Collab Chrome Extension

Tutorial: Append Daily call transcription text with the Notion REST API

Creating a transcription feature in Daily Collab, our Daily video call Chrome extension demo



Jess Mitchell

Published: 2 Sep 2021 (Updated: 14 Jan 2022) • 8 min read



> *[14/01/22] This demo app uses manifest v2. Manifest v2 is being deprecated by Chrome in 2022 and will no longer be accepted into the Chrome store.*

This tutorial is part of a series on how we built Daily Collab, our latest Chrome extension demo, which lets you embed and transcribe Daily video calls in any Notion page.

Daily built the recent Chrome extension demo, Daily Collab, to show how Daily and Notion's APIs could work together to embed a custom video chat experience in Notion. Our project goal was to embed a video widget into any Notion document, as well as to let users optionally transcribe their calls right to Notion.

Building out the video and transcription features into the Daily Collab Chrome extension requires integrating the Notion API in two main ways:

- Authorizing Notion document edits via a public integration. We covered this in a previous post on working with Notion.
- Making `PATCH` requests via the Notion API to append block children to a specific Notion document as transcription text is received.

In today's tutorial, we'll cover this second step, and see how appending transcription text was accomplished in the Daily Collab demo.

If you're interested in learning more about Daily Collab's features or how Chrome extensions work, check out our previous posts from our Daily Collab series.

Tutorial goals

Our main goals today will be to understand:

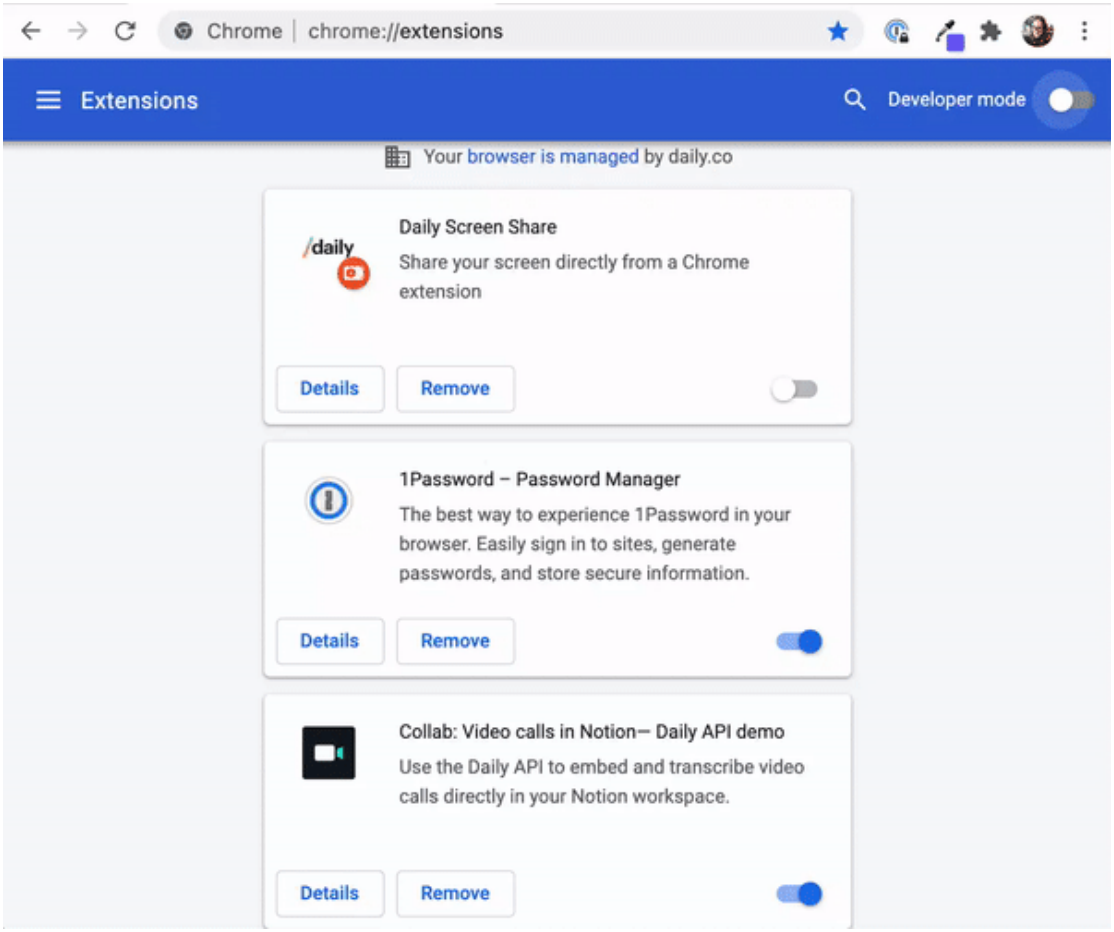
- How to use Notion's `PATCH` request to append new block children to a Notion document
- Where this `PATCH` request is incorporated into the Daily Collab structure
- How to prefix transcription text with the speaker's name

Getting started

To follow along in the full source code for Daily Collab, check out the Github repo.

If you want to test out Daily Collab locally, you will need to:

1. Clone Daily Collab locally
2. Run `npm install` to install the dependencies
3. Run `npm run start`
4. Load your extension on Chrome following:
    - Access `chrome://extensions/`
    - Check `Developer mode`
    - Click on `Load unpacked extension`
    - Select the `build` folder.



Add Daily Collab locally to your Chrome extensions

How Daily transcription text is generated and sent to the client

Before we discuss how Daily Collab handles transcription text received client-side, let's discuss how the text is even generated.

Daily uses Deepgram to get transcription text for live Daily calls that have transcription enabled. If transcription has been turned on for a call, `daily-js` will send an `app-message` participant event each time Deepgram reports new transcription text. This "app message" is a broadcast message, which means it is sent to every participant in the call.

To receive the text client-side, you will need to add an event listener to your local instance of the Daily callframe. Let's take a look now at how to create an event listener.

Note: Daily Collab is using an internal, not-so-secret Daily transcription method that should not be used in production apps. Keep an eye out for the public feature announcement coming soon. 👀

Receiving transcription text through an `app-message` Daily event

Receiving the transcription text client-side will look the same regardless of what you're planning on doing with the text. You can attach an event listener for the Daily `app-message` event and pass a callback for how you'd like to handle the text, like so:

```
import DailyIframe from '@daily-co/daily-js';

…

const handleAppMessage = (e) => {
```

```
    console.log('[APP MESSAGE]', e);
    // check if the message is a transcription message (vs. a ch
    if (e?.fromId === 'transcription' && e?.data?.is_final) {
       // handle transcription text: e.data.text
    }
};

const callFrame = DailyIframe.createCallObject({ url: DAILY_ROOM

callFrame.on('app-message', handleAppMessage);
```

Shortened version of code in Daily Collab

Notice how we're checking the `fromId` in the `app-message` event? This is because `app-message` s can be triggered from multiple sources, whether it be a chat message, transcription, or really any type of text-based data you'd like to pass between participants.

To ensure you're checking for transcription text specifically, the `fromId` should be `transcription` and the data object attached will have an `is_final` key with the value `true` .

How Daily and Notion APIs collaborate

The Daily Collab Chrome extension has several moving pieces that work together to get full feature functionality. To start, Daily Collab's content script is where all the UI (React app) lives and where the Daily `callFrame` instance is initialized. The `app-message` event listener is added in the `CallProvider` context, which is where most of the app's state is managed.

*Note: Learn more about how information is sent is our previous tutorial on the custom Daily Collab API.

In terms of how app messages are handled, Daily Collab's `app-message` event handler looks similar to the example included above:

```
const handleAppMessage = (e) => {
   console.log('[APP MESSAGE]', a);
```

```
  // Check if the message is transcription text
  if (e?.fromId === 'transcription' && e?.data?.is_final) {
    const p = daily.participants();
    const local = p.local;

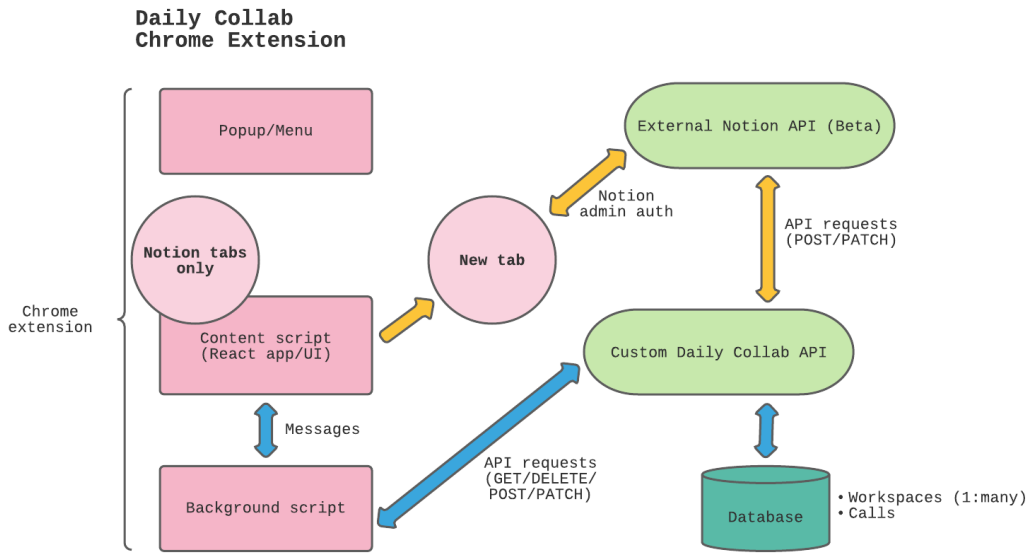
    ...
    // Each participant handles added their own text to avoid d
    if (local.session_id === e.data.session_id) {
    /**
     * Send message to background script to add the transcripti
     */
      addTextToNotionFromBackground(e.data.text, workspaceId, l
    }
    ...
  }
};
```

Any time a message is received that is transcription text, we send that text to the background script.

To do this, we need to pass the text, the Notion workspace ID, and the speaker's username, like so:

```
const addTextToNotionFromBackground = (newText, workspaceId, use
 chrome.runtime.sendMessage({
     newText,
     workspaceId,
     username,
 }, () => {});
};
```

Content/contexts/CallProvider.jsx



Transcription going from the React app and eventually to the database to store

Once the text is passed from the content script to the background script, it needs to be explicitly received. That looks like this:

```
async function handleMessageReceived(request, sender, sendRespon
   const {
     newText,
   } = request;

   …

   if (newText) {
      await addTranscriptionTextToNotionDoc(request, sender.tab)
   }
   ...
   sendResponse('message received');
}

/**
* Listen for messages and determine which message is sent in par
*/
chrome.runtime.onMessage.addListener(handleMessageReceived);
```

Content/contexts/CallProvider.jsx

All messages received by the background script are received by the same listener that then determines what to do next based on the message. If the message has a `newText` key, we know it's new transcription text to append to the current Notion doc.

To actually make the request we then send the text to the Daily Collab API, which handles all API requests.

```
async function appendText(id, text, workspaceId, username) {
   const shortId = id.slice(id.length - 32);
   const block = await fetch(`${DAILY_COLLAB_API_BASE_URL}/calls
      method: 'PATCH',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        text,
        isTranscribing: true,
        shortId,
        workspaceId,
        username,
      }),
    });
    const response = await handleTranscriptionResponse(block);
    return response;
}
```

Background/index.js

The Daily Collab custom API is what actually handles the request to Notion's API.

The request to the Notion API (`updateNotionPage`) uses a `PATCH` request to append block children to a Notion document.

To make this request, you need:

1. The Notion document ID, which is the last 32 characters of a Notion document URL. For example, in the following URL, `5ca9e2e91bd64762bfa969f843cc889c` is the Notion document ID.

`https://www.notion.so/example-domain/Proposal-5ca9e2e91bd64762bfa969f843cc889c`

2. To authenticate the request, which we do with Notion's public integration, described in one of our previous posts. (That is how

the `token` is created in the code block below.)

```
async function updateNotionPage(docId, content, token, username)
 const block = {
   object: "block",
   type: "paragraph",
   paragraph: {
     text: [
       {
         type: "text",
         text: {
           content,
         },
       },
     ],
   },
 };

 const doc = await fetch(
   `${process.env.NOTION_REST_URL}/blocks/${docId}/children`,
   {
     method: "PATCH",
     headers: {
       Accept: "application/json",
       "Content-Type": "application/json",
       Authorization: `Bearer ${token}`,
       "Notion-Version": "2021-05-13",
     },
     body: JSON.stringify({
       children: [block],
     }),
   }
 );

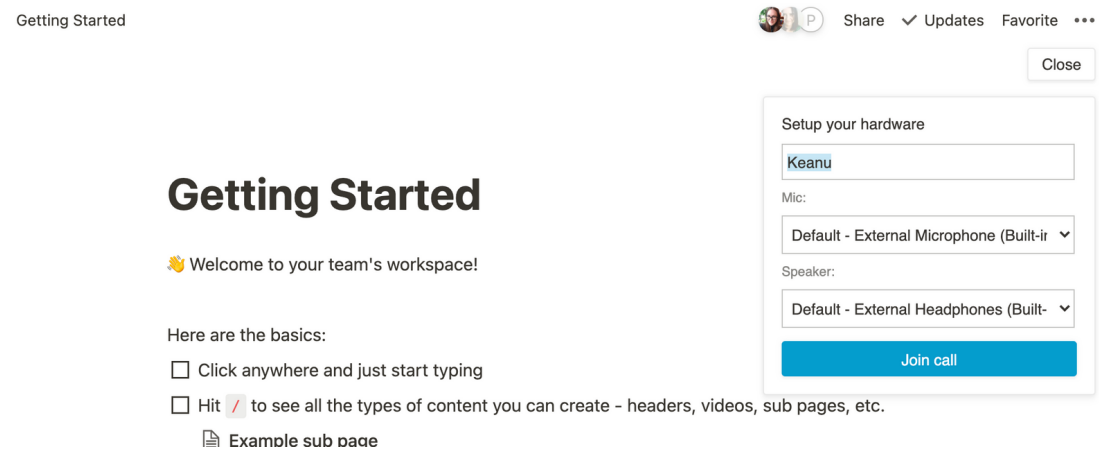
 return await doc.json();
}
```

Code example from the Daily Collab custom API (currently a private repo)

In `updateNotionPage`, we set our new block child to be a paragraph with the text content equal to the transcription text received in the client. That block is then passed in the `PATCH` request's body to be received by Notion's API.

In terms of how to format the text, you may prefer to include the call participant's name that they set in the Daily Collab UI when joining the

call. This helps the future transcription reader know who said what.



The local user can set their username before joining a call with Daily Collab

To do this, you can prepend the username to the text passed in the request's body before submitting it, like so:

```
async function updateNotionPage(docId, content, token, username)
 const block = {
   object: "block",
   type: "paragraph",
   paragraph: {
     text: [
       {
         type: "text",
         text: {
           `${username}: ${content}`,
         },
       },
     ],
   },
 };
```

Code example from the Daily Collab custom API (currently a private repo)

Once the text is sent, the `PATCH` request sent to the Notion API via the Daily Collab custom API will return a success or error response. If the request is successful, the text will have been added to the screen. If it wasn't, we'll return an error from the background script to the content script.

In the background script, we handle this like so:

```
async function addTranscriptionTextToNotionDoc(request, tab) {
  const notionId = getNotionId(tab);
  if (!notionId) return;

  const block = await appendText(
    notionId,
    request?.newText,
    request?.workspaceId,
    request?.username
  );

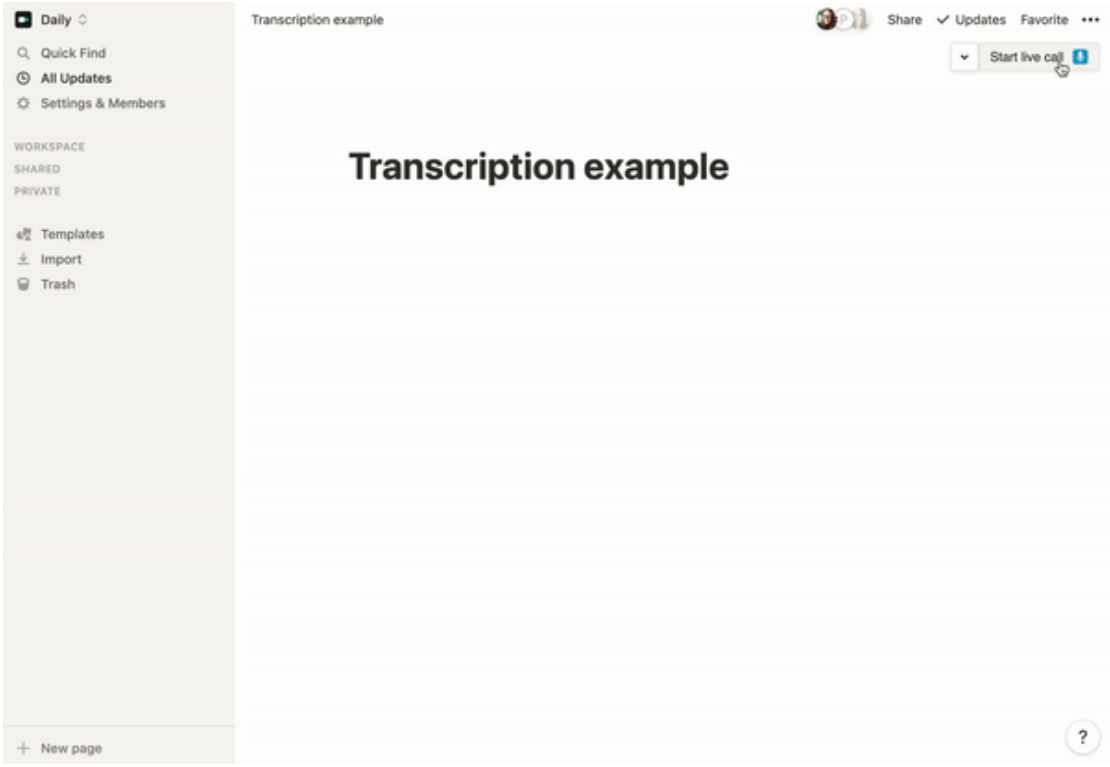
  // set error if there was an auth error
  if (block?.error) {
    const error = block.error === UNAUTHORIZED_ERROR ? block.err
    const message = {
      error,
    };

    // send error to content script
    chrome.tabs.sendMessage(tab?.id, message, noop);
  }
}
```

Background/index.js

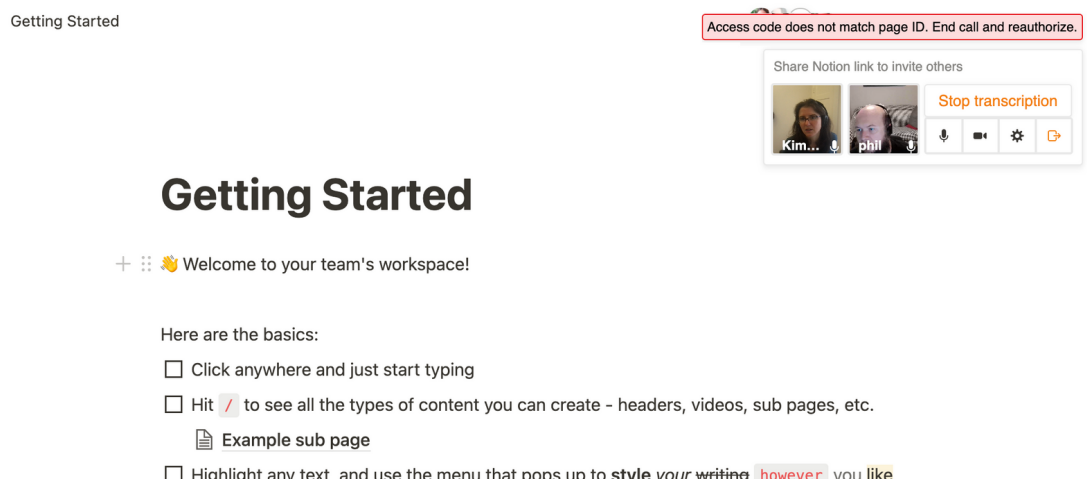
Seeing our transcription results in Notion

Once the transcription text is added, we can see the text display in the Notion document. From a user's perspective, here is what successfully appending transcription text will look like:



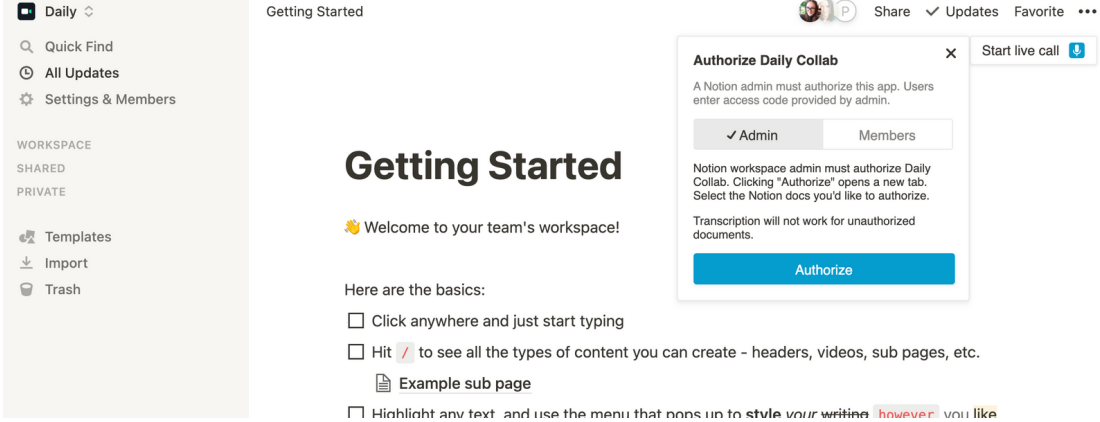
Transcription text getting added to a Notion document after joining a call and starting the transcription feature.

If the request throws an error, we do show an error message so the user knows there's an issue, as mentioned above. Transcription errors are most often due to authorization issues since updating Notion documents with Notion's API ***must*** be authenticated.



Transcription error: authentication required

To solve this issue, try reauthorizing the Notion API in Daily Collab.



Daily Collab auth form

Transcription text is taking a road trip

You may be thinking, “Wow, that’s a long trip for a little bit of text to get added to the page!” and, honestly, you’re right!

There are a few reasons we use this structure for interacting with the Notion API versus just making the API request from the content script.

Firstly, we use the Daily Collab custom API endpoints to keep all API requests related to Daily Collab in one place. It also means all application secrets (ex. API keys) related to Daily Collab can be isolated in the API, instead of needing to expose them to the client-side code, too. This is standard practice at Daily to avoid exposing application secrets unnecessarily.

Additionally, with this structure, all API requests go through the background script to the Daily Collab API to keep the background script the central “brain” of the Chrome extension. This is how we decided to organize things in this use case, but ultimately it’s up to you!

Wrapping up

In the end, getting a Daily video call and the Notion API to work together was as simple as making a Notion API request based on a call participant event (`app-message`).

This feature could be expanded even further to do something fun like make a Notion API request based on a voice command. For example,

saying, “Get all users”, could trigger a GET request to retrieve all Notion users if the transcription text matched that string.

Let us know what other feature ideas you have and how we can help you build your own Daily apps!

Get updates from Daily

Don’t miss a thing! Subscribe to our monthly newsletter and we’ll keep you up-to-date with the latest product releases, tutorials, and more from Daily.

Email*

Country*

Join peerConnection - our video/WebRTC developer community.

By clicking submit below, you consent to allow Daily.co to store and process the personal information submitted above to provide you the content requested.

Submit

More in Collab Chrome Extension

Build your own "prejoin" call UI in a custom Daily video chat app

9 Sep 2021 - 9 min read

Tutorial: Build a custom Daily video call widget and embed it into any Notion page

26 Aug 2021 - 11 min read

Authorizing Daily's Chrome extension transcription feature with Notion's API

19 Aug 2021 - 8 min read

See all 5 posts →



Code & Tutorials

Integrate Daily video chat into your Bubble app in less than 20 minutes

Use the new Daily plugin for Bubble to add a custom video widget to your next no-code Bubble app



Lola Ojabowale
7 Sep 2021 – 14 min read



Code & Tutorials

Build a video chat app with Vue and Daily Prebuilt

Embed Daily Prebuilt and add custom video call controls to any Vue app



Jess Mitchell
30 Aug 2021 – 12 min read