July 11, 2025

Using GenAI to Create a SDK from Sample Code

#development #generative ai #python #nodejs

I've been experimenting more and more with AI-assisted coding (not gonna call it vibe coding, even when I do), and the results have been incredibly interesting. Today, I decided to try something interesting - generating a SDK for an API.

There are multiple tools out there that can convert an OpenAPI specification to an SDK (and those tools I've seen are pretty darn cool), but I wanted to attempt another route. Earlier this month I blogged about the new Foxit PDF APIs I'm working with. These APIs are all *really* simple, which is good of course, but when prepping for the release, I wanted to build code samples for every possible operation.

I did so, and they're up in a repo. I've got a complete set of Python examples here: https://github.com/foxitsoftware/developerapidemos/tree/main/pdfservices/python If you look at any one of them, and then another, you'll see they are *very* similar:

- Upload a doc.
- Call an API.
- Check the API status.
- Download the result.

I was curious how well Copilot (in Visual Studio Code) could handle a request to turn this into a SDK. I began by going into my Python folder (my Node folder isn't quite complete yet), and asked:

```
this folder contains a set of samples that make use of a PDF Service API.
I'd like to build a one page Python class as a simple lightweight SDK
wrapper. It should cover at least Word to PDF and PDF to Word initially.
The developer will make an instance of this class and pass in their
credentials for ease of use.
```

Notice I focused on two examples, Word to PDF and PDF to Word. I wanted Copilot to start small. Here's part of the response, and it's near perfect in understanding:

```
I have reviewed both wordToPDF.py and pdfToWord.py. Both scripts use
similar logic for authentication, uploading documents, starting a
conversion task, polling for completion, and downloading the result.

I will now create a single Python class, PDFServiceSDK, that wraps
this logic for both Word-to-PDF and PDF-to-Word conversions. The
 user will instantiate the class with their credentials and
 methods like word_to_pdf(input_path, output_path) and
 pdf_to_word(input_path, output_path).

I will keep the implementation concise and easy to use.
```

And ... it did it. Perfectly. I did notice, of course, it wrote code a bit differently than me. I tend to use a bit more whitespace in my code, and I tend to use camel case for variables in Python. However, most Python code uses snake case. When Copilot generated the SDK, it used snake case, and honestly, that was the right call. I'd rather have something that follows the more standard Python way of doing things.

I then asked:

```
this looks good, look for all of the "to PDF" samples,
and all the "from PDF" samples, and add those other methods
```

The wording here was pretty specific. My folder has a few files that don't relate to a particular API but are more general, and this would, in theory, cover most of the API.

Once again, it did it near perfect, except that I had a demo called `pdfToThumbnail` that I didn't want included. Ditto for one that attempted to convert PDF to Markdown. I was incredibly lazy and asked Copilot to remove these instead of me just selecting and deleting. I further proved how lazy I was by asking it to alpha sort the non private members.

The end result is an SDK I can now call like so:

```
import pdf_service_sdk
import os

CLIENT_ID = os.getenv("CLIENT_ID")
CLIENT_SECRET = os.getenv("CLIENT_SECRET")

sdk = pdf_service_sdk.PDFServiceSDK( client_id=CLIENT_ID,
                                     client_secret=CLIENT_SECRET

sdk.word_to_pdf(input_path="../../inputfiles/input.docx",output_
print("Conversion completed successfully. Check the output file
```

Pretty simple, right? I then decided to push my luck:

```
can you make a copy of the Python SDK for Node.js. it should work
with the latest version of Node.js and use fetch() for network
calls. API methods should use async.
```

The initial result was CJS, so I then said:

```
please convert this to ESM support with imports.
```

And voila - it did it just fine: https://github.com/foxitsoftware/developerapidemos/blob/main/sdktesting/nodejs/PDFServiceSDK.mjs

Here's an example of using it:

```
import { PDFServiceSDK } from "./PDFServiceSDK.mjs";

const clientId = process.env.CLIENT_ID;
const clientSecret = process.env.CLIENT_SECRET;

const pdfService = new PDFServiceSDK(clientId, clientSecret);

await pdfService.wordToPDF('../../inputsfiles/input.docx', '../.
console.log("PDF conversion completed successfully.");
```

As with the Python version, this code isn't exactly as I'd write it, this time mostly with the lack of white space in methods, but I can get over it.

You can see both SDKs, and 2 sample files, here: https://github.com/foxitsoftware/developerapidemos/tree/main/sdktesting

What's Missing?

I focused on the "to" and "from" methods, added Extract, but did not get the optimization APIs in. Those will be done by hand as by this point, it's just a matter of copying and pasting one of the existing methods.

The other issue is that these SDKs basically give you a sync-like interface to our async process. I think that's fine. However, I think developers may like to have option to do things more async as well. What I'm considering is this:

Right now, when you do an operation, like `wordToPDF`, the signature looks like so: `wordToPDF(input file, output file)`. What I'm considering is simply letting the output be optional. If you don't pass it, you get a task back. With that, I'd offer two more options. You can do your own looping and pausing with `checkTask`, or use `pollTask(task, int)` to poll until completion. Both would give you a bit more control, and give you access to a result document in our storage that you could then call another operation on... which means another option - passing a document ID to the methods, not a path. Whew. AI helped me quite a bit here, but there's definitely more I'm going to need to do. Oh, and of course, getting these online so they're installing via `npm` and `pip`. Next week!

Photo by Daniel K Cheung on Unsplash