

Dynamically adding text alternatives to images with AI

Karl Groves. - 01/02/2024

By volume, missing text alternatives are among the top 5 most common accessibility issues on the web. Despite the fact that WCAG was first released 25 years ago, the simple act of adding a text alternative (or, as others call 'em: "alt tags") remains elusive to many. This is especially true of non-technical content authors. In this post, we'll walk through how to use AI to repair this common error.

First, a disclaimer

Following along with this post isn't going to result in something you can just plug right into your site. Along the way, I'll be pointing out some things you'd want to do to make this system more robust. Alternately, you can just [hire us](#) to create the production-ready version specific to your site.

Setting up the service

The most important prerequisite for this project is that you have an [Azure subscription](#) and have created a [Computer Vision](#) resource. For now, the free tier should be just fine. You will need to save a couple of things during this process: Your resource name, and your API keys.

Setting up the project

Now, head on over to the [repo on GitHub](#) to follow along. You can [download the source](#) or fork the repo, depending on your own comfort level with such things. In either case, you'll also need to have [Node and NPM installed](#) to move forward. Once you've pulled down the code, your next step is to run `npm install` to install all of the necessary dependencies.

The final step is to configure your system. Inside the project is a file called `config.example`. Rename that to `config.json`. Here's what that file looks like:

```
{
  "resource": "https://{resourceName}.cognitiveservices.azure.com/",
  "region": "eastus",
  "key1": "",
  "key2": "",
  "apiVersion": "2023-10-01",
  "features": "tags,read,caption,denseCaptions,objects,people",
  "modelVersion": "latest",
  "language": "en",
  "genderNeutralCaption": false
}
```

As you can see, that file contains a couple of areas that need your information. Specifically, those are the `resourceName`, `key1`, and `key2` which, as their names imply, are the things I told you to save from before.

Use the project

Now you're ready to turn on the service and start using it. At this point, you might be wondering why this is basically just a Proxy for the REST API from Azure. I did this for 3 reasons:

1. So that your credentials are not exposed as part of a public facing JS file
2. To simplify the API request by setting defaults
3. This gives you a head start on a final implementation that would also probably store the results for re-use, do some reporting, set some other properties, and things like that.

Please also note that in the real world, you'd also want to have some sort of authentication mechanism for this service. As it stands right now, it'll accept any **POST** request of any kind from any source and pass it over to Azure with minimal validation.

What it does

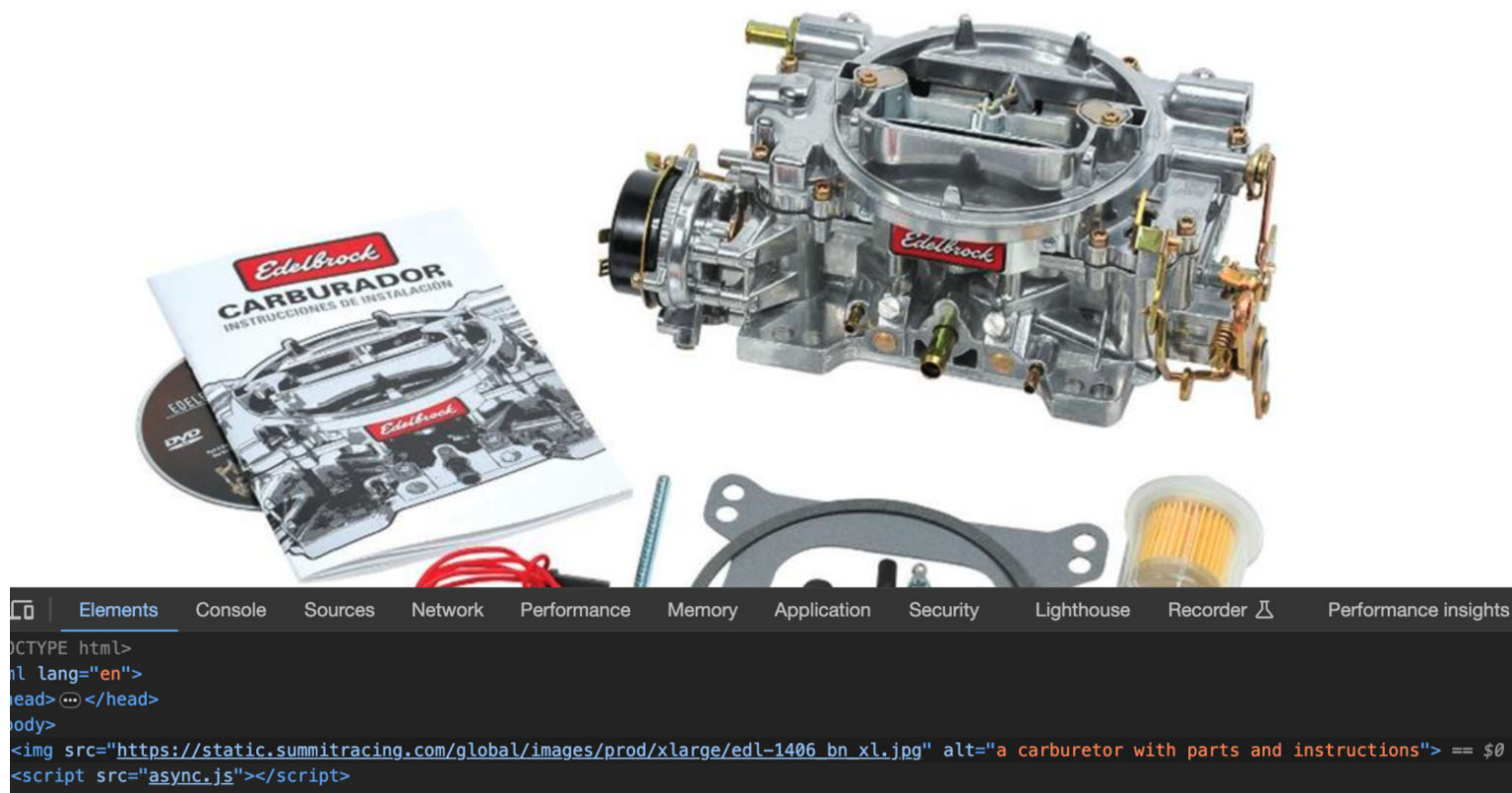
As I said above, this service accepts a **POST** request. The request must be in the form of JSON and requires only one property: **url**, which represents the URL of the image you want a text alternative for:

```
{
  "url": "https://www.example.com/images/foo.png"
}
```

Implement it on a web page

In that repo is also a **demo** folder. It has 2 files in it: **async.js** which holds the client JavaScript for making the request, and **index.html** which is a super simple HTML file with an image in it. The **** tag has no alt attribute.

The job of **async.js** is to find images that have no alt attribute at all and send each of them to the web service to retrieve the text description. Once the description is returned, that string of text is used as the value for the image's **alt** attribute. Load the file located at **demo/index.html** into a browser while [devtools is open](#) and, assuming that your configuration is correct, you'll see that the image now has an **alt** attribute.



Strengths and weaknesses of this approach

The strength to this approach is that it fixes missing text alternatives quickly. If you have a site with a ton of images, such as a retail site, the ability to get text alternatives like this is an awesome way to fix a big problem without the massive labor needed to both find the images without alternatives and have a human write descriptions for every single one. For example, we have a customer right now who has over 6,000 images without alternatives. It would take an incredibly long time to fix each one.

There are a few downsides, however. The most notable and impactful downside is, as I mentioned in a [post on my personal blog](#), that AI lacks an “opinion”. In this case, the opinion it lacks is about what the image represents *in the context of its use*. Sometimes, like on retail sites, the image is of a product and a concise, fact-based description of what is in the image is fine. In other cases, the website owner may have chosen a specific image to evoke a specific feeling about a topic or about the company itself.



The image above is taken from the website of our friends at [Scribely](#), a company that specializes in fixing alternatives for images and media. Microsoft's Computer Vision returns a response of "a group of women sitting on stairs smiling". However, Scribely's text alternative is: "Diverse group of women turn toward one another and smile as they sit outdoors on a narrow set of steps painted with abstract designs." As you can see, not only was Scribely's description more accurate, but also used language that signal to the user what kind of company Scribely is. This is more than just a company that writes alt text. They're friendly people, committed to diversity. The AI product is focused on accurately conveying what is in the image, not why the image is there.

Computer Vision, like all automation in this space, may suffer from [GIGO](#). The way all such products work is that they will return a response with one or more possible image descriptions (Microsoft calls them **caption** and **denseCaptions**) and the product’s response also includes how confident it is in its accuracy.

For example:

```
"captions":[
  {
    "text":"a city with tall buildings",
    "confidence":0.48468858003616333
  }
]
```




In the above code block, Microsoft’s computer vision says it is only 48% confident. When the image is more complex, containing more subjects, or is more “artistic”, the confidence tends to plummet.

Next steps, if you go this route

Despite the issues above, this would be a cool way to quickly fix issues with alt text quickly. That said, here are some important next steps and considerations.

1. The web service needs some form of authentication before accepting requests. You may also want to consider throttling traffic to it.
2. The web service definitely needs some way to cache results. These services aren’t free. If you’re sending a fresh request for an image description for each image on each page each time it is loaded in a user’s browser, it will get very expensive very fast.
3. Finally, you should understand that this is – at best – a temporary solution. The real solution involves taking a strategic approach to identifying which images need text alternatives and writing the appropriate text alternatives for them.

Share it with others



Related Blog Posts

[Can you really get a tax credit for making your website accessible?](#)

Yes! But the devil is in the details. The “General Business Credit” refers to a collection of various individual credits available to businesses in the United States, as outlined and governed by the Internal Revenue Service (IRS). These credits are designed to encourage certain business activities that align with broader economic, social, or policy goals, [...]

Join Our Mailing List

Enter your email

Subscribe

Powered by [Buttondown.](#)

[Home](#) [About Us](#) [Lets talk](#)

[Accessibility Risk Assessment](#)

[Services](#) [Careers](#) [Blog](#)

[Accessibility Statement](#)

©2024 Afixt. All Rights Reserved.

