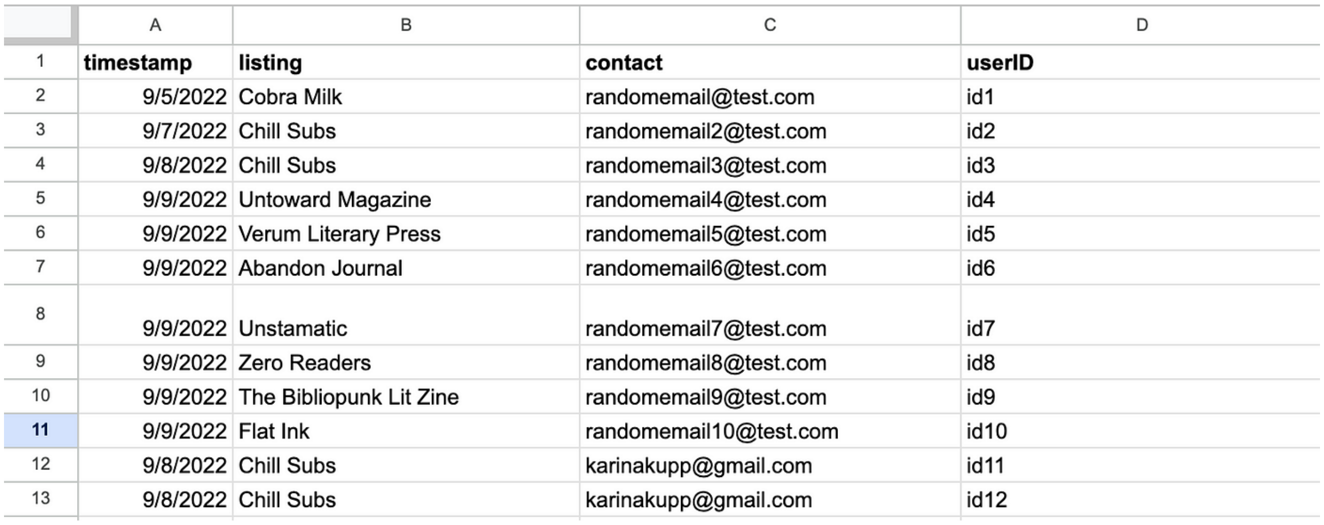
Using Google Sheets as another database in your Next.js app

levelup.gitconnected.com/using-google-sheets-as-another-database-in-your-next-js-app-ab1b2c6d8f1a

Karina Kupp

January 30, 2024



	A	B	C	D
1	**timestamp**	**listing**	**contact**	**userID**
2	9/5/2022	Cobra Milk	randomemail@test.com	id1
3	9/7/2022	Chill Subs	randomemail2@test.com	id2
4	9/8/2022	Chill Subs	randomemail3@test.com	id3
5	9/9/2022	Untoward Magazine	randomemail4@test.com	id4
6	9/9/2022	Verum Literary Press	randomemail5@test.com	id5
7	9/9/2022	Abandon Journal	randomemail6@test.com	id6
8	9/9/2022	Unstamatic	randomemail7@test.com	id7
9	9/9/2022	Zero Readers	randomemail8@test.com	id8
10	9/9/2022	The Bibliopunk Lit Zine	randomemail9@test.com	id9
11	9/9/2022	Flat Ink	randomemail10@test.com	id10
12	9/8/2022	Chill Subs	karinakupp@gmail.com	id11
13	9/8/2022	Chill Subs	karinakupp@gmail.com	id12

Hey there!

So my company (Chill Subs) lists over 4000 opportunities for writers. And managing all this isn't exactly a walk in the park.

When I started the website, I used a JSON database for over 4 months (which worked perfectly fine, highly recommend), but when I added the accounts functionality, I moved everything to MongoDB.

Still, we constantly work with spreadsheets because they are what most of my non-technical team understands, and being the only developer, I'm not able to make our internal admin panel convenient enough for them to do everything they want.

Here are some of the things we do through spreadsheets:

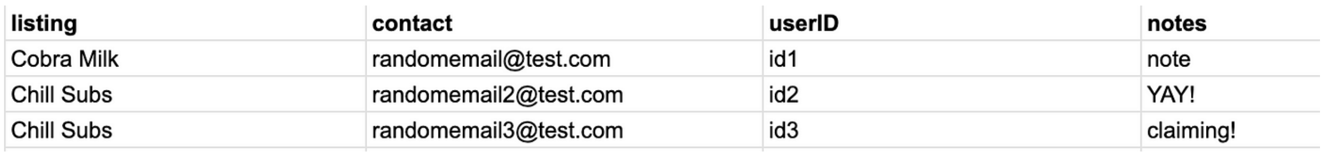
- collecting data and importing it to MongoDB
- reading directly from a spreadsheet without import
- our feedback and bug reports arrive in a spreadsheet and we manage it there
- saving our mailing list
- tracking customers purchasing something through Stripe/Shopify
- doing custom exports to create lists of opportunities for our newsletter
- and MORE.

And for all this, I had to master the art of working with Google Sheets inside my Next.js app.

So, let’s have a look!

Setting up a Google Sheets API

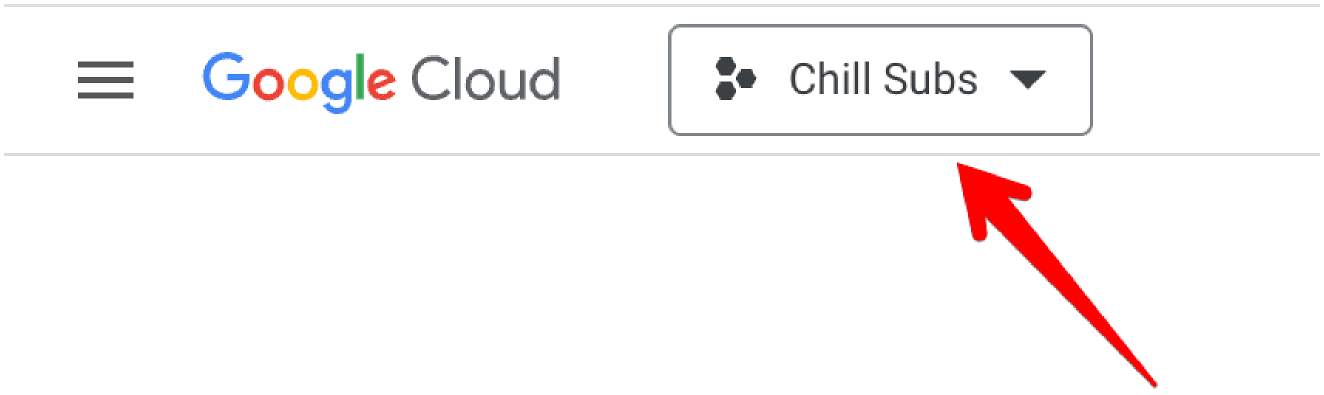
First, let’s create a random spreadsheet. In my case, I have a spreadsheet of editors claiming access to their listing on our website.



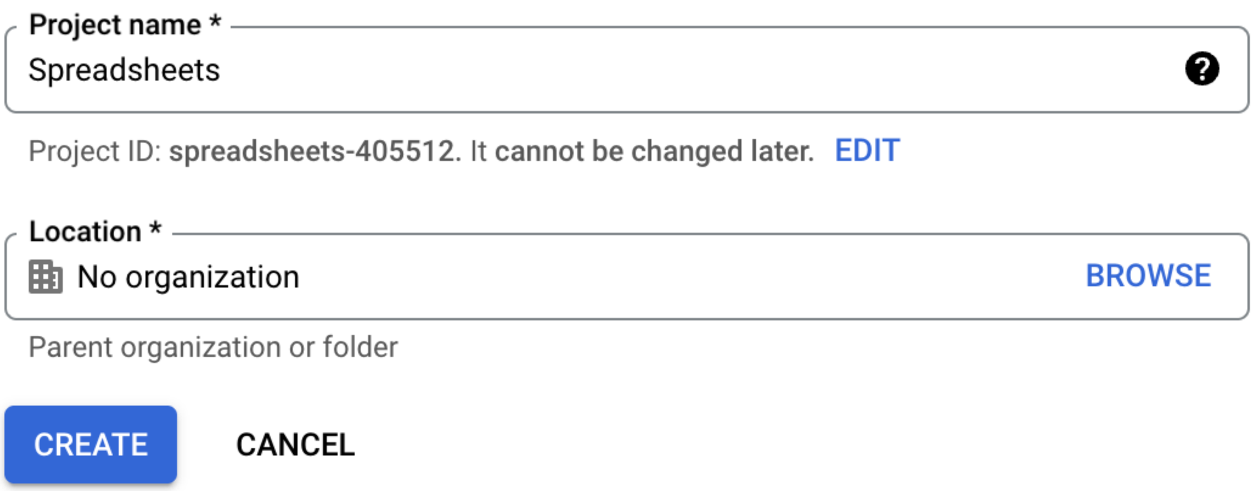
listing	contact	userID	notes
Cobra Milk	randomemail@test.com	id1	note
Chill Subs	randomemail2@test.com	id2	YAY!
Chill Subs	randomemail3@test.com	id3	claiming!

To connect to this spreadsheet from our Next.js app, let’s go to Google Cloud Platform.

There, you’ll see a project select dropdown.



If you don’t have a project yet, click on this dropdown and press “New project” in the top right corner. It will ask you for a name.



Project name *
Spreadsheets

Project ID: spreadsheets-405512. It cannot be changed later. EDIT

Location *
No organization BROWSE

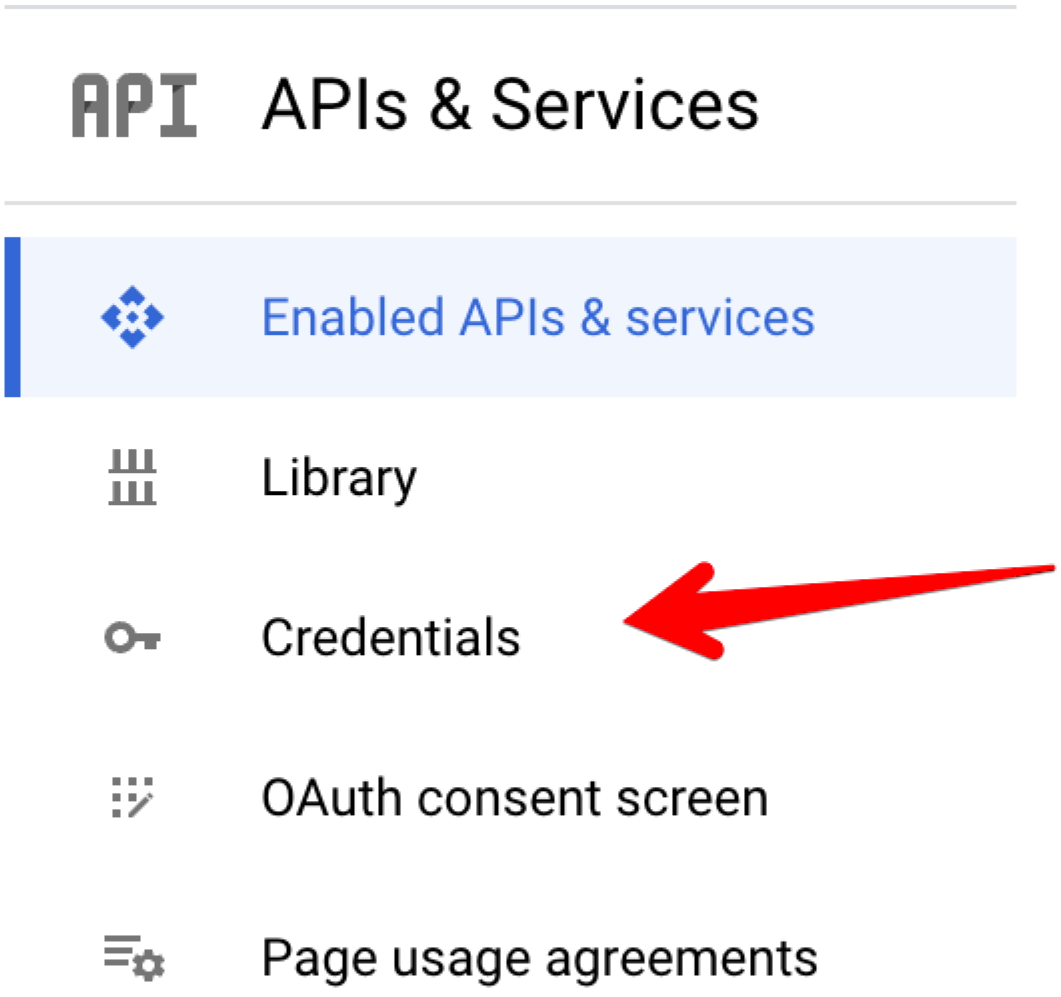
Parent organization or folder

CREATE CANCEL

Then, when Google is done creating your project, find the search bar and search for “Google Sheets API”.

When you find it, click “Enable”. This will take you to the API configuration page.

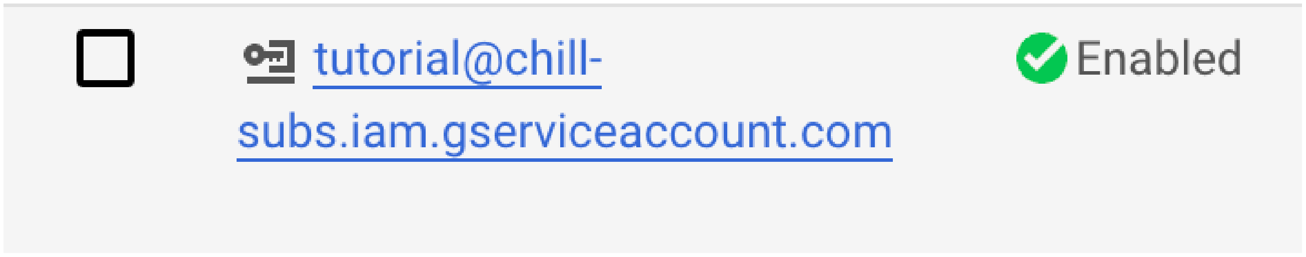
In the left sidebar, find “Credentials”.



There, under “OAuth 2.0 Client IDs” click “Manage service accounts”. And then, “Create service account”. We will use this account to talk to sheets inside our Next.js app.

Name it whatever you want and press “Create and continue”. Skip all the other steps.

Congrats, interacting with Google Console is almost over! Just copy the weird email.



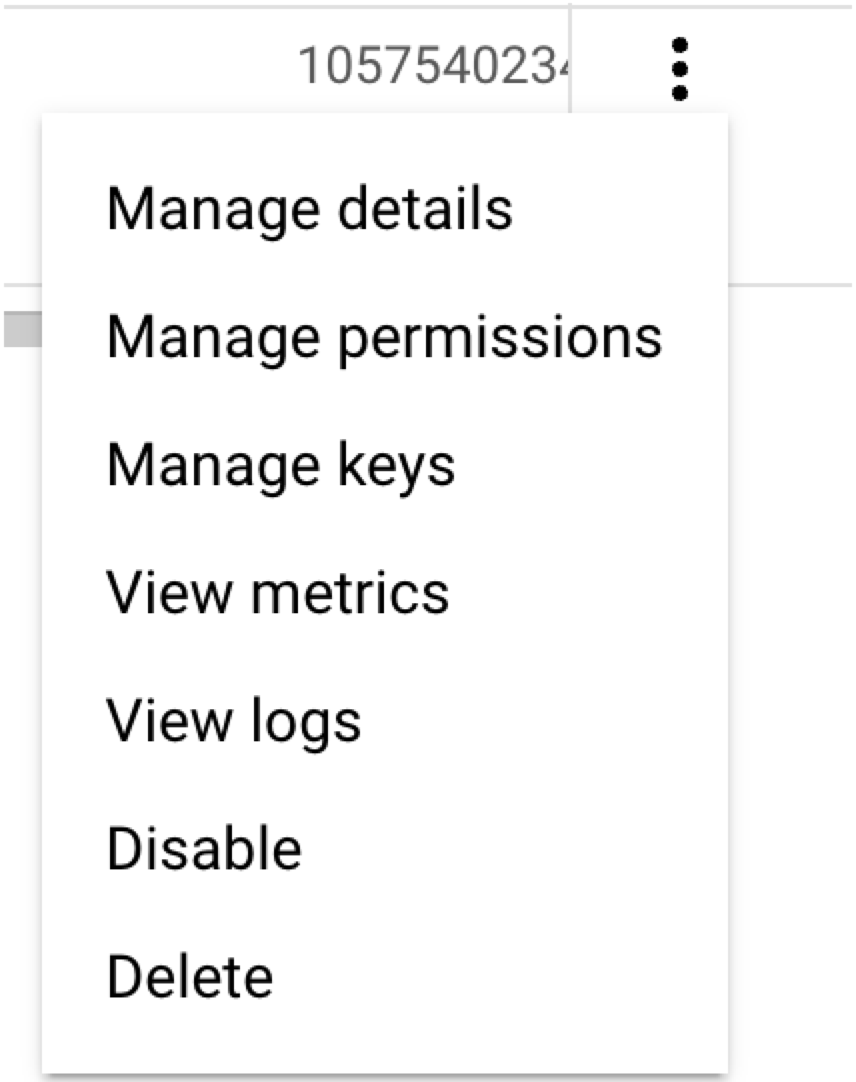
tutorial@chill-subs.iam.gserviceaccount.com

Enabled

After this, return to your spreadsheet, click “Share”, and paste that email there with the Editor role.

Boom!

Ok, the prep is almost done. Now return to Google Console, scroll to the right of the service account table, click on the Actions menu and select “Manage keys”.



There, click “Add key” -> “Create new key”, keep JSON selected, and press “Create”. This will download the key to your computer.

We can then move this to our project folder as `spreadsheet-keys.json` (which you can add to your `.gitignore`)

Connecting to your spreadsheet

First, install the `googleapis` package.

Then go to your `api` folder and create a file like `sheets.js`.

Now here’s my file (note that I’m using the `pages` router)

```
import { google } from"googleapis";
import keys from"../../spreadsheet-keys.json";



exportdefaultfunctionhandler(req, res) {
try {
const auth = await google.auth.getClient({
projectId: keys.project_id,
credentials: {
type: "service_account",
private_key: keys.private_key,
client_email: keys.client_email,
client_id: keys.client_id,
token_url: keys.token_uri,
universe_domain: "googleapis.com",
      },
scopes: [
"https://www.googleapis.com/auth/spreadsheets",
      ],
    });



const sheets = google.sheets({ version: "v4", auth });



// copy your spreadshset id here
// and update the range based on the sheet name and colums used
const data = await sheets.spreadsheets.values.get({
spreadsheetId: "1OKZHO3lc9bqfj11gbZEOOq7QbQnx_vFmgzswgJioeQN",
range: "Sheet1!A:D",
    });



// here we'll work with the data

          res.();  }  (e) {    .(e)     res.().({ : , :  });  }}
```

Working with the spreadsheet data

Reading from the spreadsheet

Let’s say, we have to get all the access requests we haven’t processed yet. Here we’re looking for rows where the status column (which has index 5) is either empty or says “Todo”.

```
const data = await sheets.spreadsheets.values.get({
spreadsheetId: "1OKZHO3lc9bqfj11gbZEOOq7QbQnx_vFmgzswgJioeQN",
range: "Sheet1!A:D",
});
```

```
const unprocessedRequests = data.data.values.filter(() => row[5] === 'Todo' ||
!row[5]);

 requests = unprocessedRequests.( {   {    : row[],    : row[],    : row[],    :
row[],  }}))
```

Modifying the spreadsheet

After giving editors access, I wanted to update the status column in the spreadsheet. In the range property, `“F”` is the letter of the column you want to update, and `${request.row}` is the row number.

```
 sheets...({  : ,  : ,  : ,  : {    : [[]],  }});
```

What if you want to create a new entry in the spreadsheet?

For example, we have a mailing list spreadsheet. We use Substack which doesn’t have an API, so when users agree to the newsletter, we collect emails in a spreadsheet and then import them to Substack.

Here’s how you can do it:

```
const spreadsheetId = "1_O9NGIGNAm0Nk82ABok8MqZz0Z3EsECX-SKJPB60PE9";
const sheetName = "some name";
```

```
let data = Object.keys(subscriber).map(() => subscriber[key]);
data.push(newDate().toISOString());

  sheets...({  spreadsheetId,  : ,  : ,  : {    : [data]  }}, {});
```

Exporting data from your internal database to CSV

My colleagues asked me for a page in the admin panel where they would be able to export a few common lists like “Magazines added last month”, “All our contests” or “User info”.

At the time I hadn’t worked closely with `googleapis` yet, so I did this using `csv-writer`.

This script generates a CSV file and sends it to the frontend.

```
import { NextApiRequest, NextApiResponse } from'next';
import path from'path';
const createCsvWriter = require('csv-writer').createObjectCsvWriter;



const magazineExample = {
'Name': 'Magazine name',
'Link': 'https://link.com',
'Website': 'https://www.instagram.com/psm.magazine/',
};



exportdefaultasync (req: NextApiRequest, res: NextApiResponse) => {
try {
const filePath = path.join(process.cwd(), 'public', 'spreadsheets', 'all_mags.csv');



const csvWriter = createCsvWriter({
path: filePath,
header: Object.keys(magazineExample).map( => ({ id: key, title: key })),
    });



await csvWriter.writeRecords(spreadsheetData);

    res.().({ :  });  }  (error) {    .(, error);    res.().({ :  });  }};
```

Then in the frontend:

```
constdownloadSpreadsheet = async (spreadsheet) => {
try {
setLoading(spreadsheet);
const response = awaitfetch(`/api/spreadsheets/`);
const { filePath } = await response.json();

                           a = .();    a.. = ;    ..(a);    a. = filePath;    a. = ;
a.();    ..(a);    ();  }  (error) {    .(, error);    ();      } };
```

Here though you would have to upload the spreadsheet to Google Drive manually.

`googleapis` has this method that creates a spreadsheet with a specified title straight in your root folder in Drive:

```
const resource = {
properties: {
    title,
  },
};

   spreadsheet =  service..({  resource,  : ,});.();
```

You can do some extra trickery to create it in a specific folder, but I haven't looked into that myself. Probably time to check this out!

Alright, I think this is pretty much it! Let me know if you have any questions.

And if you liked the article, give me those sweet claps. I'm a silly human, I depend on external validation. I need you.

You can also follow me here because I'll be sharing more about my journey of building Chill Subs and making it a profitable project :)