Replicate Proxy Docs

TLDR:

- Use this url to reached replicate API.
- https://itp-ima-replicate-proxy.web.app/api/create_n_get
- Check the API of the specific model you want to use for parameters to send

Reasons For Needing a Proxy

- Keeps API-Key Private
- Prevents CORS exception
- Has ITP/IMA Pay For Replicate
- Hosts files for Modles that take URLS a parameter

Using Javascript Fetch Function To Talk to Replicate API

```
async function askForPicture(p_prompt) {
    const replicateProxy = "https://itp-ima-replicate-proxy.web.app/api/create_n_get";
    let authToken = "";
    //The JSON that your model in replicate needs to work
    let data = {
        model: "black-forest-labs/flux-schnell",
        input: {
            prompt: p_prompt,
        },
    };
    //The JSON that the fetch function needs to work (including data json)
    let fetchOptions = {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            'Authorization': `Bearer ${authToken}`,
        },
        body: JSON.stringify(data),
    };
    console.log("sending to replicate:", fetchOptions, "url", replicateProxy);
    const response = await fetch(replicateProxy, fetchOptions);
    const prediction = await response.json();
    console.log("DO SOMETHING WITH OUTPUT", prediction);
}
```

Authentication

- You do not need to authenticate
- Authenticating gives you bigger quotas
- Only `nyu.edu` authenticated accounts are allowed.
- Press this button below and authenticate with your .nyu.edu address
- Copy and paste this token into authToken variable in the examples.
- You will have to refresh this token occasionally.

Google Login

Copy

Client Examples

- [Dynamic List of Examples for the Proxy]

Usage limits

- You can get a few creations on less expensive models without authenticating.
- There is a limit of 500 requests per day on less expensive models if you auntenticate
- There is a limit of 10 requests per day on more expensive models like video models.

Sending URLS to a Model

Just let the proxy know which fields of your JSON to turn into a URL and what kind of file it is.

```
let imageInBase64Text = myCanvas.toDataURL();
let jsonToReplicat ={

    model:"myAmazingModelThatHasMediaInputParameter"
    fieldToConvertBase64ToURL::"image"
    fileFormat:".jpg"
    {
        prompt: "fast running",
        image: imageInBase64Text
    }
}
```

You might need to send media to a model (eg. send an image file to a model that upscales an images, or send a sound file to a model that transribes sounds). You are just sending JSON text in the fetch function, and most models will ask for a URL pointing to a file rather the data of the file itself. Your user's laptop cannot be a server for that URL but the proxy is a good server for that URL so it does you the favor of creating a temporary file with the extension the the format you want (eg. .jpg, .png, .wav), hosting it with a good URL and then will replace your parameter with that URL before passing it to replicate.

You can get the media to the proxy in the first place because there actually is a way to put your media's data directly in the text of JSON called base64 encoding. This turns media data into text, using the toDataURL() function. Sadly most models don't want you to send the data this way so you have to go though this dance of host a file and getting a url for them.

Longer Story

Using a Proxy

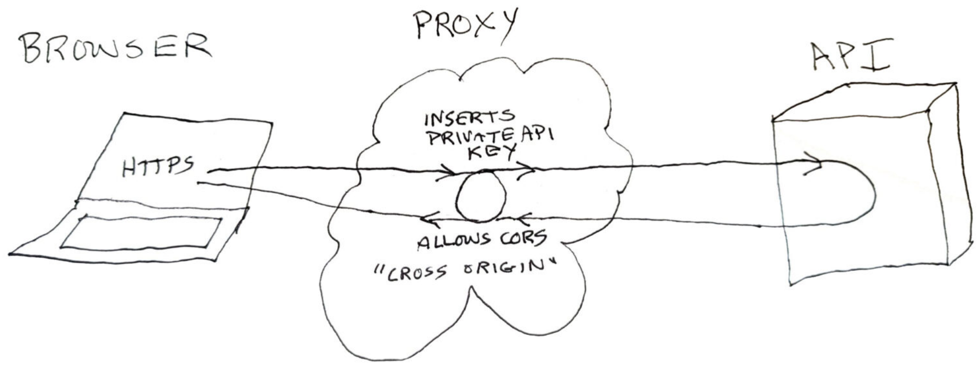
Making a web call should be the easiest thing in the world. But there are a couple of complications that will require us to have a Proxy which is a very simple intermediary program running on a server that simply relays our request on to Replicate. Eventually you might want your own proxy but for the purposes of this exercise ITP/IMA will provide a proxy for for you. Instead of you using the Replicate's API URL directly, you use our proxy and we will relay everything you say to them and return everything they say back to you. This will also save your from having, AND PAYING FOR, a replicate account. So learn to enjoy the proxy lifestyle and just use this url.

let proxyUrl = "https://itp-ima-replicate-proxy.web.app/api/create_n_get";

Here are the benefits for using a proxy software written in node.js (javascript outside the browser).

- The main reason is to get around CORS. To protect web services from being abused, you will get a CORS exception in a web browser if you try to contact a server that did not also serve the html and js making the request. But you can make that request from outside the browser and so you have node.js (javascript outside the browser) do it.
- The web is an inherently open source environment and so the API key in your code could be stolen and your account could be used by other people. No one can see your node.js code so the API key is safe in the proxy.
- We are inserting a key from a Replicate account that ITP/IMA pays for. Replicate gives us a bunch of free credits and then we pay for the rest but the main point is you don't need to make an account and pay for Replicate right away.
- There are other reasons like converting from a https to http request, hosting files for URL parameters, distributing the load across servers but CORS and hiding API keys are main reasons.

If you ever want to make your own proxy you would need to write a tiny node.js program. You could run it on your machine and you may want to write your whole app in node.js in which case you would not need a seperate proxy app at all. If you want other people to access what you made, it would be better to run it on a server which is always on and with a fixed address unlike your laptop. The code for a proxy is pretty easy because all you are doing is pass a request along and adding an API Key. You might look into services like Replit that allow to pretty easily make your own proxy and adding your own API Key number to an ENV file and get the address.



Web Requests in Javascript

You will find lots of javascript examples on the web using different functions for calling out to the web, for instance p5js has httpPost, XMLHttpRequest, Jquery Ajax, but we will be using the more modern Fetch.

Alternatively you will find some services, including Replicate and will have their own javascript client libraries to further simplify and specialize fetch to connect to their services. Currently the Replicate client library only works in nodejs (not for a web client) so we will still concentrate on Fetch.

Parameters That Your API is Expecting

To work an API you first have to find out what parameters to sent to it. If you click on a model in Replicate there will be little "playground" interface with a few fields for the parameter and a submit button to allow you to figure out what the API expects to get and what it gives back before you go through the trouble of making your own calls. Often these test pages will even have sample client code for making the calls even in multiple languages. Sadly you may see the “Javascript” or “HTTP” code seem promising but just shows Node.js (javascript outside the browser) code or event just CURL calls because, as we will see in the section below about proxies, they don’t expect you to be able to make a call directly from browser based javascript.

Usually most of the parameters will have default values that you don’t have to specify them. But if you were using, for instance a Stable Diffusion or llama 2 model you would at least have to specify the name of the model. You can find he name of the model at the top of the page and it usually looks like a path and there is a button for copying it but that comes with a lot of spaces that you need to delete. The model and, of course the text prompt, might be the only parameters you are required to supply to many models. You would package these up into json with two levels, model in the first level and and all the parameter one level deeper within an “input” object: const myData = { // if you use copy button from replicate you might have delete spaces "model":"stability-ai/stable-diffusion-3", "input: { prompt: "photorealistic painting of salvador dali", }, }; If want a specific older version of a model in Replicate, you can also specify that. let data = { //find the url in the replicate under API tab, and then under HTTP tab version: "stability-ai/stable-diffusion-3:ac732df83cea7fff18b8472768c88ad0", input: { prompt: p_prompt, negative_prompt: "", num_inference_steps: 50, guidance_scale: 7.5, seed: 52, }, };

Fetch Parameters for Sending

You have to give the fetch command instructions for how to work. The most basic instruction is of course which URL to use. and then next is POST or GET with are two different methods of storing the information you are giving out. A GET is like carry on for an airplane with it gets attached after a question mark on the URL like https://itp.nyu.edu/search?name=jehovah. A POST is for larger checked baggage that goes in a separate compartment under the plane. In addition you might put information about how to authenticate and we will talk more about that in the section about proxies. All of these parameters for fetch and the bound up together with the API parameters are put into json like this. These are the details we are spared with P5’s loadJSON function.

```
async function askForPicture(p_prompt) {
    const replicateProxy = "https://itp-ima-replicate-proxy.web.app/api/create_n_get";
    let authToken = "";
    //The JSON that your model in replicate needs to work
    let data = {
        model: "black-forest-labs/flux-schnell",
        input: {
            prompt: p_prompt,
        },
    };
    //The JSON that the fetch function needs to work (including data json)
    let fetchOptions = {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
            'Authorization': `Bearer ${authToken}`,
        },
        body: JSON.stringify(data),
    };
    console.log("sending to replicate:", fetchOptions, "url", replicateProxy);
    const response = await fetch(replicateProxy, fetchOptions);
    const prediction = await response.json();
    console.log("DO SOMETHING WITH OUTPUT", prediction);
}
```

Receiving Stuff Back (Async and Await)

It takes time for stuff to come back so you should take advantage of javascript’ asynchronous functions. This is yet another way of dealing with things that will happen in the future. The first strategy we learned was callbacks for receiving serial communication and responding user interface objects like buttons. The Arduino used “polling” where it just checked over and over in the loop. Now there is a new way in javascript. Putting “async” in before “function” in your function definition allows you to use “await” inside the function to wait for one line to finish before going on to the next. This async and await saves you from having to make callback functions when you just want a couple of lines of code to run one after another. In order to use await you have to declare the function async so that things like your draw loop do not get held up by waiting for each line to finish. Related to this you might hear talk of “promises” and “then” but for now just put async at the start of the function and await in in front of the function you want to wait for. You will see that you first await the response which is like the header information which they send right away in case there was a problem. Then you have to await the meat of the message in json.

```
async function functionName(p_prompt) {
    const response = await fetch(replicateProxy, fetchOptions);
    const prediction = await response.json();
}
```

What you do with the json that comes back from the model will differ for every model and applicaiton. The first move is to print it out in the browser console and try unfolding the json until you find what you want. For example is what you might do if an image comes back in a list in a json field called "output"

```
loadImage(prediction.output[0], (incomingImage) => {
        img = incomingImage;  //put the new image in a global variable to be used elesewhere
        console.log ("Image generated!");
    });
```