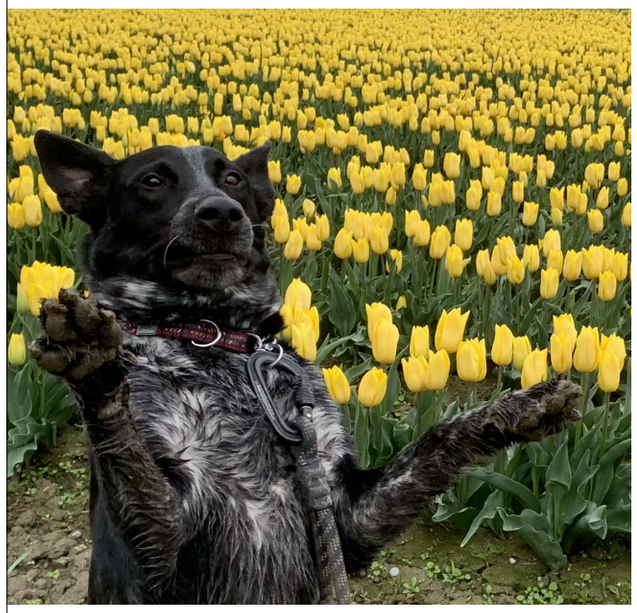
ISABELA MOREIRA

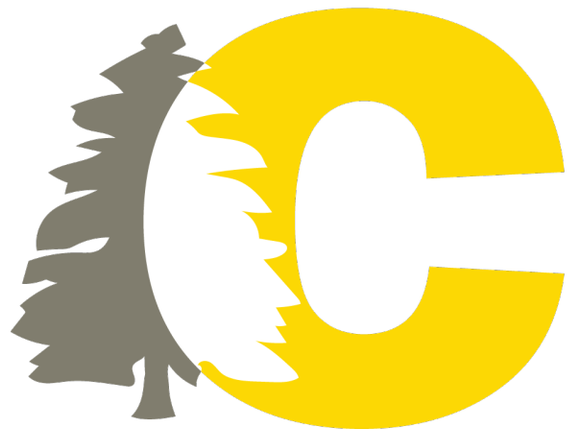
THE STARTUP'S GUIDE TO CREATING A DESIGN SYSTEM

ChicagoJS 2019

@isabelacmor

HI 👋



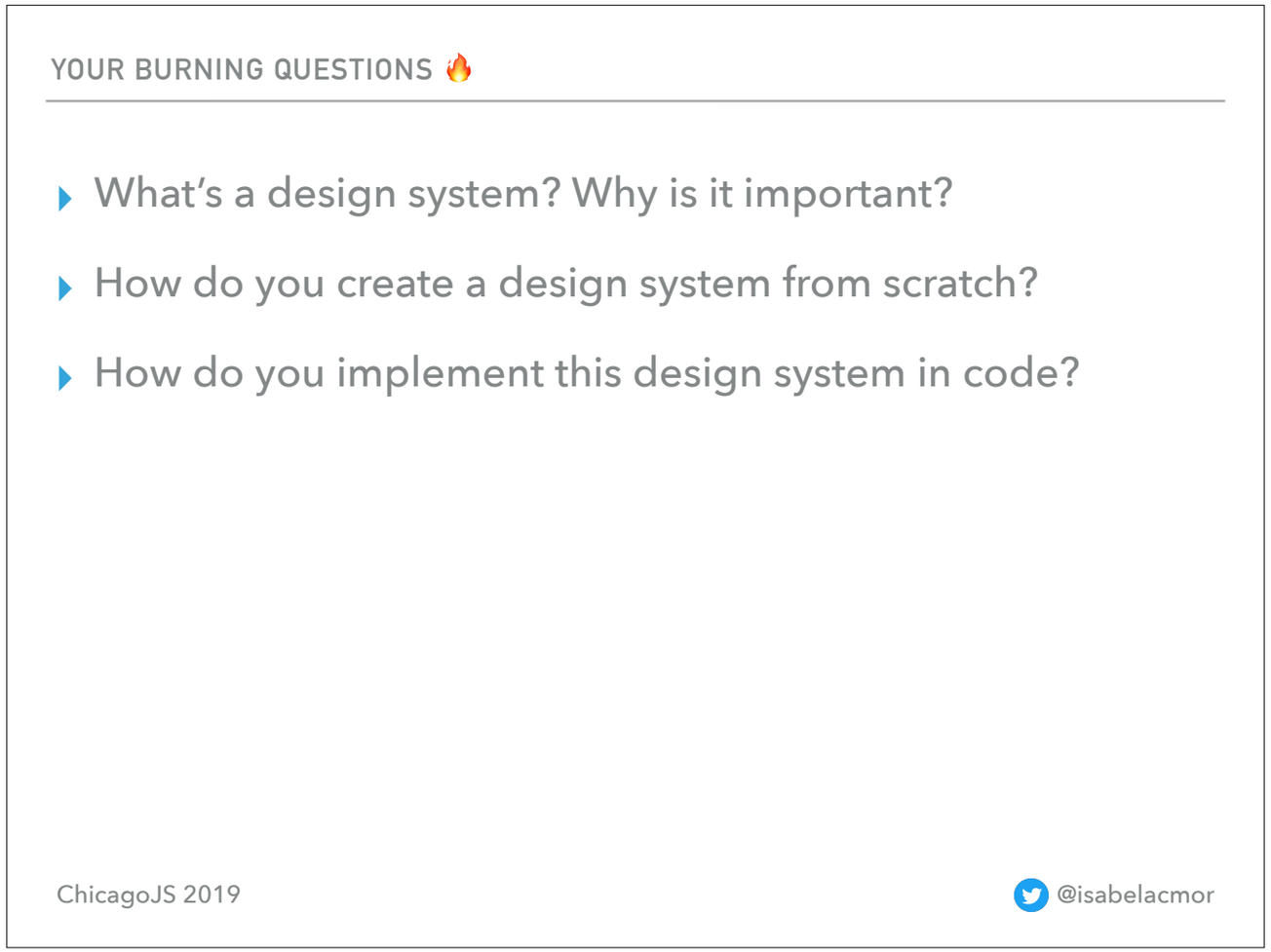


ChicagoJS 2019

@isabelacmor

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

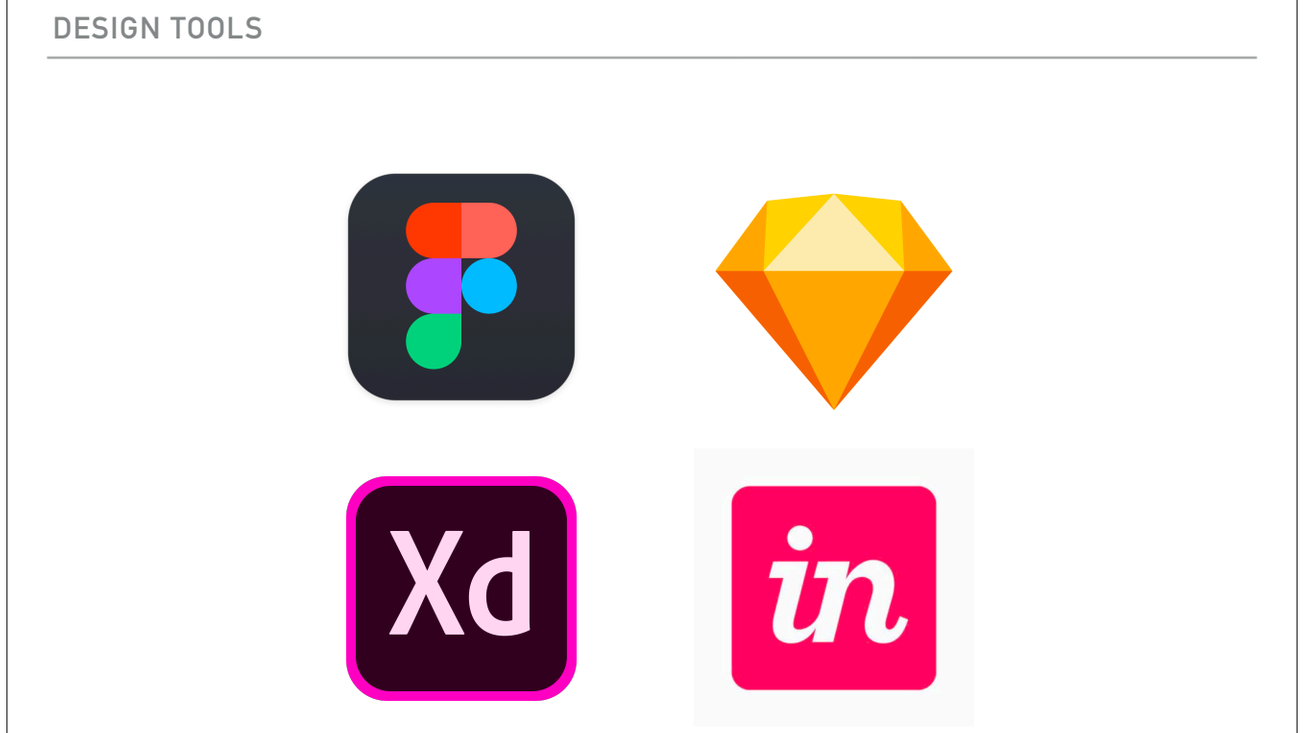
1. This is my dog, Atlas. He's very excited about this talk and I hope you all are too!
2. I work at a tiny startup in Seattle called CedarAI. We work on railroad technology and the entire industry is still run on pen and paper. This is a huge UX challenge and it's even harder to tackle for a brand new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



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

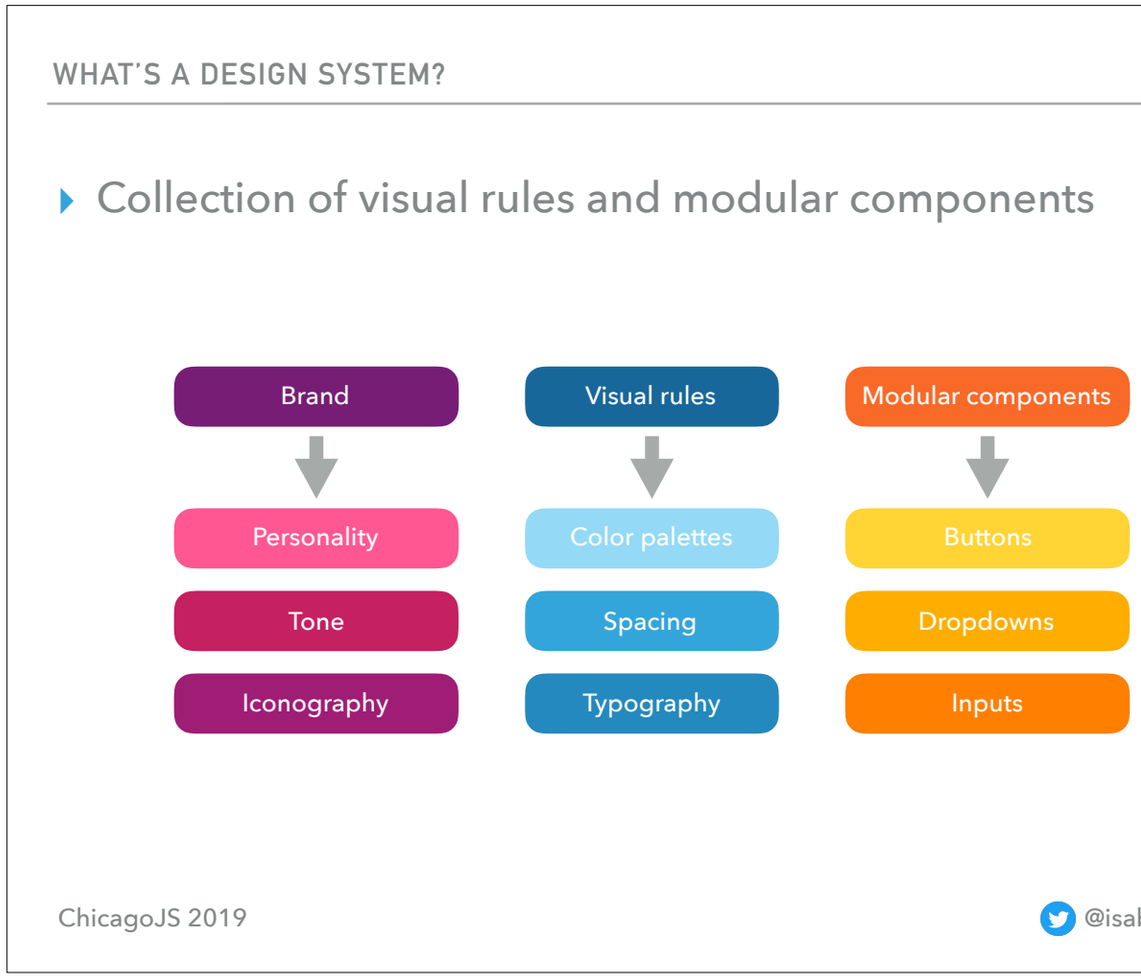
Most of this talk will show the tools I use specifically at work, but the strategies we talk about are pretty platform and tool agnostic.



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.

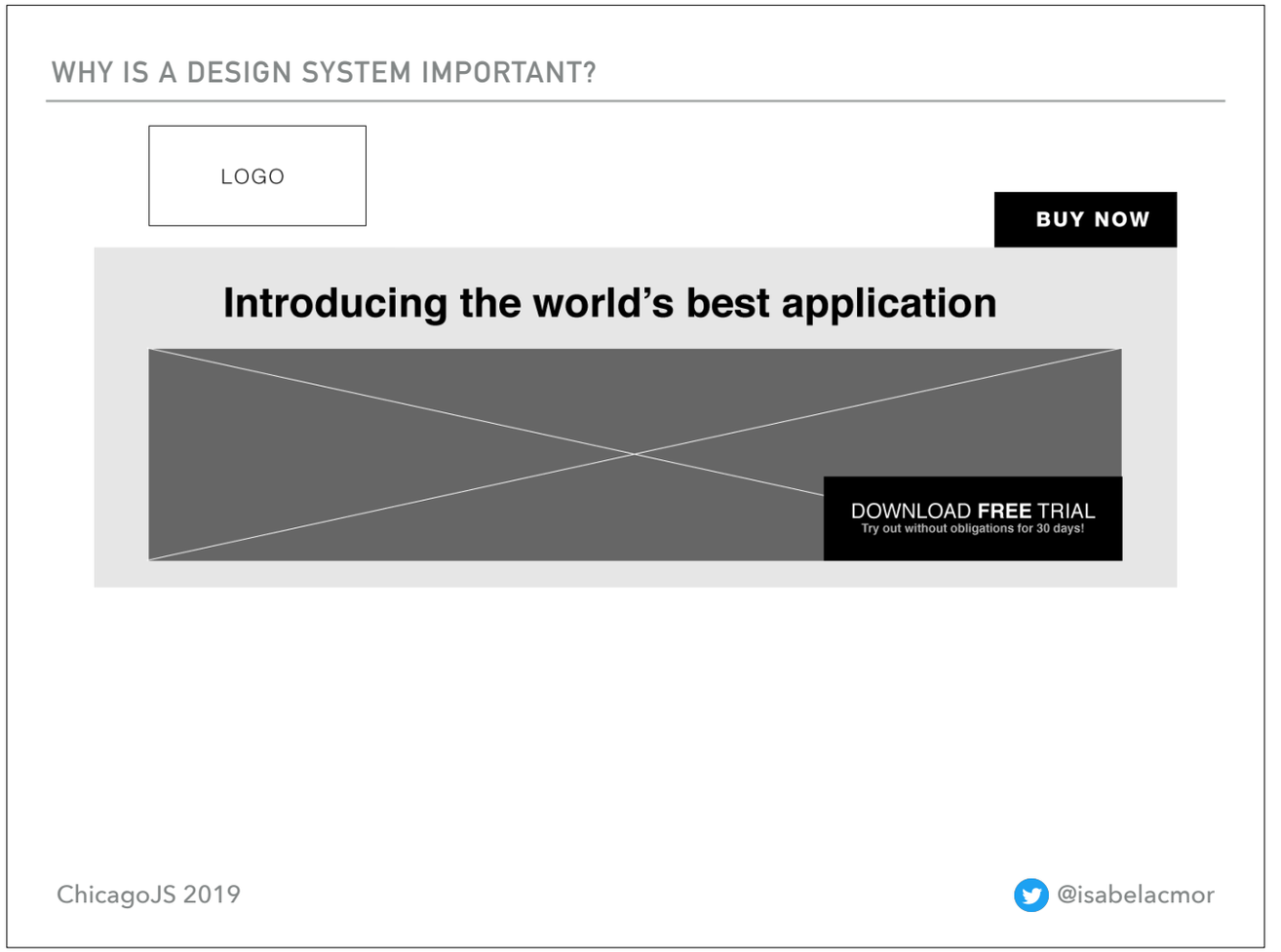
A lot of the screenshots I'm going to share are from a design tool called Figma, just because it's what I personally use at work.

DESIGN TOOLS



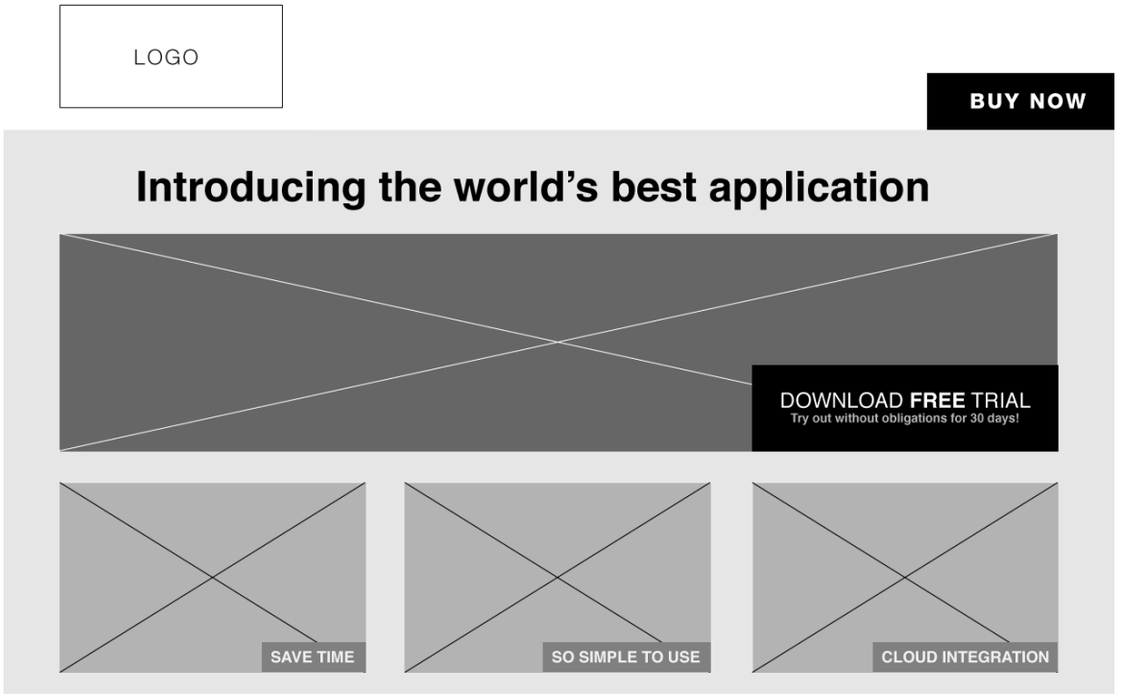
Very simply put, a design system is a collection of visual rules and modular components to guide and direct a product's design, from interaction models, to interfaces, to a product's personality and tone.

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.



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?

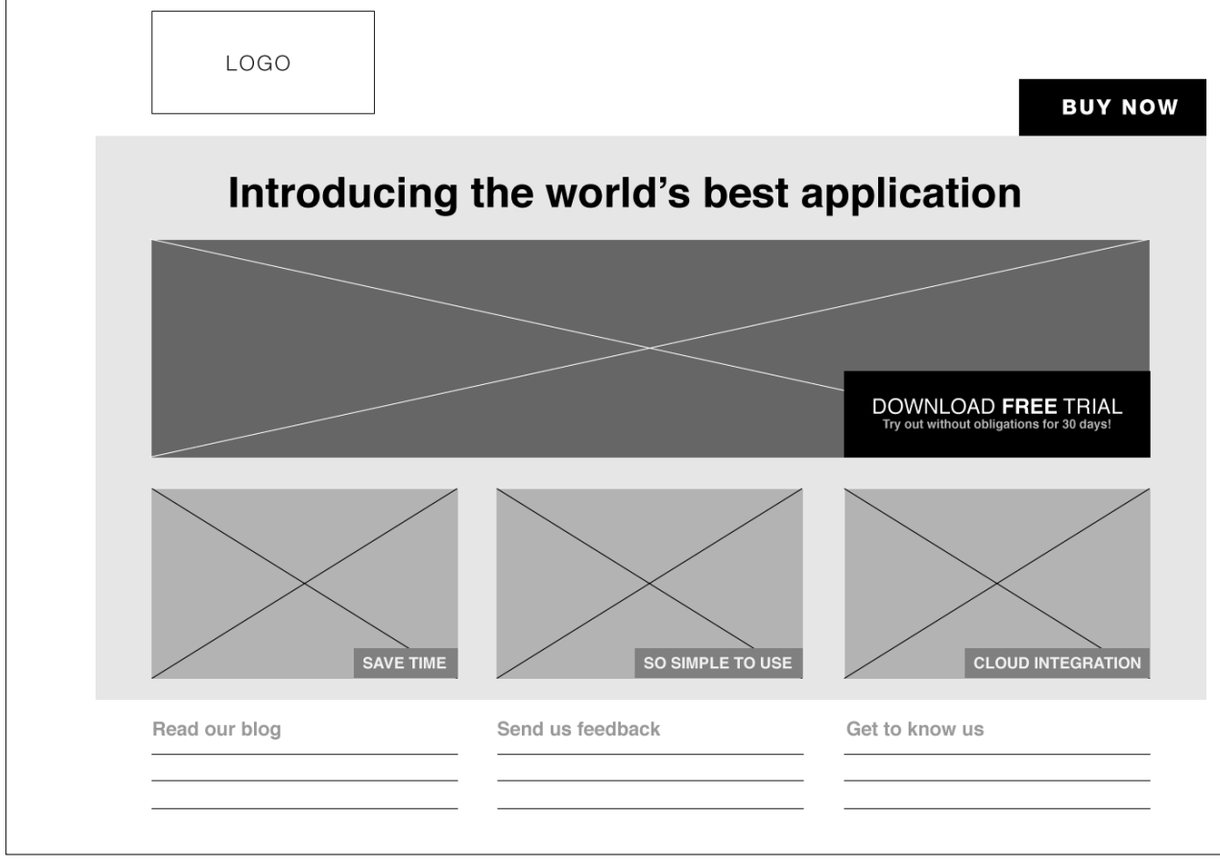


ChicagoJS 2019

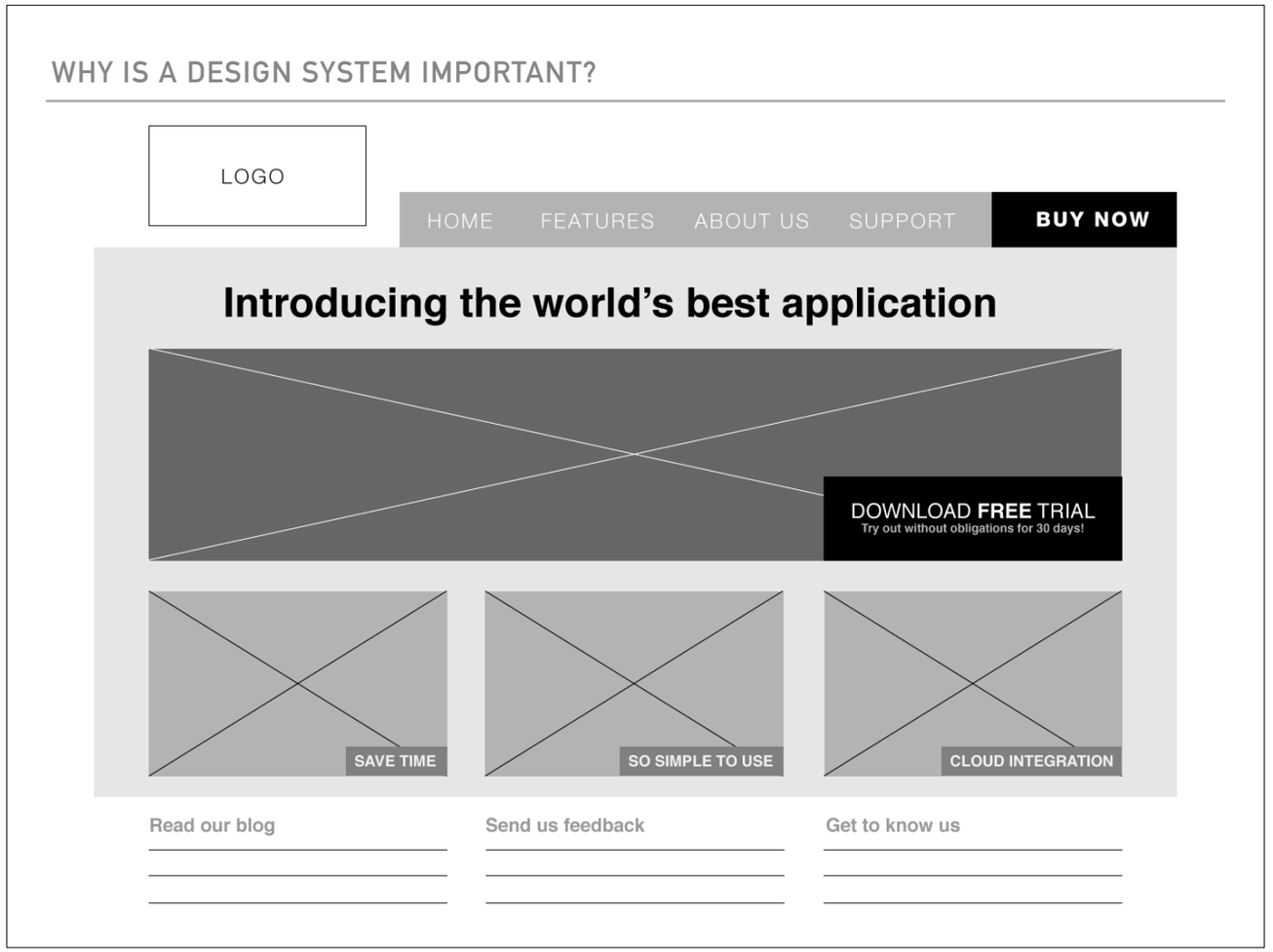
@isabelacmor

But then you start adding features…

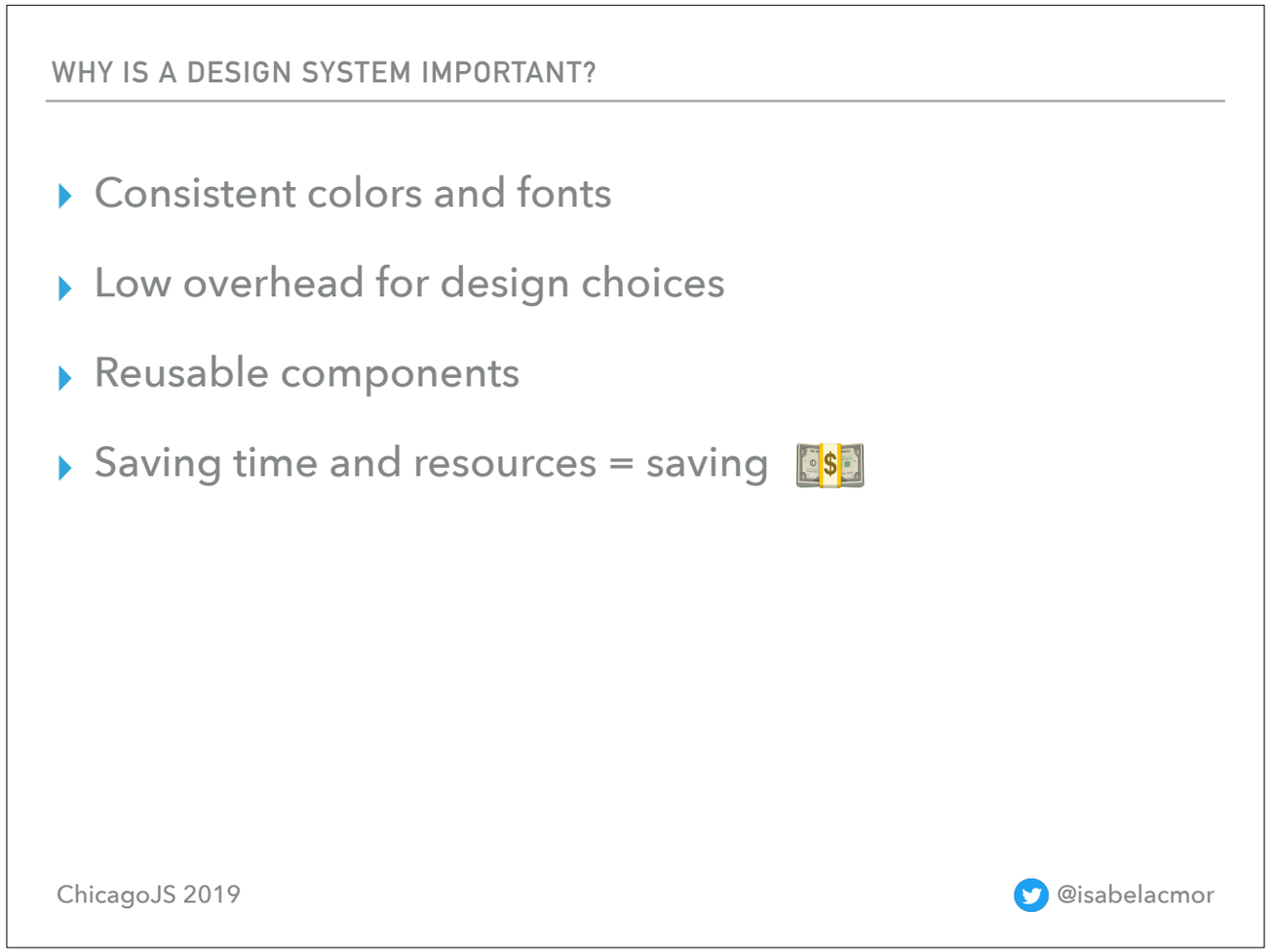
WHY IS A DESIGN SYSTEM IMPORTANT?



...



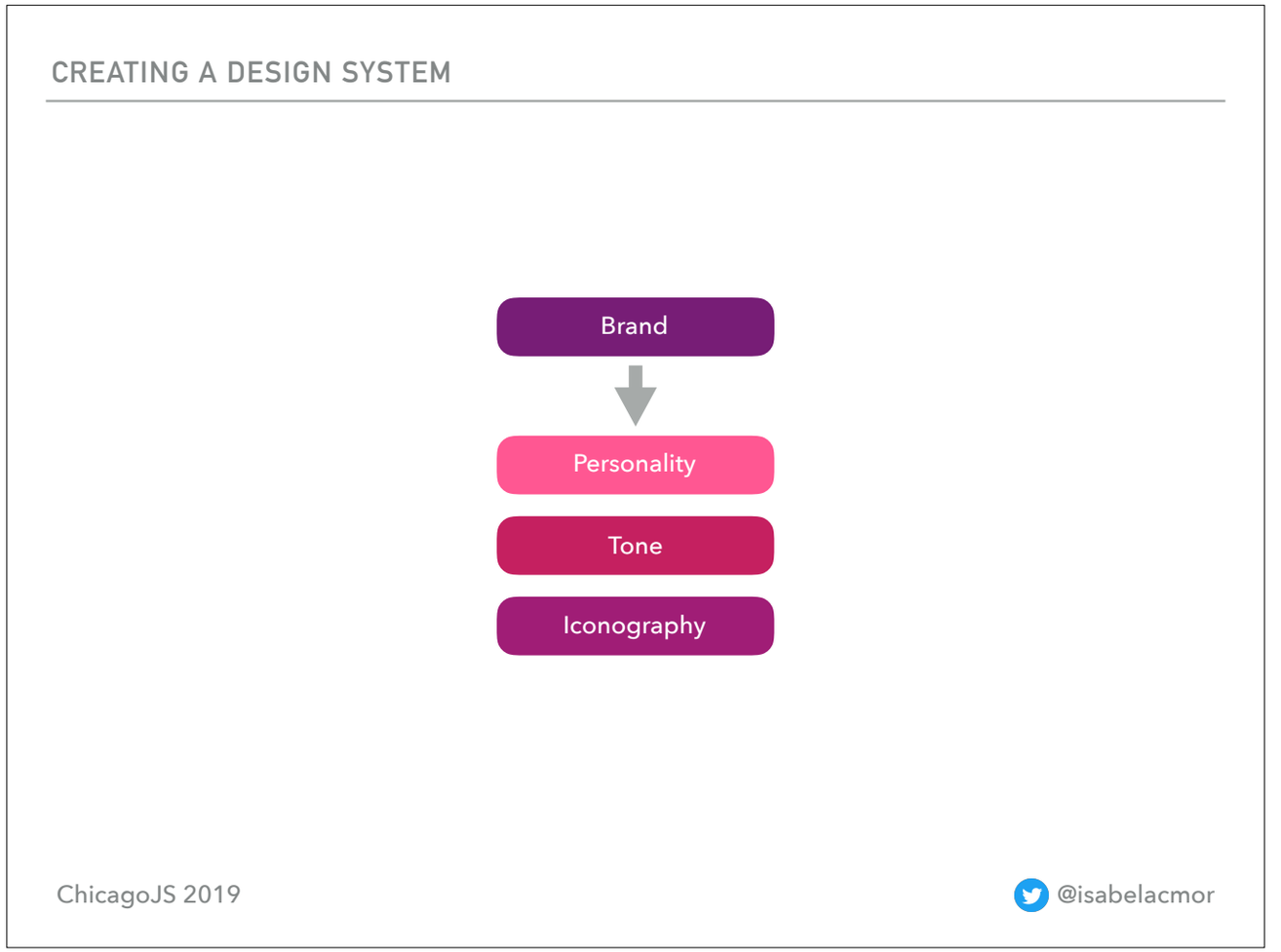
… and once you're done with your one page, you decide to add a few more pages. By this point, you probably have styles duplicated everywhere. There are subtle inconsistencies across your UI. Maybe your buttons have slightly different padding. Or your header text is a different size on different pages.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



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!

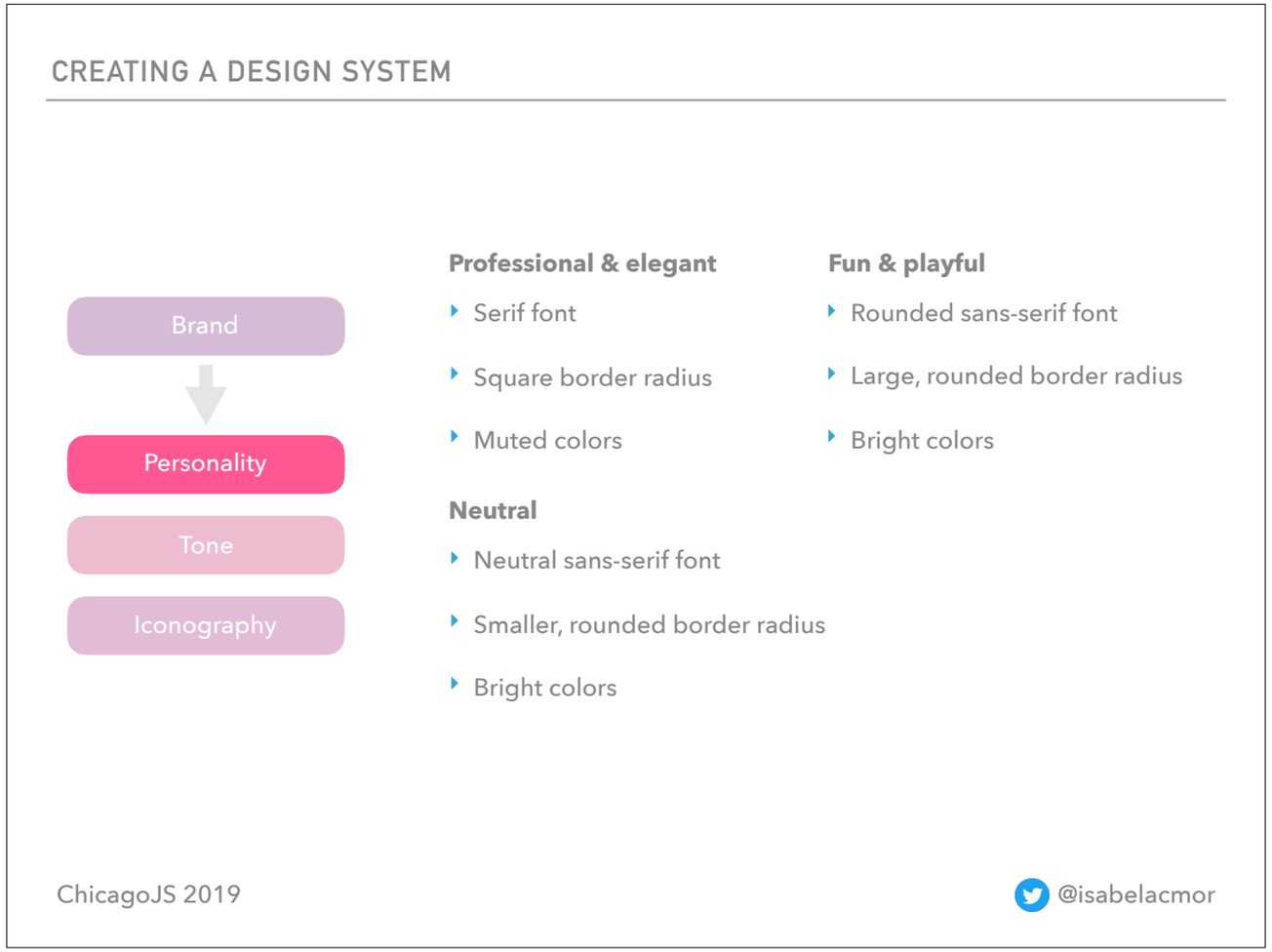
These choices don't apply to just the current project you're working on. These apply across all your products! If you create a fully-featured design system, you'll be able to use it across all your products, even if they use different technology stacks. The tools you choose to implement your design system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

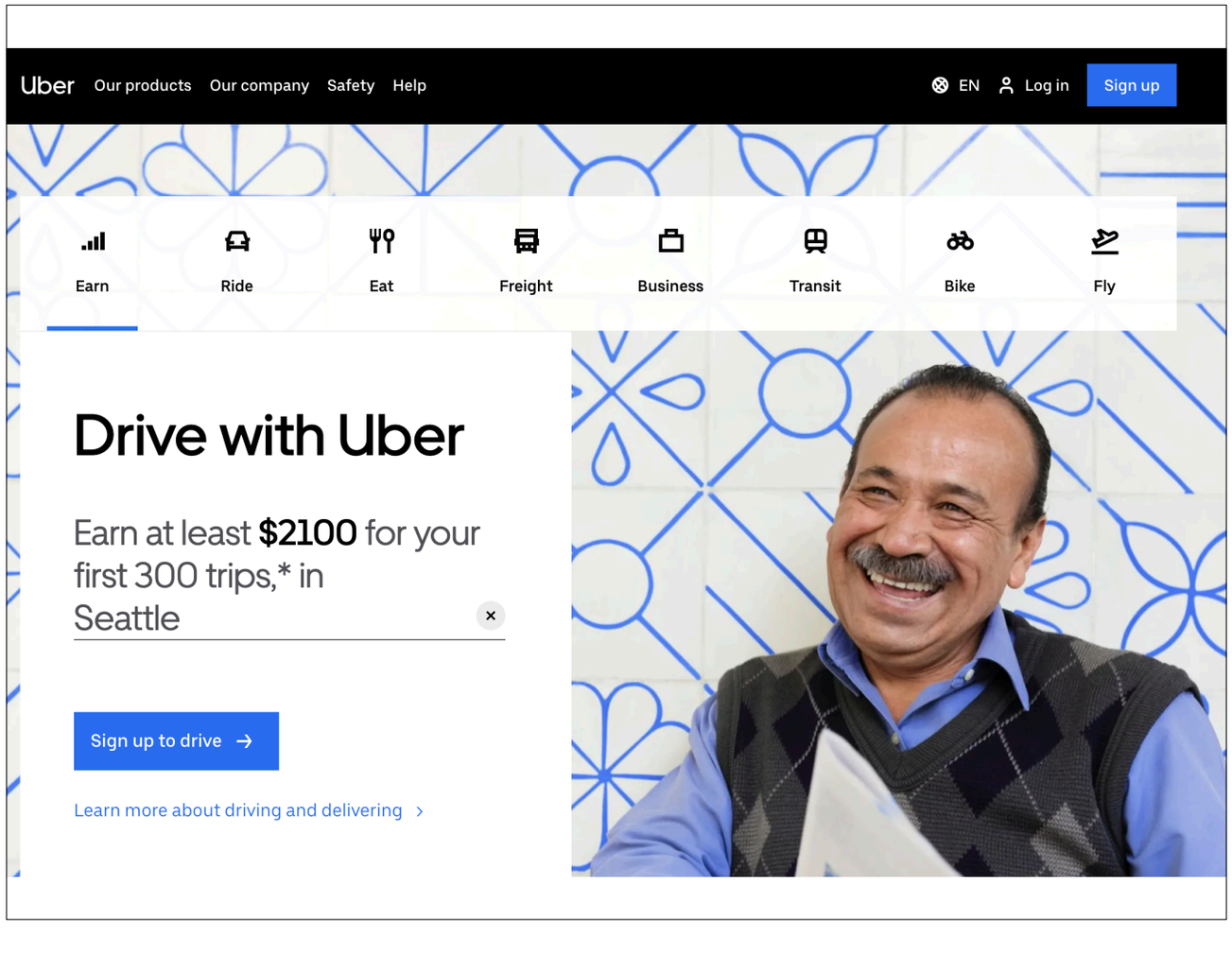


When we first start creating our design system, we'll need to start with the smallest, most foundational building blocks before working our way up to larger and more complex pieces.

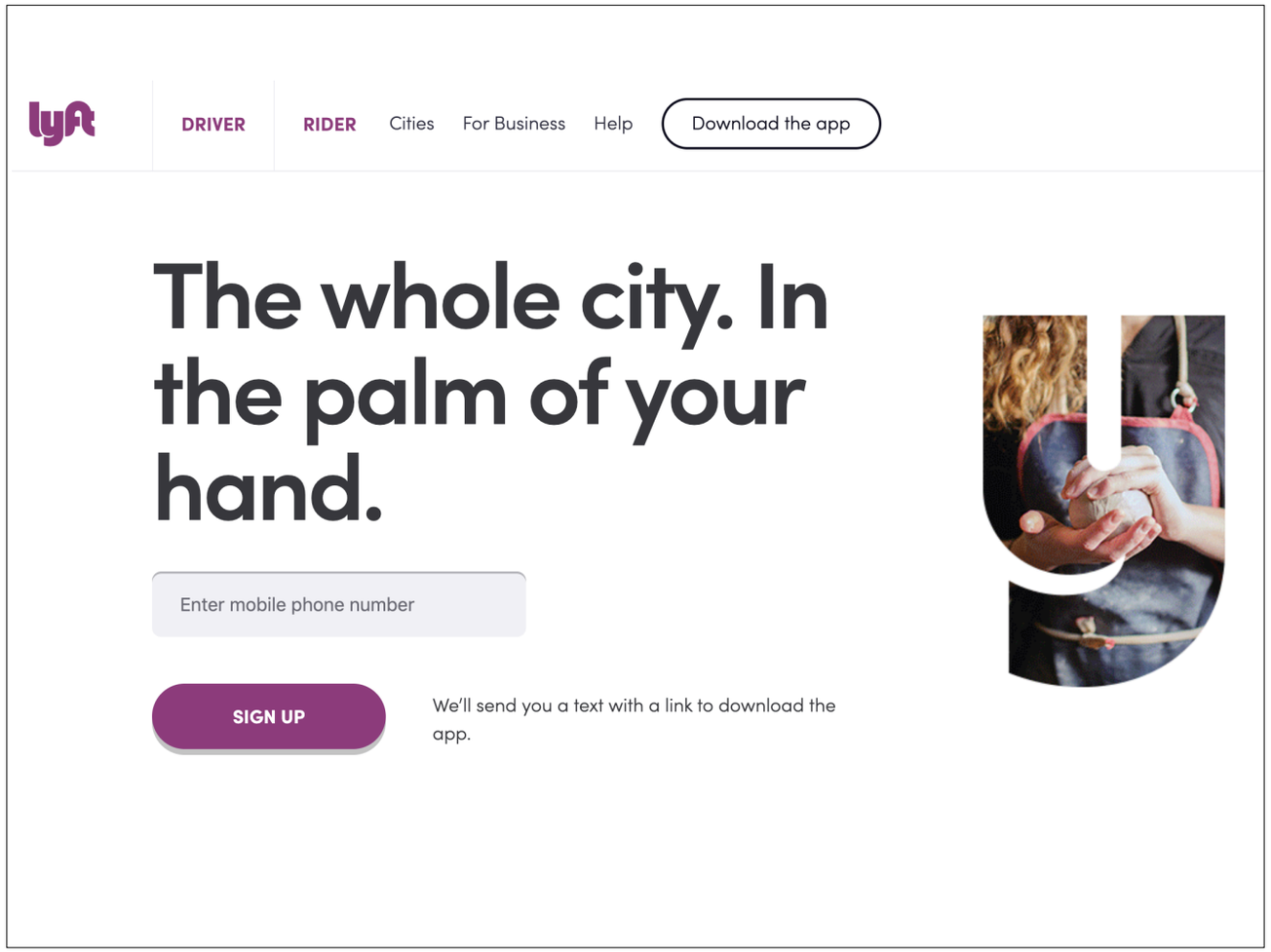
We'll do this by defining our brand.



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

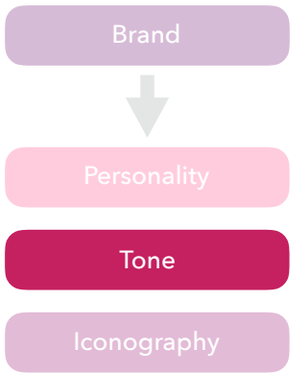


First, we'll want to decide our product's personality. This will influence choices like color, fonts, and borders.



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."

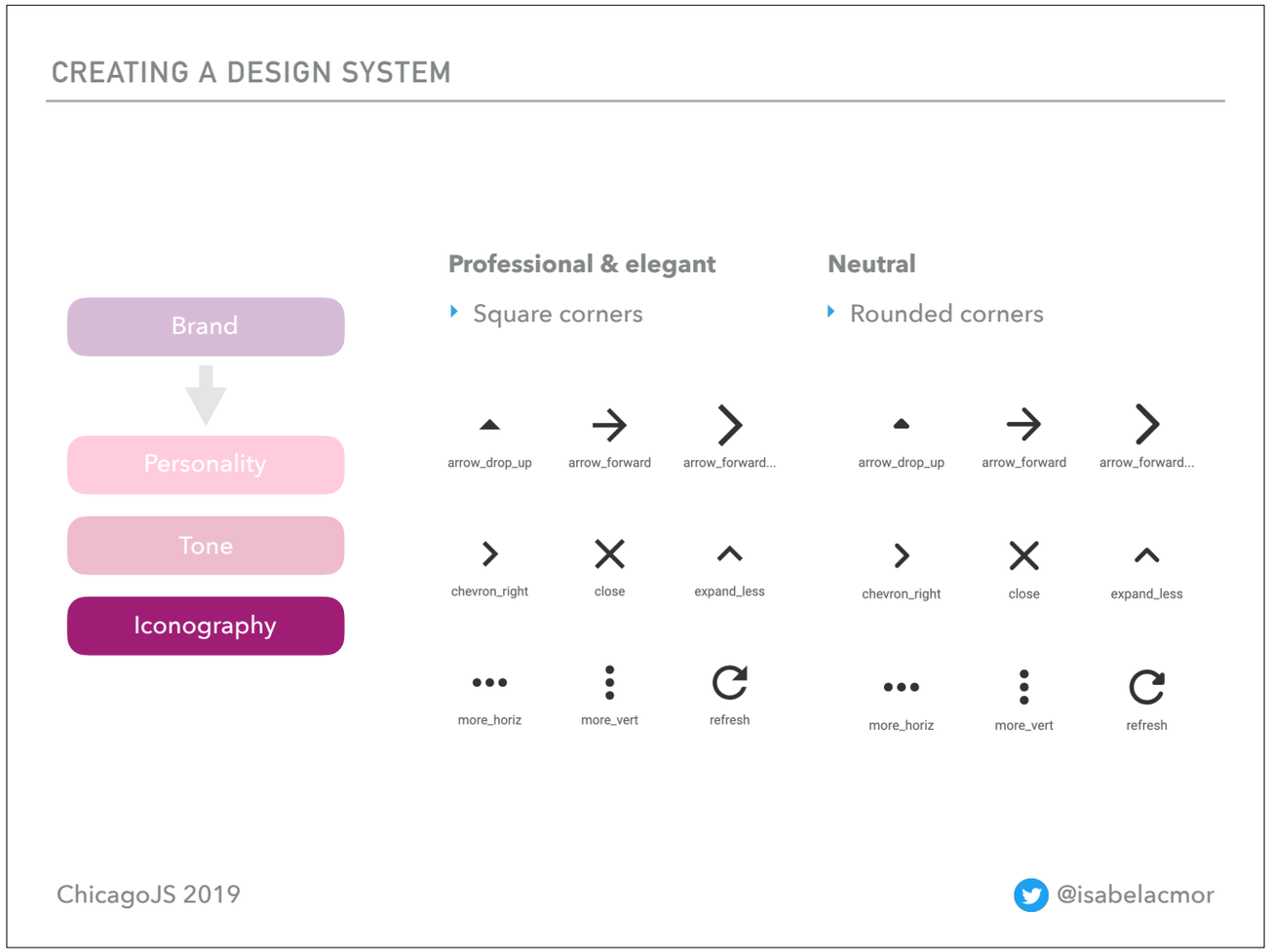
Neutral

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

Fun & playful

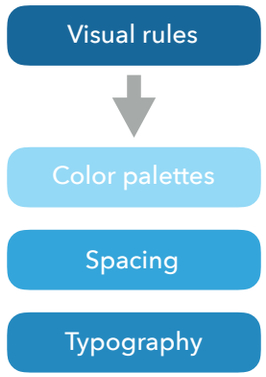
- More casual language
- "Just to make sure this is really you, where can we send you a verification code?"

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.

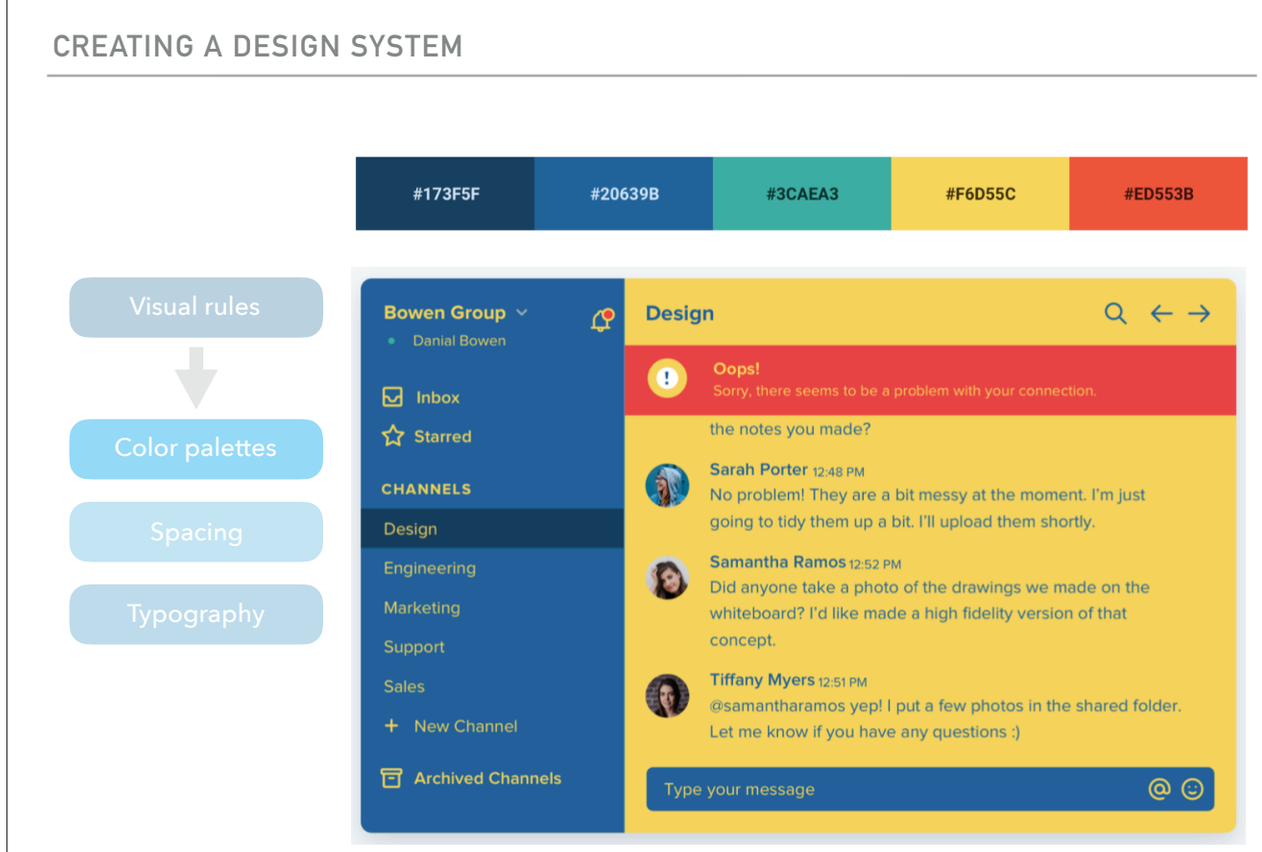


Finally, we should decide on iconography. If you're a small team or company, you probably don't have the resources to design an entire icon set and it's probably not a good use of your time any way.

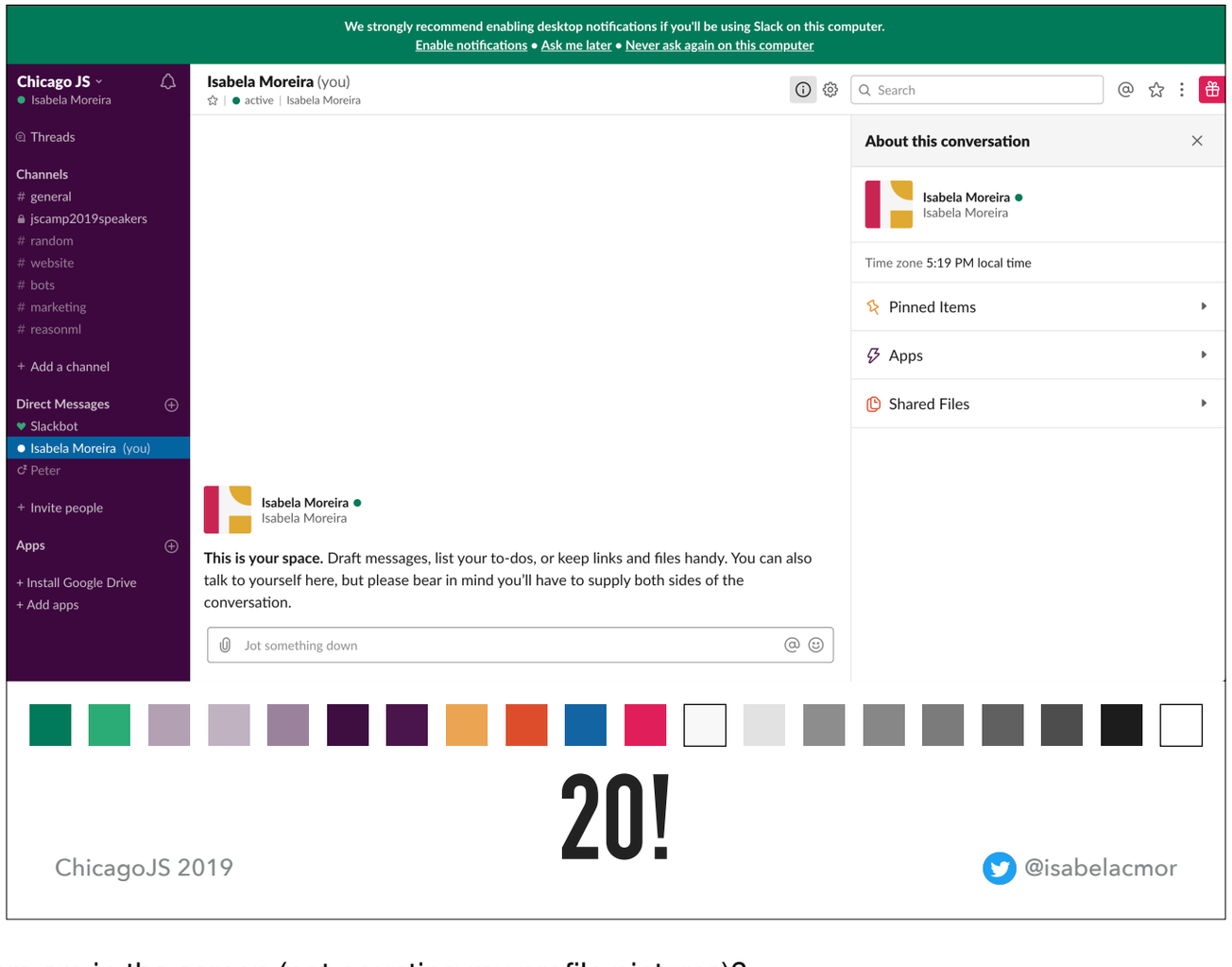
CREATING A DESIGN SYSTEM



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.



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.

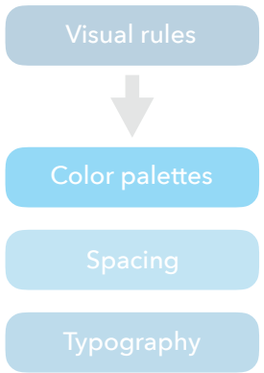


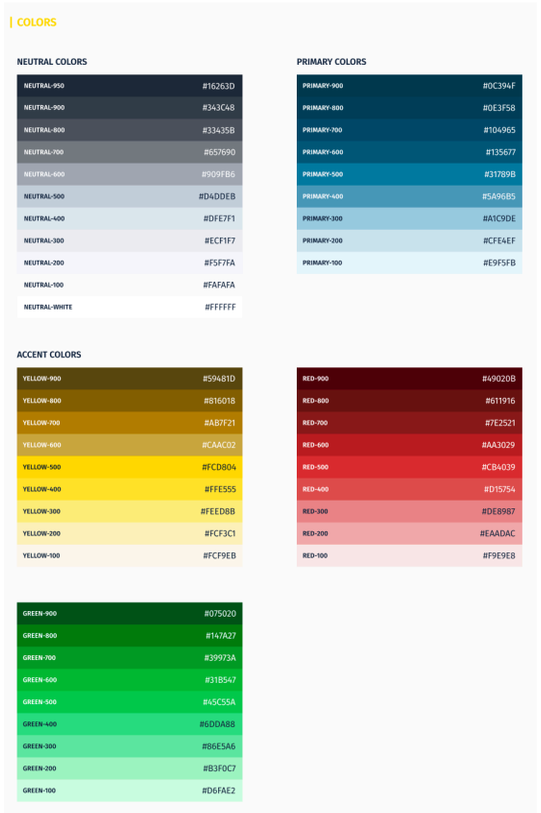
Can I get any guesses of how many colors are in the screen (not counting my profile pictures)?

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

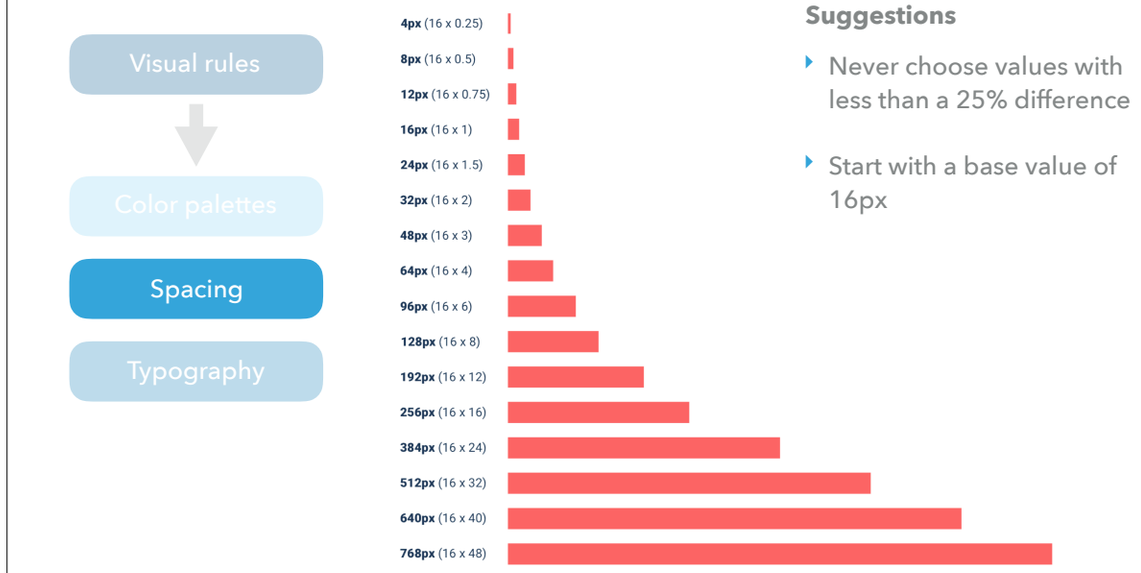
I counted 20 different colors in this one screen alone! Imagine if you were trying to keep track of all these colors by hand.

CREATING A DESIGN SYSTEM





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.

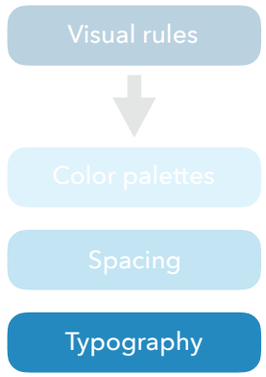


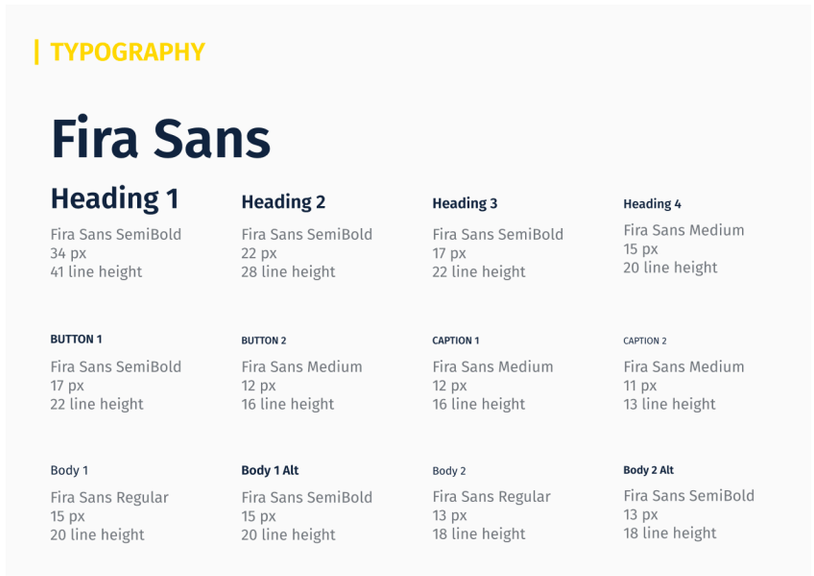
One of the most frustrating things in design can be choosing between different sizes and spacing on a pixel level and not knowing if you should have a padding of 16px or 18px. Not having a consistent spacing guideline leaves you with a huge range of hardcoded arbitrary values and you'll end up wasting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

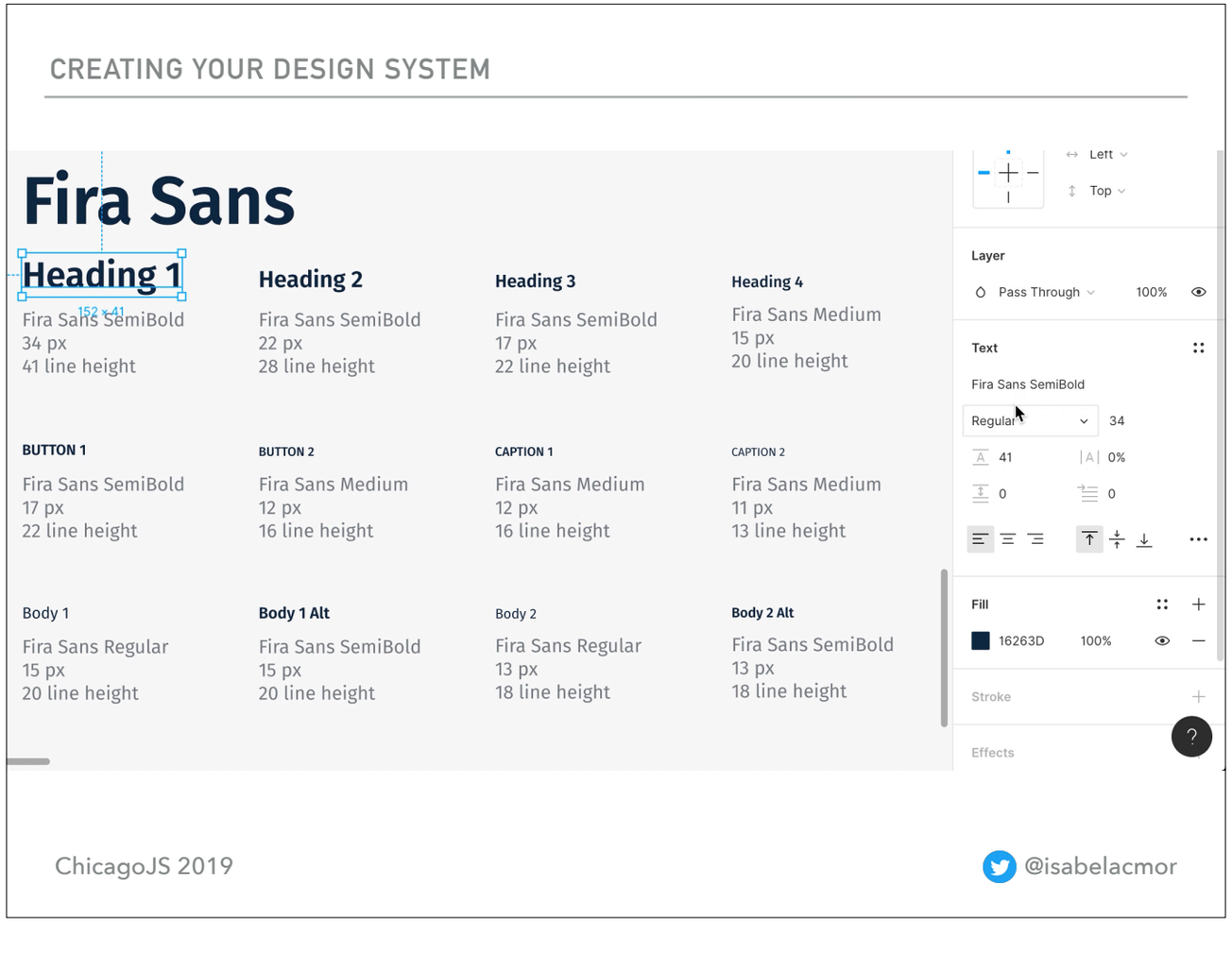
1. Never choose values with less than a 25% difference. You don't want to be picking between 16px and 18px.
2. When creating your spacing guideline from scratch, start with a base of 16px. This is an easily divisible number and the default font size in browsers.

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.





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.



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

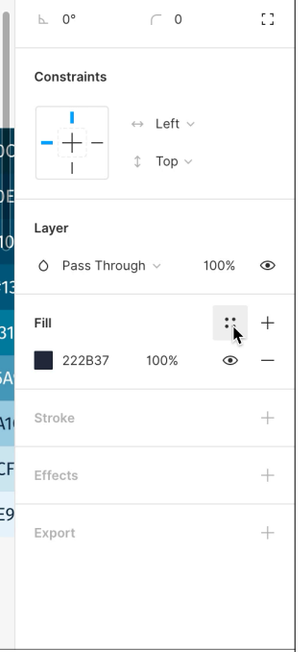
COLORS

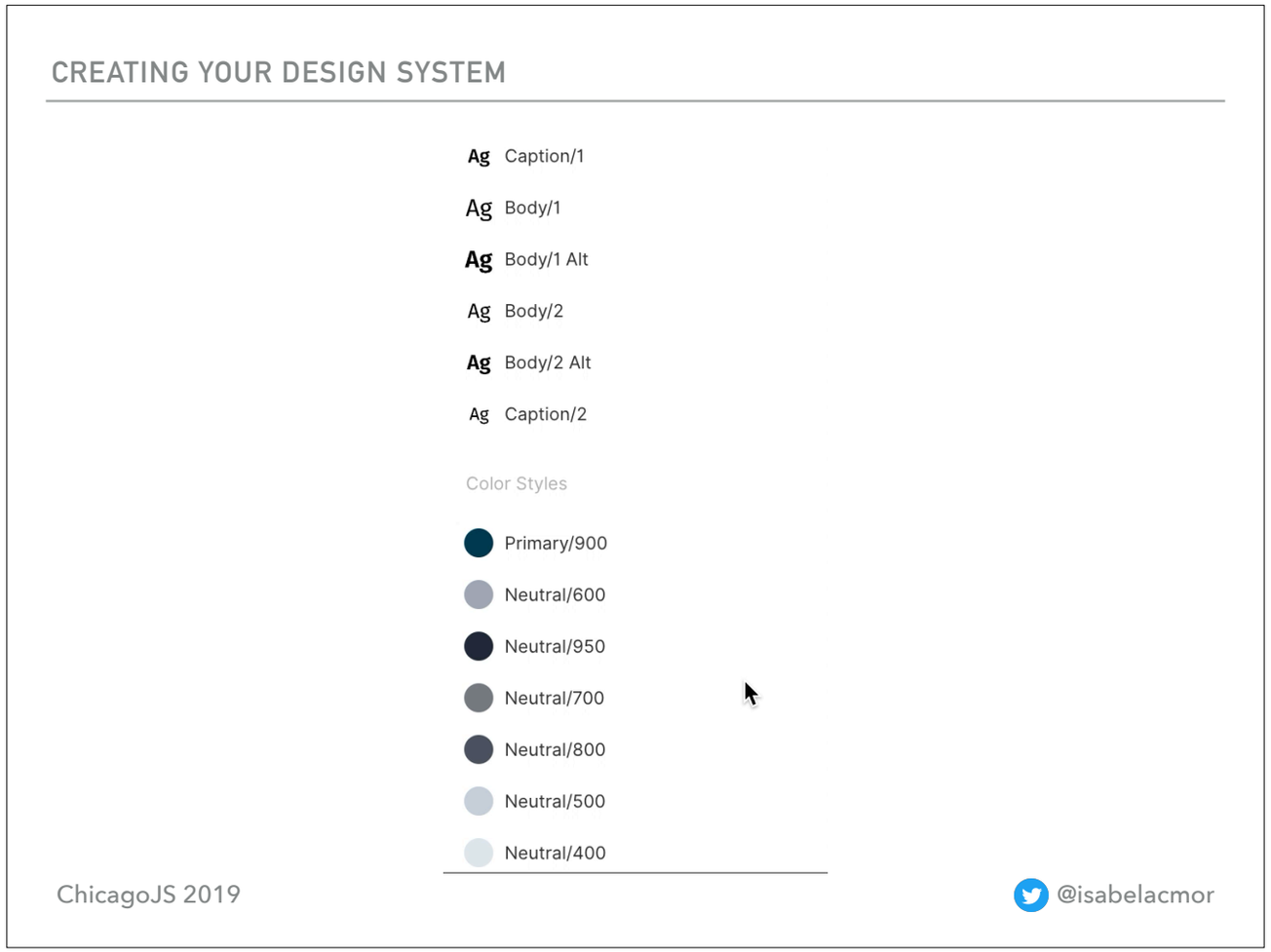
NEUTRAL COLORS

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

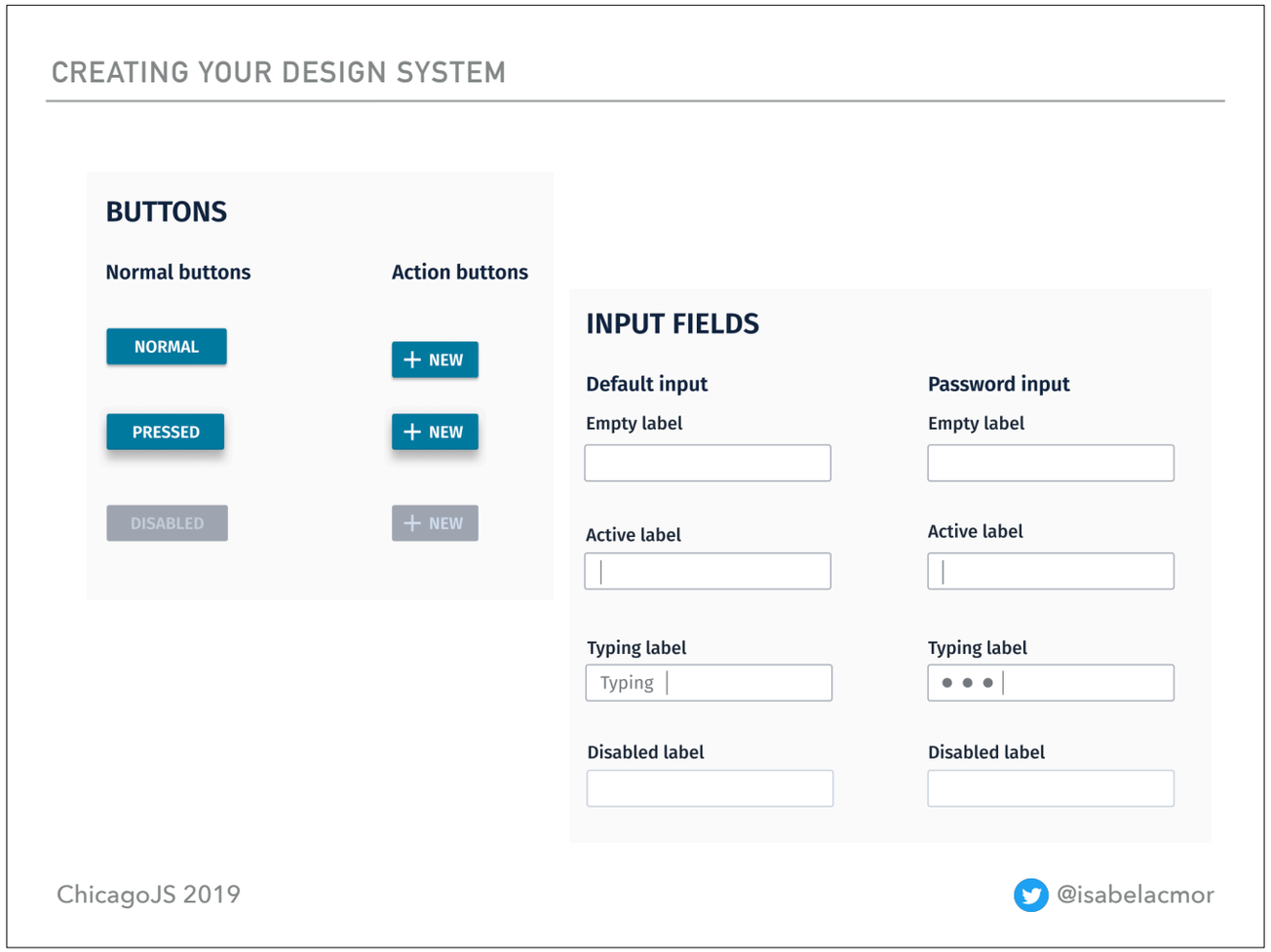
PRIMARY COLORS

PRIMARY-900	#0C
PRIMARY-800	#0E
PRIMARY-700	#10
PRIMARY-600	#13
PRIMARY-500	#31
PRIMARY-400	#5A
PRIMARY-300	#A1
PRIMARY-200	#CF
PRIMARY-100	#E9

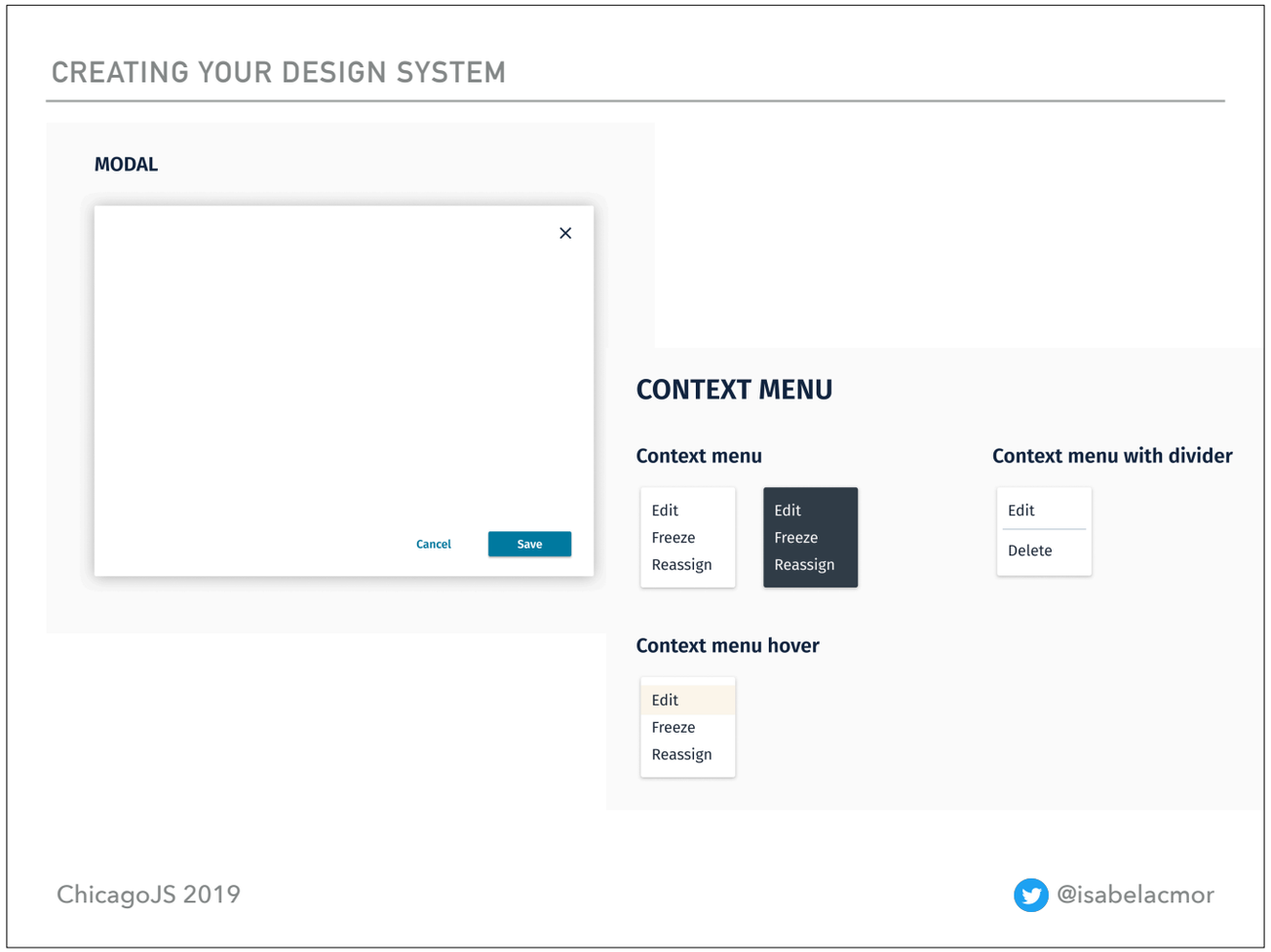




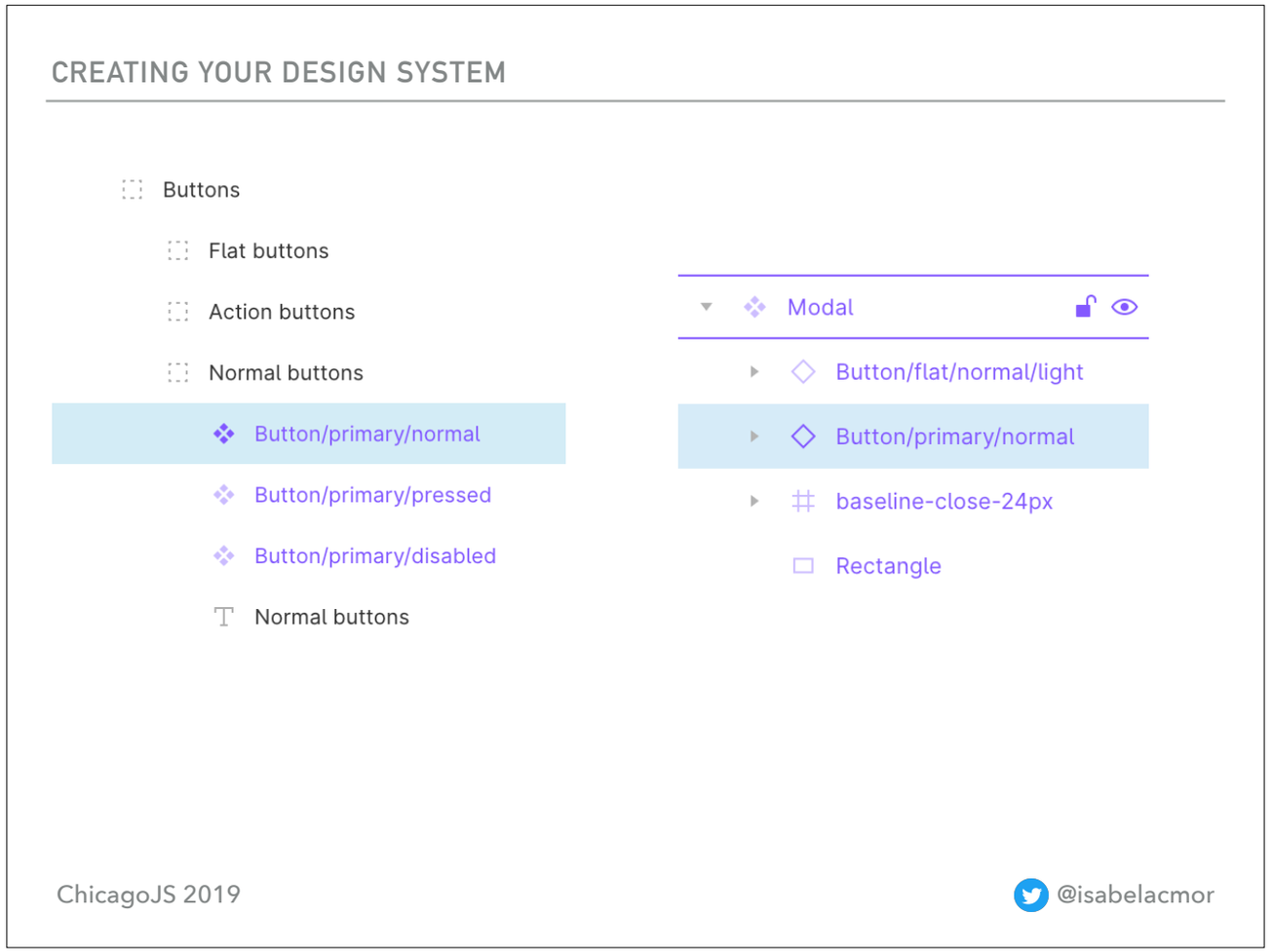
In the end, you end up with a library of all the styles you'll be using throughout your design system.



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.



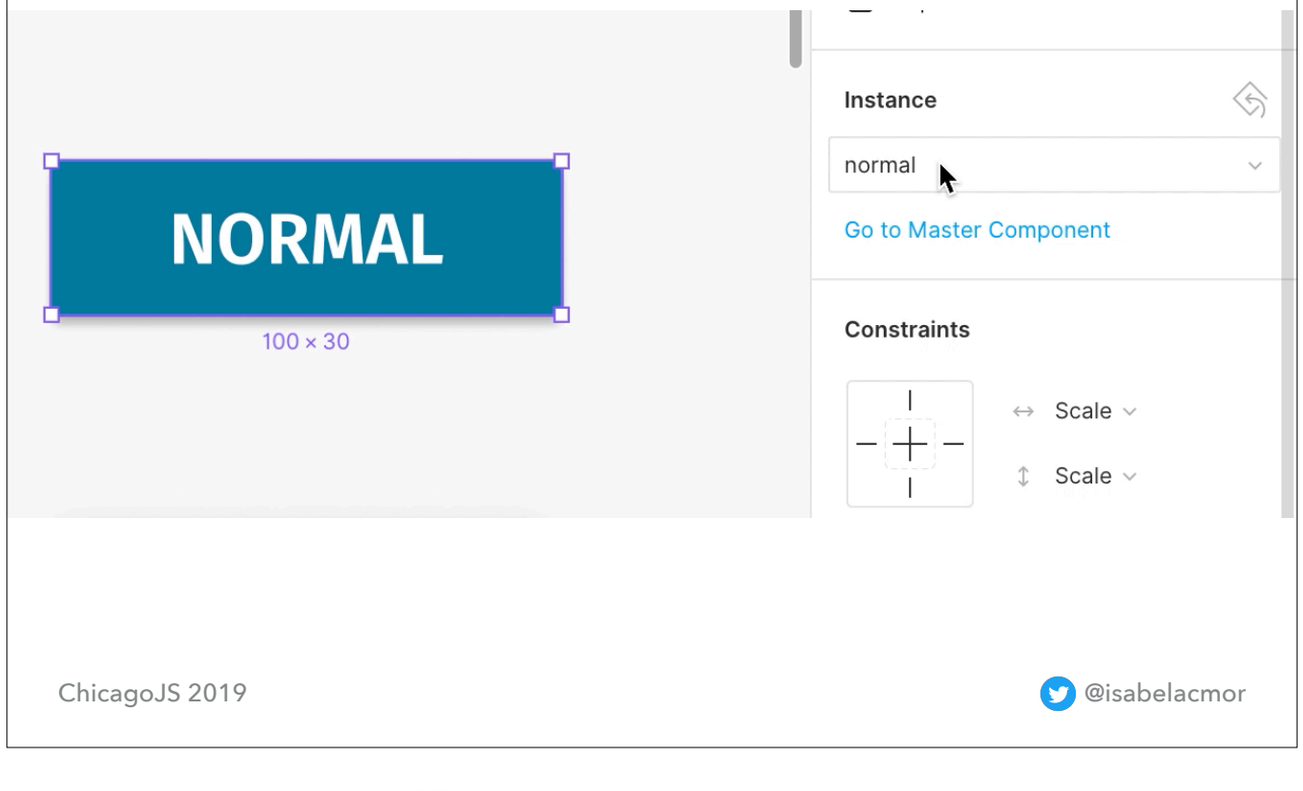
Once I have some foundational components designed, I can start designing more complex pieces like modals and context menus. All the smaller components should live in your library and be reused to build up larger pieces. For example, for this modal, we don’t create a brand new button. We simple copy our 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.



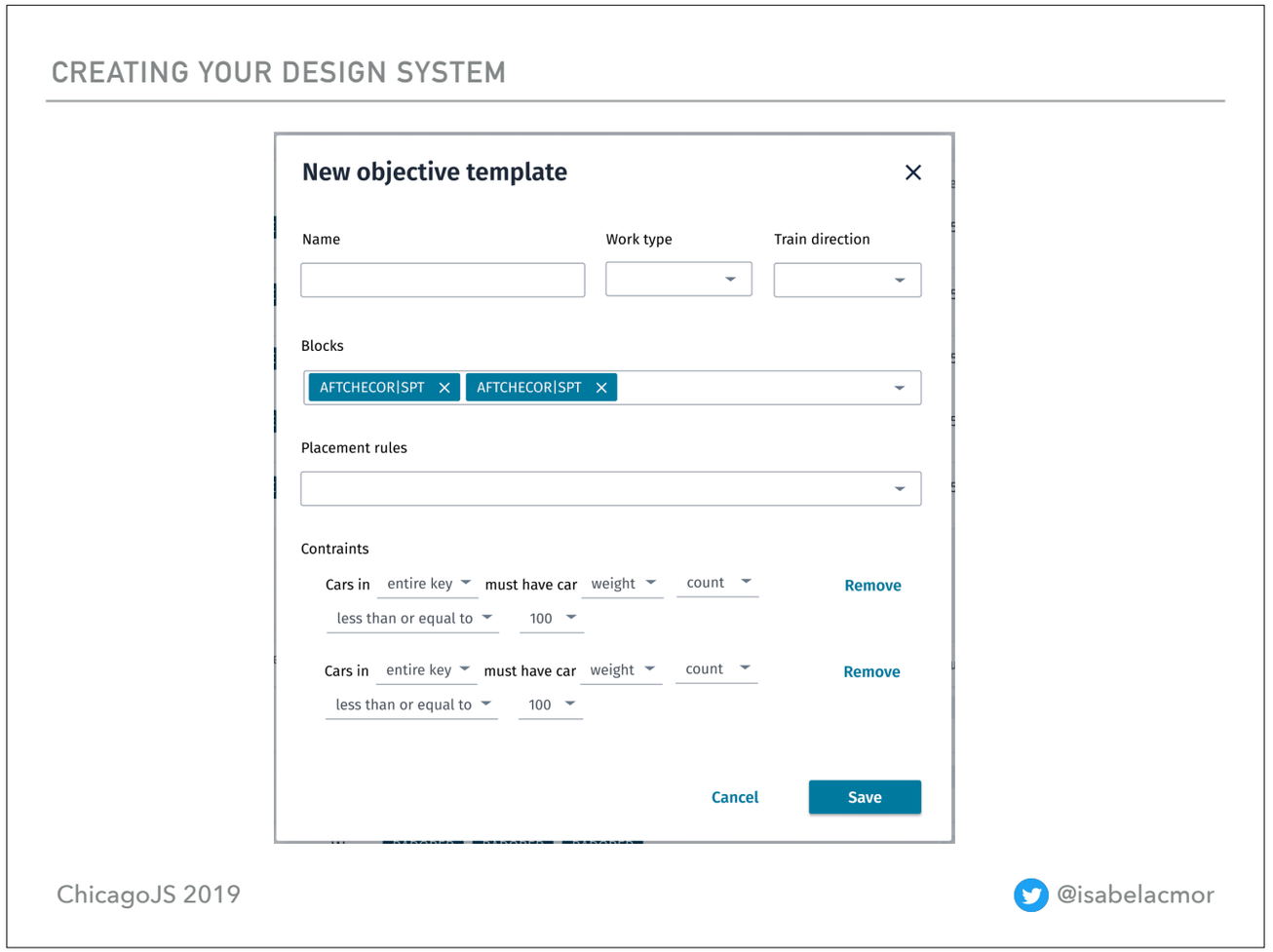
Just like the library we created for colors and fonts, we can group all the pieces needed for a particular component into one symbol that represents that component in our library. For example, a button is made up of two pieces - a text label and a background. I've grouped these two pieces into a single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

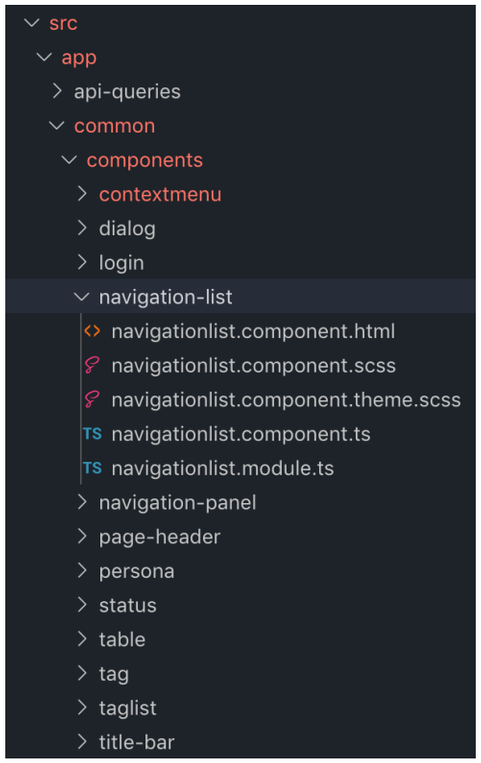


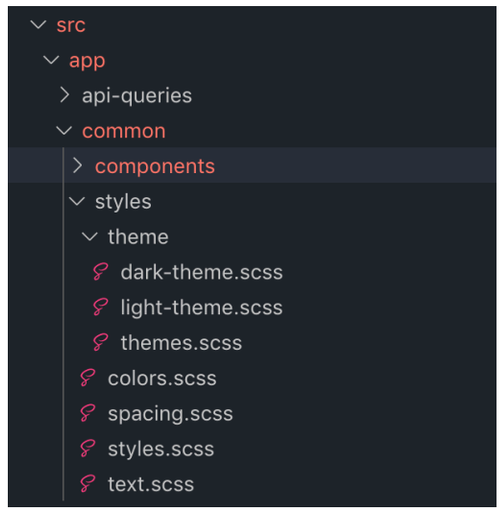
Reusing components like this also lets me quickly change between different instances of components, without needing to dig into the actual design. If I wanted to change to a pressed state for my button, all I would need to do is change which instance of the main button I'm using.



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

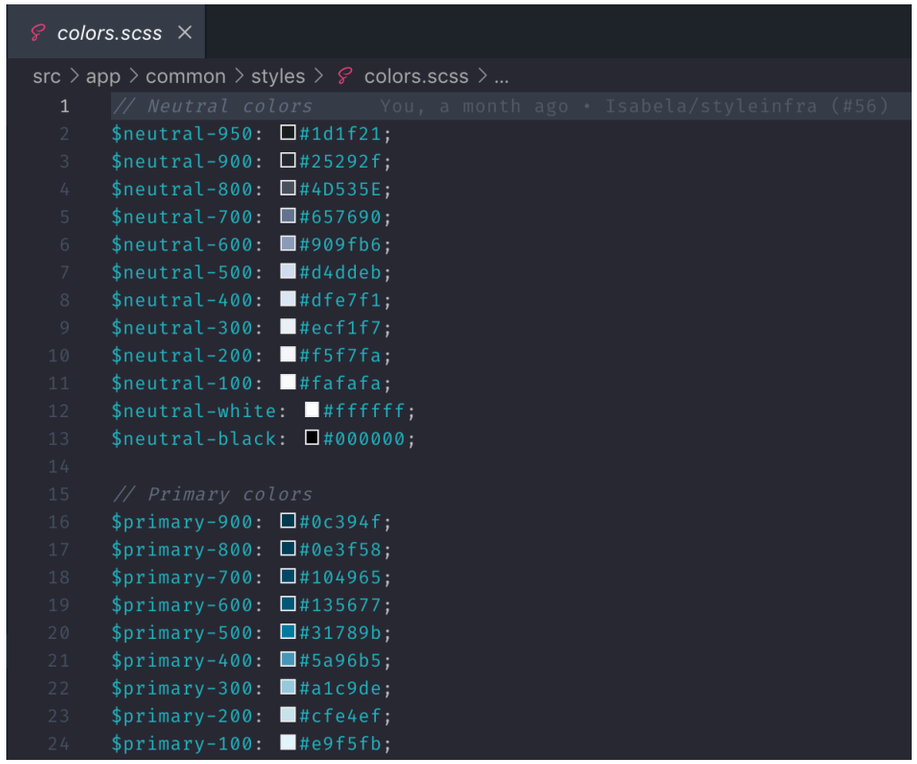




When you’re ready to move your designs into code, you should follow the same organizational structure you did in your design system. Your specific approach may different depending on your codebase or company, but what I’ve found works well for me and my team is the following file structure.

I have all common, reusable components under src/app/common/components. Each components gets its own folder so that it’s 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



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 > ...

```scss
$padding-xxs: 4px;      You, a month ago • Isabela/styleinfra (#56)
$padding-xs: 8px;
$padding-sm: 12px;
$padding-md: 20px;
$padding-lg: 30px;

$border-radius: 2px;
$border-radius-round: 50%;

$card-spacing: 50px;
```

text.scss

src > app > common > styles > text.scss > .body-sm-alt

```scss
@import url("https://fonts.googleapis.com/css?family=Fira+Sans:400,500,600&display=swap");

h1 {
  font-family: "Fira Sans", sans-serif;
  font-weight: 600;
  font-size: 34px;
  line-height: 41px;
}

h2 {
  font-family: "Fira Sans", sans-serif;
  font-weight: 600;
  font-size: 22px;
  line-height: 28px;
}
```


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: $neutra
app-header-border-color: $neut
app-header-text-color: $neutra
persona-background: $yellow-60
persona-text-color: $neutral-w

// 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,
```

```
.page-header {
  @extend h2;
  color: $page-header-text-color;
  margin-bottom: $padding-lg;
}
```


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

```
<div class="nav-panel-container">
    <h2>{{activeCustomer}}</h2>

    <!-- List of yards -->
    <h4 class="section-header">YARDS</h4>
    <NavigationList class="scrollable spacing" [items]="yards$ | async" [isLoading]="isLoading"></NavigationList>

    <!-- Settings actions -->
    <h4 class="section-header">SETTINGS</h4>
    <NavigationList class="scrollable" [items]="settingsItems"></NavigationList>
```

navigationlist.component.html

src > app > common > components > navigation-list > navigationlist.component.html > div

```

<div class="nav-list-container">
    <span *ngIf="isEmpty()" class="empty-message">No yards found</span>
    <nav *ngIf="hasItems()" mat-tab-nav-bar selectedIndex="0">
        <a *ngFor="let item of items; index as i;" mat-tab-link [routerLink]="item.routeUrl" routerLinkActive
            #rla="routerLinkActive" [active]="rla.isActive">{{item.name}}</a>
    </nav>
</div>
```


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>
  ));
```


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>
  ));
```


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.

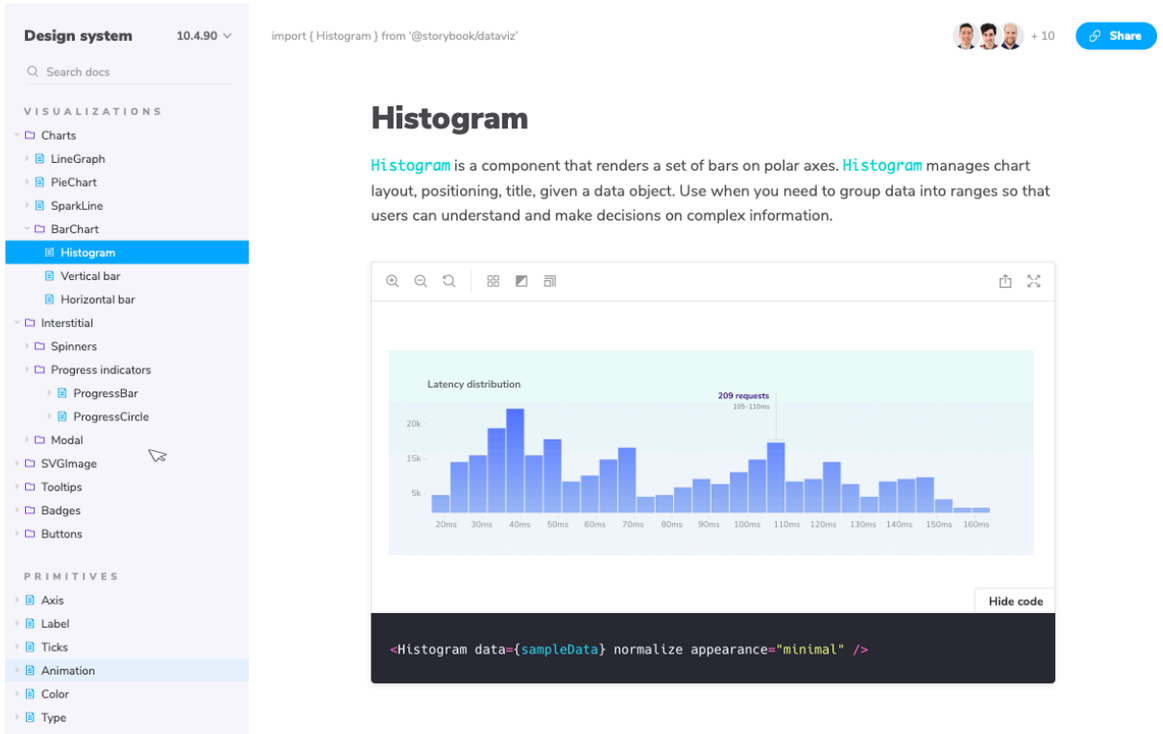
DOCUMENTING YOUR DESIGN SYSTEM

```
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>
  ));
```


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



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.

THANK YOU 🎉

- 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.