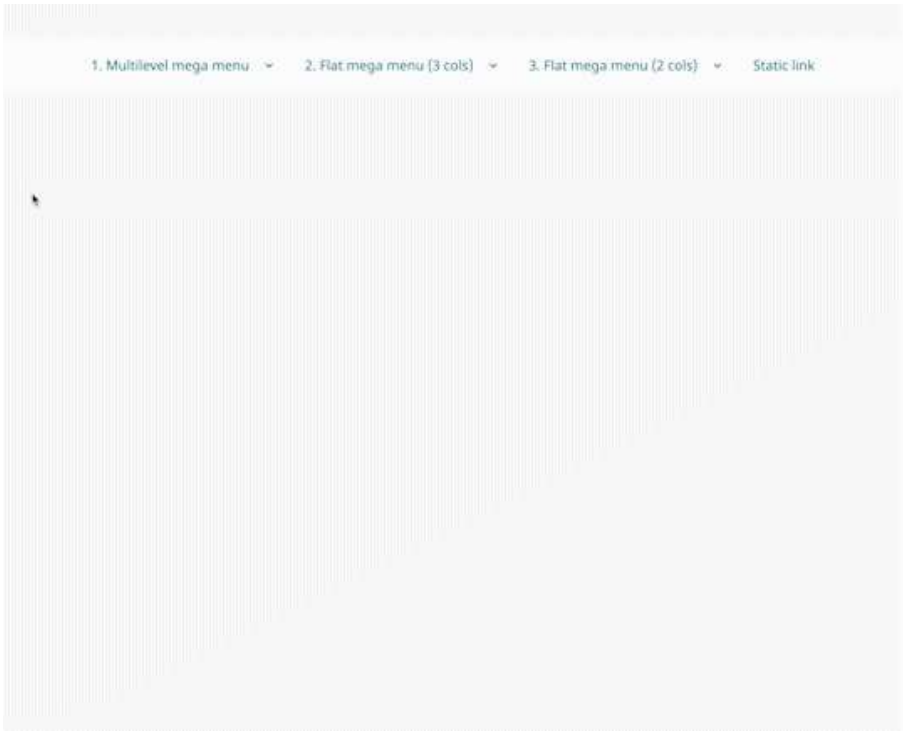
Published in The Startup

Victoria Kronsell Follow
Feb 14, 2020 · 10 min read · Listen

Building a CSS only responsive multi-level mega menu

> *A CSS only approach to building a responsive multi-level mega menu, a component which is traditionally built with JavaScript.*



Why CSS only?

- No DOM manipulations needed
- Framework agnostic — whether you're using React, Angular or just plain ol' HTML and CSS, it works

Accessibility

Using valid and well structured markup helps with accessibility. However, we would need JS to do the following:

- Add full keyboard support
- Toggle attributes such as aria-expanded

Browser support

Different browser apply :focus states differently, Safari, for example, does not apply a :focus state on click. There is also limited support for :focus-within. This means that this solution will not work in every browser.

UX

This is to show what's *possible* to do with *just CSS*, but it doesn't mean that this approach has the best user experience. JavaScript gives you a lot more options to refine interactions.

Setting up

Because we're only using CSS (LESS) and HTML, there is nothing that needs to be set up. Just spin up an HTML page and a CSS file and hit go. Or head to CodePen and follow along using LESS. This article assume a good understanding of advanced CSS (and LESS). Some of the CSS features that we will be using to create this is:

- Various states such as :focus, :hover and :focus-within
- CSS sibling selectors
- The *pointer-events* property
- CSS animations & transitions
- The *transform* property

If you are unfamiliar with any of these, I'd suggest reading up about them before continuing.

Without any styling, it will just look like this:

Open mobile menu

- 1. Multilevel mega menu
  - 1.1 Flyout link
    - 1.1.1 Page link
    - 1.1.2 Flyout link
      - 1.1.2.1 Page link
      - 1.1.2.2 Page link
    - 1.1.3 Page link
  - 1.2 Flyout link
    - 1.2.1 Page link
    - 1.2.2 Page link
  - 1.3 Flyout link
    - 1.3.1 Page link
    - 1.3.2 Page link
  - 1.4 Page link
  - Back
- 2. Flat mega menu (3 cols)
  - 2.1 Page link header
    - 2.1.1 Page link
    - 2.1.2 Page link
    - 2.1.3 Page link
  - 2.2 Page link header
    - 2.2.1 Page link
    - 2.2.2 Page link
    - 2.2.3 Page link
  - 2.3 Page link header
    - 2.2.1 Page link
    - 2.2.2 Page link
  - Back
- 3. Flat mega menu (2 cols)
  - 3.1 Page link header
    - 3.1.1 Page link
      Short decription of link
    - 3.1.2 Page link
      Short decription of link
    - 3.1.2 Page link
      Short decription of link
  - 3.2 Page link header

    This is just static content. You can add anything here. Images, text, buttons, your grandma's secrect recipe.
  - Back
- Static link
- Home

Some things to note regarding the markup:

- A link that has a mega menu dropdown or a flyout needs to have the attribute “aria-haspopup” set to true, we also need to prevent the default behaviour (this is technically using JS, I know — another option would be to use <button> instead of <a>)
- A lot of the CSS is dependent on specific semantics, so it’s important that the proper markup is used

Before we get started, let’s list the requirements for our menu:

- Multilevel mega menu with flyout menus
- Flat mega menus with flexible column layout
- Ability to add static content
- Full responsiveness; on mobile the menu will be a fixed slide-in menu

Creating the desktop menu

Basic styling and layouts

First, let’s add some basic styling and layouts to the main parts of our menu, including the menu bar, menu links and the mega menu dropdown.


Note the following:

- The menu bar is a flexbox container and has relative position
- The mega menu dropdown is a flexbox container and has absolute positioning so it will be displayed below the menu bar
- Dropdown/flyouts link must have aria-haspopup=true and their sibling lists will be hidden

Dropdown and flyout functionality

To create the functionality traditionally created with JavaScript, we will use a combination CSS capabilities.

Mega menu dropdown

The first functionality we will add is the ability to open the mega menu dropdown menu. To make sure our solution is robust, this will be done using a combination of approaches:

- On **:focus** on the link
- On **:focus-within** the list item
- On **:hover** on the mega menu itself, this ensure that it stays open if 1) the link loses focus or 2) it's used in a browser that does not support :focus-within


This is all a bit clunky at this point, so let's add a bit of animation. We know that simulating the slide down animation of jQuery is very hard unless we have a fixed height (or a fixed max height), so instead we will use transform's scale property. By setting transform-origin to top and only animating on the Y axis, we can simulate a slide down animation.

The good news about using transform, and especially scale, for animations is that it's cheap for the browser to animate, which means good performance.

The animation will only be added on :focus on the link, since the rest is only used to keep the mega menu drop open.

Finally, we want to **add active states** to our menu bar links. To make sure the active states stick while the menu is open, we will use a combination of approaches here as well:

- On **:hover** and **:focus-within** on the list item
- On **:focus** on the link

Even though we are using the hover state on the list item, the styling will still only be applied to the link itself.

Now, our menu should look something like this:

Flyouts

Next step is to create the flyout functionality in our multi level mega menu. In the previous step, we hid all siblings of links with flyouts.

To create the functionality to open these nested flyouts, we will use a similar approach to our mega menu dropdown. The flyouts will be opened on hover, but if preferred, this can be done on focus instead.

This is also looking a bit clunky, so we'd want to add some animations here. The same problem applies here; we can't animate width without any fixed values, so we'll use transform scale here as well. Because the flyouts will be sliding out sideways, we'll set transform origin to left and animate the X axis. Note that it's applied on the :hover of the list-item, this ensures the animation isn't re-start when moving the cursor from the opened flyout back to the link (because we are still hover the list-item when we're hovering the flyout because it's a child).

Before we see what it looks like now, let’s add some active states as well.

Et voila:

Flat mega menu styling

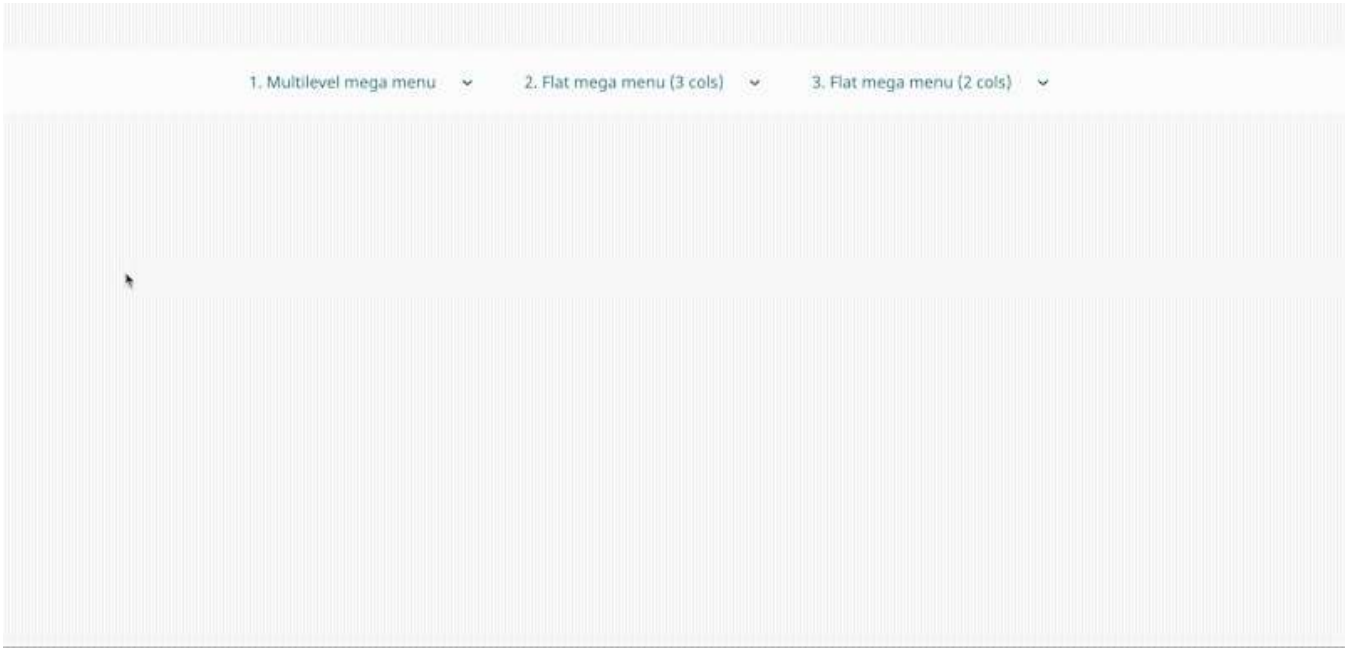
Because our mega menu is already a flexbox, all we need to do is make sure that all the children of our flat mega menu takes up the same amount of space. We'll also add some styling to our header links.

Desktop result

That's it for the functionality on the desktop version of the menu. We'll add a few more styles to it:

- Down arrows for menu bar dropdown links
- Right arrows for mega menu links with flyouts
- Borders between mega menu columns

And with that, we've got a pretty nice looking menu:



Creating the mobile menu

Before continuing to the mobile menu, let's do a bit of cleaning up. I've taken all the code in the previous examples and consolidated it. I moved desktop specific styles to inside a media query so we don't have to overwrite those styles for the mobile version. Here is our starting point for the next step:

Basic styling and layouts

We will start setting up our mobile menu by adding some basic styles and layouts to our different elements. On mobile, the menu will be a fixed left aligned menu that slides in when a button is being tapped. The multi-level mega menu's subsequent menus will be displayed as accordions.

The only thing we'll see on the page now is our mobile menu trigger button. The menu itself is positioned outside of the screen, and the mega menus are positioned outside of the mobile menu. We are actually animating the left position of the menu here, and not using transform. It seems that in iOS, if we use transform to offset the menu, clicking on the "Open mobile menu" button also registers a click on the "Home" link within the menu.

A note on :focus on mobile

Mobile devices don't actually support :focus states, but :hover can be used to simulate the same functionality. This means that we can't use the same code for our mobile menu used on a desktop, and our mobile menu used on touch device. The former is probably an edge case, but for the sake of thoroughness I will add support for both.

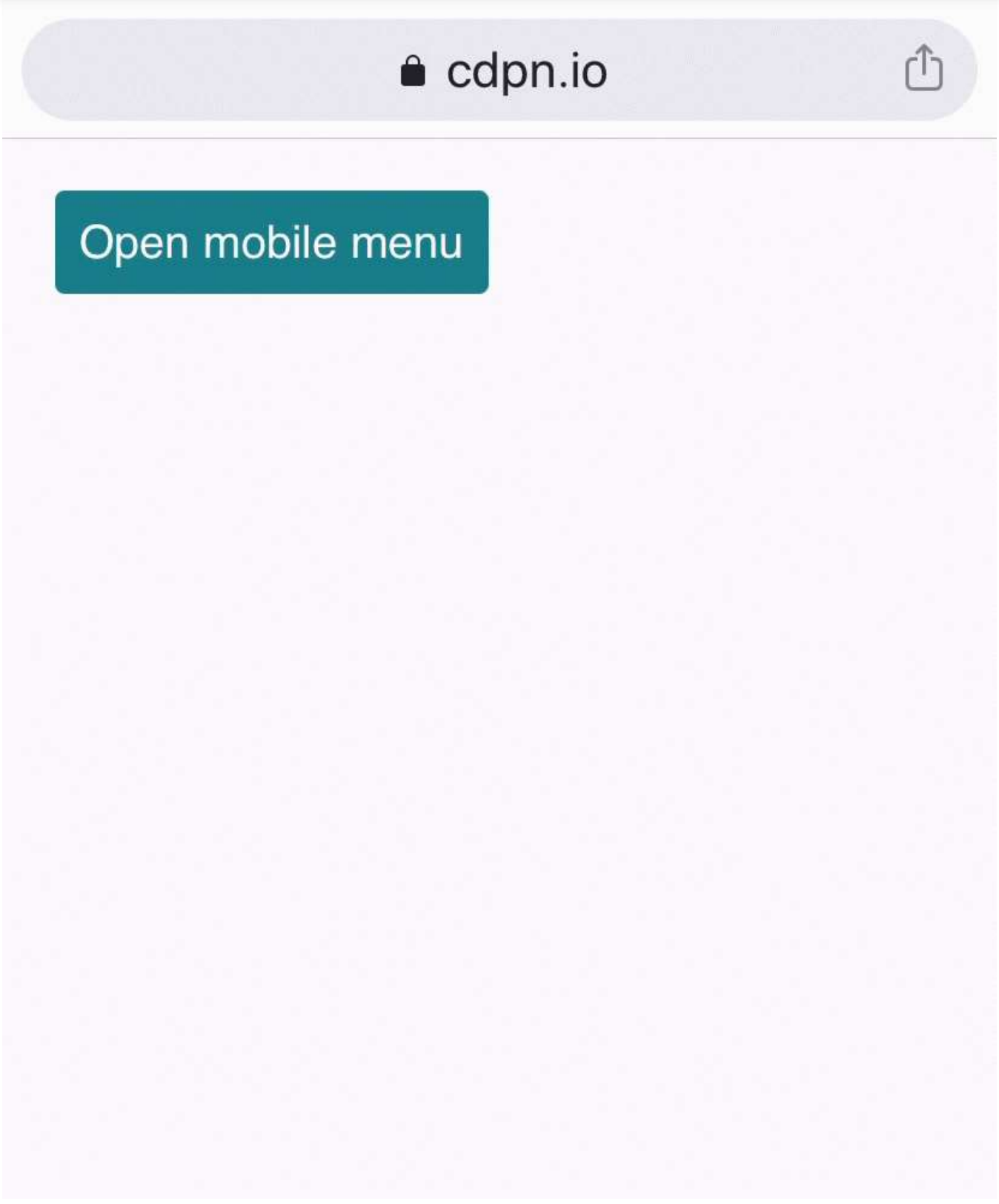
To target just touch devices, we can use the hover media feature, which checks if the user's primary input mechanism can hover over elements.

Opening the mobile menu

For this to work, the menu must be a sibling of the mobile menu button. As you might have guessed, we will use :hover to trigger a position changed of the menu as we tap the button, but to support non-touch devices as well, we will also use :focus.

To avoid the menu closing as soon we start interacting with it (because as soon as we do that, our button will lose focus/hover), we're going to use :hover and :focus-within on the menu itself to keep it open.

We've now got basic functionality to open the mobile menu

Opening the mega menus

We will use the same approach to open our mega menus as well.

Back buttons

This is one of the trickier parts to do without JavaScript. Remember, the only way of opening our menus, is using :focus or :hover on a sibling or a parent of the menu. Our back buttons are part of that menu, and because of the way that CSS works, we can't target the parent of the back button to change the position of the menu.

So, how can we make sure that the menu that contains the back button that has been clicked, loses its focus?

First, we need to remove the back button from the list itself. To do this, we need to set a fixed height on the button and then use the same value to negatively offset it vertically, and positively offset the mega menu. We'll also add some basic styling to the back button.

Now, it will look the same, but the back button is actually sitting outside of its list. Notice the **pointer-events: none;** — this prevents all clicks and states on an element. This means in theory, when we click the back button, we are actually clicking *what's behind it*. In this case, that is the previous menu. This also means, that when we click it, the opened mega menu will lose its focus.

This causes another issue though… Because we are clicking what's behind the back button, we are actually clicking the "Home" link in the first level of our menu which triggers the default behaviour of that link, which takes us to a page.

To avoid this, we're gonna set the visibility of the home link to hidden when a mega menu is open. At the time we click the back button, the link is hidden, but as soon as

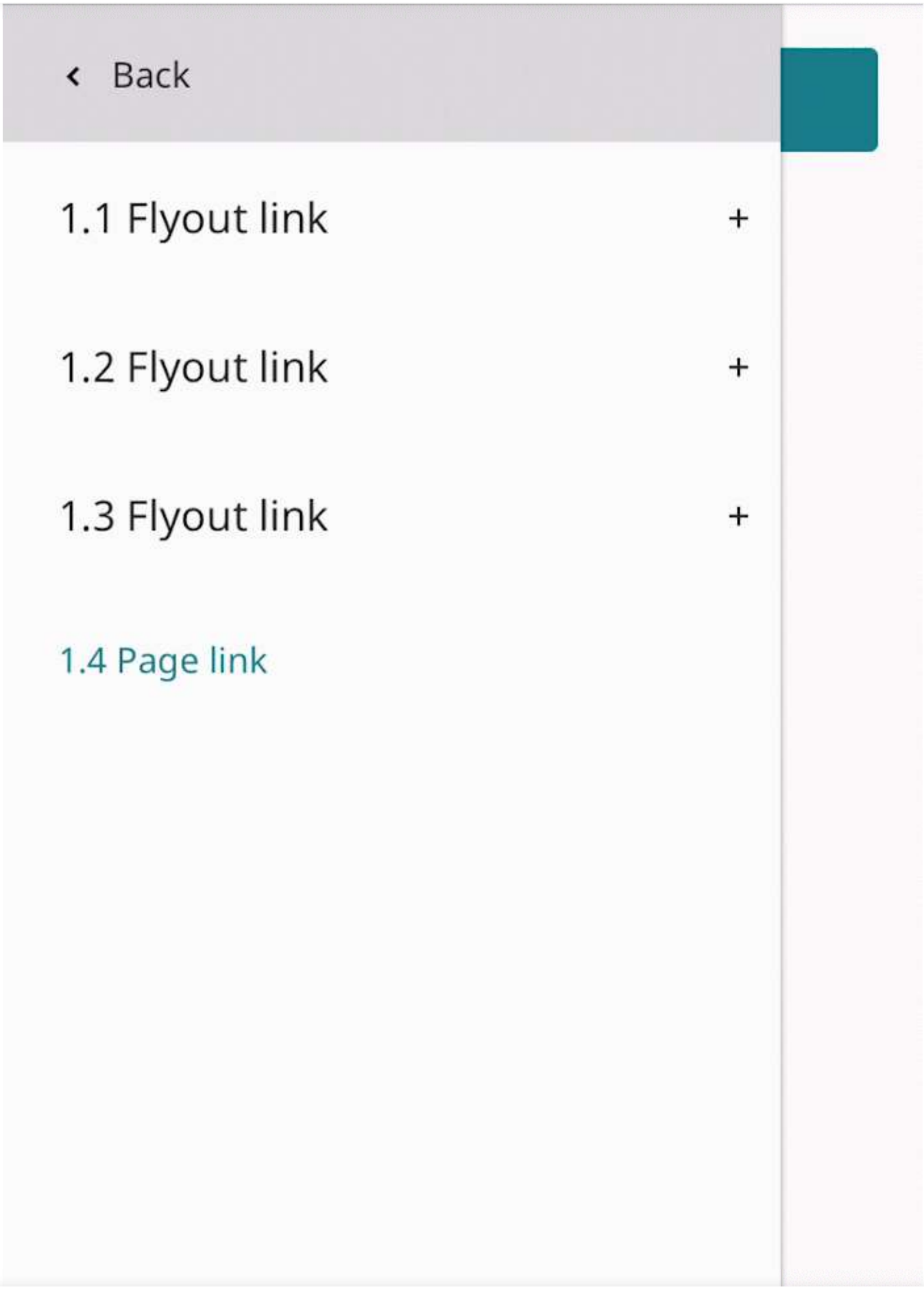

Expanding flyouts

Next we will create the functionality to expand our subsequent flyouts. Let's first add some styling.

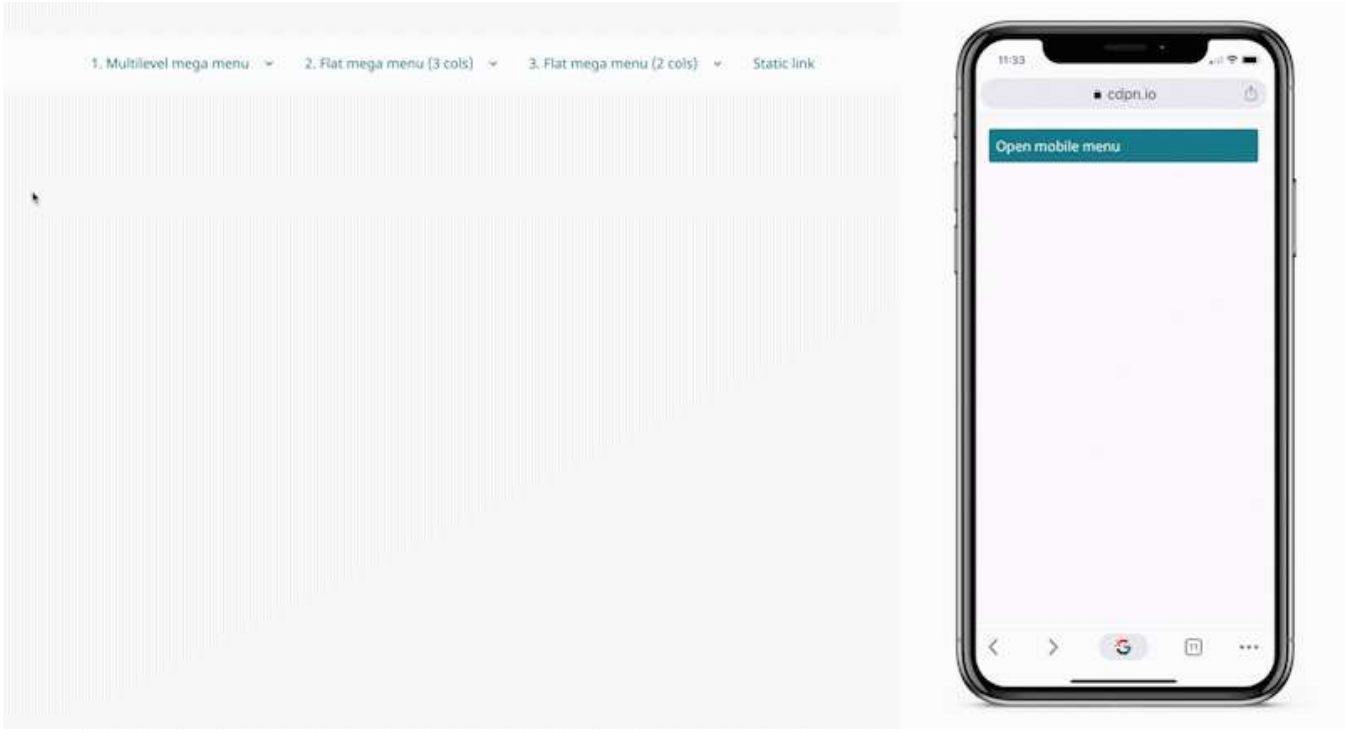
We will create a nested accordion-like functionality using a combination of transitions and animations. Let’s add our functionality before going through it.

We will be re-using our dropdown animation that we use for our desktop mega menu. The issue we had previously when trying to use transform to open the mobile menu and having the click on the button also trigger a click on the link inside menu is also an issue here. If we would only use transform, when we click to expand the second item, for example, it will actually trigger a click on the third item.

That's where max-height comes in. This is a bit of a hack, but it works. Because we're using max-height as a transition, there is a slight (.1s) delay to the previously opened accordion closing. This prevents the click to be triggered on several items.



That's it. We've created our CSS only multi-level responsive mega menu.



Codepen here: https://codepen.io/vixxofsweden/pen/xxGGYOE

Conclusion and next steps

CSS is very powerful, and I believe in many cases underestimated. I think traditionally we're inclined to turn to JavaScript for functionality and interactivity, but this project shows that if you just get creative, the capabilities of CSS can really surprise you.

JavaScript enhancements

Creating a CSS only version was an experiment to see how far I could get without JS. To create a production ready component of this, I will be adding JavaScript to ensure same functionality in all browsers, create a smoother experience and remove some of the issues caused by a CSS only approach.