Blogging in Typst is not that hard

i have been using Typst for writing my university lab assignment reports for a while now. it works *perfectly* for that, i have a nice template for the title page, i can make graphs in the same file as all other content, etc. for (some) programming-related classes i can even do the calculations *in the same file* as my report! frankly, it's **awesome**.

so naturally i had a thought: what if i could use it for online writing as well? surely, it wouldn't be too bad an experience. *right?*

right?

thankfully (or so i thought), Typst already has HTML export built-in! in fact, it was announced **with Typst 0.13 in February**, and has been **improved in 0.14**.

but here's the thing: according to the **HTML module reference**, it exports no styles. here's how the **index page** looks like with HTML export:

hello there

here's a bunch of links: my github, my fedi

not even *nearly* the same as the page when rendered to, for example, PNG:

i'm not complaining that it doesn't work exactly how i personally want it to work, especially since the behavior is clearly outlined in the docs. *but* sadly i didn't want to write styles *twice*. what other options do we have?

SVG export

Typst has native SVG export! and SVG is treated just as a vector image format (which it ofc is), so it renders *perfectly*. problem solved?

not yet. let's look at the (simplified) structure of an exported SVG:

```
<svg class="typst-doc">
  <g>
    <g class="typst-group">
      <g>
        <g class="typst-text">
          <use xlink:href="#..." x="0" y="0"/>
          <use xlink:href="#..." x="9.5326" y="0"/>
          <use xlink:href="#..." x="17.063200000000002" y="0"/>
          <use xlink:href="#..." x="22.0682" y="0"/>
          <use xlink:href="#..." x="27.0732" y="0"/>
          <use xlink:href="#..." x="39.4086" y="0"/>
          <use xlink:href="#..." x="44.9218" y="0"/>
          <use xlink:href="#..." x="54.4544" y="0"/>
          <use xlink:href="#..." x="61.985" y="0"/>
          <use xlink:href="#..." x="68.453" y="0"/>
        </g>
      </g>
  </g>
</svg>
```

where's the actual text?! turns out, to make sure that everything is rendered *perfectly*, Typst turns text into a group of vector glyphs. a pretty clever solution, albeit pretty bad if i want to do anything except print the text on a piece of paper. selecting it in a web browser? nope, there's no actual text! reading with a screen reader? nope, for the same reason!

granted, Typst wasn't made for rendering webpages.

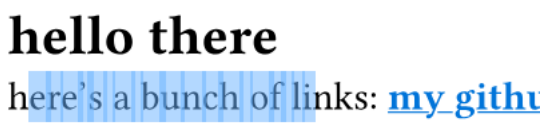
and thankfully, it's **open-source**, meaning if i need a feature i can just add it myself and make a PR so that others benefit from it too! which is **exactly what i did**.

one might ask me, did i revert all this beautiful rendering and made Typst rely on **gasp** the renderers in browsers?! nope. instead, my PR makes Typst add an invisible `<text>` tag to each piece of text it emits:

```
<g class="typst-text">
  <use xlink:href="#..." x="0" y="0"/>
  <!-- ... -->
  <text fill="transparent">the actual text goes here!</text>
</g>
```

it's a bit more involved since i also have to make the selection boxes line up (which turned out to be a *lot* harder than i thought it would be), but you get the idea. if you compile a document into SVG with the forked version of Typst and open that in a browser, you can select text! sadly, browsers are kinda shit at implementing the SVG standard. for example, Firefox **does**

not implement `lengthAdjust` on `tspan` elements, which leads to selection boxes overlapping because the font used to *measure* the text by the browser is not the same as the font used to *render* the text by Typst:



this is what i (and as it turns out, you, the one reading this unless you've decided it wasn't worth the hassle and are currently reading this in PDF) have to live with if i don't want to bundle a whole PDF renderer written in JS (which i *could*, and it would most probably work beautifully, *but* that'd require JS enabled to view the pages at all).

and now let's jump to conclusions like everyone else on the internet:

insert the opposite to "introduction" (outroduction?) here

would a reasonable person do this? no. would i do this again if i knew the problems beforehand? yes, since i'm not a reasonable person.

to be quite honest, writing the wrapper around Typst that gently wrangles it into generating all the content i need and extracts the metadata i need almost felt harder than making the compiler PR... Typst is a nice codebase to work on, i genuinely like it. (other than CLion inserting some random `use` statement which made me think i needed to update Rust while i did *not* need to update Rust).

but hey, if you're reading this, that means the work wasn't all for nothing! yay!