# Using ChatGPT to Generate Fake Data in Your Spreadsheet with Google Apps Script

### ChatGPT is pretty much a universal API: one endpoint to get any data.

ChatGPT is all the craze nowadays, and unless you’ve been living under a rock, you’ve seen all kinds of content on this topic, so I’m not going to explain what it is. One of its greatest advantages is its ease of use, and it’s perfectly usable in Google Apps Script. So here is one use case that we can implement: generating fake data.

Fake data is useful for testing things out. Sometimes real data is too sensitive, sometimes you don’t have access to real data before production, but you need to start working with something. Here is where fake data comes in.

We used to have a great library for fake data called [Faker.js](https://www.npmjs.com/package/faker), but at one point Marak, the author, got so frustrated that he could not monetize the solution that he [pushed an update that broke everything](https://shay-ag13200.medium.com/what-really-happened-to-faker-js-cd3b464beb7), which led to him being blocked onGitHub and NPM, so the library is no longer maintained. ChatGPT, however, is not going anywhere.

Take note that ChatGPT is a paid service. However at the time of writing when you register you get a free credit of 18 USD for 3 months, which means you’ll be able to follow along this tutorial if you (hopefully) decide to do so.

As usual, the full source code can be found at [this GitHub repository](http://github.com).

## Create an Account and Login into Open AI

Open AI is the company behind ChatGPT. Just [go to their website](https://openai.com/), create an account and then you can [access the chat](https://chat.openai.com/chat).

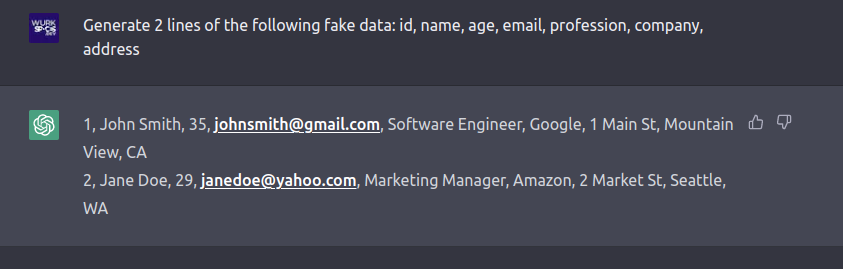
## Defining and Fine-Tuning the Prompt

The first thing to do is to decide what exactly we want as a result. Let’s say we want to generate data for a spreadsheet that contains the following columns:

* id
* name
* age
* email
* profession
* company
* address

Let’s try this prompt:

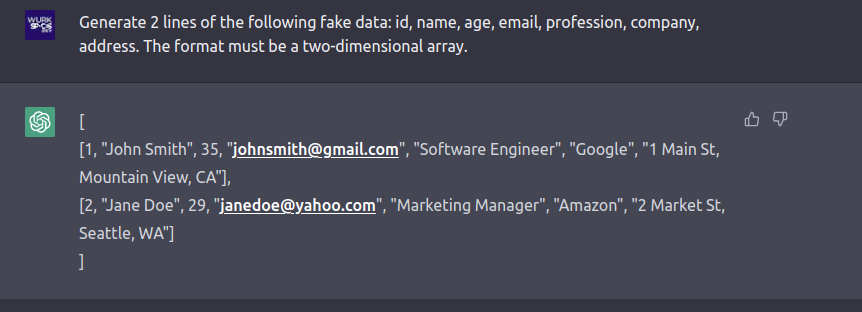
*Generate 2 lines of the following fake data: id, name, age, email, profession, company, address*



OK, that’s not bad, we have the data in comma-separated format. We could work with that, but let’s try and make ChatGPT do some more heavy lifting. Let’s make it return a two-dimensional array that we can then directly paste in a Google spreadsheet.

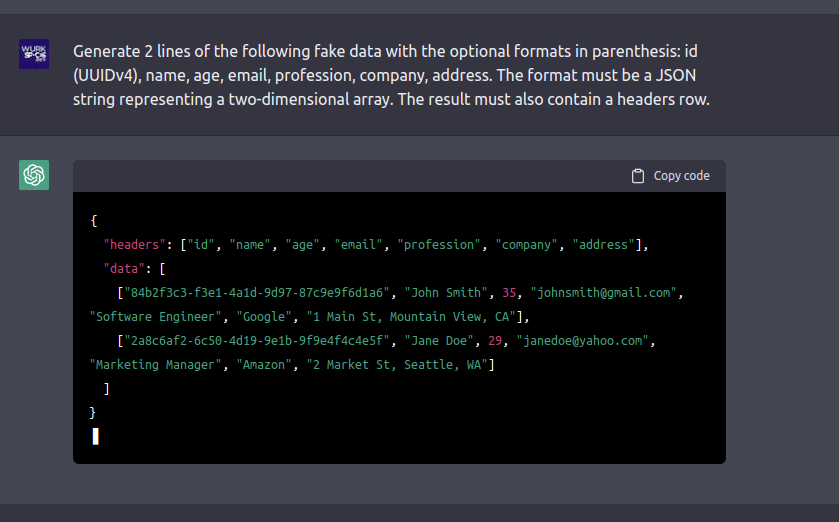
Let try a more complex prompt:

*Generate 2 lines of the following fake data: id, name, age, email, profession, company, address. The format must be a two-dimensional array.*



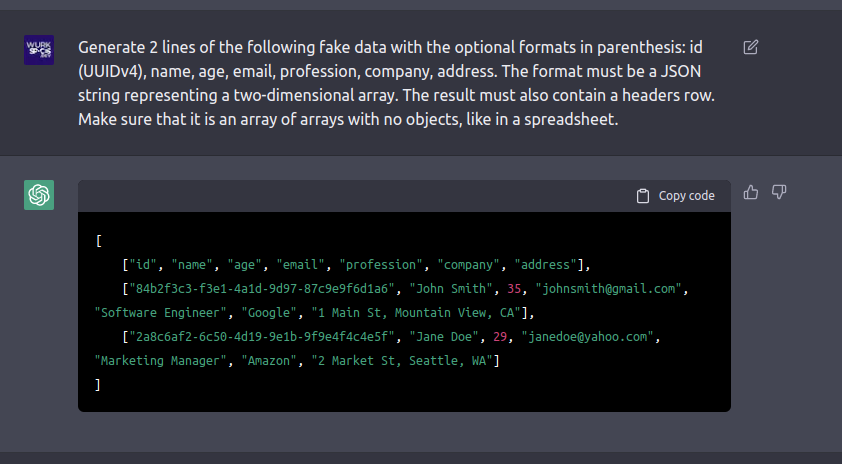
OK, wow, this is actually really cool. Now let's add more requirements. Let’s make sure we have a headers row, that our **id** is in [UUIDv4](https://fr.wikipedia.org/wiki/Universally_unique_identifier) format and the result is a JSON string:

*Generate 2 lines of the following fake data with the optional formats in parenthesis: id (UUIDv4), name, age, email, profession, company, address. The format must be a JSON string representing a two-dimensional array. The result must also contain a headers row.*



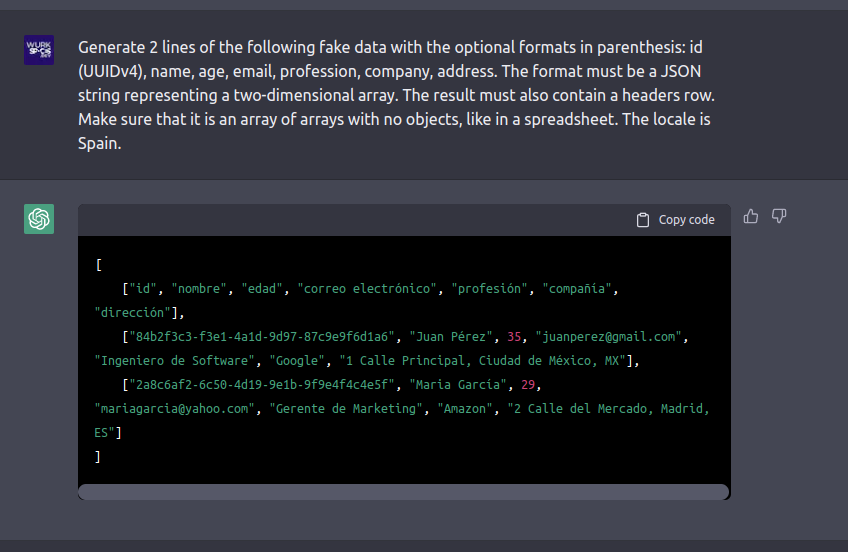
OK wow, it got a little creative there. This is a perfectly valid JSON, it does contain headers and the **id**s are in the required UUIDv4 format, but let’s make sure we just get the two-dimensional array again so that we can just use it as-is without any manipulation. Here’s the next version:

*Generate 2 lines of the following fake data with the optional formats in parenthesis: id (UUIDv4), name, age, email, profession, company, address. The format must be a JSON string representing a two-dimensional array. The result must also contain a headers row. Make sure that it is an array of arrays with no objects, like in a spreadsheet.*



OK, this is pretty amazing. However, it seems to be only generating data for the United States. What if we wanted to specify another locale? Let’s try this:

*Generate 2 lines of the following fake data with the optional formats in parenthesis: id (UUIDv4), name, age, email, profession, company, address. The format must be a JSON string representing a two-dimensional array. The result must also contain a headers row. Make sure that it is an array of arrays with no objects, like in a spreadsheet. The locale is Spain.*

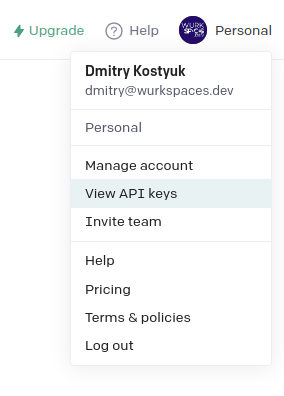


OK, this is really cool, we can work with that!

The next step is to take this prompt and link it with the ChatGPT API in Google Apps Script. To do this, the first thing we will need is a developer key.

## Getting a Developer Key

Go back to you account on Open AI, click on your avatar in the top right corner and select “View API keys”



Generate a new key and copy it. We will be using it for **Bearer** authentication. If you want to know more about how APIs and the Bearer authentication works, check out my previous article [How to Build an API Wrapper in Google Apps Script](https://medium.com/geekculture/how-to-build-an-api-wrapper-in-google-apps-script-904be20f0dd7).

Not that you’ve got your key, create a file that will contain your environment variables. I like to call mine **ENV.js** and create an object called **ENV** that contains keys, file ids and other globals. Put the key there:

**const ENV = {**

**openAIKey: 'sk-xI1pAyx6…',**

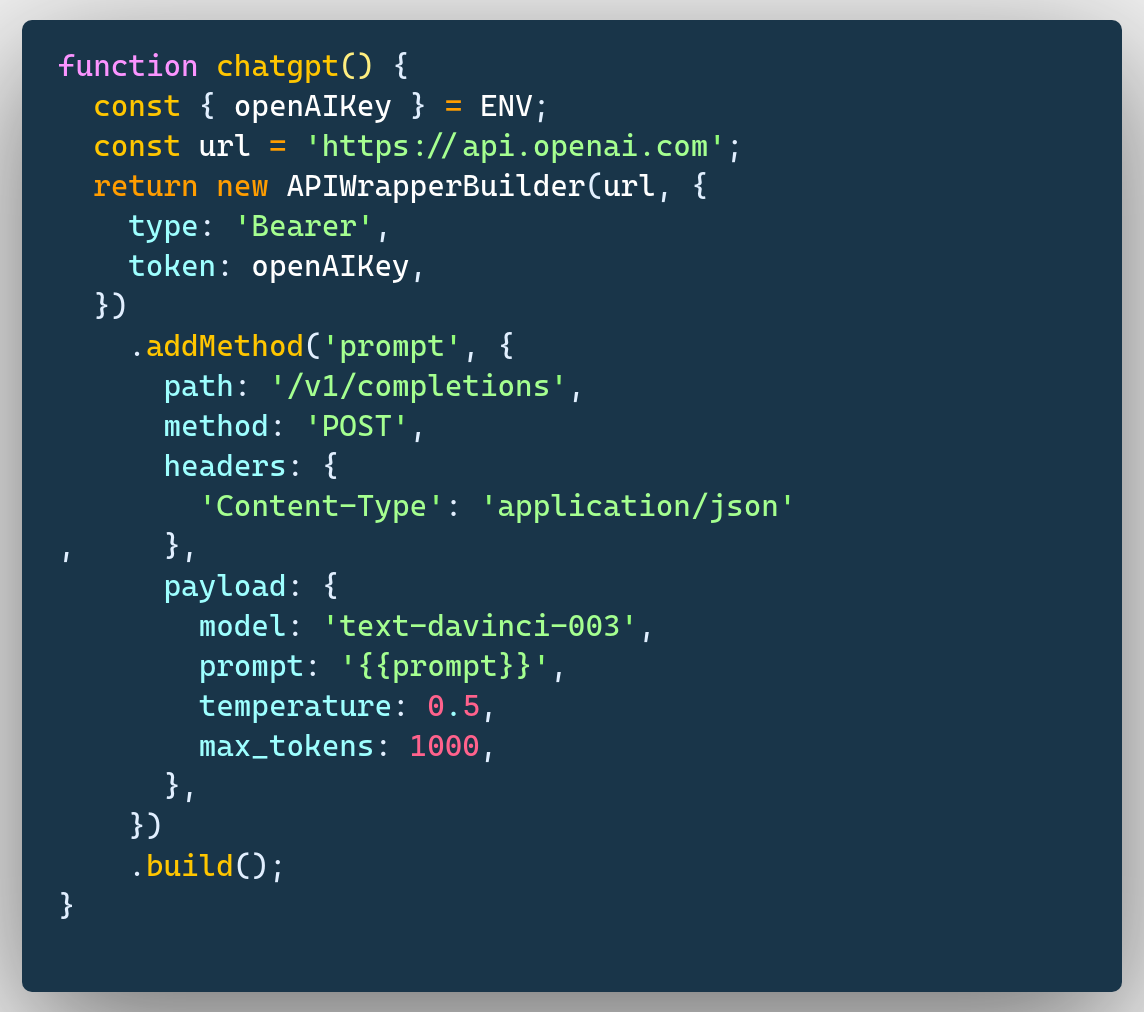
**};**

Now we are ready to start writing some code.

## Querying the API

The utility used to query APIs in Google Apps Script is [**UrlFetchApp**](https://developers.google.com/apps-script/reference/url-fetch/url-fetch-app). It’s a pretty straight forward fetch utility and you are free to use it in its vanilla version. Here, I will be using my library [**API Wrapper**](https://github.com/WildH0g/gas-api-wrapper) that allows you to wrap any API and handles authentication easily. As mentioned above, you can also read my [previous article](https://medium.com/geekculture/how-to-build-an-api-wrapper-in-google-apps-script-904be20f0dd7) about it.

To get started, copy the **APIWrapper.js** file [from the repo](https://github.com/WildH0g/gas-api-wrapper) into your project. Then create the following **chatgpt()** function:



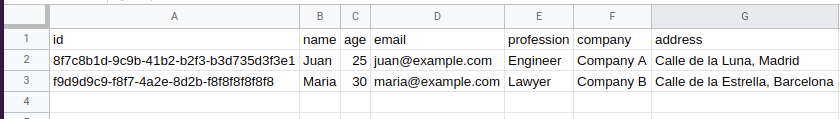
So there are a couple of things going on here. **ENV.openAIKey** is where our API key is stored. We create a wrapper by initiating an instance of **APIWrapperBuilder** and we tell it to connect to the Open AI URL using our token and **Bearer** authentication. We then define a method called **prompt** that connects to the **/v1/completions** end point via a POST request.

The payload contains a few interesting entries. The **model** is **text-davinci-003**, which is the most advanced one Open AI has. The **temperature** is how creative you want ChatGPT to be, 0 being very repetitive and 1 pretty wild; I settled at 0.5 for this test. **max\_tokens** is how much data you want to use, as this will impact the pricing. And the **prompt** is our actual prompt that we will be sending. Notice the moustache notation in the prompt value, this allows to pass the prompt as an argument to the **prompt()** method. Refer to the documentation if you feel like you need more details on that.

Let’s try and run a test with our prompt from earlier.



Simply execute the **main** function from the online IDE. This will paste the following result into the spreadsheet:

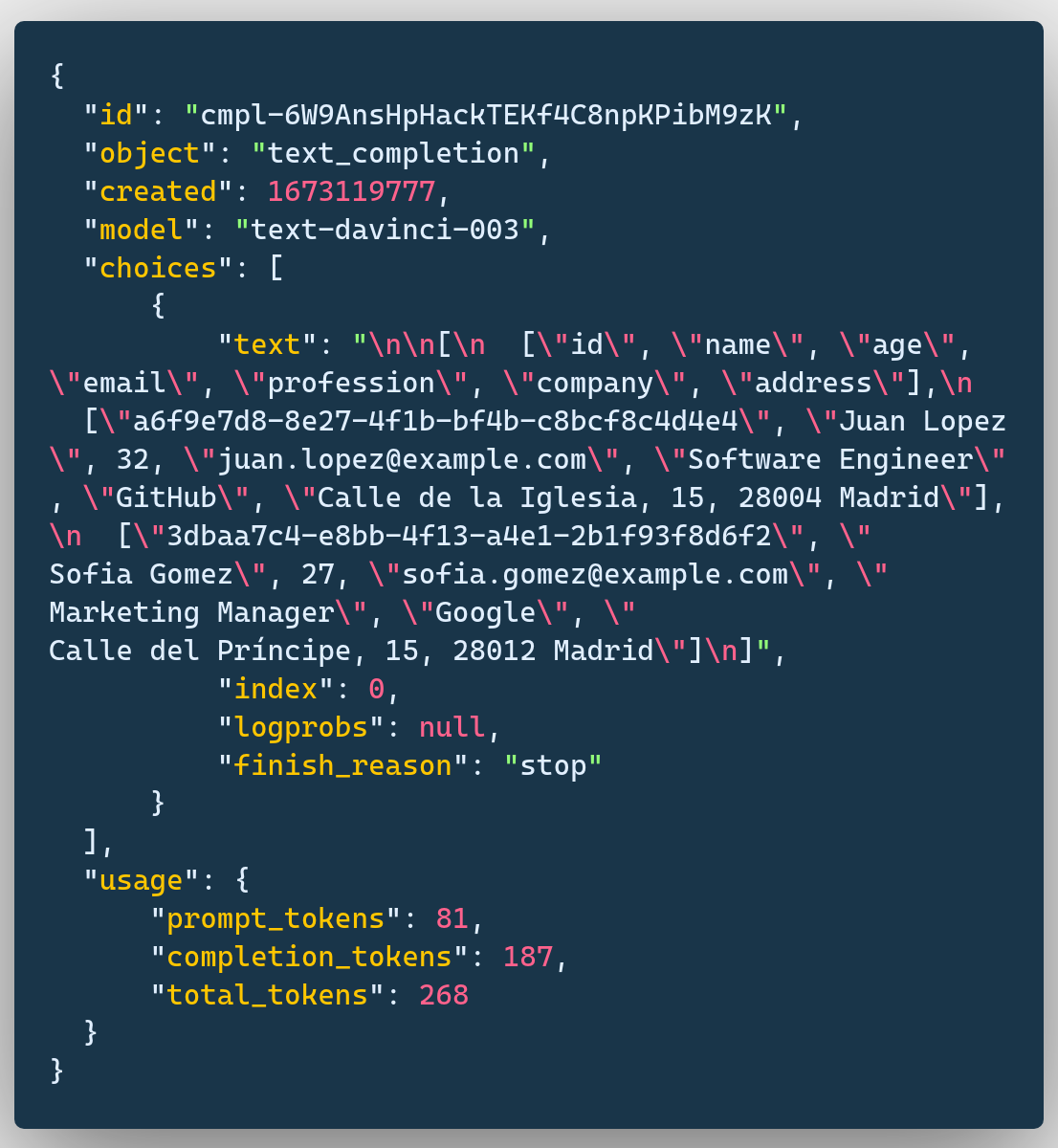


OK, this is what we want, that’s great.

This syntax, however, may look a little weird:

const ar = JSON.parse(JSON.parse(res).choices[0].text);

This is because the ChatGPT returns a JSON string in the following format:



Because it’s a JSON *string* we first need to parse it to an object with the inner **JSON.parse(res)**, then we access the first element in its **choices** array and grab the **text** property. However, the **text** property has been doubly stringified, hence we need to **JSON.parse** it again.

Now that the back-end is working, let’s build a front end that will allow the user to generate any data they need.

## Building the UI

Let’s go back to our prompt and think which values you want to be dynamic. We want to be able to define:

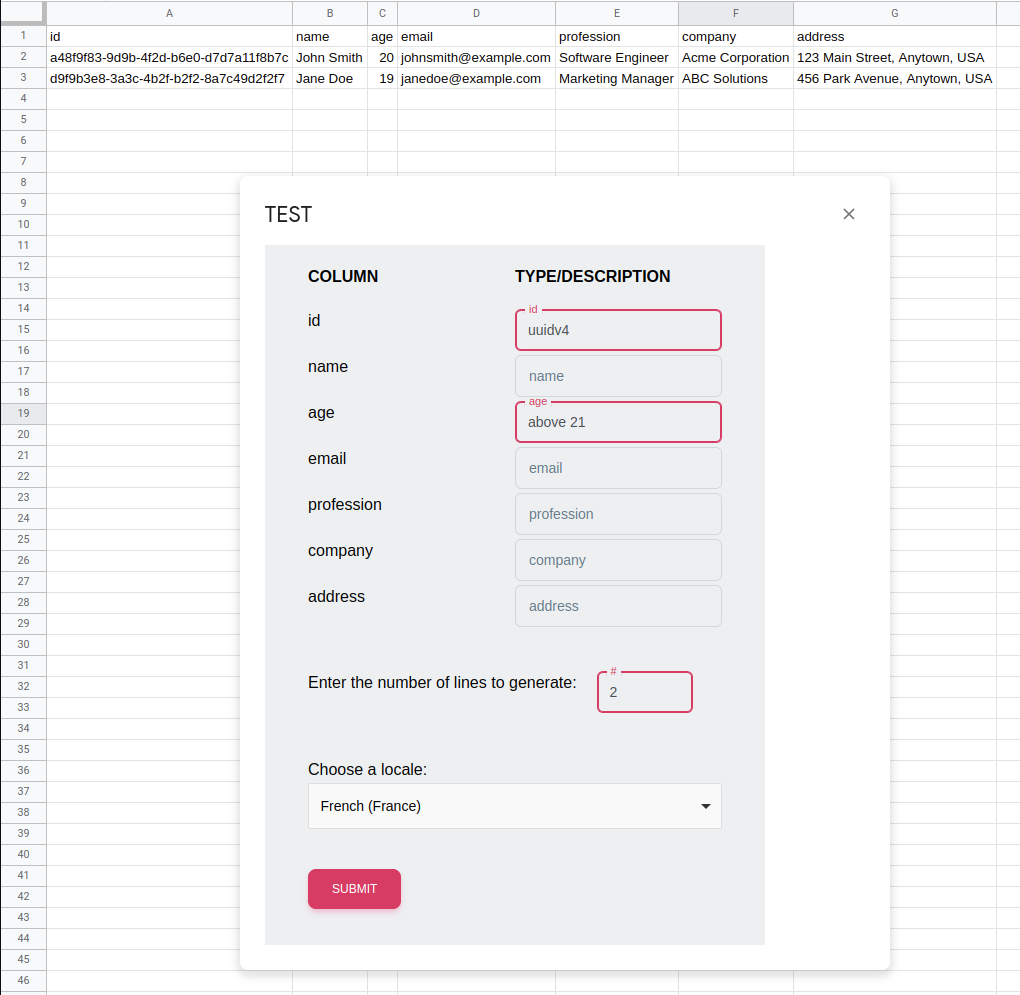
* the column names and their optional formats
* the number of rows in our result set
* the locale

Let’s start with the column names and their formats.

### Column Names and Formats

Previously we asked ChatGPT for certain columns and requiredit to include the header row. Let’s revisit that. Let's assume that the first row in the active sheet already contains predefined headers and we will simply ask the user to specify the formats and descriptions.

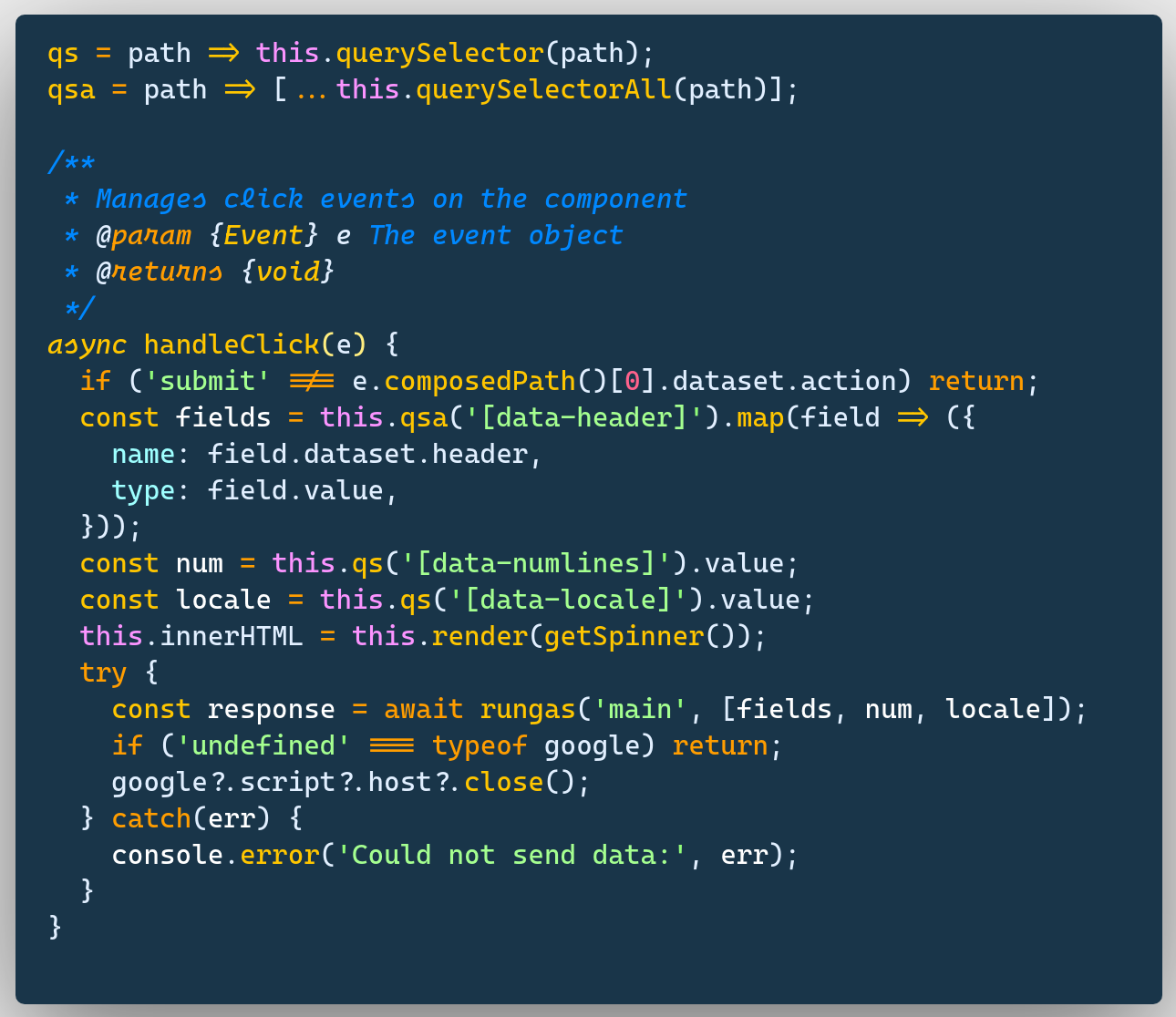
This is the UI we are going for:



As I said, we have our headers in the first row and when we open the modal, it lists them automatically. For every header we are able to specify a type and/or a description. We can also specify how many lines we want generated and which locale to use.

Building the UI per se is outside the scope of this article (by the way, do let me know if you are interested in this topic specifically). You can always check the [source code](http://github.com) if you are interested. For information, it was built with [native web components](https://developer.mozilla.org/en-US/docs/Web/Web_Components), [Tailwind CSS](https://tailwindcss.com/) and [Material Tailwind](https://www.material-tailwind.com/) frameworks, bundled with [Vite](https://vitejs.dev/) and [vite-plugin-singlefile](https://www.npmjs.com/package/vite-plugin-singlefile).

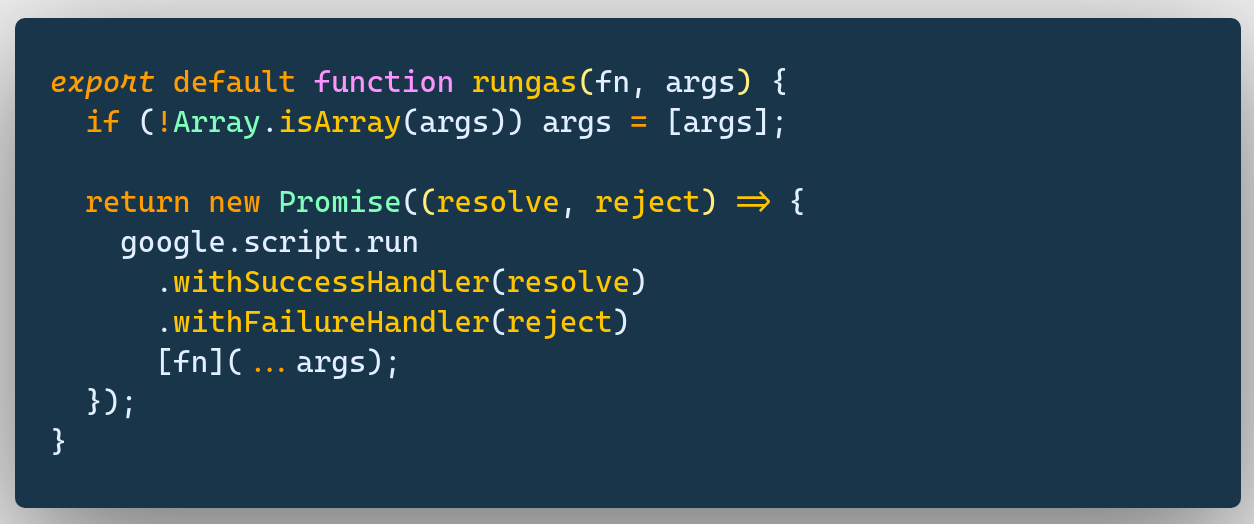
However, we do need to go over the functions that read user input and send it to Apps Script.



The above methods are part of a web component, which, in its turn is a class, this is why you don’t see the **function** keyword.

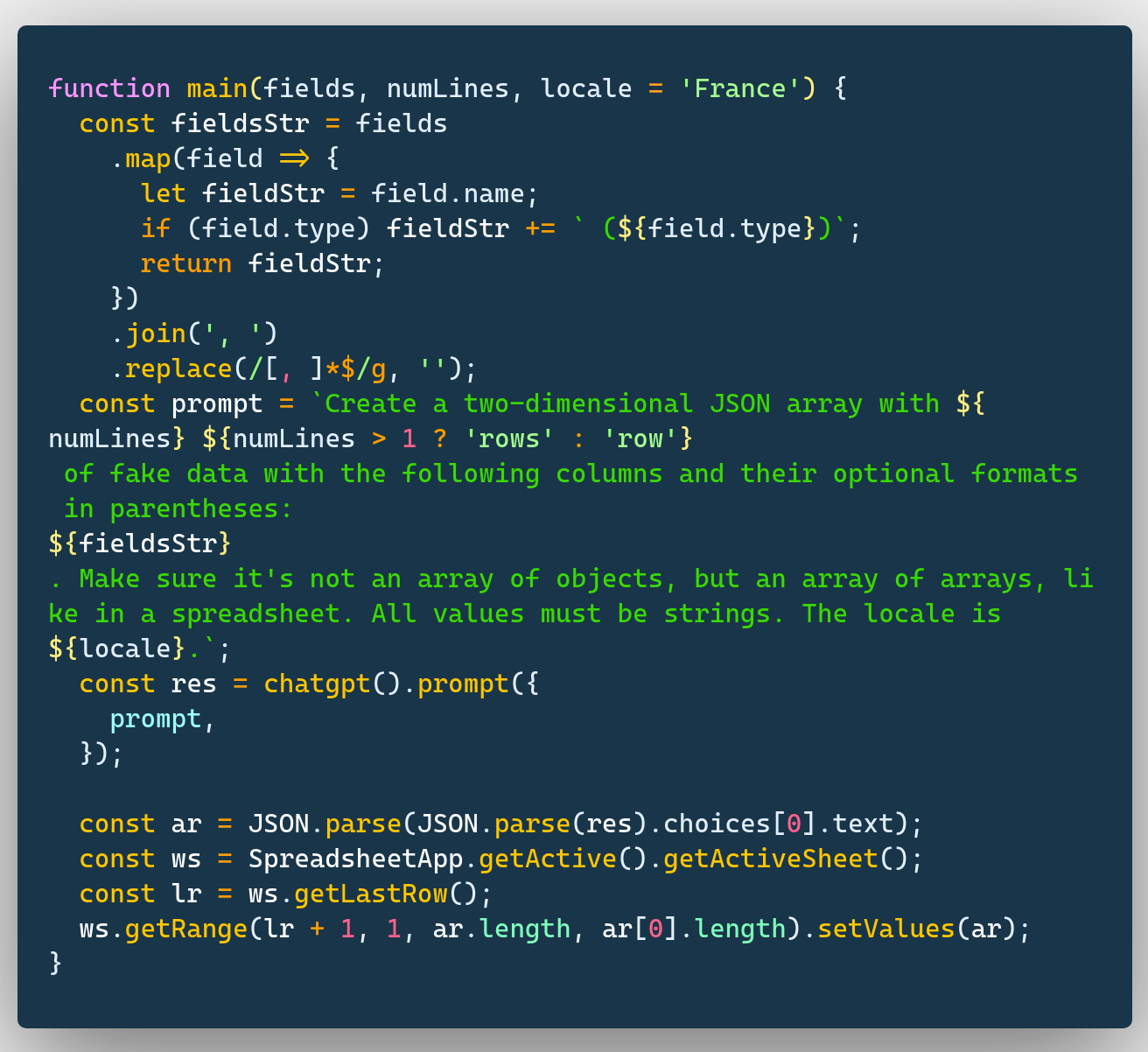
What **handleClick** does is when it’s triggered by a **click** event, it fetches field names and types, the number of lines to generate and the locale from the modal dialog. Then it calls the **rungas** function with two arguments, **main** and the inputs we’ve just listed. As you may have guessed, **main** corresponds to the name of our Google Apps Script function.

Let’s now look at the **rungas** function:



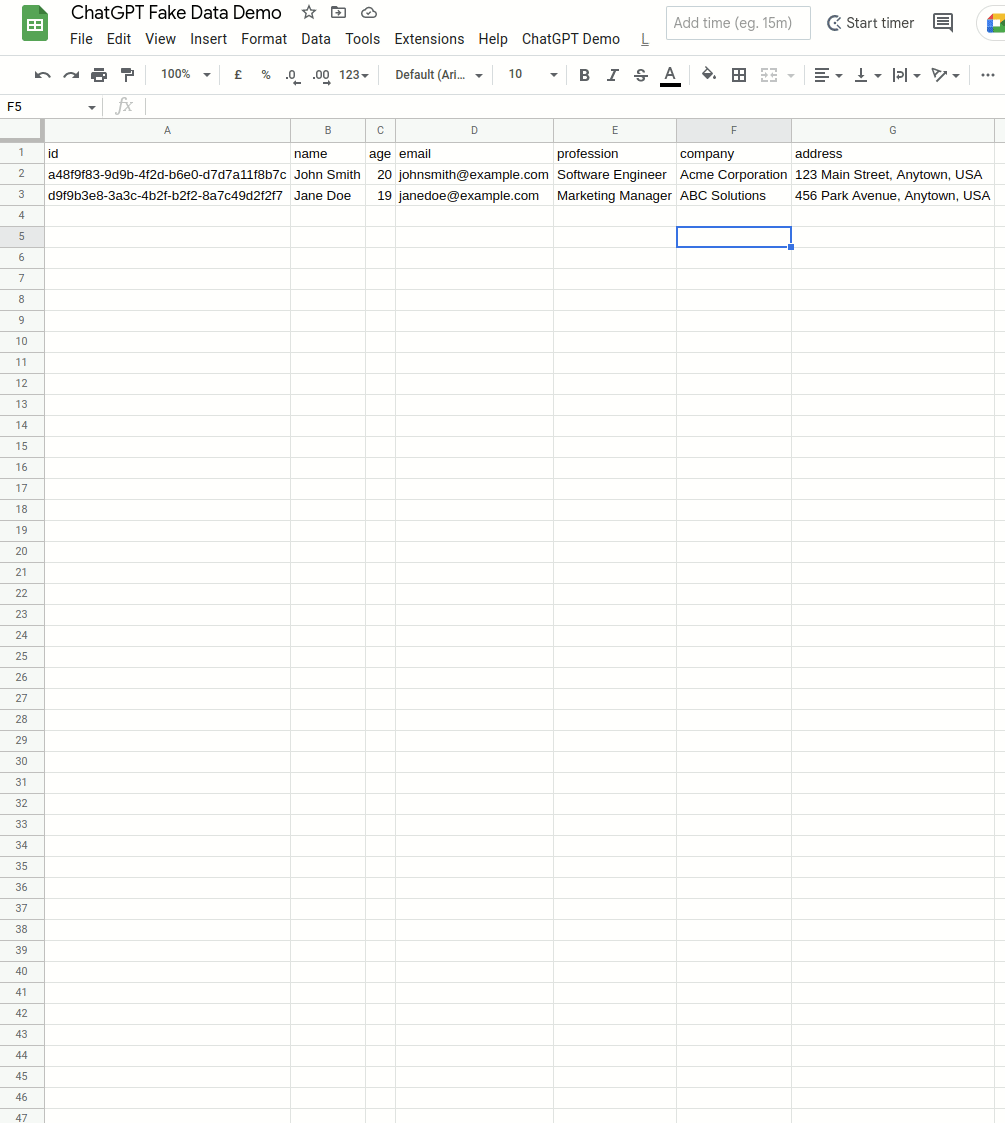
It simply wraps our usual **google.script.run** command in a promise, so that we are able to use native JavaScript syntax. I always recommend doing this as it makes the code much cleaner. As it’s probably clear from the syntax, when we execute **await rungas('main', [fields, num, locale])** we are in fact calling **main(fields, num, locale)** from the Apps Script side.

This means we have to refactor our **main** function a little bit:



The main function now takes the three arguments we pass and generates a custom prompt. The headers are parsed into a string with comma-separated values then integrated into the prompt. The number of lines and the locale are integrated as they are. Then we send the prompt to the ChatGPT API and append the result to the end of the file.

Let’s see the whole process in action:



Great, we have a fully functional solution now!

## What Else Can It Do?

We just scratched the surface in this article. You now have all the mechanics that will allow you to build your own ChatGPT integrations. You can build custom formulas, prompts or anything that comes to your mind. And you don’t have to limit yourself to sheets. You can have it write emails in Gmail or draft agreements in Google Docs, or you can even just connect to the prompt in Google Chat so that you don’t have to switch environments when working in Google Workspace. ChatGPT is like a universal API that can return anything and you can talk to it in a human language.

## So is ChatGPT Going to Replace Developers?

I have to give my two cents on this 🙂 The short answer is no. It will, however, change the way we work. This tool, as you can see, was able to do a lot of heavy lifting for us: it fetched the data, it formatted the data and we could customise the prompt for better results. We could have gone even further and asked it to write some or all of the source code for this app. So yes, a lot of mundane, repetitive tasks will likely be automated with ChatGPT and the likes. However, it’s still up to us to define our business needs and to build the proper architecture. So there is no reason to be afraid, but you do need to embrace it.

## About Me

I am a full-time Google Workspace and Google Cloud Platform developer, and the Founder of [Wurkspaces.dev](https://www.wurkspaces.dev/). | [Hire me.](https://wurkspaces.dev/contact)