

Organizing and Implementing Storybook with Chromatic in a Monorepo

Author: Danyon Satterlee

Claps: 21

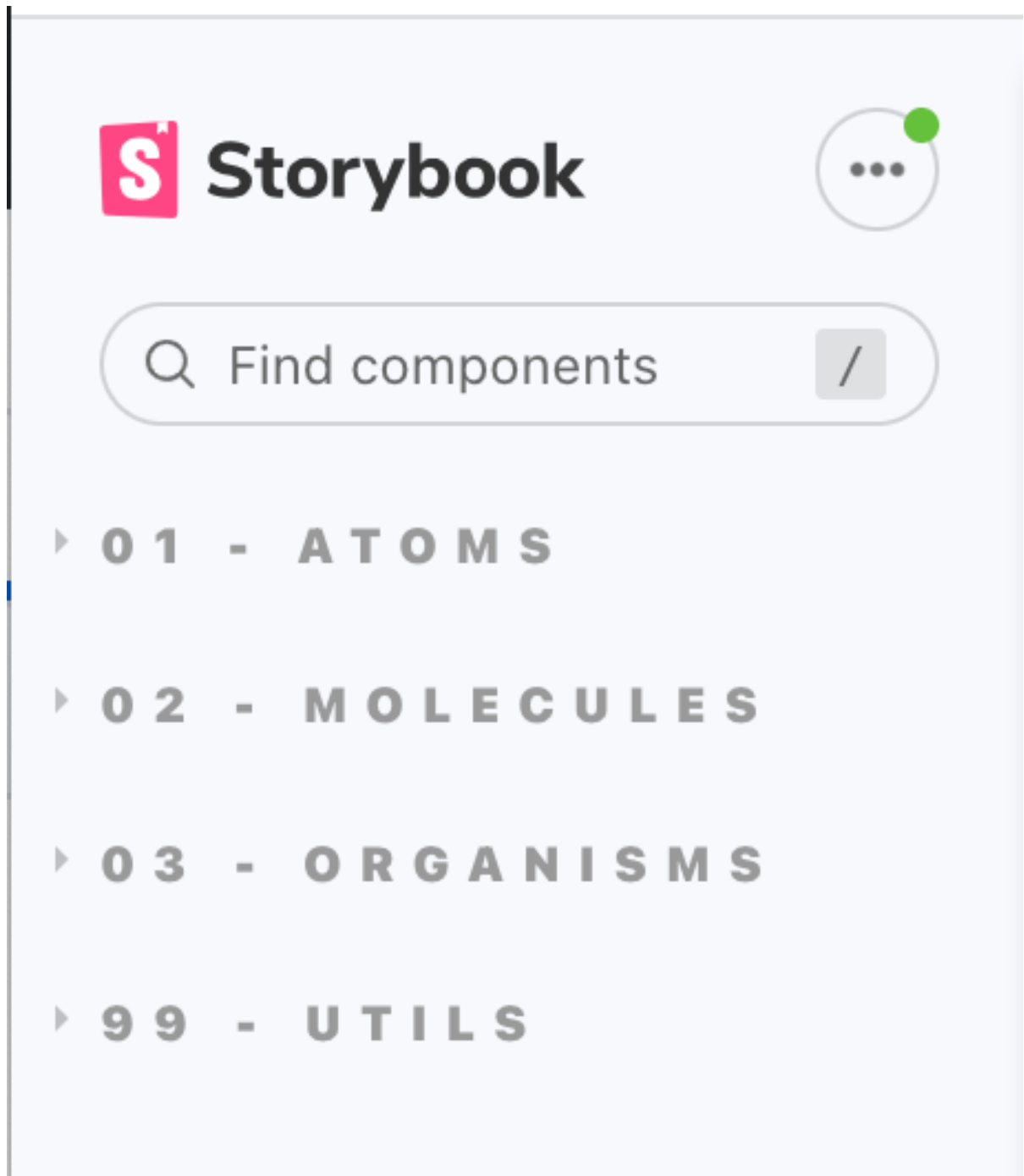
Date: Oct 2

Here at Klaviyo, our engineering team develops our customer products *and* products that make our own business run better. My team, K-Ops Marketing, is responsible for the technology of our external sites such as klaviyo.com (the marketing website), its international relatives, and help.klaviyo.com. We want all of these websites to look and feel the same to give our users a consistent experience. To achieve this, we house all of the sites in one monorepo and share a component library.

To help us maintain consistency, we use [Storybook](#) to build out our React components in isolation, and then use [Chromatic UI Tests](#) for visual regression testing. Our growing library currently has 106 components with 1229 variations of those components. In this post I discuss how we stay organized by categorizing components, rendering multiple variations of a component in a single story, and being efficient with Chromatic snapshots.

Atomic Design

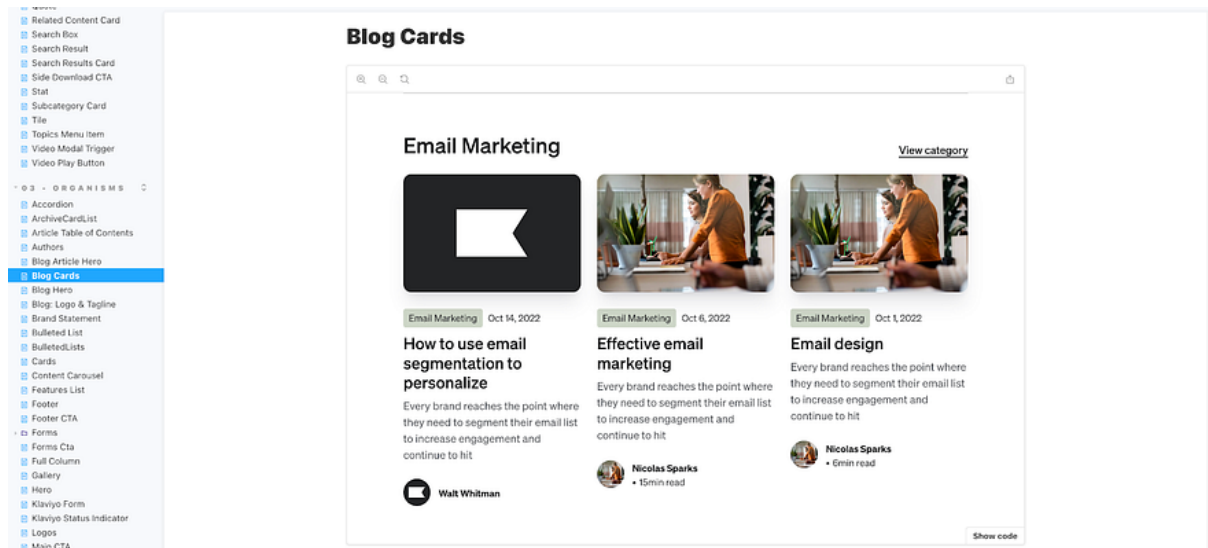
Development, reuse, and testing are all aided by arranging React components in a hierarchy and organizing Storybook and the source tree according to a consistent methodology. We chose [Atomic Design](#).



Screenshot from Storybook showing our four categories.

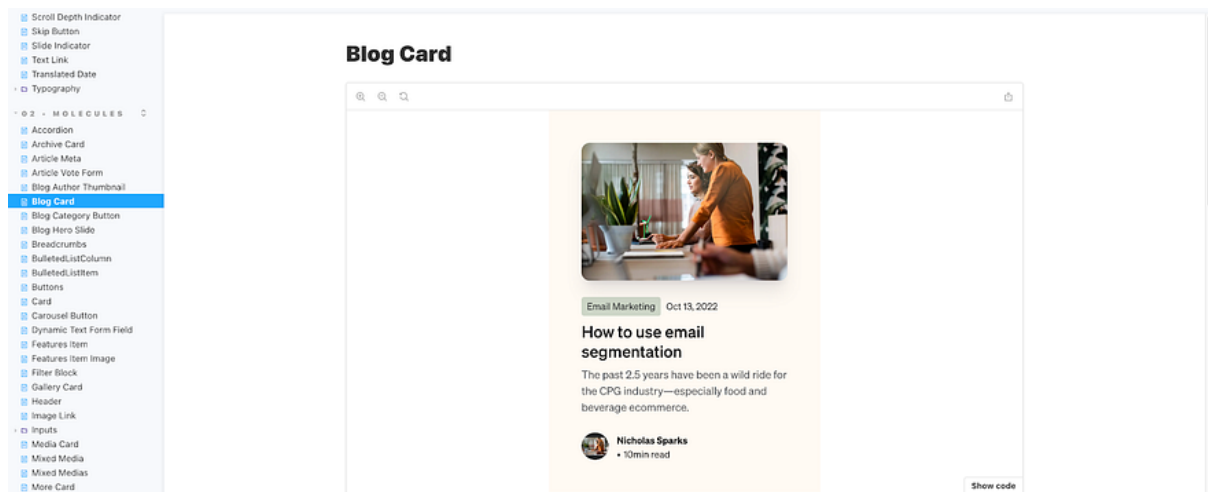
The most fundamental components are *atoms*. Examples of atoms are elements like buttons, links, and typography. *Molecules* are components with multiple atoms, for example, a breadcrumb component. *Organisms* are a combination of atom and molecule components and can stand alone on a page. (It’s also at this level that designers hand off components to our team.)

Here’s an example. At the organism level, our Blog Cards component looks like this:

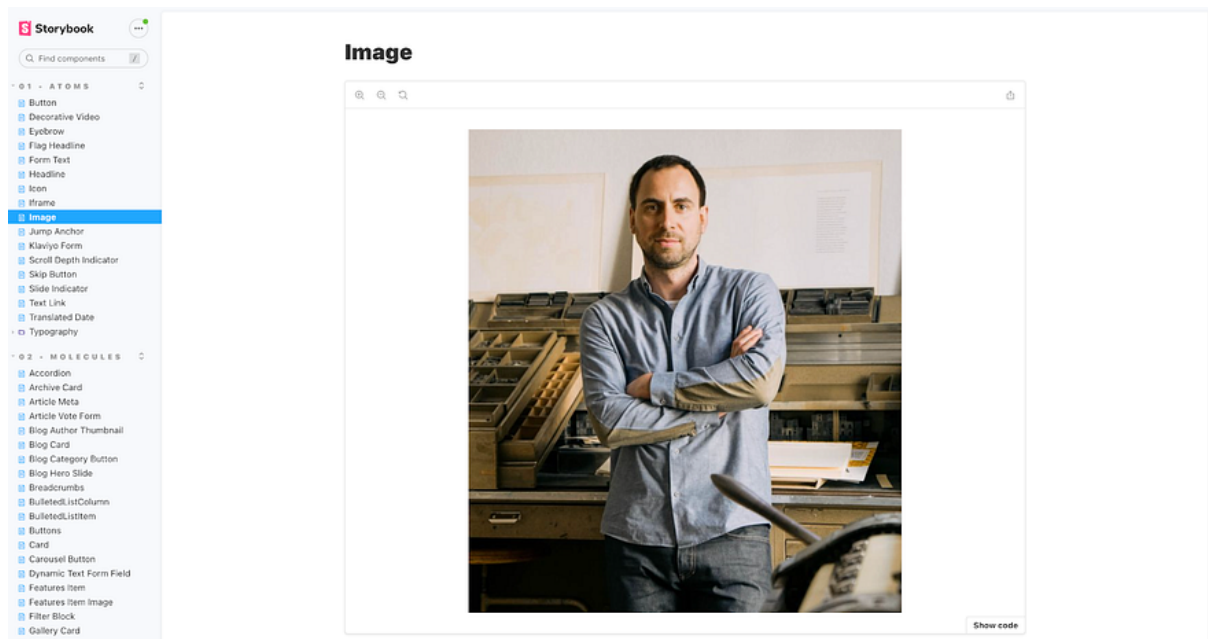


Screenshot from Storybook showing our Blog Cards component. You can get a sense for our other organism-level components from the index at left.

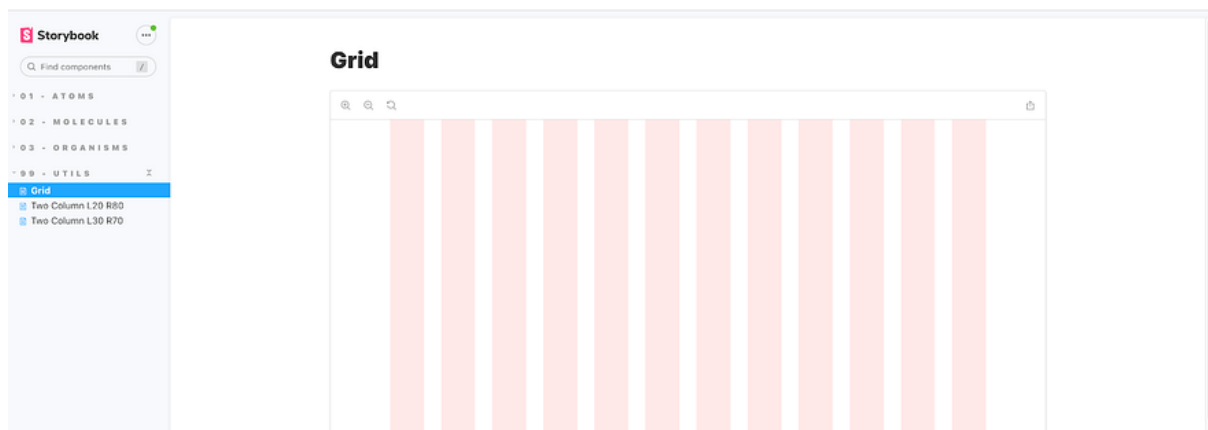
The primary molecules of Blog Cards are the repeating individual cards. The Blog Card (singular) is its own molecule and has its own Story:



Finally, we break down the individual Blog Card into its atoms. One example of an atom inside the Blog Card is Image:



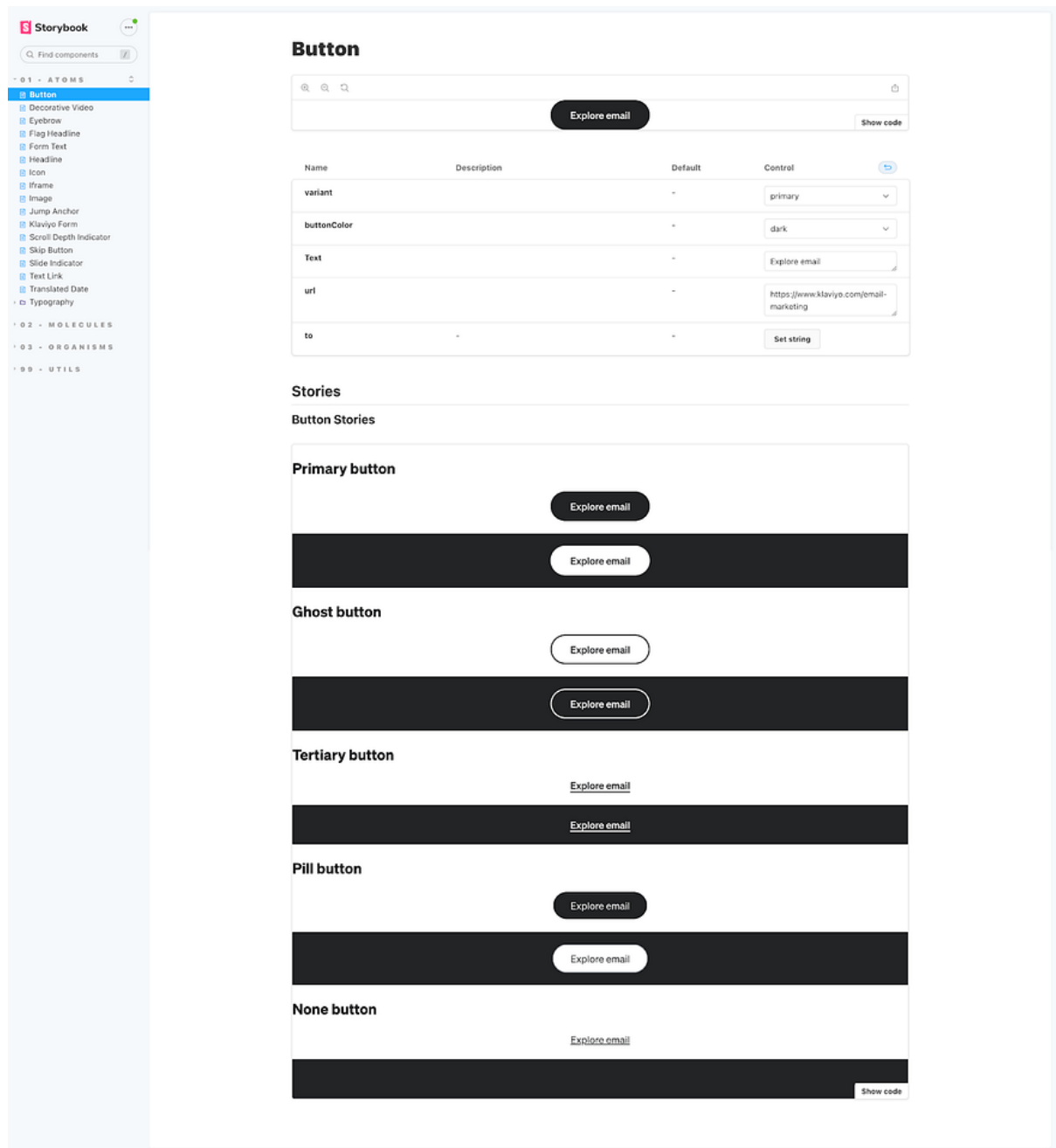
We also have a Utils category. This is not in the typical Atomic Design system. We use it to house layout elements like grids.



Storybook file setup

Now that we understand how we break down our components, let's discuss how we prepare our Storybook files so that we can do automated visual regression testing. We use Component Storybook Format 3 (CSF 3). In the file we have two stories, *Default* and *Variations*. The Default story is for our engineers and other stakeholders to play with the component and read documentation about each prop type. The Variations story shows all of the variations of the component on a single canvas and it's primarily intended for Chromatic snapshots.

As an example, this screenshot shows the Default story and Variations story for our Button component:



Approach to Chromatic builds

If you're operating at even moderate scale, and you follow the default Chromatic advice of running tests on every git push, you'll quickly run into snapshot limits. For example, our K-Ops Marketing team is currently allocated 60,000 snaps per month (out of a larger Klaviyo account), and if we didn't optimize, we would hit our cap within a day or two. So to make our builds and tests fast, avoid clutter, and save money, we configured our process as follows.

Use TurboSnap

As Chromatic says, [Turbosnap](#) relies on using Git and Webpack's [dependency graph](#). It identifies component files and dependencies that have changed, then intelligently snapshots only the stories associated with those changes. This cuts down on the number of snapshots taken,

and it helps identify where the changes are. When we review components, we only have to review the ones that actually changed, or if none have changed, then it tells us that as well.

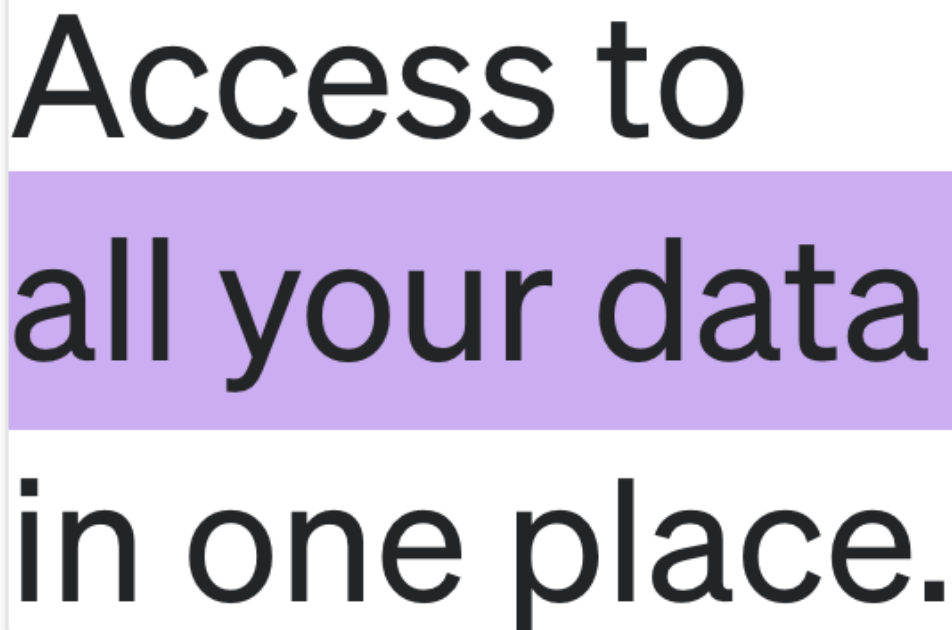
Customize the GitHub Action

We do not want Chromatic to run on every push. This is our criteria for the GitHub action. We should note that we utilize GitHub's [squash and merge](#) feature. When using this feature with the proper setup with Chromatic, [Chromatic uses the most recent commit](#) to our `main` branch as the baseline. This is the criteria for our GitHub action to run:

- It should only run on an open PR against `main`
- It should run on every push to the opened PR against `main`
- It should only run if files have changed within the directory where the components live.
- It should not run if the user is `dependabot`
- It should not run if the PR is opened as a [draft](#)

Use a single Variations story

I described this set up above. The approach also helps limit snapshots. For example, our Flag Headline component has over 120 variations with optional features. By rendering all of them in our Variations story, we keep the file simple, avoid clutter, and get testing coverage while limiting the number of stories snapped to 2 rather than 121. We've found that using nested loops to render variations works well. For example, here's our Flag Headline component:



Access to
all your data
in one place.

And here's the code used to render the Variations story:

```
export const flagColors = [  
  'charcoal', 'lavender', 'lavenderShade', 'lemon',  
  'lightSand', 'ocean', 'poppy', 'sage', 'sageShade', 'sageTint',  
  'sand', 'sandShade', 'sandTint', 'transparent', 'white',  
];  
  
export const headlineVariants = [  
  'charcoal', 'lavender', 'lavenderShade', 'lemon',  
  'lightSand', 'ocean', 'poppy', 'sage', 'sageShade', 'sageTint',  
  'sand', 'sandShade', 'sandTint', 'transparent', 'white',  
];
```

```

    'display1', 'display2', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'
  ];

const StyledAsVariations = ({ ...args }) => (
  <>
    {flagColors.map((flagColor)=> (
      <s.StackedItem>
        <s.TypographyContainer>
          <h2>Flag headline with a {flagColor} flag color</h2>
          {headlineVariants.map((heading) => (
            <s.StackedItem key={`_${heading}-${args.flagColor}`}>
              <FlagHeadline
                content={args.content}
                {...args}
                styledAs={heading as TStyledAs}
                flagColor={flagColor as DesignSystemColorTypes}
              />
            </s.StackedItem>
          ), )}
        </s.TypographyContainer>
      </s.StackedItem>
    ))}
  </>
);

export const FlagHeadlineStory: TStory = {
  render: ({ ...args }) => <StyledAsVariations {...args} />,
};

```

This simple looping technique saves time, makes for more maintainable story code, and often lets us visually spot a combination we wouldn't have thought to test manually.

What counts as a variation

We use Atomic Design to guide what counts as a variation. At each level of the hierarchy, we only test what is customizable at that level. For example, at the *organism* level, we may be rendering it with or without a headline, and with or without an image, but we don't look at variations of the *molecules* and *atoms*. Those variations will be captured in the stories for the molecules and atoms. This way we follow a consistent pattern for our components and we avoid double testing of a component.

Screen size

A navigation component can change drastically from a desktop view to a mobile view. The desktop may have navigation items listed out for the user to see, whereas a mobile nav typically has a hamburger menu to see most of the content. To capture these scenarios we take snapshots of each organism component at our designated viewport sizes. We do not take snapshots of molecules or atoms at the different screen sizes because they do not typically change much from desktop to mobile views.

Conclusion

K-Ops Marketing is responsible for a growing number of Klaviyo web properties including our main external site. We structured our component library so that we can achieve quality, consistency, and attractive design across pages, sites, and time. We use Storybook and Chromatic and came up with best practices that let us be efficient and stay organized!