

ISABELA MOREIRA

THE STARTUP'S GUIDE TO CREATING A DESIGN SYSTEM

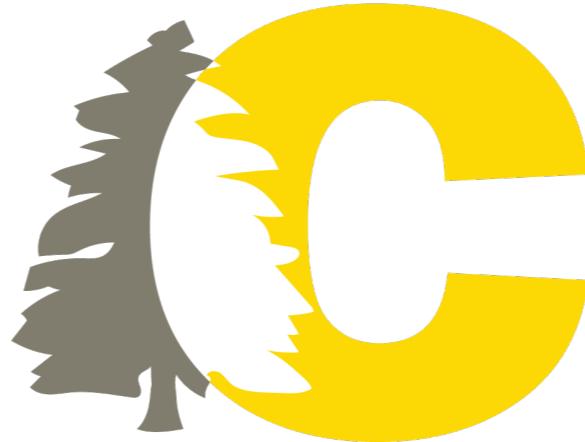
ChicagoJS 2019

 @isabelacmor

HI 🙌



ChicagoJS 2019



 @isabelacmor

Before we dive in, I want to tell you guys a bit more about me. The two most important things:

1. This is my dog, Atlas. He's very excited about this talk and I hope you all are too!
2. I work at a tiny startup in Seattle called CedarAI. We work on railroad technology and the entire industry is still run on pen and paper. This is a huge UX challenge and it's even harder to tackle for a brand new product in a brand new space. In this talk, I'll be sharing how to use a design system to take on this kind of challenge.

YOUR BURNING QUESTIONS 🔥

- ▶ What's a design system? Why is it important?
- ▶ How do you create a design system from scratch?
- ▶ How do you implement this design system in code?

ChicagoJS 2019

 @isabelacmor

Some of the key points we'll cover in this talk will be:

- * Learning what a design system is and why it's useful
- * Creating a design system from scratch
- * Implementing a design system from scratch

Most of this talk will show the tools I use specifically at work, but the strategies we talk about are pretty platform and tool agnostic.

DESIGN TOOLS



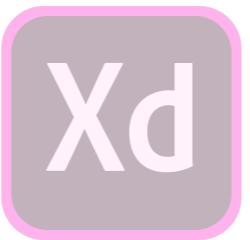
ChicagoJS 2019

 @isabelacmor

Before we jump in, I want to set some context here for anyone who doesn't have a design background. A design tool is similar to Photoshop, which I'm sure everyone here's familiar with, and there's a bunch of different ones that are popular in the community. They usually let you design and do prototyping.

A lot of the screenshots I'm going to share are from a design tool called Figma, just because it's what I personally use at work.

DESIGN TOOLS

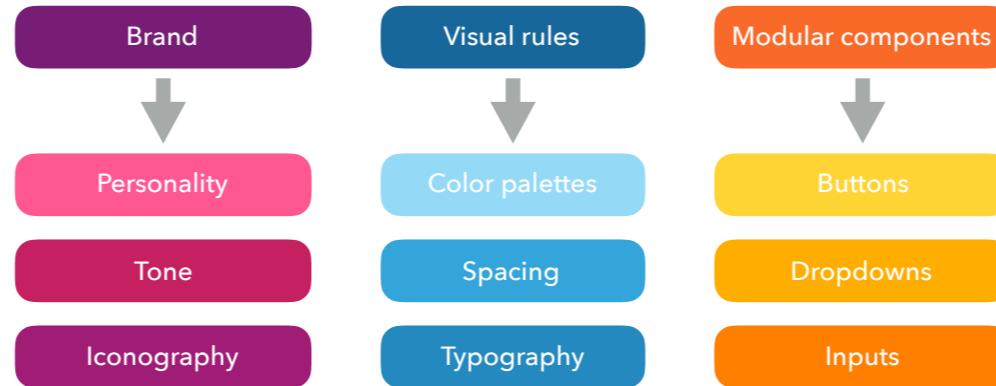


ChicagoJS 2019

 @isabelacmor

WHAT'S A DESIGN SYSTEM?

- ▶ Collection of visual rules and modular components



ChicagoJS 2019

 @isabelacmor

Very simply put, a design system is a collection of visual rules and modular components to guide and direct a product's design, from interaction models, to interfaces, to a product's personality and tone.

When you define your product's brand, you influence the visual rules you'll follow, which in turn influence the kinds of components you'll build.

WHY IS A DESIGN SYSTEM IMPORTANT?

LOGO

BUY NOW

Introducing the world's best application

DOWNLOAD FREE TRIAL
Try out without obligations for 30 days!

ChicagoJS 2019

 @isabelacmor

So you may be wondering why any of this is even important. You can just pick a few colors, pick a font, and be done with it. After all, you may only be designing and building a single page.

WHY IS A DESIGN SYSTEM IMPORTANT?

LOGO

BUY NOW

Introducing the world's best application

DOWNLOAD FREE TRIAL
Try out without obligations for 30 days!

SAVE TIME

SO SIMPLE TO USE

CLOUD INTEGRATION

ChicagoJS 2019

 @isabelacmor

But then you start adding features...

WHY IS A DESIGN SYSTEM IMPORTANT?

LOGO

BUY NOW

Introducing the world's best application

DOWNLOAD FREE TRIAL
Try out without obligations for 30 days!

SAVE TIME

SO SIMPLE TO USE

CLOUD INTEGRATION

Read our blog

Send us feedback

Get to know us

...

WHY IS A DESIGN SYSTEM IMPORTANT?

The image shows a wireframe of a website page. At the top left is a placeholder for a logo. To its right is a horizontal navigation bar with links: HOME, FEATURES, ABOUT US, SUPPORT, and a prominent black button labeled BUY NOW. Below the navigation is a large heading: "Introducing the world's best application". Underneath this heading is a large dark gray rectangular area with a white 'X' through it. In the bottom right corner of this area is a call-to-action button: "DOWNLOAD FREE TRIAL" followed by the smaller text "Try out without obligations for 30 days!". Below this main section are three smaller boxes, each with a large white 'X' through it: "SAVE TIME", "SO SIMPLE TO USE", and "CLOUD INTEGRATION". At the bottom of the page are three links with three-line dropdown menus: "Read our blog", "Send us feedback", and "Get to know us".

... and once you're done with your one page, you decide to add a few more pages. By this point, you probably have styles duplicated everywhere. There are subtle inconsistencies across your UI. Maybe your buttons have slightly different padding. Or your header text is a different size on different pages. It starts to get very hard to keep track of all your components and make sure all of these look the same.

WHY IS A DESIGN SYSTEM IMPORTANT?

- ▶ Consistent colors and fonts
- ▶ Low overhead for design choices
- ▶ Reusable components
- ▶ Saving time and resources = saving 

ChicagoJS 2019

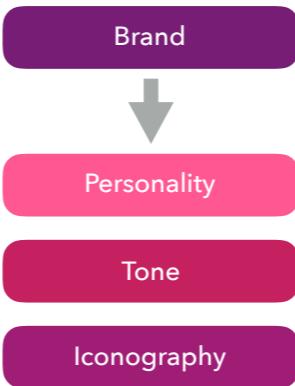
 @isabelacmor

These are the exact problems design systems are built to solve. With a design system, you'll get consistent colors and fonts across your product because you've already done all the overhead of the work to make these choices. You won't have to make decisions like "should I make this header 13px or 14px?" because you'll have made all those decisions already. Most importantly, you'll end up with a library of reusable components so if you ever decide to change a border radius or font-size, you'll only need to make a change in one place!

These choices don't apply to just the current project you're working on. These apply across all your products! If you create a fully-featured design system, you'll be able to use it across all your products, even if they use different technology stacks. The tools you choose to implement your design system in shouldn't matter - the end result of your mocks should be the same.

Hopefully by now I've convinced you to create and use a design system, so let's jump into how we can get this done.

CREATING A DESIGN SYSTEM



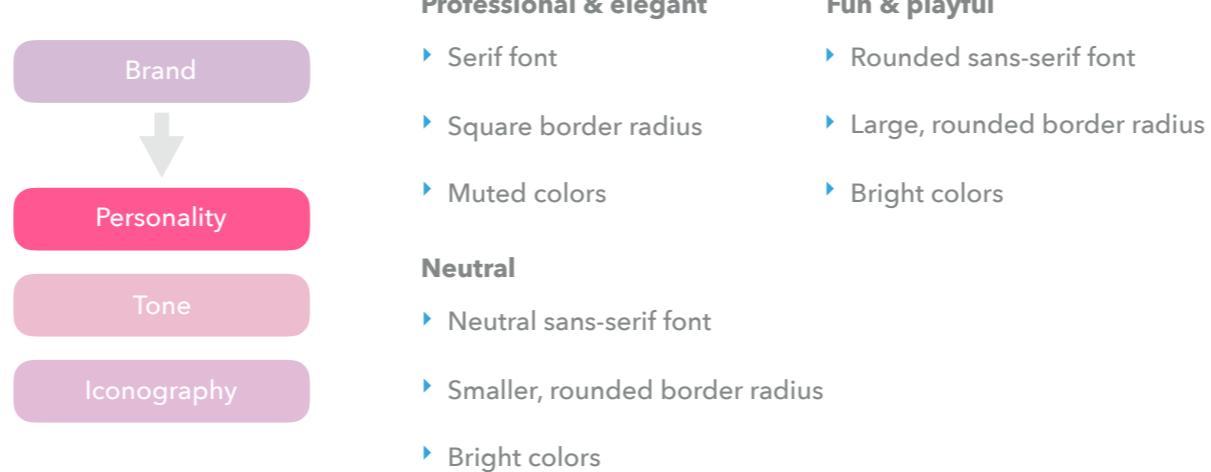
ChicagoJS 2019

 @isabelacmor

When we first start creating our design system, we'll need to start with the smallest, most foundational building blocks before working our way up to larger and more complex pieces.

We'll do this by defining our brand.

CREATING A DESIGN SYSTEM



ChicagoJS 2019

 @isabelacmor

First, we'll want to decide our product's personality. This will influence choices like color, fonts, and borders.

The screenshot shows the Uber website homepage. At the top, there's a black navigation bar with the Uber logo, links for "Our products", "Our company", "Safety", and "Help", and buttons for "EN", "Log in", and "Sign up". Below the navigation is a horizontal menu bar with icons for "Earn", "Ride", "Eat", "Freight", "Business", "Transit", "Bike", and "Fly". The "Earn" icon is highlighted with a blue underline. The main content area features a large image of a smiling man with a mustache, wearing a grey argyle sweater vest over a blue shirt, holding a white envelope. To his left, the text "Drive with Uber" is displayed in a large, bold, black font. Below it, a promotional message reads: "Earn at least \$2100 for your first 300 trips,* in Seattle". There are two calls-to-action: a blue button labeled "Sign up to drive →" and a link "Learn more about driving and delivering >".

First, we'll want to decide our product's personality. This will influence choices like color, fonts, and borders.

[DRIVER](#)[RIDER](#)[Cities](#)[For Business](#)[Help](#)[Download the app](#)

The whole city. In the palm of your hand.

Enter mobile phone number[SIGN UP](#)

We'll send you a text with a link to download the
app.



First, we'll want to decide our product's personality. This will influence choices like color, fonts, and borders.

CREATING A DESIGN SYSTEM



Professional & elegant

- ▶ Less personal
- ▶ "To confirm your identity, please provide a phone number where we can send you a verification code."

Fun & playful

- ▶ More casual language
- ▶ "Just to make sure this is really you, where can we send you a verification code?"

Neutral

- ▶ Neutral language
- ▶ "To get started, please provide a phone number to send a verification code."

ChicagoJS 2019

 @isabelacmor

We should also decide what tone we want our product to have. This will not only affect all the copy on our website, but also any graphics or images we might display.

CREATING A DESIGN SYSTEM

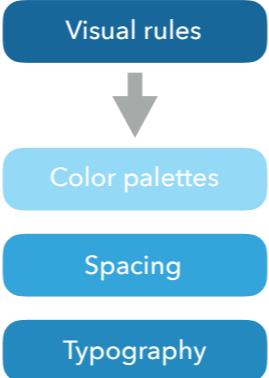


ChicagoJS 2019

 @isabelacmor

Finally, we should decide on iconography. If you're a small team or company, you probably don't have the resources to design an entire icon set and it's probably not a good use of your time any way.

CREATING A DESIGN SYSTEM

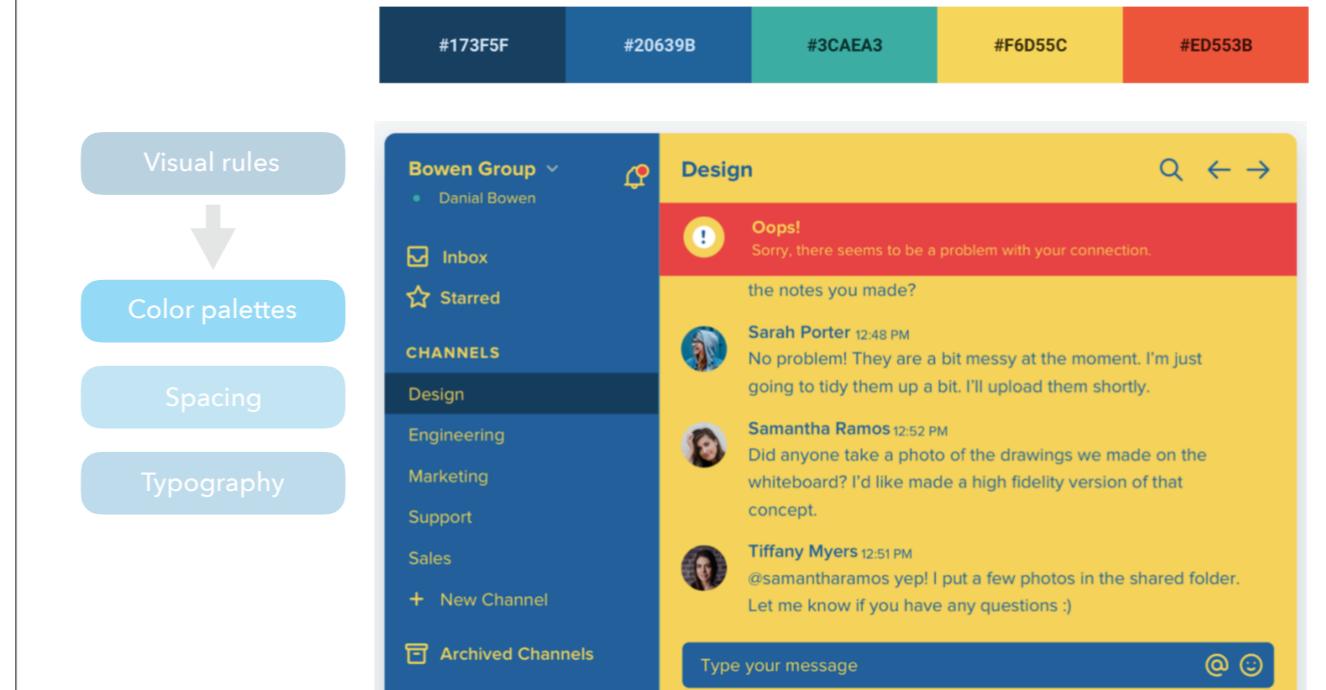


ChicagoJS 2019

 @isabelacmor

Once we have a general idea of the personality we want to portray in our product, we can start thinking about the basic visual rules we want to follow.

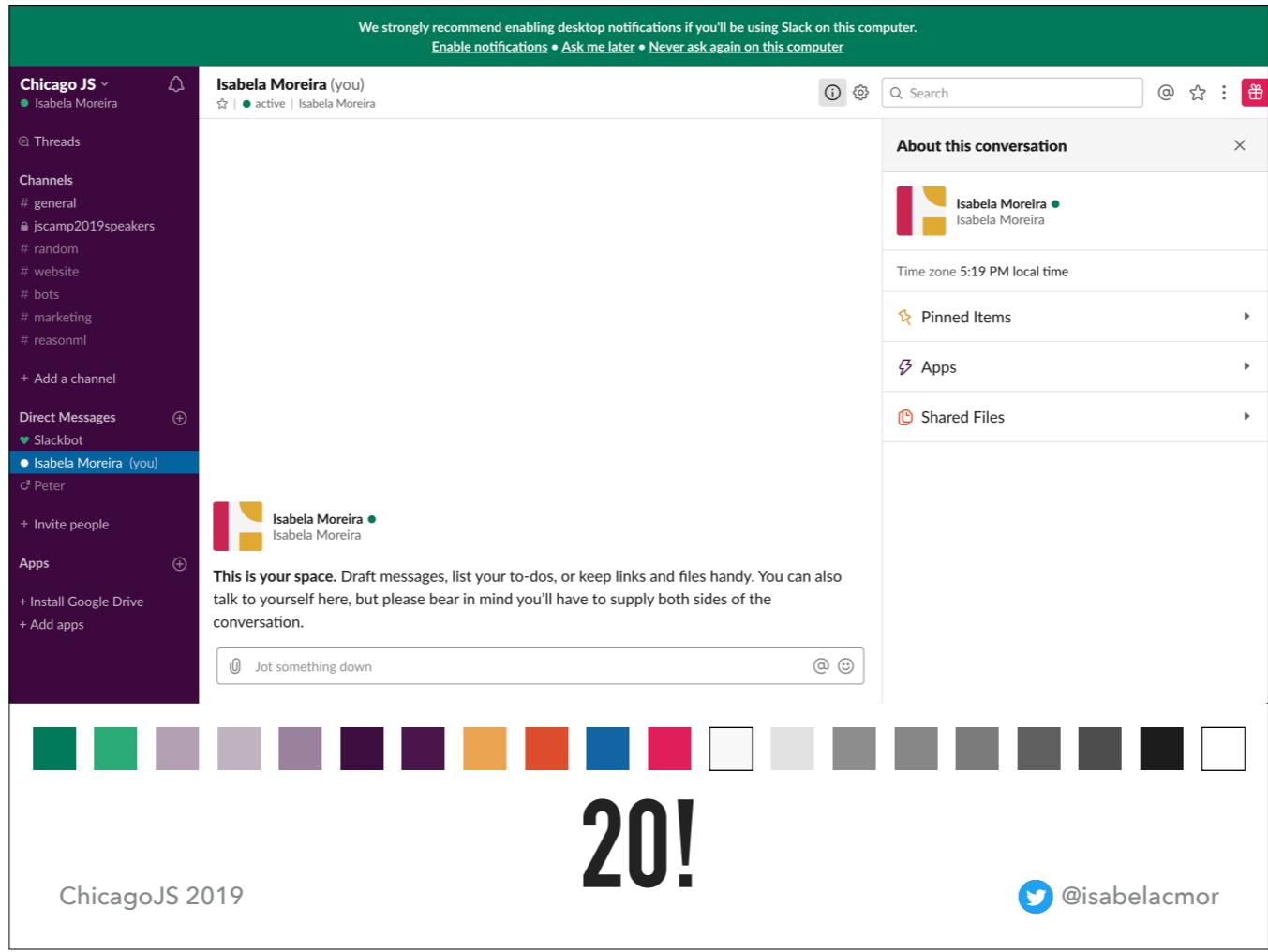
CREATING A DESIGN SYSTEM



ChicagoJS 2019

 @isabelacmor

One of the first things we should consider is our color palette. You might be tempted to head to your favorite color palette generator and grab a nice-looking palette. This is fine as a starting point, but we still need to create more colors. I mean, just imagine creating a UI with only these 5 colors.



Can I get any guesses of how many colors are in the screen (not counting my profile pictures)?

How many of you think it's more than 5? More than 10?

I counted 20 different colors in this one screen alone! Imagine if you were trying to keep track of all these colors by hand.

CREATING A DESIGN SYSTEM

Visual rules



Color palettes

Spacing

Typography

COLORS

NEUTRAL COLORS

NEUTRAL-900	#16263D
NEUTRAL-800	#343CA8
NEUTRAL-700	#33A35B
NEUTRAL-600	#265769
NEUTRAL-500	#909FB6
NEUTRAL-400	#D4DDEB
NEUTRAL-300	#DFE7F1
NEUTRAL-200	#ECF1F7
NEUTRAL-100	#F5F7FA
NEUTRAL-WHITE	#FFFFFF

PRIMARY COLORS

PRIMARY-900	#0C394F
PRIMARY-800	#0E3F58
PRIMARY-700	#104965
PRIMARY-600	#135677
PRIMARY-500	#31789B
PRIMARY-400	#5A96B5
PRIMARY-300	#A1C9D8
PRIMARY-200	#CFEAEE
PRIMARY-100	#E9F5FB

ACCENT COLORS

YELLOW-900	#59481D
YELLOW-800	#816018
YELLOW-700	#AB7F21
YELLOW-600	#CAC022
YELLOW-500	#FCB804
YELLOW-400	#FFE555
YELLOW-300	#FEEDB8
YELLOW-200	#FCF3C1
YELLOW-100	#FCAFEB

RED-900	#49020B
RED-800	#611916
RED-700	#7E2521
RED-600	#AA3029
RED-500	#CB4039
RED-400	#D15754
RED-300	#DE8987
RED-200	#EAAEAC
RED-100	#F9E9E8

ChicagoJS 2019

 @isabelacmor

Instead of keeping track of HEX values manually or using a very limited color palette, we'll want a pretty fully fleshed out color palette. There's several great guides online on how to create these, but as a general rule of thumb, you'll need about 5-10 total shades of each color you choose.

CREATING A DESIGN SYSTEM



One of the most frustrating things in design can be choosing between different sizes and spacing on a pixel level and not knowing if you should have a padding of 16px or 18px. Not having a consistent spacing guideline leaves you with a huge range of hardcoded arbitrary values and you'll end up wasting a lot of time just choosing between two similar options.

A lot of the spacing guidelines can be left up to you and your product's use case, but there are a few suggestions to keep in mind.

1. Never choose values with less than a 25% difference. You don't want to be picking between 16px and 18px.
2. When creating your spacing guideline from scratch, start with a base of 16px. This is an easily divisible number and the default font size in browsers.

When you need to add some space under an element, you can just grab a value from your scale and try it out. If it's not quite enough, the next value will probably be perfect.

| TYPOGRAPHY

Visual rules

Color palettes

Spacing

Typography

Fira Sans

Heading 1	Heading 2	Heading 3	Heading 4
Fira Sans SemiBold 34 px 41 line height	Fira Sans SemiBold 22 px 28 line height	Fira Sans SemiBold 17 px 22 line height	Fira Sans Medium 15 px 20 line height
BUTTON 1	BUTTON 2	CAPTION 1	CAPTION 2
Fira Sans SemiBold 17 px 22 line height	Fira Sans Medium 12 px 16 line height	Fira Sans Medium 12 px 16 line height	Fira Sans Medium 11 px 13 line height
Body 1	Body 1 Alt	Body 2	Body 2 Alt
Fira Sans Regular 15 px 20 line height	Fira Sans SemiBold 15 px 20 line height	Fira Sans Regular 13 px 18 line height	Fira Sans SemiBold 13 px 18 line height

ChicagoJS 2019

@isabelacmor

Just like for spacing, the last thing you want to do is be choosing between different pixel values for your text. Instead, your design system should include a type ramp of all possible font styles for your product. This will let you simply grab values from your type ramp to build up larger components without having to fiddle with specific values.

CREATING YOUR DESIGN SYSTEM

The screenshot shows the Figma interface with a design system for the Fira Sans font family. On the left, there's a grid of style cards:

- Heading 1:** Fira Sans SemiBold, 34 px, 41 line height.
- Heading 2:** Fira Sans SemiBold, 22 px, 28 line height.
- Heading 3:** Fira Sans SemiBold, 17 px, 22 line height.
- Heading 4:** Fira Sans Medium, 15 px, 20 line height.
- BUTTON 1:** Fira Sans SemiBold, 17 px, 22 line height.
- BUTTON 2:** Fira Sans Medium, 12 px, 16 line height.
- CAPTION 1:** Fira Sans Medium, 12 px, 16 line height.
- CAPTION 2:** Fira Sans Medium, 11 px, 13 line height.
- Body 1:** Fira Sans Regular, 15 px, 20 line height.
- Body 1 Alt:** Fira Sans SemiBold, 15 px, 20 line height.
- Body 2:** Fira Sans Regular, 13 px, 18 line height.
- Body 2 Alt:** Fira Sans SemiBold, 13 px, 18 line height.

The right side of the interface shows the Figma properties panel for the selected "Heading 1" style:

- Layer:** Pass Through, 100% opacity.
- Text:** Fira Sans SemiBold, Regular weight, 34 size, 41 line height, 0 baseline shift, 0 letter spacing, 0 word spacing.
- Fill:** Dark blue color (hex 16263D), 100% opacity.
- Stroke:** No stroke.
- Effects:** None.

At the bottom left is the text "ChicagoJS 2019" and at the bottom right is a Twitter icon followed by the handle "@isabelacmor".

Once you've laid out your styles in your design tool, you can add them to your styles library. Here you can see how I do it in Figma.

CREATING YOUR DESIGN SYSTEM

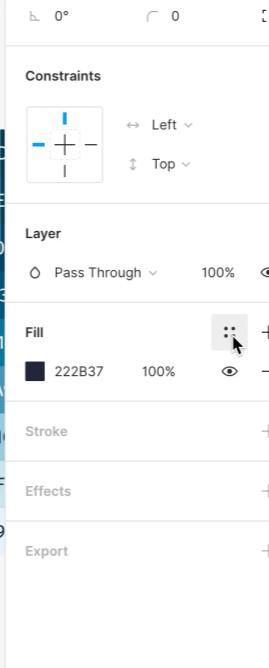
COLORS

NEUTRAL COLORS

NEUTRAL-950	#16263D
NEUTRAL-900	#343C48
NEUTRAL-800	#33435B
NEUTRAL-700	#657690
NEUTRAL-600	#909FB6
NEUTRAL-500	#D4DDEB
NEUTRAL-400	#DFE7F1
NEUTRAL-300	#ECF1F7
NEUTRAL-200	#F5F7FA
NEUTRAL-100	#FAFAFA
NEUTRAL-WHITE	#FFFFFF

PRIMARY COLORS

PRIMARY-900	#001A31
PRIMARY-800	#0E1E3D
PRIMARY-700	#102A4D
PRIMARY-600	#133A6B
PRIMARY-500	#314A8B
PRIMARY-400	#5A6AAB
PRIMARY-300	#A1BABA
PRIMARY-200	#CFD9E9
PRIMARY-100	#E9F2F9



ChicagoJS 2019

 @isabelacmor

CREATING YOUR DESIGN SYSTEM

Ag Caption/1

Ag Body/1

Ag Body/1 Alt

Ag Body/2

Ag Body/2 Alt

Ag Caption/2

Color Styles

● Primary/900

● Neutral/600

● Neutral/950

● Neutral/700

● Neutral/800

● Neutral/500

● Neutral/400

ChicagoJS 2019

 @isabelacmor

In the end, you end up with a library of all the styles you'll be using throughout your design system.

CREATING YOUR DESIGN SYSTEM

BUTTONS

Normal buttons

NORMAL

PRESSED

DISABLED

Action buttons

+ NEW

+ NEW

+ NEW

INPUT FIELDS

Default input

Empty label

Active label

Typing label

 Typing |

Disabled label

Password input

Empty label

Active label

Typing label

 • • • |

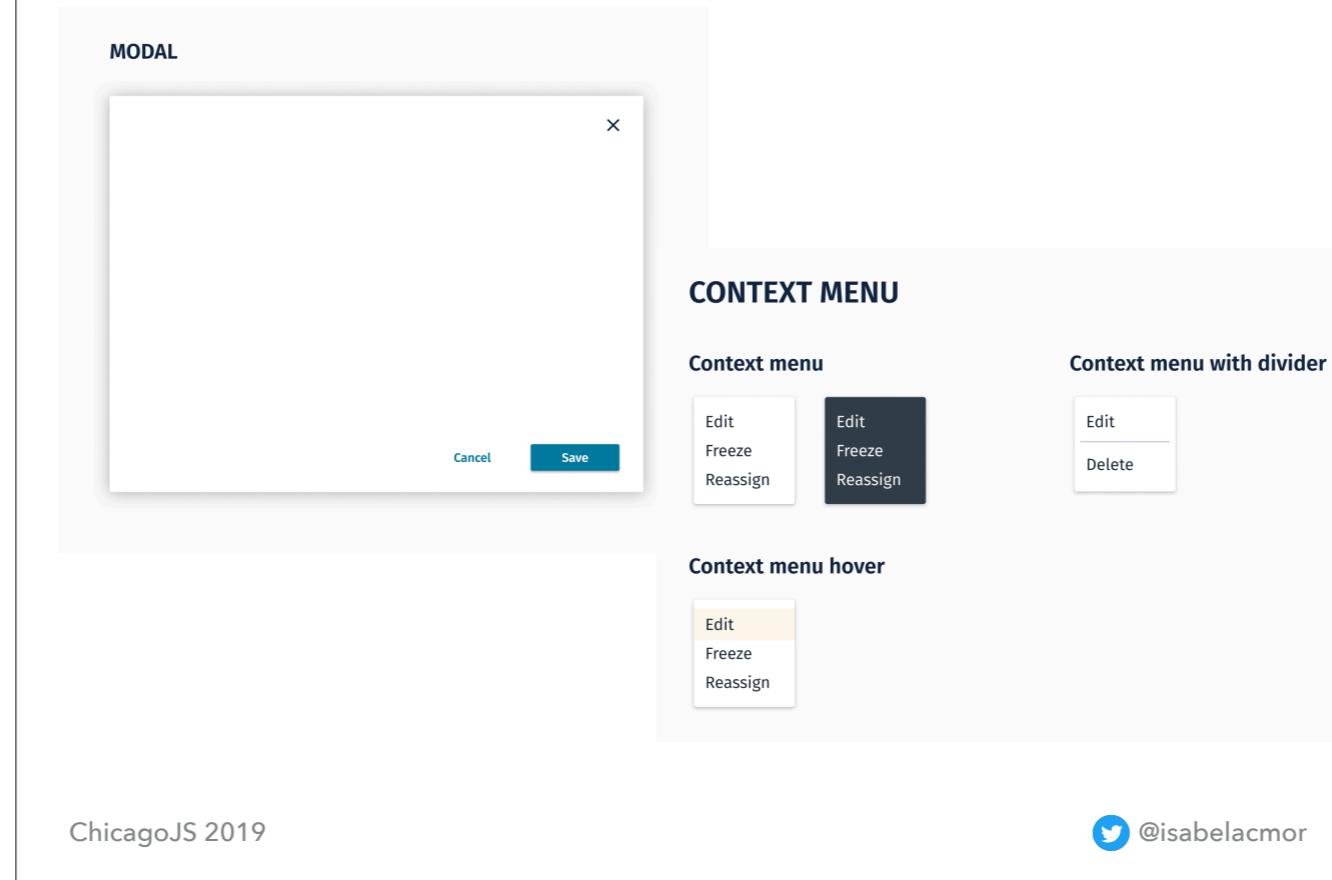
Disabled label

ChicagoJS 2019

 @isabelacmor

From there, you can use these styles to start building more complex components like buttons... input fields... I usually like to start with some very rough sketches on paper to get an idea of what kind of components I need to focus on designing. For example, my product might be very data-heavy and I want to design a lot of tables and graphs. I wouldn't want to waste time designing a date picker that may never get used.

CREATING YOUR DESIGN SYSTEM



ChicagoJS 2019

 @isabelacmor

Once I have some foundational components designed, I can start designing more complex pieces like modals and context menus. All the smaller components should live in your library and be reused to build up larger pieces. For example, for this modal, we don't create a brand new button. We simply copy our existing button symbol and use it as a reference. If any changes get made to the master button component, they'll be reflected across the entire library.

CREATING YOUR DESIGN SYSTEM

- Buttons
 - Flat buttons
 - Action buttons
 - Normal buttons
 - ❖ Button/primary/normal
 - ❖ Button/primary/pressed
 - ❖ Button/primary/disabled
 - Normal buttons



ChicagoJS 2019

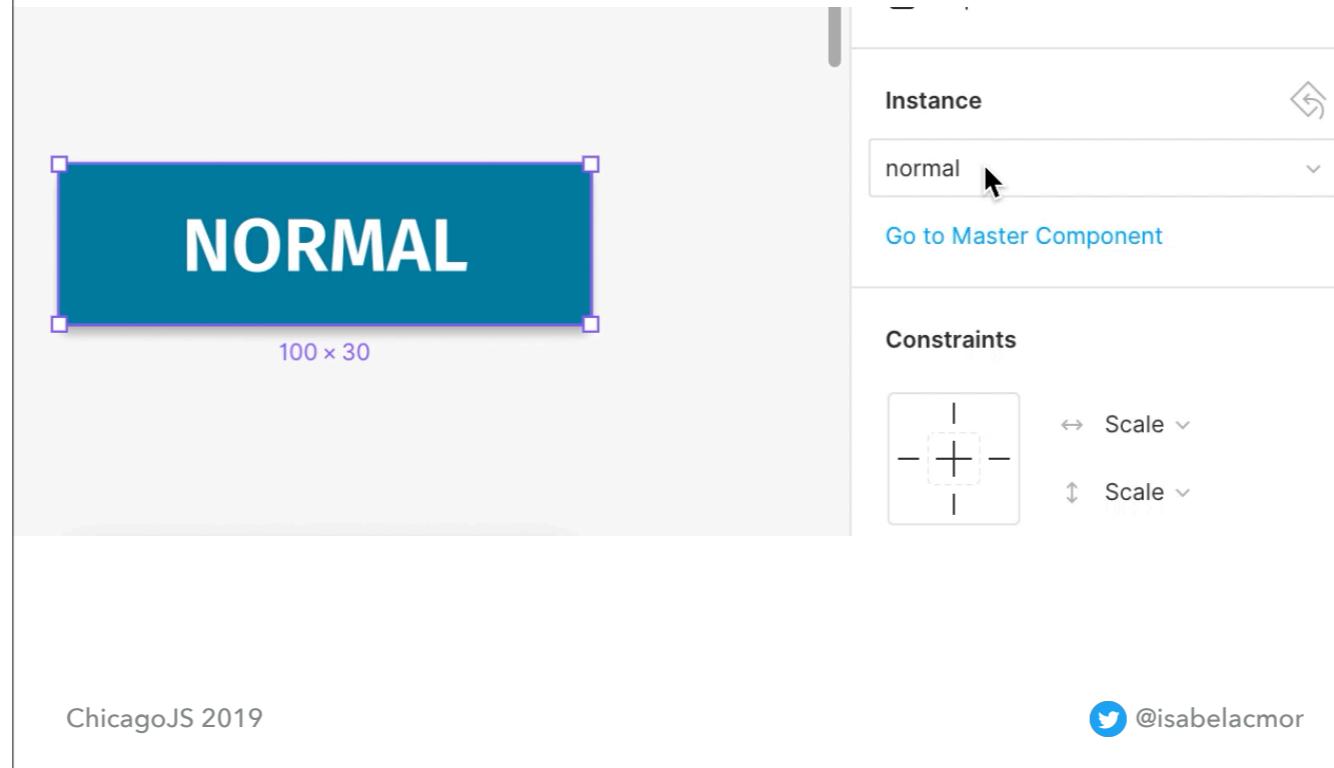
 @isabelacmor

Just like the library we created for colors and fonts, we can group all the pieces needed for a particular component into one symbol that represents that component in our library. For example, a button is made up of two pieces - a text label and a background. I've grouped these two pieces into a single symbol called Button/primary/normal, pressed, and disabled.

When I go to use the button in another component, I simply use a reference to the symbol in my library. Notice how here, the modal is also a symbol, but it's composed of multiple references to other symbols.

For example, for this modal, we don't create a brand new button. We simple copy our existing button symbol and use it as a reference. If any changes get made to the master button component, they'll be reflected across the entire library.

CREATING YOUR DESIGN SYSTEM



Reusing components like this also lets me quickly change between different instances of components, without needing to dig into the actual design. If I wanted to change to a pressed state for my button, all I would need to do is change which instance of the main button I'm using.

CREATING YOUR DESIGN SYSTEM

New objective template X

Name Work type Train direction Train direction

Blocks AFTCHECOR|SPT X AFTCHECOR|SPT X

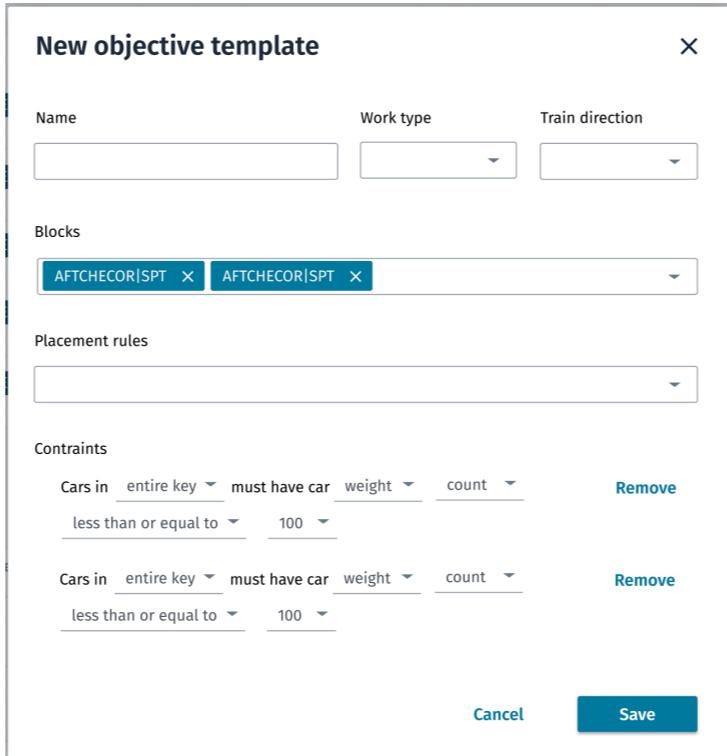
Placement rules

Constraints

Cars in entire key must have car weight count Remove
less than or equal to 100

Cars in entire key must have car weight count Remove
less than or equal to 100

Cancel Save



ChicagoJS 2019

 @isabelacmor

Eventually, you'll have all the individual components you need to simply import them into a mockup and use them without having to worry about duplicated styles and components. This nicely reflects what our coding process will look like as well when we start translating our designs into code.

IMPLEMENTING YOUR DESIGN SYSTEM

```
src
└── app
    ├── api-queries
    └── common
        ├── components
        │   ├── contextmenu
        │   ├── dialog
        │   ├── login
        │   ├── navigation-list
        │   │   ├── navigationlist.component.html
        │   │   ├── navigationlist.component.scss
        │   │   └── navigationlist.component.theme.scss
        │   ├── navigationlist.component.ts
        │   └── navigationlist.module.ts
        ├── navigation-panel
        ├── page-header
        ├── persona
        ├── status
        ├── table
        ├── tag
        ├── taglist
        └── title-bar
```

```
src
└── app
    ├── api-queries
    └── common
        ├── components
        └── styles
            ├── theme
            │   ├── dark-theme.scss
            │   ├── light-theme.scss
            │   └── themes.scss
            ├── colors.scss
            ├── spacing.scss
            ├── styles.scss
            └── text.scss
```

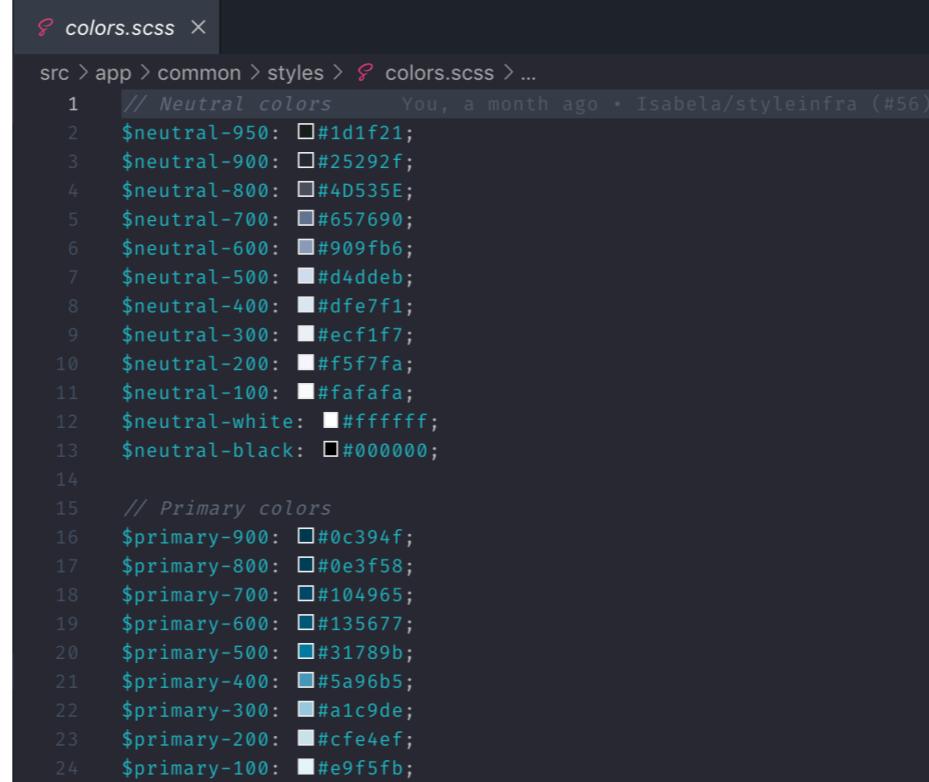
ChicagoJS 2019

 @isabelacmor

When you're ready to move your designs into code, you should follow the same organizational structure you did in your design system. Your specific approach may differ depending on your codebase or company, but what I've found works well for me and my team is the following file structure.

I have all common, reusable components under `src/app/common/components`. Each component gets its own folder so that its related files are all contained. My common folder also has a subfolder called `styles`. This is where all my shared styles live. Let's dig into these style files a bit more, since that's where I usually start when translating my design system into code.

IMPLEMENTING YOUR DESIGN SYSTEM



```
colors.scss x
src > app > common > styles > colors.scss > ...
1 // Neutral colors You, a month ago · Isabela/styleinfra (#56)
2 $neutral-950: #1d1f21;
3 $neutral-900: #25292f;
4 $neutral-800: #4d535e;
5 $neutral-700: #657690;
6 $neutral-600: #909fb6;
7 $neutral-500: #d4ddeb;
8 $neutral-400: #dfe7f1;
9 $neutral-300: #ecf1f7;
10 $neutral-200: #f5f7fa;
11 $neutral-100: #fafafa;
12 $neutral-white: #ffffff;
13 $neutral-black: #000000;
14
15 // Primary colors
16 $primary-900: #0c394f;
17 $primary-800: #0e3f58;
18 $primary-700: #104965;
19 $primary-600: #135677;
20 $primary-500: #31789b;
21 $primary-400: #5a96b5;
22 $primary-300: #a1c9de;
23 $primary-200: #cfefef;
24 $primary-100: #e9f5fb;
```

ChicagoJS 2019

 @isabelacmor

The first thing I usually do is define my color palettes. You never want to have colors, or any other values, hardcoded in your codebase because it makes modifications and maintenance so much harder. Imagine I want to change my neutral-900 color tomorrow. Instead of needing to dig throughout my entire codebase for the hardcoded hex value and then manually verify that I haven't missed any, I can just edit the value in this one file.

IMPLEMENTING YOUR DESIGN SYSTEM

```
Spacing scss
src > app > common > styles > Spacing scss > ...
1 $padding-xxs: 4px; You, a month ago · Isabela/styleinfra (#56)
2 $padding-xs: 8px;
3 $padding-sm: 12px;
4 $padding-md: 20px;
5 $padding-lg: 30px;
6
7 $border-radius: 2px;
8 $border-radius-round: 50%;
9
10 $card-spacing: 50px;
```

```
Text scss
src > app > common > styles > Text scss > body-sm-alt
1 @import url("https://fonts.googleapis.com/css?family=Fira+Sans:400,500,600&display=swap");
2
3 ∵ h1 {
4   font-family: "Fira Sans", sans-serif;
5   font-weight: 600;
6   font-size: 34px;
7   line-height: 41px;
8 }
9
10 ∵ h2 {
11   font-family: "Fira Sans", sans-serif;
12   font-weight: 600;
13   font-size: 22px;
14   line-height: 28px;
15 }
```

ChicagoJS 2019

 @isabelacmor

The same goes for my spacing values, fonts, and any other values I want to be able to set and forget.

IMPLEMENTING YOUR DESIGN SYSTEM

```
// Navigation colors
navigation-panel-background: $neutral-900,
navigation-panel-border-color: $neutral-950,
navigation-panel-header-color: $neutral-200,
navigation-panel-section-color: $neutral-600,
navigation-list-item-active: $yellow-600,
navigation-list-item-active-indicator: $yellow-500,
navigation-list-item-color: $neutral-white,
navigation-list-item-active-color: $neutral-900,

// Header colors
app-header-background: $neutral-950;
app-header-border-color: $neutral-900;
app-header-text-color: $neutral-200;
persona-background: $yellow-600;
persona-text-color: $neutral-white;

// Main page colors
page-background: $neutral-950,
page-header-text-color: $neutral-300,
page-card-header-text-color: $neutral-600,
page-card-subheader-text-color: $neutral-400,
page-text-color: $neutral-100,
```

ChicagoJS 2019

 @isabelacmor

Once I have my variables defined from my design system, I start creating very semantic variables based off those original variables.

For example, in my global styles.scss file, I define a class called page-header. I base it off my defined h2 style and set the color and margin from my other semantic variables. Now, whenever I want to make a page header element, I just add this class to my element and all the styling will look the same across all my elements with this class name. You'd never want to duplicate this style in a css file local to the page you're using. All your styles should live either in your global stylesheet or component stylesheet.

IMPLEMENTING YOUR DESIGN SYSTEM

```
navigationpanel.component.html ●
src > app > common > components > navigation-panel > navigationpanel.component.html > div.nav-panel-container
1   <div class="nav-panel-container">
2     <h2>{{activeCustomer}}</h2>
3
4     <!-- List of yards -->
5     <h4 class="section-header">YARDS</h4>
6     <NavigationList class="scrollable spacing" [items]="yards$ | async" [isLoading]="isLoading"></NavigationList>
7
8     <!-- Settings actions -->
9     <h4 class="section-header">SETTINGS</h4>
10    <NavigationList class="scrollable" [items]="settingsItems"></NavigationList>
```

```
navigationlist.component.html ✘
src > app > common > components > navigation-list > navigationlist.component.html > div
18
19   <div class="nav-list-container">
20     <span *ngIf="isEmpty()" class="empty-message">No yards found</span>
21     <nav *ngIf="hasItems()" mat-tab-nav-bar selectedIndex="0">
22       <a *ngFor="let item of items; index as i;" mat-tab-link [routerLink]="item.routeUrl" routerLinkActive
23         #rla="routerLinkActive" [active]="rla.isActive">{{item.name}}</a>
24     </nav>
25   </div>
```

ChicagoJS 2019

 @isabelacmor

Really quickly, here's a great example of building components from other components. Here I've defined a navigation panel, which is made up of two navigation list components. This navigation list component is composed of simple HTML elements. Both live in the common/components directory.

DOCUMENTING YOUR DESIGN SYSTEM

```
npm install @storybook/react --save-dev
```

```
import React from 'react';
import { storiesOf } from '@storybook/react';
import { Button } from '@storybook/react/demo';

storiesOf('Button', module)
  .add('with text', () => (
    <Button>Hello Button</Button>
  ))
  .add('with emoji', () => (
    <Button><span role="img" aria-label="so cool">😊 😎 🤘 💯 </span></Button>
  ));

```

ChicagoJS 2019

 @isabelacmor

Once you have your common components implemented, I'd suggest creating documentation that encompasses both your design system and the technical aspects of your design system. The most recommended tool I've seen is Storybook.

Setting up a Storybook is super simple. You just npm install the package and create a new JS file for each page you want in your book. That page will contain your custom components. In this case, it's just the Button component.

DOCUMENTING YOUR DESIGN SYSTEM

```
npm install @storybook/react --save-dev
```

```
import React from 'react';
import { storiesOf } from '@storybook/react';
import { Button } from '@storybook/react/demo';

storiesOf('Button', module)
  .add('with text', () => (
    <Button>Hello Button</Button>
  ))
  .add('with emoji', () => (
    <Button><span role="img" aria-label="so cool">😊 😎 🤘 💯 </span></Button>
  ));

```

ChicagoJS 2019

 @isabelacmor

Setting up a Storybook is super simple. You just npm install the package and create a new JS file for each page you want in your book. That page will contain your custom components. In this case, it's just the Button component.

DOCUMENTING YOUR DESIGN SYSTEM

```
import React from 'react';
import { storiesOf } from '@storybook/react';
import { Button } from '../../common/components/button';

storiesOf('Button', module)
  .add('with text', () => (
    <Button>Hello Button</Button>
  ))
  .add('with emoji', () => (
    <Button><span role="img" aria-label="so cool">😎 😎 🤘 💯 </span></Button>
  ));

import React from 'react';
import { storiesOf } from '@storybook/react';
import { Button } from '@storybook/react/demo';

storiesOf('Button', module)
  .add('with text', () => (
    <Button>Hello Button</Button>
  ))
  .add('with emoji', () => (
    <Button><span role="img" aria-label="so cool">😎 😎 🤘 💯 </span></Button>
  ));
```

ChicagoJS 2019

 @isabelacmor

You can include your Storybook documentation either in your product's repo and import your components locally, or in a separate repo if you publish your common components to a public or private npm package.

DOCUMENTING YOUR DESIGN SYSTEM

The screenshot shows the Storybook interface for the 'Histogram' component. On the left, a sidebar lists various components under 'VISUALIZATIONS' (e.g., LineGraph, PieChart, SparkLine, BarChart, Histogram) and 'PRIMITIVES' (e.g., Axis, Label, Ticks, Animation). The 'Histogram' item is selected, highlighted with a blue bar at the top of the list. The main content area displays a histogram titled 'Latency distribution' showing the number of requests versus latency time (ms). A callout highlights a peak at 40ms with the text '209 requests 105-110ms'. Below the chart is a code snippet:

```
<Histogram data={sampleData} normalize appearance="minimal" />
```

. At the top right, there are user icons and a 'Share' button.

Design system 10.4.90

Search docs

VISUALIZATIONS

- Charts
 - LineGraph
 - PieChart
 - SparkLine
 - BarChart
 - Histogram
 - Vertical bar
 - Horizontal bar
- Interstitial
- Spinners
- Progress indicators
 - ProgressBar
 - ProgressCircle
- Modal
- SVGImage
- Tooltips
- Badges
- Buttons

PRIMITIVES

- Axis
- Label
- Ticks
- Animation
- Color
- Type

Histogram

Histogram is a component that renders a set of bars on polar axes. **Histogram** manages chart layout, positioning, title, given a data object. Use when you need to group data into ranges so that users can understand and make decisions on complex information.

Latency distribution

209 requests
105-110ms

Hide code

```
<Histogram data={sampleData} normalize appearance="minimal" />
```

ChicagoJS 2019

@isabelacmor

Your Storybook should give an overview on how to use your component, including code samples and APIs. Storybook just provides the framework to spin up all the table of contents and UI for nicely representing both the component in action and the documentation. All the boilerplate code we wrote before ends up giving us this really nice UI with a table of contents and everything.

- ▶ Useful for all projects, from personal websites to prod apps
- ▶ Minimizes maintenance with low startup overhead
- ▶ Provides living documentation for designers and devs

ChicagoJS 2019

 @isabelacmor

Design systems minimize maintenance as your product grows and matures, because you'll have a solid foundation based on a central source of truth for styles. Your design system will also be living documentation for designers and developers alike. In the context of a startup, we also use it for customer demos because it's super easy to create clickable prototypes directly in most design tools.

I hope I've been able to convince you that design systems are useful for projects of all sizes and give you the confidence to use this as another tool in your developer toolbox.