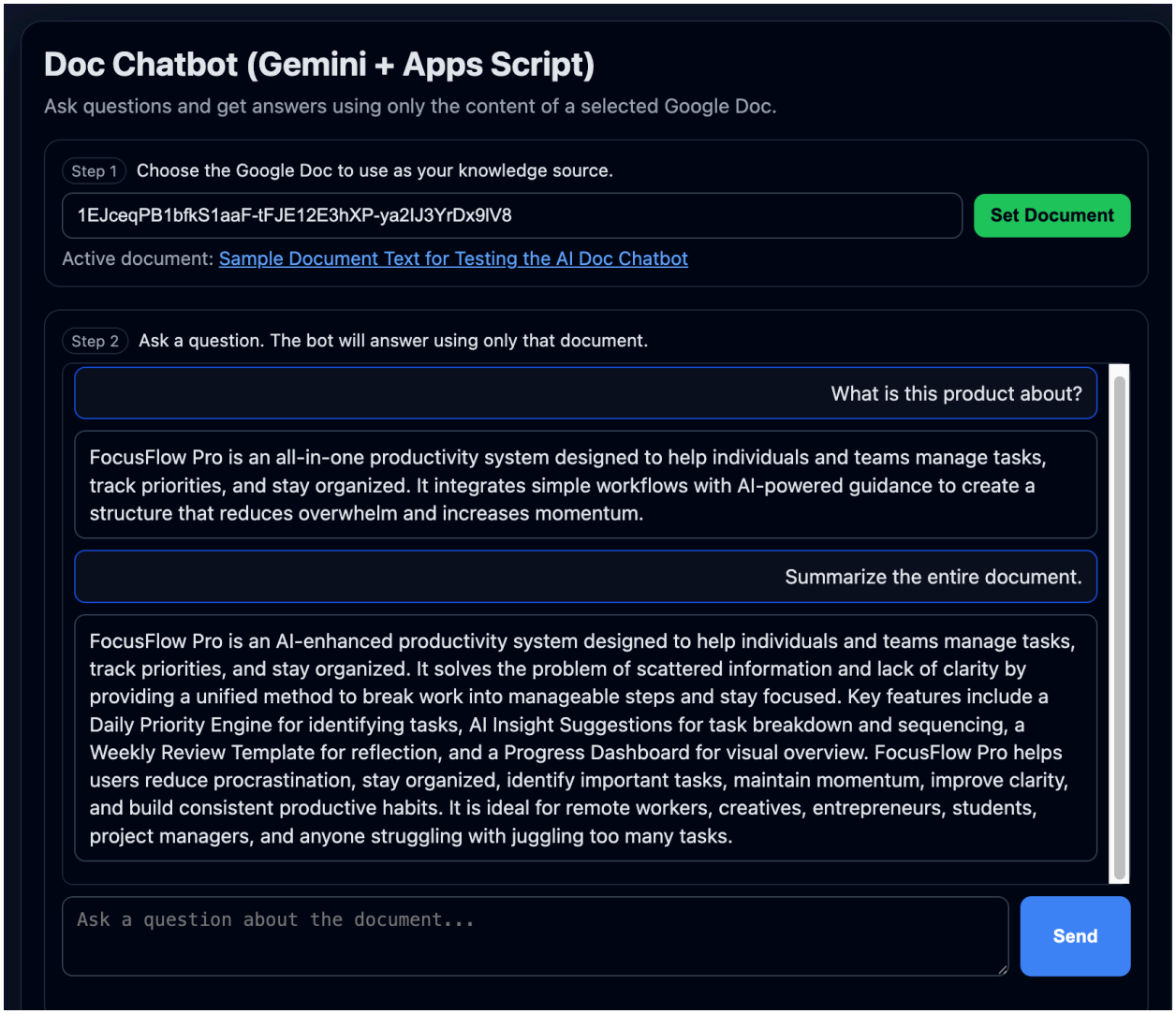
Build a Google Docs AI Chatbot with Apps Script and Gemini



Have you ever wished you could drop a Google Doc into a chatbot and ask questions about it—*and* have the bot answer only from that document?

In this post, you'll build exactly that:

> An Apps Script-powered web app that lets the user select a Google Doc, then chat with an AI agent (Gemini) that answers questions using **only** the content in that document.

We'll cover:

- How the solution is structured

- The full Apps Script (server) code
- The HTML/JS (client) chat UI
- How the Gemini API call is built
- A simple exercise to help you extend it

What You'll Build

The final result is a small web app that:

1. Lets you paste a **Google Doc URL or ID** and click **Set Document**.
2. Saves that document as your "active" knowledge source.
3. Provides a chat box where you can ask questions.
4. Calls Gemini with:
    - The document text
    - Your question
    - A system instruction telling Gemini to only use the doc
5. Returns a concise answer back into the chat.

Each user gets their own active document (stored in `UserProperties`), so multiple people can use the same deployed web app with different docs.

Project Setup

1. Go to **script.google.com** and create a **new standalone project**.
2. Enable the **Google Docs** and **Drive** services (they're usually available by default in Apps Script).
3. Get a **Gemini API key** from Google AI Studio and add it as a Script Property:
    - In Apps Script, open **Project Settings** → **Script properties** → **Add property**
    - Key: `GEMINI_API_KEY`
    - Value: your actual key

4. You'll create two files:
    - `Code.gs` – server-side Apps Script
    - `Index.html` – client UI

Let's go through each piece.

Server-Side: `Code.gs`

Below is the core Apps Script that:

- Handles the web app entry point (doGet)
- Stores the active document per user
- Reads the document text
- Calls the Gemini API
- Returns an answer

```
/**
 * Simple AI Agent for a Selected Google Doc using Gemini.
 */

const GEMINI_MODEL_ID = 'gemini-2.5-flash'; // adjust if needed
const MAX_DOC_CHARS = 60000; // safety limit so we don't exceed
payload size

/**
 * Entry point for web app.
 */
function doGet() {
  return HtmlService.createHtmlOutputFromFile('Index')
    .setTitle('Doc Chatbot (Gemini)')
    .setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);
}

/**
 * Get Gemini API key from Script Properties.
 */
function getGeminiApiKey_() {
  const props = PropertiesService.getScriptProperties();
  const key = props.getProperty('GEMINI_API_KEY');
  if (!key) {
    throw new Error('GEMINI_API_KEY is not set in Script
Properties.');
  }
  return key;
}

/**
 * Extract a Doc ID from a pasted URL or raw ID.
 */
function extractDocId_(input) {
  if (!input) throw new Error('No document value provided.');
  const trimmed = input.trim();
  // Try to find a 25+ char ID from a Docs URL
  const match = trimmed.match(/[-\w]{25,}/);
  return (match && match[0]) || trimmed;
```

```
}

/**
 * Store the active doc for the current user.
 */
function setActiveDoc(docInput) {
  const docId = extractDocId_(docInput);

  // Validate that this is actually a Google Doc
  let file;
  try {
    file = DriveApp.getFileById(docId);
  } catch (e) {
    throw new Error('Could not find a file with that ID. Please check
the URL/ID.');
  }

  if (file.getMimeType() !== MimeType.GOOGLE_DOCS) {
    throw new Error('The selected file is not a Google Doc.');
  }

  const userProps = PropertiesService.getUserProperties();
  userProps.setProperty('ACTIVE_DOC_ID', docId);

  const doc = DocumentApp.openById(docId);
  return {
    id: docId,
    title: doc.getName(),
    url: doc.getUrl()
  };
}

/**
 * Return info about current active doc for this user.
 */
function getActiveDocInfo() {
  const userProps = PropertiesService.getUserProperties();
  const docId = userProps.getProperty('ACTIVE_DOC_ID');
  if (!docId) {
    return null;
  }

  try {
    const doc = DocumentApp.openById(docId);
```

```
    return {
      id: docId,
      title: doc.getName(),
      url: doc.getUrl()
    };
  } catch (e) {
    userProps.deleteProperty('ACTIVE_DOC_ID');
    return null;
  }
}

/**
 * Core chat function: answer question using ONLY the active doc.
 *
 * @param {string} userMessage User's question.
 * @returns {Object} { answer: string }
 */
function chatWithDoc(userMessage) {
  if (!userMessage || !userMessage.trim()) {
    throw new Error('Please enter a question.');
  }

  const userProps = PropertiesService.getUserProperties();
  const docId = userProps.getProperty('ACTIVE_DOC_ID');
  if (!docId) {
    throw new Error('No document selected yet. Please set a document
first.');
  }

  // Fetch document text
  const doc = DocumentApp.openById(docId);
  let docText = doc.getBody().getText() || '';
  if (!docText) {
    throw new Error('The selected document is empty.');
  }

  // Limit doc size to avoid extremely large payloads
  if (docText.length > MAX_DOC_CHARS) {
    docText = docText.slice(0, MAX_DOC_CHARS);
  }

  const apiKey = getGeminiApiKey_();
```

```
  const url =
`https://generativelanguage.googleapis.com/v1beta/models/${GEMINI_MODE
L_ID}:generateContent`;

  const systemInstruction = [
    'You are a helpful assistant that answers questions using ONLY the
information in the document below.',
    'If the answer is not clearly in the document, reply: "I don\'t
see that in the document."',
    'Be concise and quote or refer to relevant sections when helpful.'
  ].join(' ');

  const promptText =
    systemInstruction +
    '\n\n=== DOCUMENT CONTENT START ===\n' +
    docText +
    '\n=== DOCUMENT CONTENT END ===\n\n' +
    'User question: ' + userMessage;

  const payload = {
    contents: [
      {
        role: 'user',
        parts: [{ text: promptText }]
      }
    ],
    generationConfig: {
      temperature: 0.3,
      maxOutputTokens: 512
    }
  };

  const options = {
    method: 'post',
    muteHttpExceptions: true,
    contentType: 'application/json',
    headers: {
      'x-goog-api-key': apiKey
    },
    payload: JSON.stringify(payload)
  };

  const response = UrlFetchApp.fetch(url, options);
  const code = response.getResponseCode();
```

```
  if (code !== 200) {
    throw new Error('Gemini API error: HTTP ' + code + ' - ' +
response.getContentText());
  }

  const data = JSON.parse(response.getContentText());

  const answer =
    (data.candidates &&
      data.candidates[0] &&
      data.candidates[0].content &&
      data.candidates[0].content.parts &&
      data.candidates[0].content.parts
        .map(function (p) {
          return p.text || '';
        })
        .join('')) ||
    'No response from Gemini.';

  return { answer: answer };
}
```

Now let's break down what's happening.

Understanding the Server Code

1. `doGet()` – Web App Entry

```
function doGet() {
  return HtmlService.createHtmlOutputFromFile('Index')
    .setTitle('Doc Chatbot (Gemini)')
    .setXFrameOptionsMode(HtmlService.XFrameOptionsMode.ALLOWALL);
}
```

- This function is called when someone opens the web app URL.
- It loads `Index.html` and sends it to the browser.
- `setTitle` sets the page title.
- `setXFrameOptionsMode(ALLOWALL)` allows embedding this UI in iframes if needed.

---

2. `getGeminiApiKey_()` – Reading the API Key

```
function getGeminiApiKey_() {
  const props = PropertiesService.getScriptProperties();
  const key = props.getProperty('GEMINI_API_KEY');
  if (!key) {
    throw new Error('GEMINI_API_KEY is not set in Script
Properties.');
  }
  return key;
}
```

- Uses `ScriptProperties` to fetch your API key, so it's not hard-coded in the script.
- If the key is missing, it throws a clear error message.

---

3. `extractDocId_()` – Handling URLs and IDs

```
function extractDocId_(input) {
  if (!input) throw new Error('No document value provided.');
  const trimmed = input.trim();
  const match = trimmed.match(/[-\w]{25,}/);
  return (match && match[0]) || trimmed;
}
```

- Users can paste either a full **Google Docs URL** or just the **ID**.
- This function uses a regex to extract the long document ID (usually 44+ characters).
- If it doesn't find a match, it assumes the whole string is the ID.

---

4. `setActiveDoc()` – Saving the Selected Doc

```
function setActiveDoc(docInput) {
  const docId = extractDocId_(docInput);

  let file;
  try {
    file = DriveApp.getFileById(docId);
  } catch (e) {
    throw new Error('Could not find a file with that ID. Please check
the URL/ID.');
  }
```

```
  if (file.getMimeType() !== MimeType.GOOGLE_DOCS) {
    throw new Error('The selected file is not a Google Doc.');
  }

  const userProps = PropertiesService.getUserProperties();
  userProps.setProperty('ACTIVE_DOC_ID', docId);

  const doc = DocumentApp.openById(docId);
  return {
    id: docId,
    title: doc.getName(),
    url: doc.getUrl()
  };
}
```

This function:

1. Extracts the doc ID from user input.
2. Uses `DriveApp.getFileById()` to verify the file exists.
3. Checks that the MIME type is a **Google Doc**.
4. Stores the doc ID in `UserProperties` under key `ACTIVE_DOC_ID`.
5. Returns some handy metadata (title + URL) back to the client.

> **Key idea:** `UserProperties` are per-user. That means each user can pick their own active document, even with the same web app URL.

---

5. `getActiveDocInfo()` – Showing Current Doc

```
function getActiveDocInfo() {
  const userProps = PropertiesService.getUserProperties();
  const docId = userProps.getProperty('ACTIVE_DOC_ID');
  if (!docId) {
    return null;
  }

  try {
    const doc = DocumentApp.openById(docId);
    return {
      id: docId,
      title: doc.getName(),
      url: doc.getUrl()
    };
```

```
  } catch (e) {
    userProps.deleteProperty('ACTIVE_DOC_ID');
    return null;
  }
}
```

- Used when the page loads to show whether a document has already been set.
- If the doc doesn't exist or can't be opened, the property is cleared.

---

6. `chatWithDoc()` – The AI Brain

This is the heart of the app.

1. **Validate input**
2. **Fetch active doc ID**
3. **Read doc text**
4. **Trim to `MAX_DOC_CHARS`**
5. **Build a prompt for Gemini**
6. **Call the Gemini REST API**
7. **Parse and return the answer**

Key piece: the prompt structure.

```
const systemInstruction = [
  'You are a helpful assistant that answers questions using ONLY the
information in the document below.',
  'If the answer is not clearly in the document, reply: "I don\'t see
that in the document."',
  'Be concise and quote or refer to relevant sections when helpful.'
].join(' ');

const promptText =
  systemInstruction +
  '\n\n=== DOCUMENT CONTENT START ===\n' +
  docText +
  '\n=== DOCUMENT CONTENT END ===\n\n' +
  'User question: ' + userMessage;
```

This tells Gemini:

- What its role is

- The only allowed knowledge source (the document)
- What to do when it doesn't find the answer

Then the payload is sent to the Gemini API using `UrlFetchApp.fetch()`.

Client-Side: `Index.html`

The HTML file creates a minimal but pleasant chat UI:

- A section to paste/set a document URL or ID
- A live "active document" label
- A scrollable chat window
- A text area and send button

Here's the full file:

```
<!DOCTYPE html>
<html>
  <head>
    <base target="_top" />
    <meta charset="UTF-8" />
    <style>
      body {
        font-family: system-ui, -apple-system, BlinkMacSystemFont,
'Segoe UI', sans-serif;
        margin: 0;
        padding: 16px;
        background: #0f172a;
        color: #e5e7eb;
      }
      .app {
        max-width: 800px;
        margin: 0 auto;
        background: #020617;
        border-radius: 16px;
        padding: 16px;
        box-shadow: 0 10px 40px rgba(0, 0, 0, 0.6);
        border: 1px solid #1f2937;
      }
      h1 {
        font-size: 1.5rem;
        margin-top: 0;
```

```
  margin-bottom: 0.5rem;
}
.subtitle {
  font-size: 0.9rem;
  color: #9ca3af;
  margin-bottom: 1rem;
}
.doc-section,
.chat-section {
  margin-bottom: 16px;
  padding: 12px;
  border-radius: 12px;
  background: #020617;
  border: 1px solid #1f2937;
}
.doc-row {
  display: flex;
  gap: 8px;
  flex-wrap: wrap;
  align-items: center;
}
.doc-row input {
  flex: 1;
  padding: 8px 10px;
  border-radius: 8px;
  border: 1px solid #374151;
  background: #020617;
  color: #e5e7eb;
}
.doc-row button {
  padding: 8px 12px;
  border-radius: 8px;
  border: none;
  background: #22c55e;
  color: #020617;
  font-weight: 600;
  cursor: pointer;
}
.doc-row button:disabled {
  opacity: 0.6;
  cursor: default;
}
.doc-info {
  margin-top: 6px;
```

```
  font-size: 0.85rem;
  color: #9ca3af;
}
.doc-info a {
  color: #60a5fa;
}
.messages {
  max-height: 360px;
  overflow-y: auto;
  padding: 8px;
  background: #020617;
  border-radius: 8px;
  border: 1px solid #1f2937;
  margin-bottom: 8px;
  font-size: 0.9rem;
}
.message {
  margin-bottom: 8px;
  padding: 8px 10px;
  border-radius: 8px;
  line-height: 1.4;
  white-space: pre-wrap;
}
.message.user {
  background: #0b1120;
  border: 1px solid #1d4ed8;
  text-align: right;
}
.message.bot {
  background: #020617;
  border: 1px solid #334155;
}
.input-row {
  display: flex;
  gap: 8px;
  margin-top: 4px;
}
.input-row textarea {
  flex: 1;
  resize: vertical;
  min-height: 40px;
  max-height: 100px;
  padding: 8px 10px;
  border-radius: 8px;
```

```
      border: 1px solid #374151;
      background: #020617;
      color: #e5e7eb;
    }
    .input-row button {
      padding: 8px 12px;
      border-radius: 8px;
      border: none;
      background: #3b82f6;
      color: white;
      font-weight: 600;
      cursor: pointer;
      min-width: 80px;
    }
    .input-row button:disabled {
      opacity: 0.6;
      cursor: default;
    }
    .status {
      font-size: 0.8rem;
      color: #9ca3af;
      margin-top: 4px;
      min-height: 1.2em;
    }
    .error {
      color: #f97373;
    }
    .small {
      font-size: 0.8rem;
    }
    .badge {
      display: inline-block;
      padding: 2px 6px;
      border-radius: 999px;
      border: 1px solid #1f2937;
      font-size: 0.7rem;
      color: #9ca3af;
    }
  </style>
</head>
<body>
  <div class="app">
    <h1>Doc Chatbot (Gemini + Apps Script)</h1>
    <div class="subtitle">
```

```
        Ask questions and get answers using only the content of a
selected Google Doc.
      </div>

      <div class="doc-section">
        <div class="small" style="margin-bottom: 6px;">
          <span class="badge">Step 1</span>
           Choose the Google Doc to use as your knowledge source.
        </div>
        <div class="doc-row">
          <input
            id="docInput"
            type="text"
            placeholder="Paste Google Doc URL or ID here..."
          />
          <button id="setDocBtn" onclick="onSetDoc()">Set
Document</button>
        </div>
        <div id="docInfo" class="doc-info">No document selected
yet.</div>
      </div>

      <div class="chat-section">
        <div class="small" style="margin-bottom: 6px;">
          <span class="badge">Step 2</span>
           Ask a question. The bot will answer using only that
document.
        </div>
        <div id="messages" class="messages"></div>
        <div class="input-row">
          <textarea
            id="userInput"
            placeholder="Ask a question about the document..."
          ></textarea>
          <button id="sendBtn" onclick="onSend()">Send</button>
        </div>
        <div id="status" class="status"></div>
      </div>
    </div>

    <script>
      const messagesEl = document.getElementById('messages');
      const userInputEl = document.getElementById('userInput');
      const statusEl = document.getElementById('status');
```

```
      const docInfoEl = document.getElementById('docInfo');
      const setDocBtn = document.getElementById('setDocBtn');
      const sendBtn = document.getElementById('sendBtn');

      function addMessage(text, sender) {
        const div = document.createElement('div');
        div.className = 'message ' + sender;
        div.textContent = text;
        messagesEl.appendChild(div);
        messagesEl.scrollTop = messagesEl.scrollHeight;
      }

      function setLoading(isLoading, which) {
        if (which === 'doc') {
          setDocBtn.disabled = isLoading;
          setDocBtn.textContent = isLoading ? 'Setting…' : 'Set
Document';
        } else if (which === 'chat') {
          sendBtn.disabled = isLoading;
          sendBtn.textContent = isLoading ? 'Thinking…' : 'Send';
        }
      }

      function onSetDoc() {
        const value =
document.getElementById('docInput').value.trim();
        if (!value) {
          statusEl.textContent = 'Please paste a document URL or ID.';
          statusEl.classList.add('error');
          return;
        }
        statusEl.textContent = '';
        statusEl.classList.remove('error');
        setLoading(true, 'doc');

        google.script.run
          .withSuccessHandler(function (info) {
            setLoading(false, 'doc');
            if (info) {
              docInfoEl.innerHTML =
                'Active document: <a href="' +
                info.url +
                '" target="_blank">' +
                info.title +
```

```
          '</a>';
      } else {
        docInfoEl.textContent = 'No document selected yet.';
      }
    })
    .withFailureHandler(function (err) {
      setLoading(false, 'doc');
      statusEl.textContent = err.message || String(err);
      statusEl.classList.add('error');
    })
    .setActiveDoc(value);
}

function onSend() {
  const text = userInputEl.value.trim();
  if (!text) {
    statusEl.textContent = 'Please enter a question.';
    statusEl.classList.add('error');
    return;
  }
  statusEl.textContent = '';
  statusEl.classList.remove('error');

  addMessage(text, 'user');
  userInputEl.value = '';
  setLoading(true, 'chat');

  google.script.run
    .withSuccessHandler(function (res) {
      setLoading(false, 'chat');
      addMessage(res.answer, 'bot');
    })
    .withFailureHandler(function (err) {
      setLoading(false, 'chat');
      statusEl.textContent = err.message || String(err);
      statusEl.classList.add('error');
    })
    .chatWithDoc(text);
}

function init() {
  google.script.run
    .withSuccessHandler(function (info) {
      if (info) {
```

```
            docInfoEl.innerHTML =
              'Active document: <a href="' +
              info.url +
              '" target="_blank">' +
              info.title +
              '</a>';
          } else {
            docInfoEl.textContent = 'No document selected yet.';
          }
        })
        .getActiveDocInfo();
    }

    document.addEventListener('DOMContentLoaded', init);
  </script>
 </body>
</html>
```

How the Client Code Works

1. Displaying Messages

```
function addMessage(text, sender) {
  const div = document.createElement('div');
  div.className = 'message ' + sender;
  div.textContent = text;
  messagesEl.appendChild(div);
  messagesEl.scrollTop = messagesEl.scrollHeight;
}
```

- Adds a new `.message user` or `.message bot` div to the chat log.
- Scrolls the container to the bottom so the latest message is visible.

---

2. Handling Loading State

```
function setLoading(isLoading, which) {
  if (which === 'doc') {
    setDocBtn.disabled = isLoading;
    setDocBtn.textContent = isLoading ? 'Setting…' : 'Set Document';
  } else if (which === 'chat') {
```

```
    sendBtn.disabled = isLoading;
    sendBtn.textContent = isLoading ? 'Thinking…' : 'Send';
  }
}
```

- Disables buttons while the server is doing work to prevent duplicate clicks.

---

3. Setting the Active Doc

```
function onSetDoc() {
  const value = document.getElementById('docInput').value.trim();
  if (!value) {
    statusEl.textContent = 'Please paste a document URL or ID.';
    statusEl.classList.add('error');
    return;
  }
  statusEl.textContent = '';
  statusEl.classList.remove('error');
  setLoading(true, 'doc');

  google.script.run
    .withSuccessHandler(function (info) {
      setLoading(false, 'doc');
      if (info) {
        docInfoEl.innerHTML =
          'Active document: <a href="' +
          info.url +
          '" target="_blank">' +
          info.title +
          '</a>';
      } else {
        docInfoEl.textContent = 'No document selected yet.';
      }
    })
    .withFailureHandler(function (err) {
      setLoading(false, 'doc');
      statusEl.textContent = err.message || String(err);
      statusEl.classList.add('error');
    })
    .setActiveDoc(value);
}
```

- Reads the input field.
- Uses `google.script.run` to call the server function `setActiveDoc`.
- On success, shows the active doc's title and link.
- On failure, shows an error message.

---

4. Sending a Question to Gemini

```
function onSend() {
  const text = userInputEl.value.trim();
  if (!text) {
    statusEl.textContent = 'Please enter a question.';
    statusEl.classList.add('error');
    return;
  }
  statusEl.textContent = '';
  statusEl.classList.remove('error');

  addMessage(text, 'user');
  userInputEl.value = '';
  setLoading(true, 'chat');

  google.script.run
    .withSuccessHandler(function (res) {
      setLoading(false, 'chat');
      addMessage(res.answer, 'bot');
    })
    .withFailureHandler(function (err) {
      setLoading(false, 'chat');
      statusEl.textContent = err.message || String(err);
      statusEl.classList.add('error');
    })
    .chatWithDoc(text);
}
```

- Adds the user's message to the chat immediately.
- Calls `chatWithDoc` on the server.
- Displays the AI answer or any error returned.

---

5. Initializing the Active Doc on Load

```
function init() {
```

```
  google.script.run
    .withSuccessHandler(function (info) {
      if (info) {
        docInfoEl.innerHTML =
          'Active document: <a href="' +
          info.url +
          '" target="_blank">' +
          info.title +
          '</a>';
      } else {
        docInfoEl.textContent = 'No document selected yet.';
      }
    })
    .getActiveDocInfo();
}

document.addEventListener('DOMContentLoaded', init);
```

- When the page loads, it asks the server: “Do I already have an active document?”
- This makes the experience smoother for returning users.

Hands-On Exercise: Extend the Agent

Once you have this working, here are some guided exercises to deepen your understanding.

Exercise 1 – Add a “Doc Excerpt” to the Answer

Goal: Show a short snippet from the document under the AI’s answer to give more transparency.

Hints:

- Modify `chatWithDoc` to also return a short chunk of the document text around the first match of the user’s keywords.

Change the return object from `{ answer: string }` to something like:

```
return {
  answer: answer,
  excerpt: someSnippet
};
```

-

In `Index.html`, update the success handler:

```
.withSuccessHandler(function (res) {
  setLoading(false, 'chat');
  addMessage(res.answer, 'bot');
  if (res.excerpt) {
    addMessage('Excerpt: ' + res.excerpt, 'bot');
  }
})
```

-

You'll learn how to:

- Extend server responses
- Pass multiple fields back to the client
- Work with substrings and searches in Apps Script

Exercise 2 – Handle Documents Larger Than the Limit

Right now the script truncates the doc at `MAX_DOC_CHARS`.

Challenge:

- Instead of truncating blindly, split the document into paragraphs.
- Select only the paragraphs that contain one or more keywords from the user's question.
- Join those paragraphs and send that as `docText` to Gemini.

This gives you a first taste of **basic retrieval-augmented generation** (RAG) using only Apps Script and Gemini.

Exercise 3 – Add a "Reset Chat" Button

Add a small button that clears the chat window.

Add a button in the HTML:

```
<button onclick="clearChat()">Clear Chat</button>
```

Implement `clearChat()` in the `<script>`:

```
function clearChat() {
  messagesEl.innerHTML = '';
}
```

You'll get practice adding new UI behaviors and reinforcing the client–side flow.

How to Deploy and Test

1. In Apps Script, go to **Deploy** → **New deployment** → **Web app**.
2. Set **Who has access** to "Anyone with the link" (or your preferred option).
3. Copy the URL.
4. Open it in your browser.
5. Paste a Google Doc URL/ID (for example, your test doc with the FocusFlow Pro text).
6. Ask questions like:
    - "What problem does this product solve?"
    - "Who is the ideal user?"
    - "Summarize the key features."

You should see Gemini answering based on your document content.