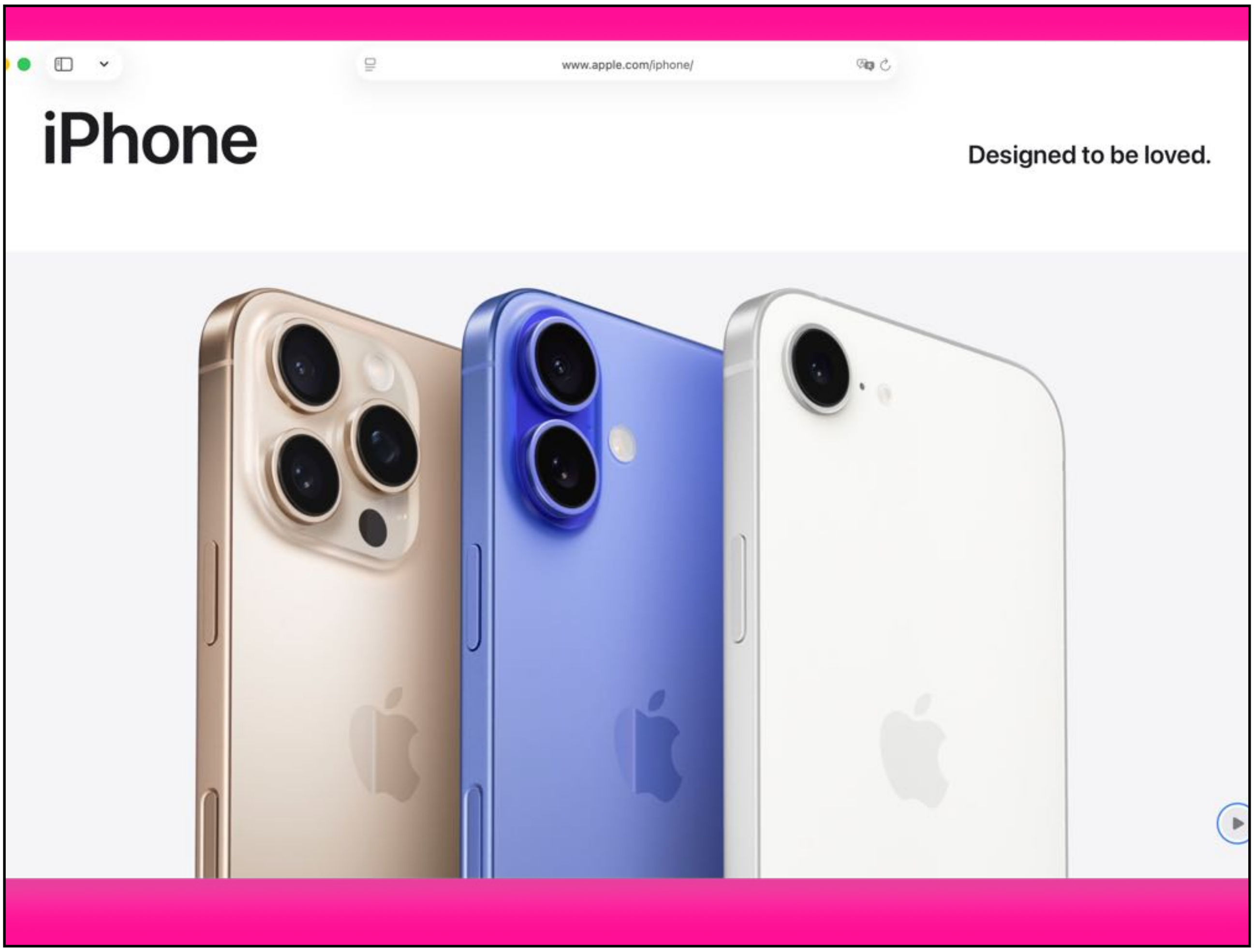
For-Profit Library

How I Learned to Stop Worrying and Love Gleam

Benji

Background



www.apple.com/iphone/

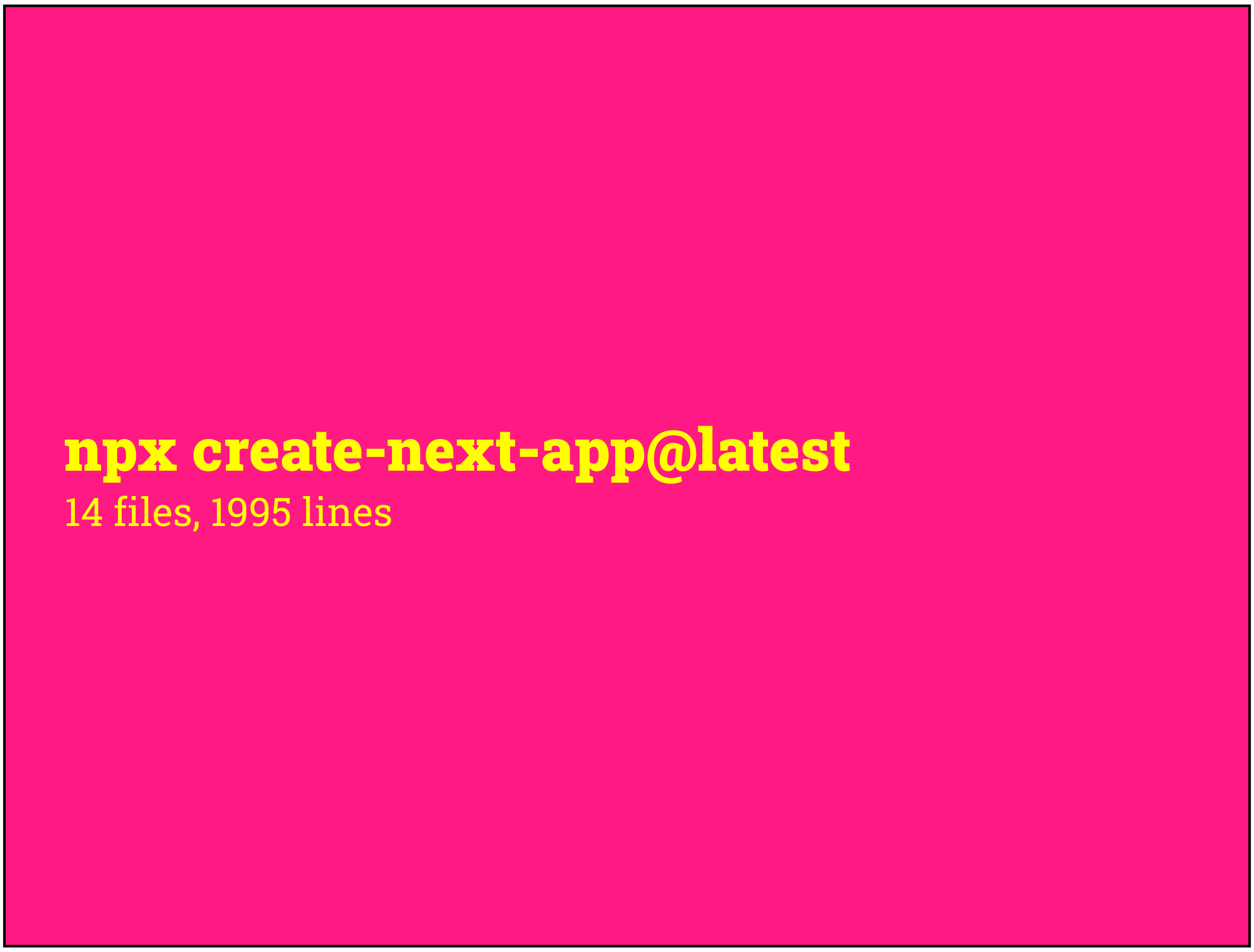
iPhone

Designed to be loved.

We're all familiar with this. Well, the best way to make anything on for these and their brothers and sisters—the Mac, the Vision Pro, the Apple Watch, the iPad if you must—is

SwiftUI. It's the shit. Look at this:

That's a whole app in SwiftUI, with a beautiful interface, live camera feed, and orientation detection with the gyroscope. By comparison...



npx create-next-app@latest

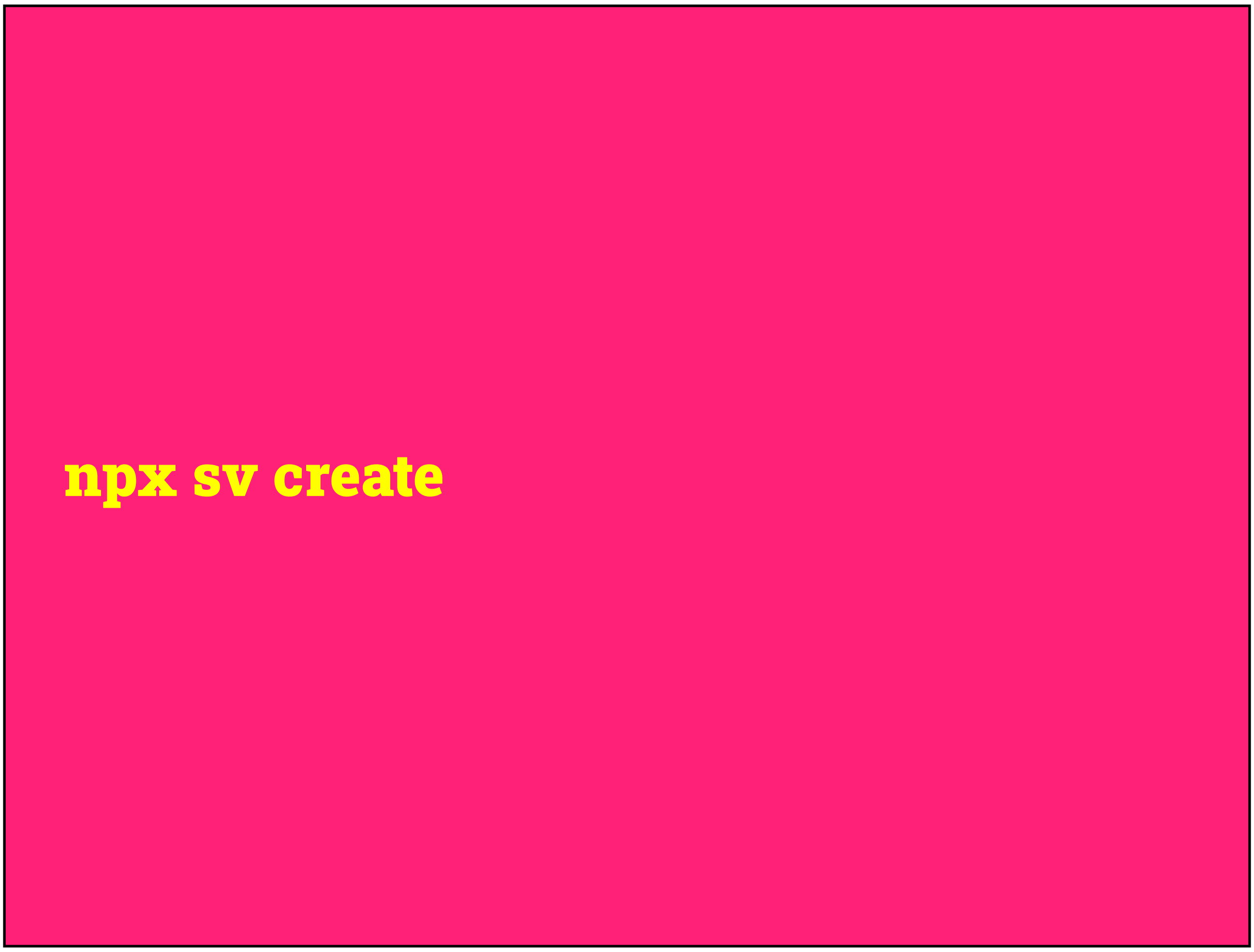
14 files, 1995 lines

You've got Next.js,

rails new

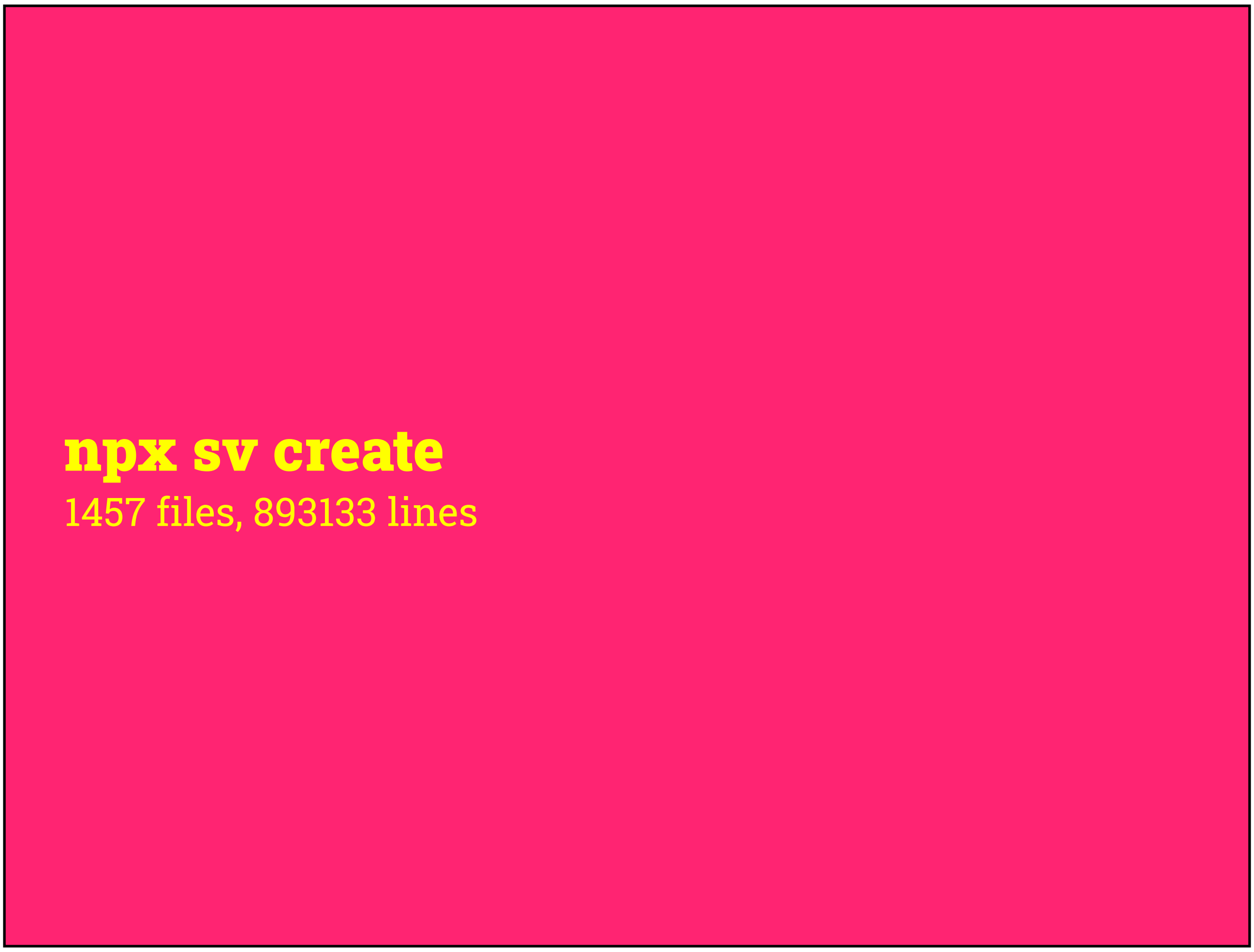
63 files, 1357 lines

you've got Rails, which makes vastly more files but still leaves you with less code.



```
npx sv create
```

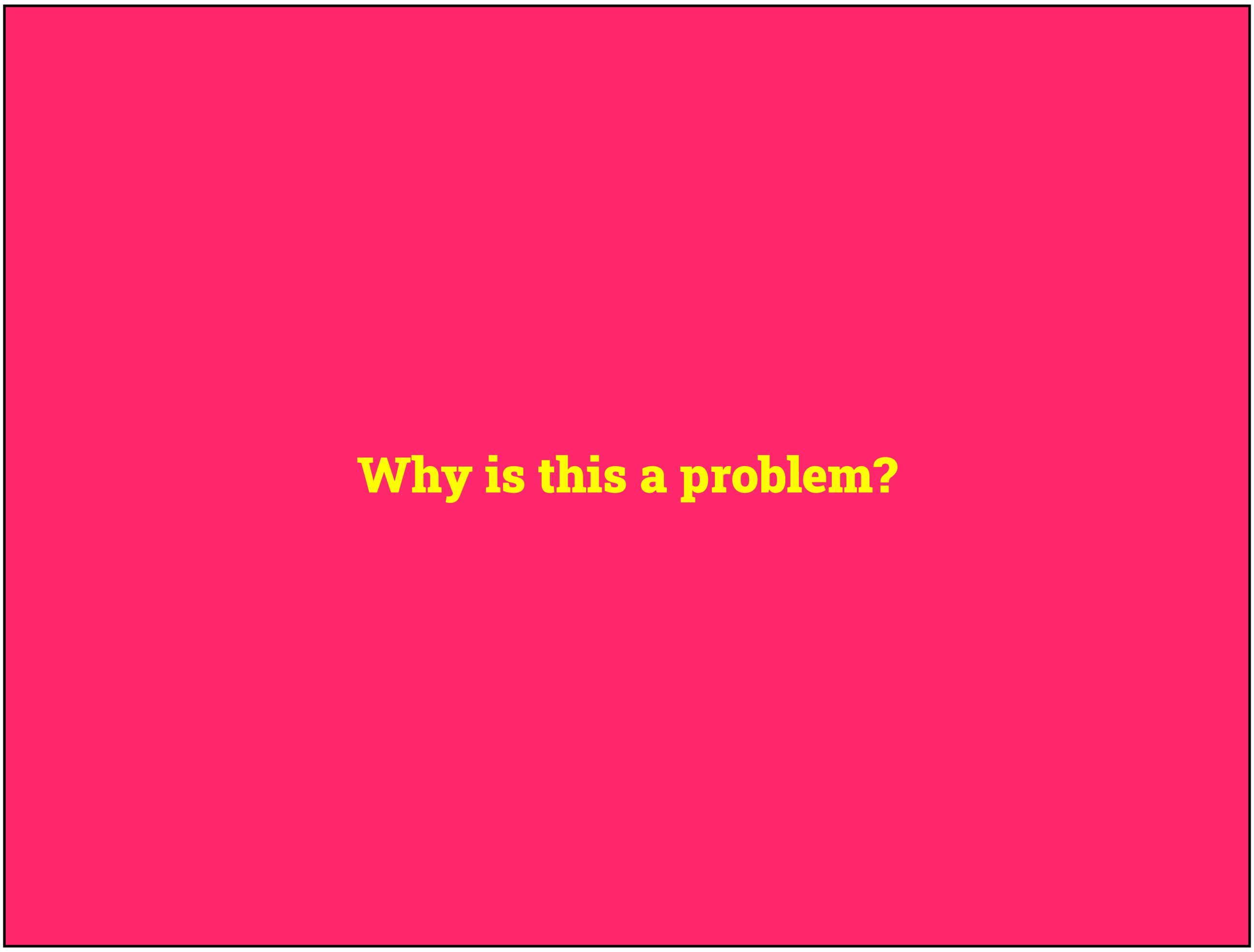
You've got Svelte, right, which writes on its website the definition of that word, meaning "attractively thin, graceful and stylish."



npx sv create

1457 files, 893133 lines

So yeah. And that's not even touching on the fact that these frameworks only cover a patchwork of features and are written in such lovely languages as those that invented the triple-equals mark to fight their demons.



Why is this a problem?

There are super stupid measurements and done totally haphazardly with `tokei` mostly to prove a point. In particular if you generate a Svelte app with `Deno` it's actually the smallest at only 143 lines. But nevertheless:

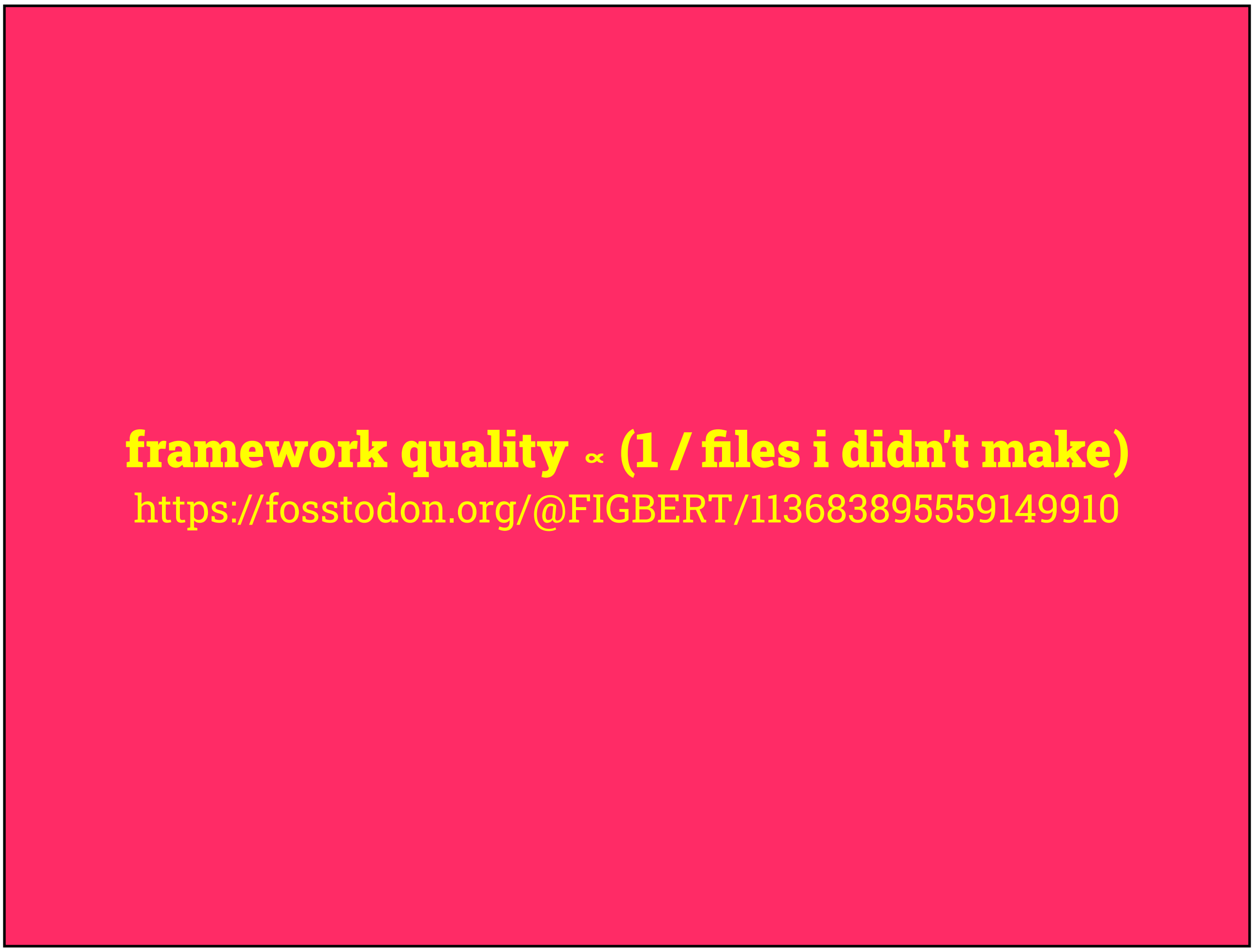
I fundamentally do not want to manage someone else's code. I am responsible for everything that lives in my repository—by implication, everything there should be present for a reason, something I can only be familiar with having written it myself. When using these frameworks that's not the case.

I am not saying you should not have dependencies. I am saying that dependencies should be self-contained interfaces and not leaky abstractions.

These frameworks somehow manage to combine the disadvantages of having remote dependencies with the disadvantages of vendoring.

And I hate to beat a dead horse but this is not the case with SwiftUI.

And all of this has lead me to the really popular approximation that everyone's referencing these days and people think is really cool



framework quality ∝ (1 / files i didn't make)

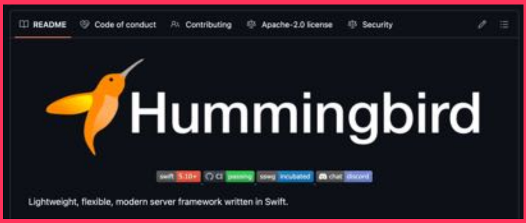
https://fosstodon.org/@FIGBERT/113683895559149910

Benji's Law of Frameworks. And beyond that, it lead me to a really dark place.

maybe i can just use rails in one file

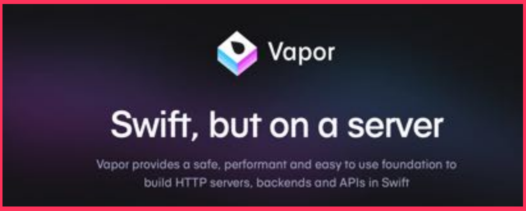
sinatra, roda, phlex...

literally swiftui in the browser



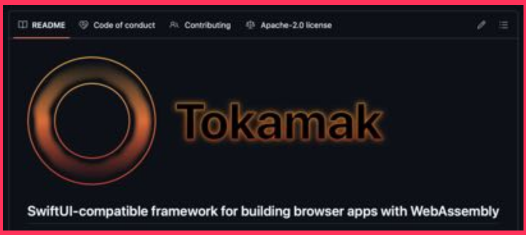
Hummingbird

Lightweight, flexible, modern server framework written in Swift.



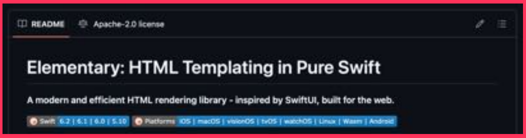
Swift, but on a server

Vapor provides a safe, performant and easy to use foundation to build HTTP servers, backends and APIs in Swift



Tokamak

SwiftUI-compatible framework for building browser apps with WebAssembly


README
Apache-2.0 license
Swift 6.2 | 6.1 | 6.0 | 5.10
Platforms iOS | macOS | visionOS | tvOS | watchOS | Linux | Wasm | Android

Elementary: HTML Templating in Pure Swift

A modern and efficient HTML rendering library - inspired by SwiftUI, built for the web.

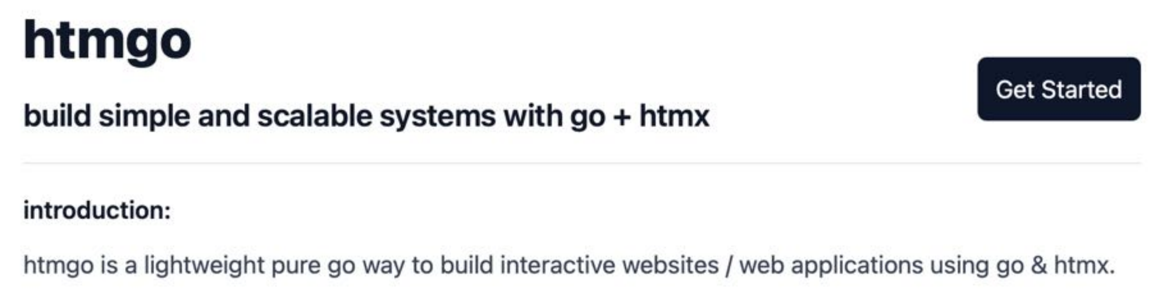
the fundamentally good shores

Modern web applications in pure Python

Built on solid web foundations, not the latest fads - with FastHTML you can get started on anything from simple dashboards to scalable web applications in minutes.

Most concisely: the problem with FastHTML is that it over-indexes on the rapid prototyping phase, but doesn't provide an escape hatch for when that's done and you want something elegant and maintainable.

near misses with perfection



htmgo

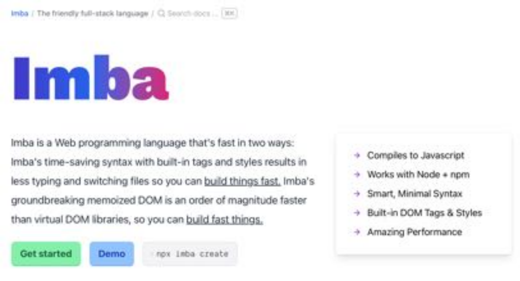
build simple and scalable systems with go + htmx

introduction:

htmgo is a lightweight pure go way to build interactive websites / web applications using go & htmx.

12 files, 292 lines

here be dragons



Imba / The friendly full-stack language / Search docs ... ⌘K

Imba

Imba is a Web programming language that's fast in two ways: Imba's time-saving syntax with built-in tags and styles results in less typing and switching files so you can build things fast. Imba's groundbreaking memoized DOM is an order of magnitude faster than virtual DOM libraries, so you can build fast things.

- → Compiles to Javascript
- → Works with Node + npm
- → Smart, Minimal Syntax
- → Built-in DOM Tags & Styles
- → Amazing Performance

```
npx imba create
```

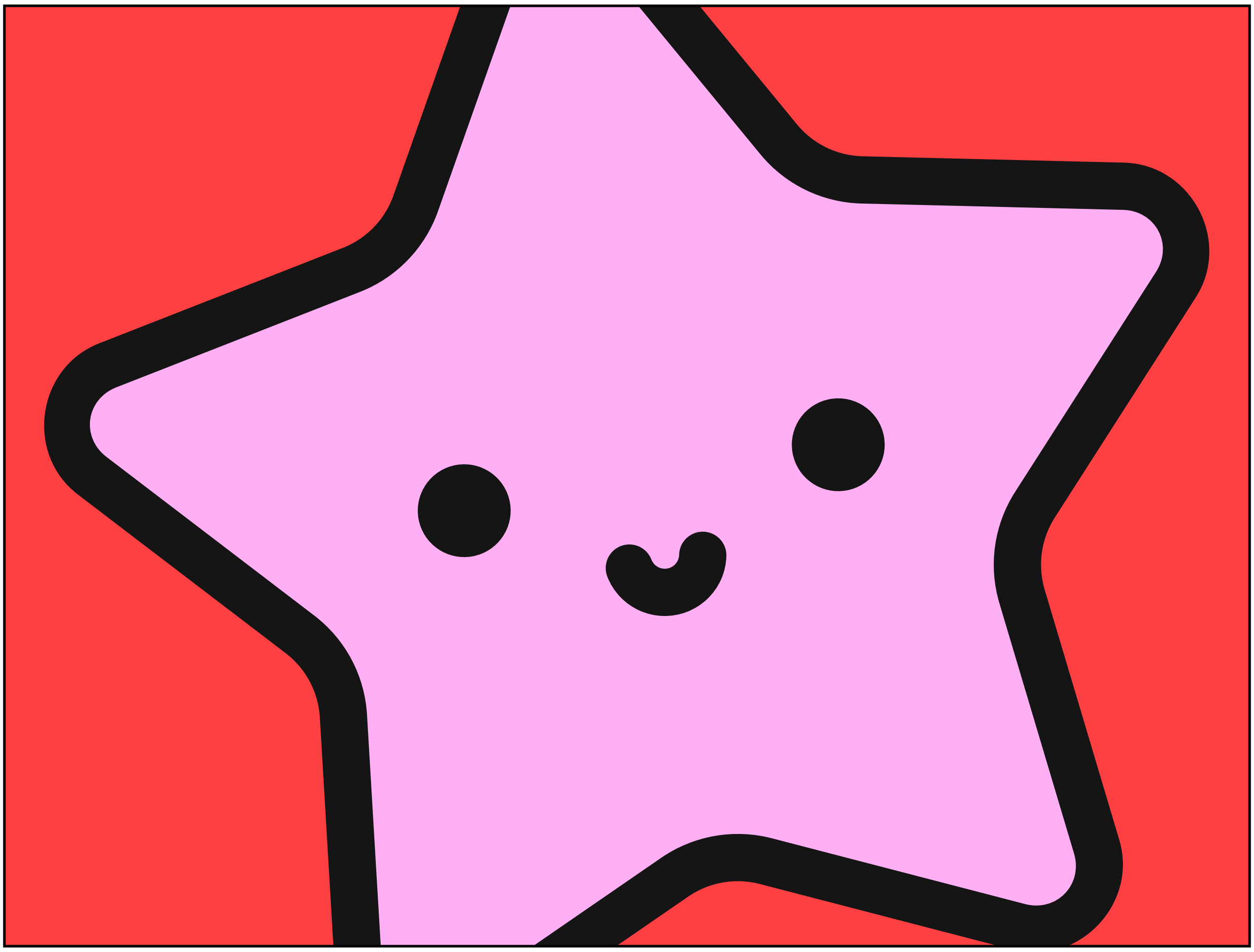
Web Origami

A language for making websites where you can understand how they're made

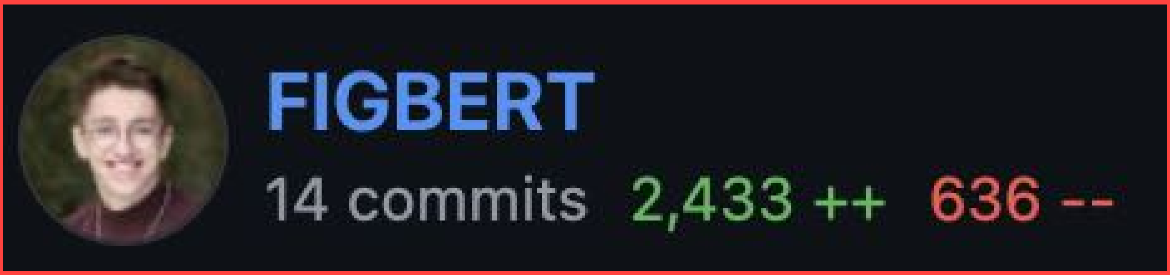
and even places where no man had before ventured.

Keith And The Big Lie

Which was something like "you should try this cool new language that is inspired by Elm with a syntax like Go." I can't imagine a better sentence for me on this quest.



That language was Gleam. Keith has never programmed in Gleam.

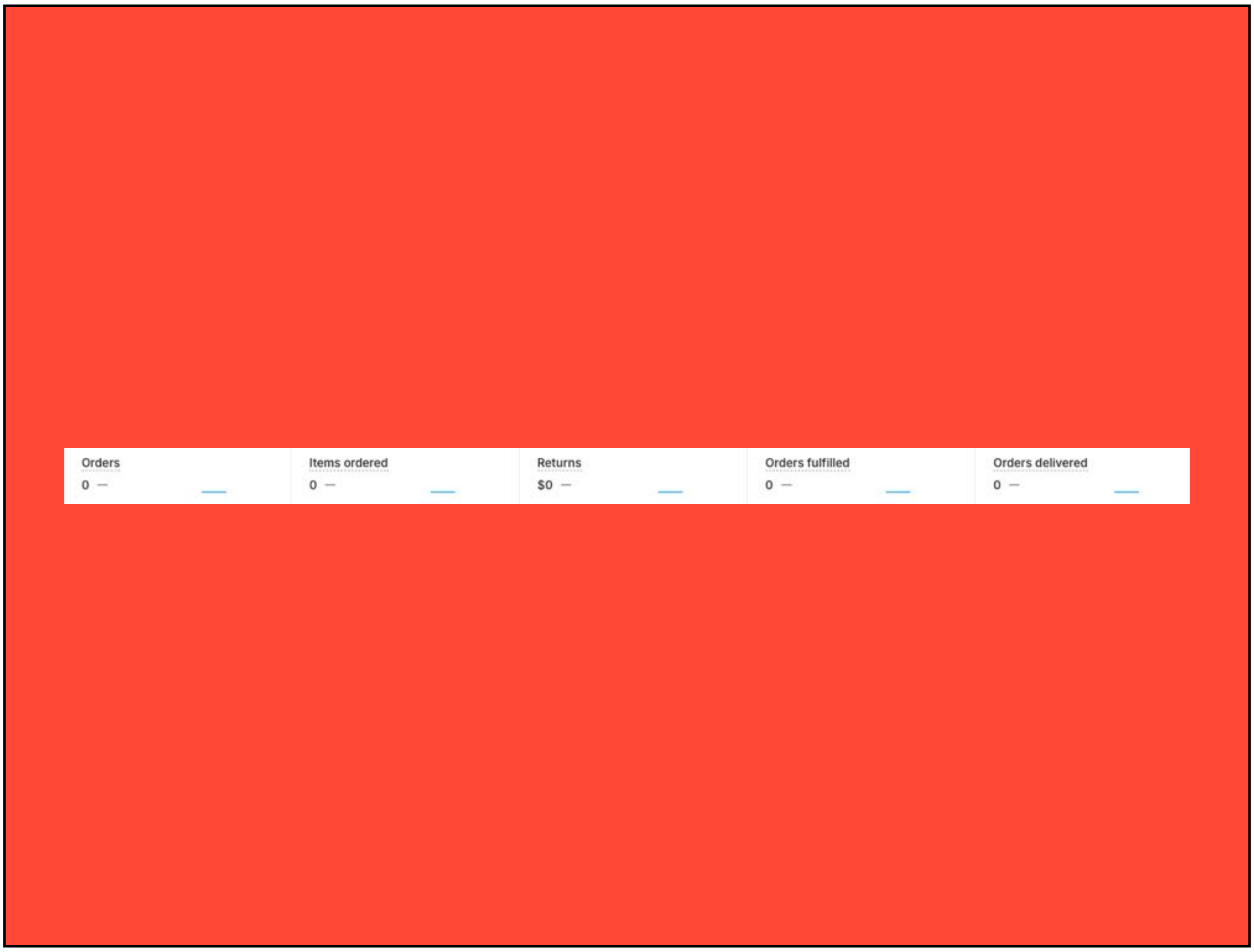


FIGBERT

14 commits 2,433 ++ 636 --

For the last three days, I have. But what have I been building? Many people don't know this about me, but when I'm not a programmer, I'm actually

a librarian. I own a bookstore—that's me.
It's fun! It's called "Blind Date A Book" and the gimmick is that you have no idea what book you're getting, you just read a one-sentence description and buy based on that.
My initial plan was to build this AI hyperscaling monstrosity but I couldn't figure out how to dropship books so I just asked my friends for their favorite books and to write pitch and made a website based on that. When people order I run out to the bookstore and ship them everything manually.



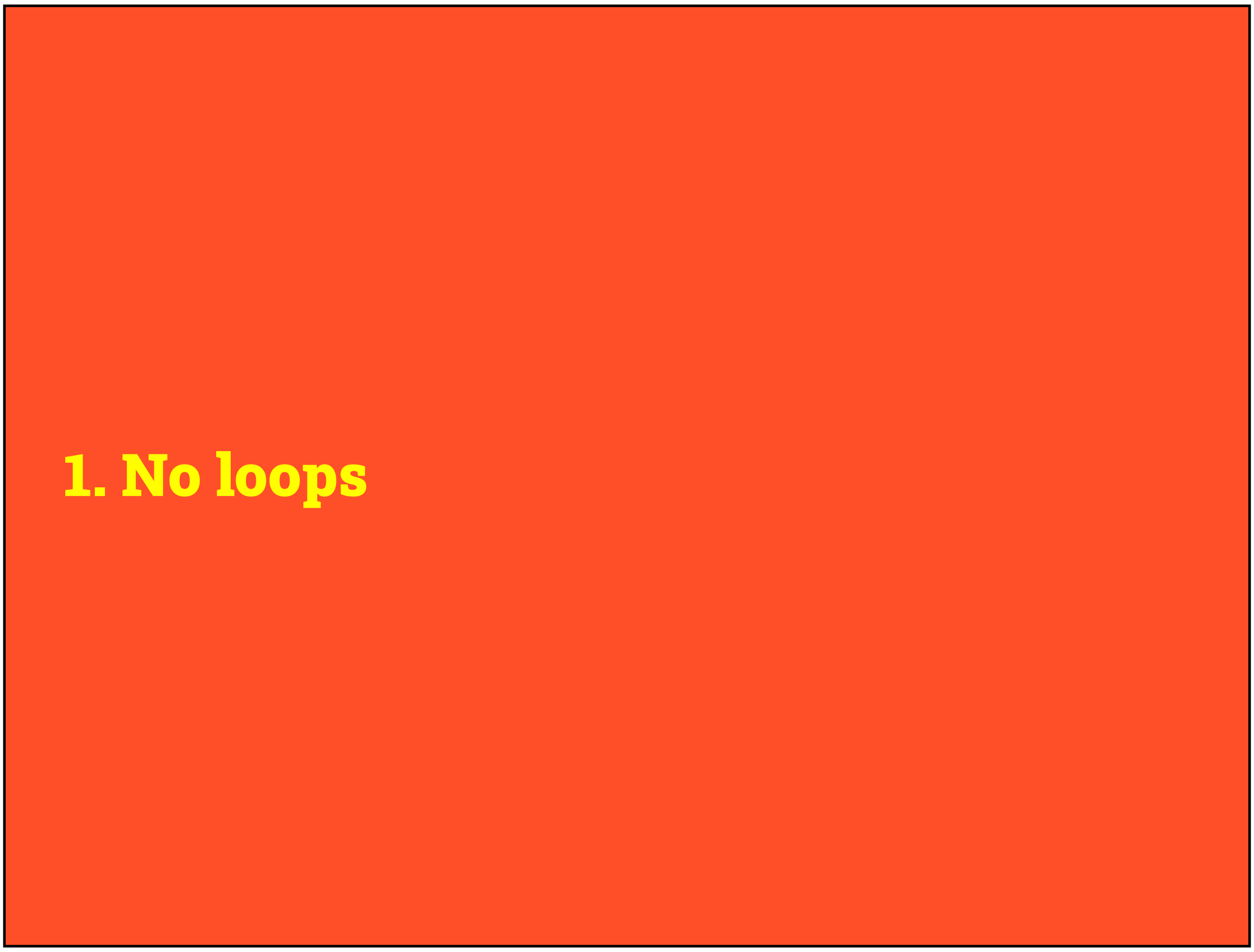
Orders	Items ordered	Returns	Orders fulfilled	Orders delivered
0 —	0 —	$0 —	0 —	0 —

It's going really well!

Hmmm. No one knows this exists. And I'm paying $40/month to Shopify to run my checkout!!
My for-profit bookstore is making quite nearly -$500/year. Bad.
And the website is slow because I made it with Web Origami which is a really weird tool, as previously stated.

Gleam

So Gleam! Remember that, from five slides ago? I rewrote the website in Gleam. A few things to know about Gleam that I found out the hard way.



1. No loops

Only recursion.

2. No if statements

Just exhaustive switch statements.

3. No lists*

Or rather, no arbitrary indexing into lists. So perhaps better stated, no lists as they are often/traditionally understood.

4. This weird thing

```
fn example(callback: fn(Int) -> Nil) -> Nil {
  callback(3)
}

fn main() -> Nil {
  use int <- example()
  io.println(int)
}
```

Demo

Cool things I will try and talk about:

1. Elm architecture
2. Epic types
3. "One Path" (unofficial)
4. FFI

Frameworks used:

1. lustre (html)
2. sketch (css)
3. glen (server)