Sam Kramer Follow

May 4 · 5 min read · Listen

Stop wasting your Heroku dyno hours

How my portfolio wakes sleeping projects so I don't have to pay to keep them awake.

The problem:

If you're like me, you've got a bunch of projects hosted on Heroku. It was a very attractive option when they offered a free tier with 1000 dyno hours per month (after verifying your credit card), more than enough to keep a single project at 100% uptime. There are plenty of scripts out there to visit your project every half hour or so, much longer than that and your dynos will go to sleep. They take an annoying 15–30 seconds to wake up from idle, spending the whole time appearing as yet another broken project to visitors, recruiters, and potential employers.

In the other direction, if you try to keep more than one project awake 24/7, you'll have at best 3 weeks of uptime before you run out of dyno hours. When those run out, you can show a custom error page explaining what's going on, but you can't show a working project. Heroku's new Eco tier offers the same 1000 hours, so if you want multiple apps running continuously, you're going to have to pay.

The solution:

A strategy for how to spend those precious dyno hours. If each of your projects will do the courtesy of waking up the others when it gets visited, you can let the others sleep while one stands guard. This is the one you will set up automated requests to keep awake. Choose which project is (ho get that first visit, and to be already awake.

Keep that project highlighted on your portfolio, on your LinkedIn profile, on your resumé, anywhere that a potential employer might visit from. It needs to be the same one everywhere because it's the only one ready to handle the task, it's the only one that's awake all the time.

The next step is to actually wake them all up.

```
const autoRun = () => {
  // Gather all your project links...
  const projLinks = document.querySelectorAll('.projLink');
  for (let link of projLinks) {
    // ... And send each one a wakeup request
    fetch(`${link.href}api/wakeup`, {mode: 'no-cors'});
  }
}

autoRun();
```

When this code runs after the page loads, it makes an array of all the project links (in the case of my portfolio, all the elements with the class "projLink") and iterates through, sending a GET request to /api/wakeup on each of my projects. It's not a real API route, and it doesn't have to be. In order for the backend to check if it's valid, the dynos have to wake up and start the backend, and that's literally all we care about. Within moments after my portfolio loading in a potential employer's browser, every one of my projects is starting to wake up.

I have code like this on nearly each of my projects, though not necessarily links to every other project for iterating through. In those cases, it's fine to hardcode a list of destinations for those wakeup requests.

The response:

That `fetch` in the first step generates a promise, and when it gets a 404 back from the server, the promise is fulfilled. In other words, *the script can detect when your project's dynos have woken up*. The whole thing we're trying to avoid is having someone stare at their empty browser as your project wakes up and thinking it's broken, but your project *tells us* when it's awake, and we can `await` that.

```
const autoRun = () => {
  const projLinks = document.querySelectorAll('.projLink');
  for (let link of projLinks) {
    // Listen for clicks on your project links
    link.addEventListener("click", projClickHandler(link.href));
    fetch(`${link.href}api/wakeup`, {mode: 'no-cors'});
  }
}

// Generate a custom click handler for each URL
const projClickHandler = url => {
  return async e => {
    // Don't just open the page...
    e.preventDefault();
    // ... Show a "Please wait" modal instead ...
    showModal(url);
    const apiUrl = `${url}api/wakeup`;
    // ... And wait for the 404 response ...
    const result = await fetch(apiUrl, {mode: 'no-cors'});
    // ... Before redirecting to the project
    window.location.href = url;
  }
}

const showModal = (url) => {
  // This code brings up a loading modal, specific to the url of each project
}

autoRun();
```

That `await` is doing all the heavy lifting here. It pauses the execution of all your synchronous code, and only resumes after the promise stored as `result` is fulfilled (or rejected) asynchronously.

The outcome:

You now are covered for any visit to anything that's live all the time. For me, that's a static portfolio hosted on Github pages, plus one project on Heroku. I put those two links higher up on everything — resumé, LinkedIn, Wellfound, etc… — and hope that nobody clicks something else first.

In a worst-case 31-day month, that costs 744 dyno hours out of thousand, leaving a coincidentally aesthetic 256 hours left for other projects. That's ten and a half days that you could leave another project running for, but why would you? You have a *system* now.

And paranoia that someone might click the sleeping ones first.

The overkill:

The next step is to intercept *every click* on *every project* from *every place*. Ambitious? Absolutely not. There's already code in place to do all the hard parts, all it needs is a URL.

```
const autoRun = () => {
  const projLinks = document.querySelectorAll('.projLink');
  for (let link of projLinks) {
    link.addEventListener("click", projClickHandler(link.href));
    fetch(`${link.href}api/wakeup`, {mode: 'no-cors'});
  }
  // Save the GET parameters from the URL
  const params = new URLSearchParams(window.location.search.toLowerCase());
  // Look for a ?modalfor= parameter
  if (params.get('modalfor')) {
    // Save that parameter's value
    const forUrl = params.get('modalfor');
    // Generate a function to launch the modal, using that URL
    const launcher = projClickHandler(forUrl);
    // launch the modal, passing in a dummy event so handler doesn't error out
    launcher({preventDefault: () => {}});
  }
}

const projClickHandler = url => {
  return async e => {
    e.preventDefault();
    showModal(url);
    const apiUrl = `${url}api/wakeup`;
    const result = await fetch(apiUrl, {mode: 'no-cors'});
    window.location.href = url;
  }
}
```

```
autoRun();
```

There's no reason that URL has to come from a link in your portfolio, it can take it from a GET parameter, and now this can check if *any website* is awake before redirecting to it. Wrap this around your project links on everything that links to your projects, and each click will go through your modal first to show something deliberate to the visitor while you make sure it's up and running.

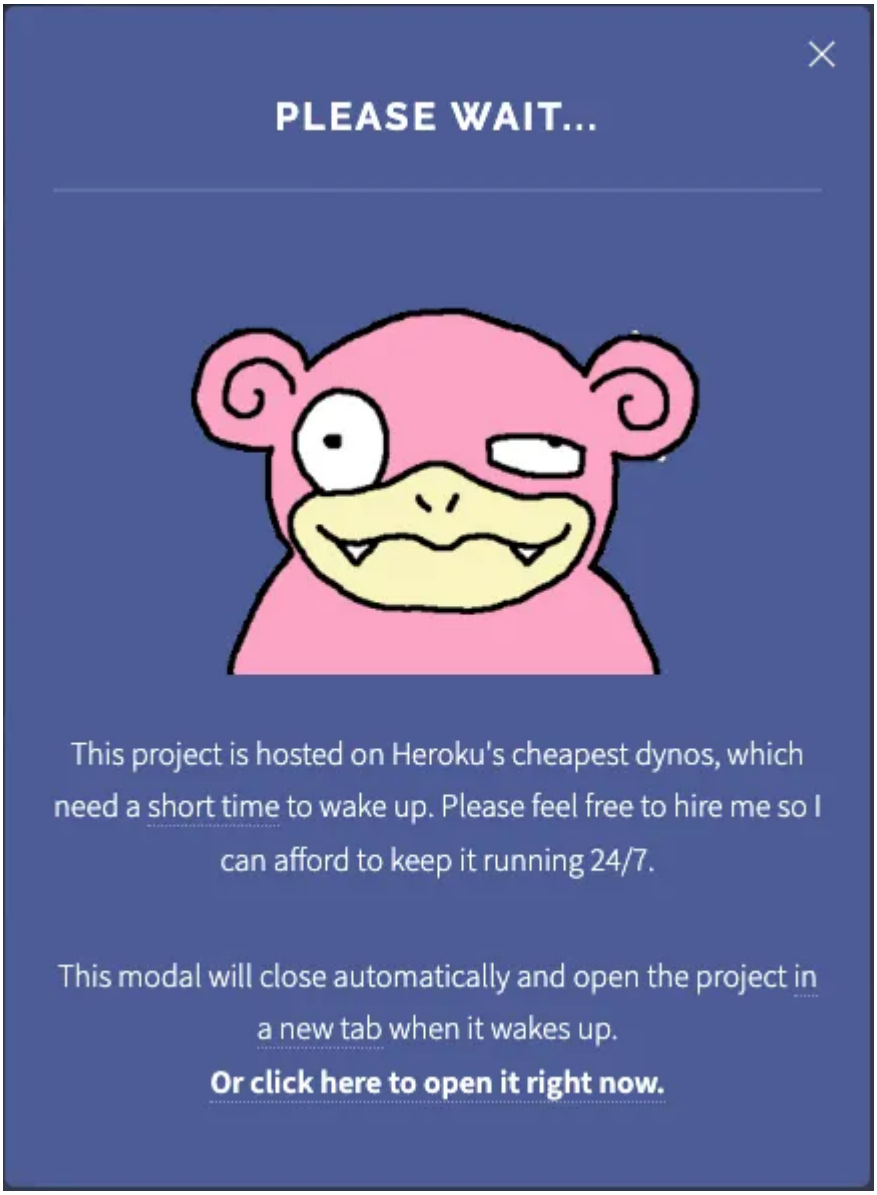
The tl;dr

If you direct all clicks to something that's awake 24/7, it's the only thing that needs to be awake 24/7. It can then wake up everything else, and redirect only after it's awake.

See the link intercept on my portfolio using a demo url (https://example.test) which will not redirect, or visit a real project to see the click handler in action!

The developer

is eager to hear from you if you found this useful, downright terrible, worth offering a job interview, or otherwise life-changing. Email me at SKramerCodes@gmail.com and find me on LinkedIn.



Heroku Programming JavaScript Optimization Efficiency

Get an email whenever Sam Kramer publishes.

Your email