Refactoring UI

by Adam Wathan & Steve Schoger

Contents

Starting from Scratch

Start with a feature, not a layout

When you start the design for a new app idea, what do you design first? If it's the navigation bar at the top of the page, you're making a mistake.

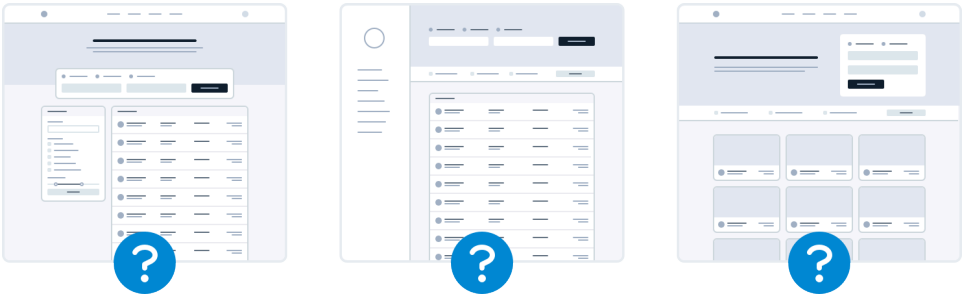
The easiest way to find yourself frustrated and stuck when working on a new design is to start by trying to "design the app." When most people think about "designing the app", they're thinking about the *shell*.

Should it have a top nav, or a sidebar?

Should the navigation items be on the left, or on the right?

Should the page content be in a container, or should it be full-width?

Where should the logo go?



The thing is, an "app" is actually a collection of *features*. Before you've designed a few features, you don't even have the information you need to make a decision about how the navigation should work. No wonder it's frustrating!

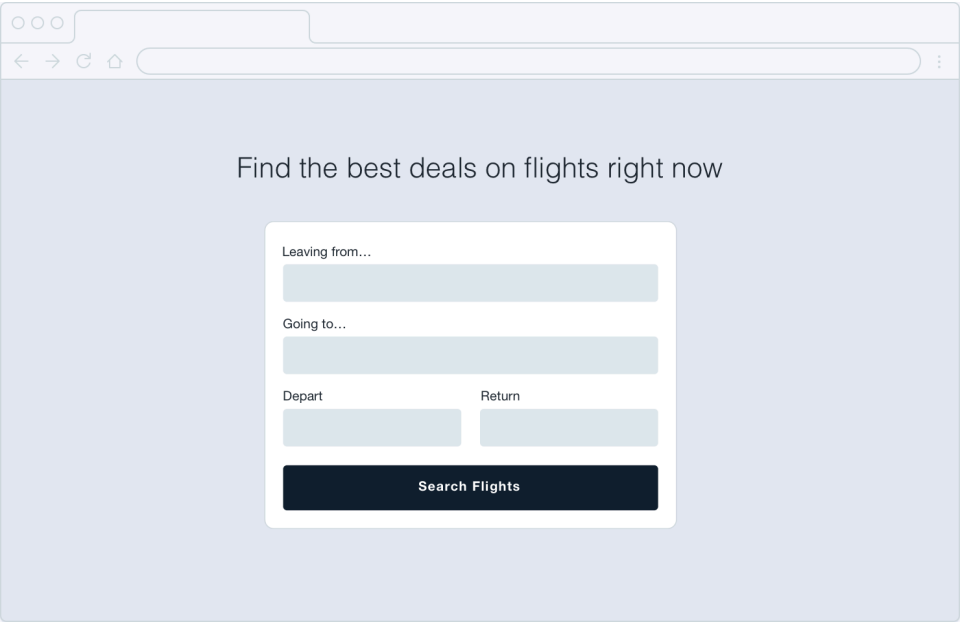
Instead of starting with the shell, start with a piece of actual functionality.

For example, say you're building a flight booking service. You could start with a feature like "searching for a flight".

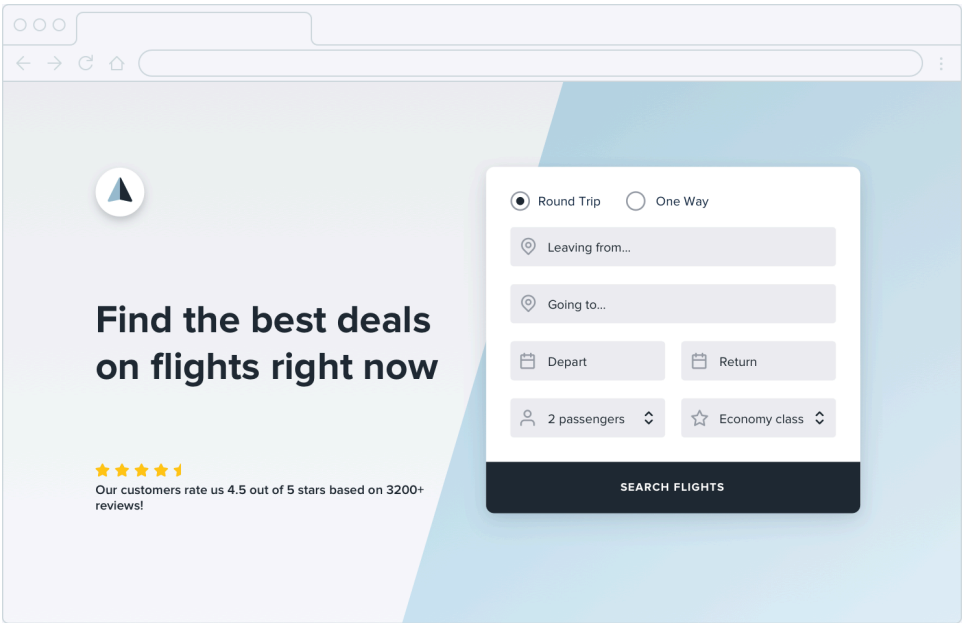
Your interface will need:

- A field for the departure city
- A field for the destination city
- A field for the departure date
- A field for the return date
- A button to perform the search

Start with that.



Hell, you might not even need that other stuff anyways — it worked for Google.



Detail comes later

In the earliest stages of designing a new feature, it's important that you don't get hung up making low-level decisions about things like typefaces, shadows, icons, etc.

That stuff will all matter eventually, but it doesn't matter right now.

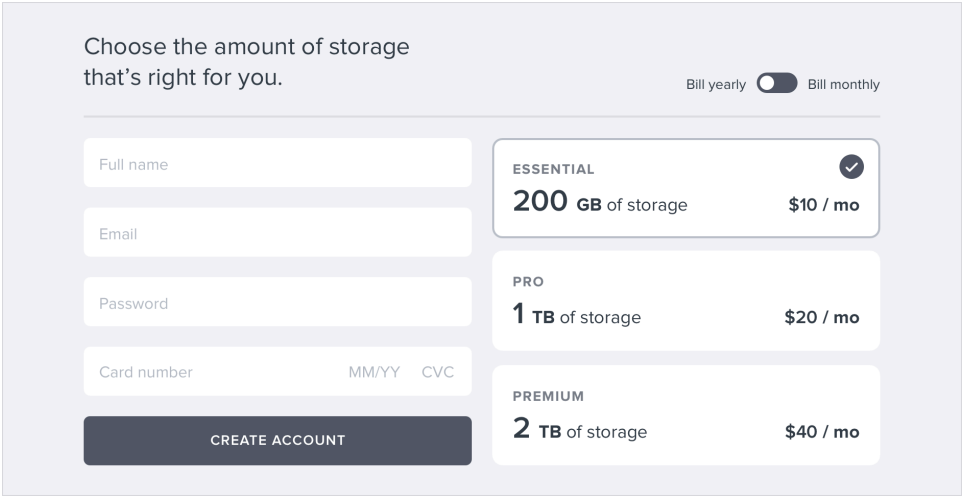
If you have trouble ignoring the details when working in a high fidelity environment like the browser or your favorite design tool, one trick Jason Fried of Basecamp likes to use is to design on paper using a thick Sharpie.

Obsessing over little details just isn't possible with a Sharpie, so it can be a great way to quickly explore a bunch of different layout ideas.

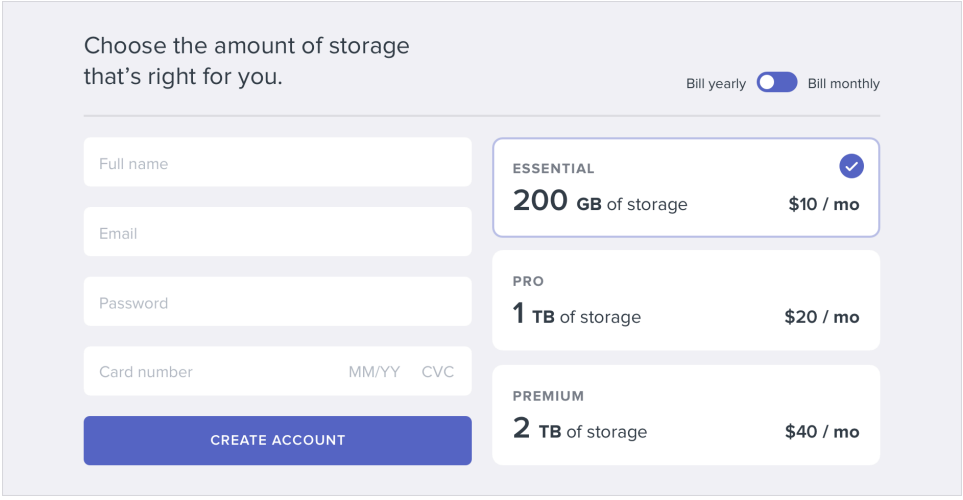
Hold the color

Even when you're ready to refine an idea in higher fidelity, resist the temptation to introduce color right away.

By designing in grayscale, you're forced to use spacing, contrast, and size to do all of the heavy lifting.



It's a little more challenging, but you'll end up with a clearer interface with a strong hierarchy that's easy to enhance with color later.



Don’t over-invest

The whole point of designing in low-fidelity is to be able to move fast, so you can start building the real thing as soon as possible.

Sketches and wireframes are disposable — users can’t do anything with static mockups. Use them to explore your ideas, and leave them behind when you’ve made a decision.

Don’t design too much

You don’t need to design every single feature in an app before you move on to implementation; in fact, it’s better if you don’t.

Figuring out how every feature in a product should interact and how every edge case should look is really hard, especially in the abstract.

How should this screen look if the user has 2000 contacts?

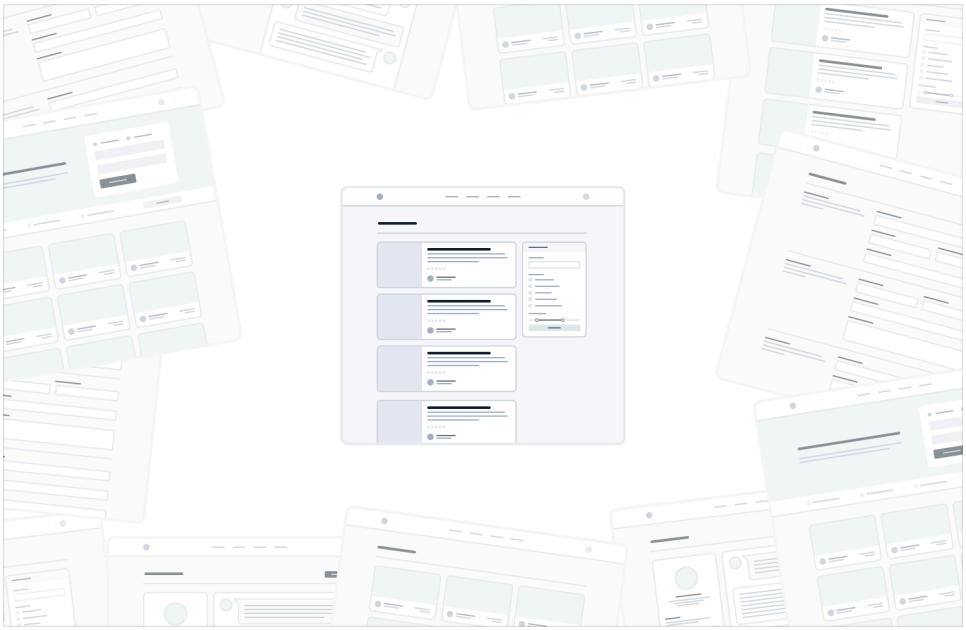
Where should the error message go in this form?

How should this calendar look when there are two events scheduled at the same time?

You’re setting yourself up for frustration by trying to figure this stuff out using only a design tool and your imagination.

Work in cycles

Instead of designing everything up front, work in short cycles. Start by designing a simple version of the next feature you want to build.

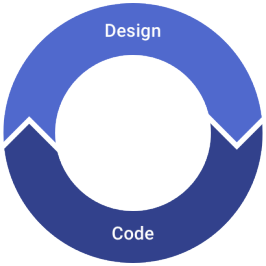


Once you’re happy with the basic design, *make it real*.

You’ll probably run into some unexpected complexity along the way, but that’s the point — it’s a lot easier to fix design problems in an interface you can actually use than it is to imagine every edge case in advance.

Iterate on the working design until there are no more problems left to solve,

then jump back into design mode and start working on the next feature.

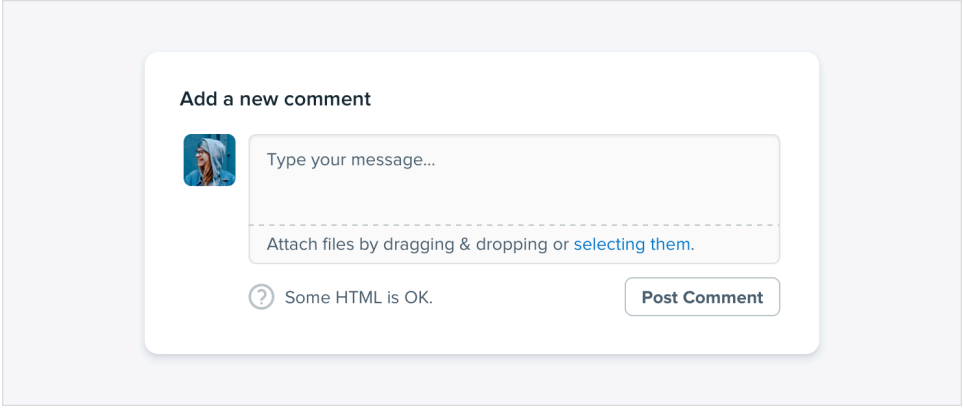


Don’t get overwhelmed working in the abstract. Build the real thing as early as possible so your imagination doesn’t have to do all the heavy lifting.

Be a pessimist

Don’t imply functionality in your designs that you aren’t ready to build.

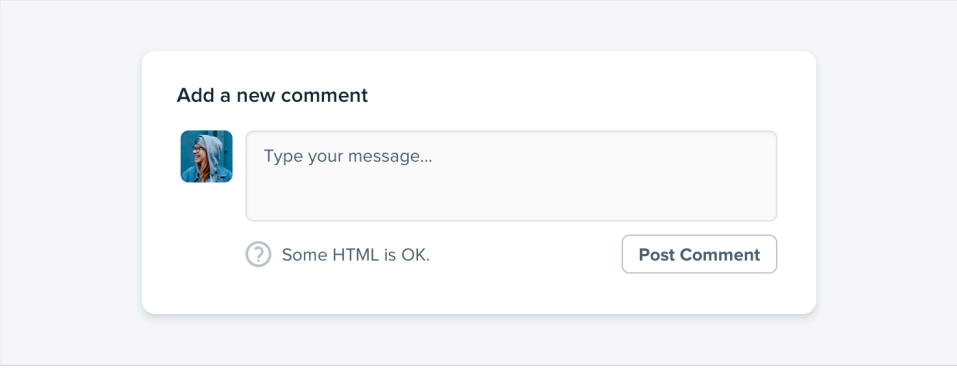
For example, say you’re working on a comment system for a project management tool. You know that one day, you’d like users to be able to attach files to their comments, so you include an attachments section in your design.



You get deep into implementation only to discover that supporting attachments is going to be *a lot* more work than you anticipated. There’s no way you have time to finish it right now, so the whole commenting system sits on the backburner while you take care of other priorities.

The thing is, a comment system with no attachments would still have been better than no comment system at all, but because you planned to include it from day one you’ve got nothing you can ship.

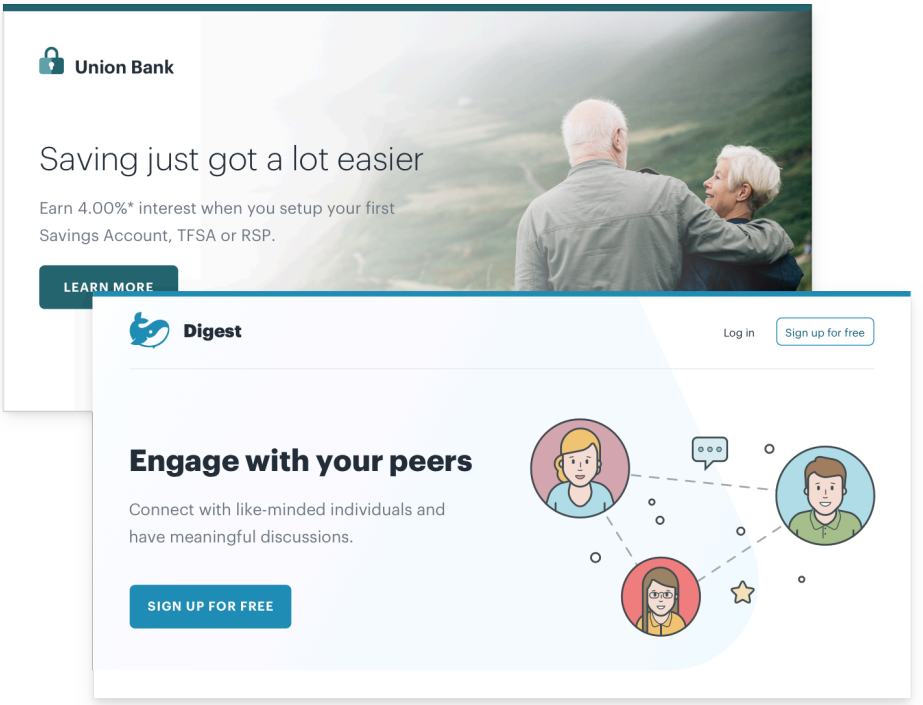
When you’re designing a new feature, **expect it to be hard to build.** Designing the smallest useful version you can ship reduces that risk considerably.



If part of a feature is a “nice-to-have”, **design it later**. Build the simple version first and you’ll always have something to fall back on.

Choose a personality

Every design has some sort of personality. A banking site might try to communicate *secure* and *professional*, while a trendy new startup might have a design that feels *fun* and *playful*.

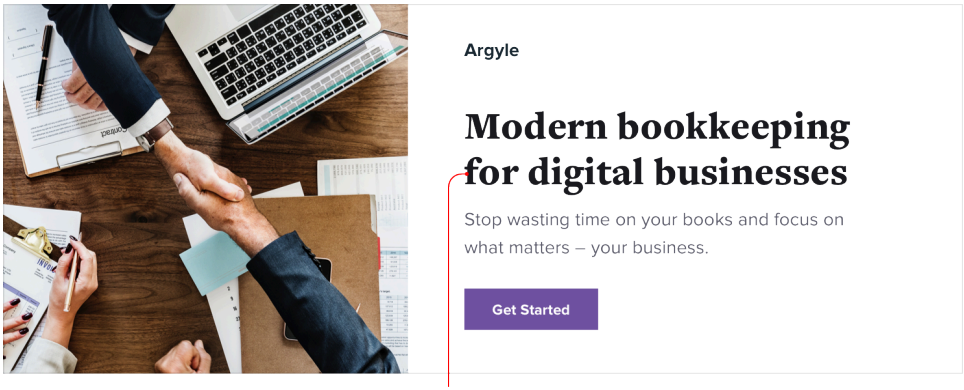


On the surface, giving a design a particular personality might sound abstract and handwavy, but a lot of it is determined by a few solid, concrete factors.

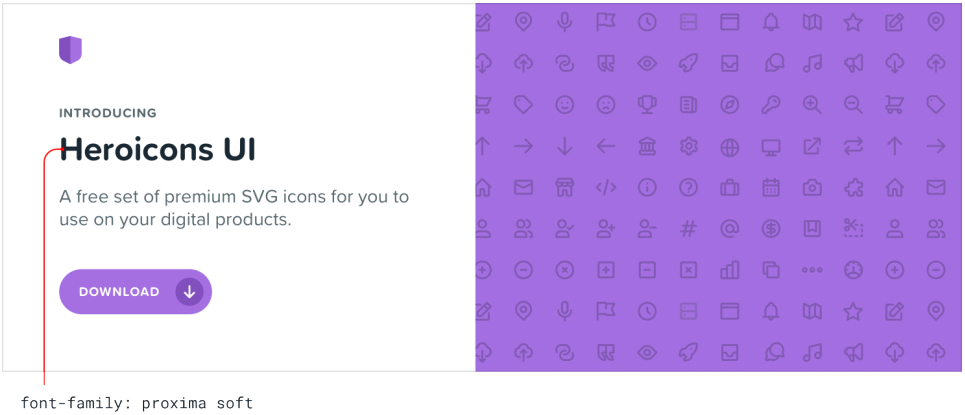
Font choice

Typography plays a huge part in determining how a design feels.

If you want an elegant or classic look, you might want to incorporate a serif typeface in your design:



For a playful look, you could use a rounded sans serif:



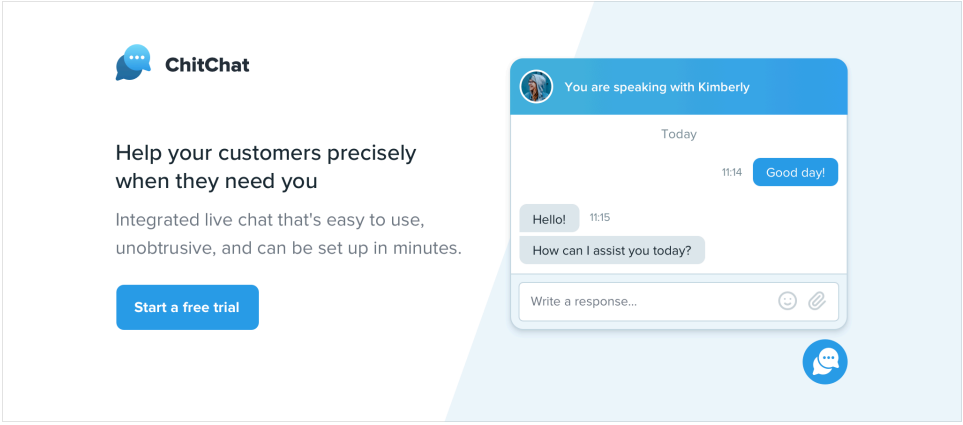
If you're going for a plainer look, or want to rely on other elements to provide the personality, a neutral sans serif works great:


```
font-family: freight sans;
```

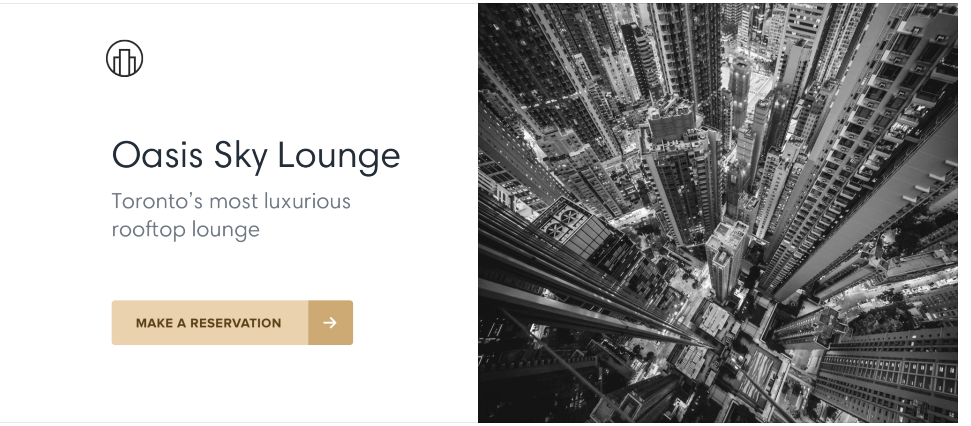
Color

There's a lot of science out there on the psychology of color, but in practice, you really just need to pay attention to how different colors feel to you.

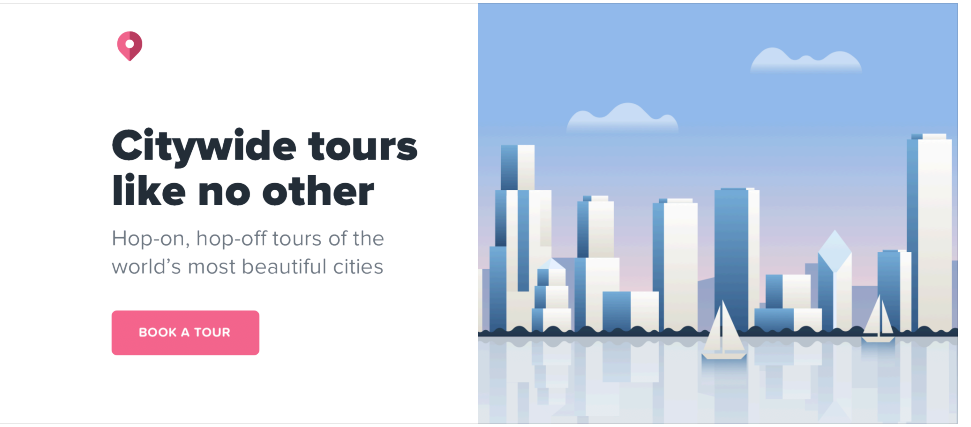
Blue is safe and familiar — nobody ever complains about blue:



Gold might say “expensive” and “sophisticated”:



Pink is a bit more fun, and not so serious:

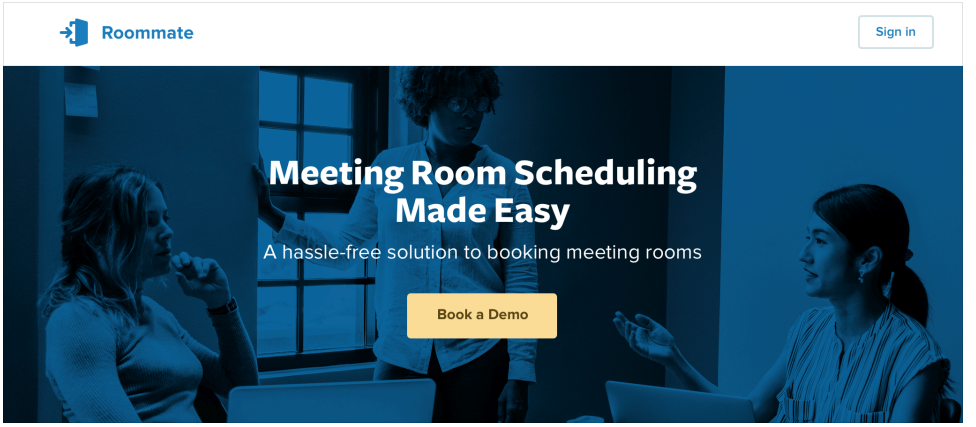


While trying to choose colors using *only* psychology isn’t super practical — a lot of it is just about what looks good to you — it can be helpful to think about when you’re trying to understand *why* you think a color is the right fit.

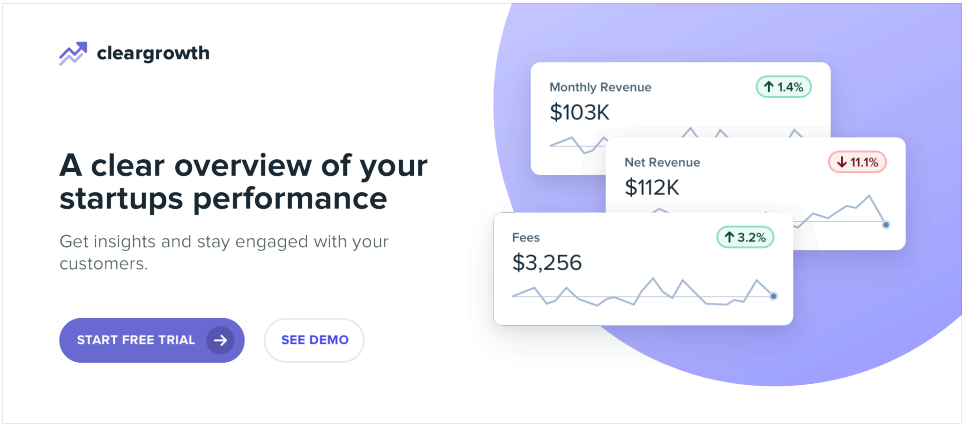
Border radius

As small of a detail as it sounds, if and how much you round the corners in your design can have a big impact on the overall feel.

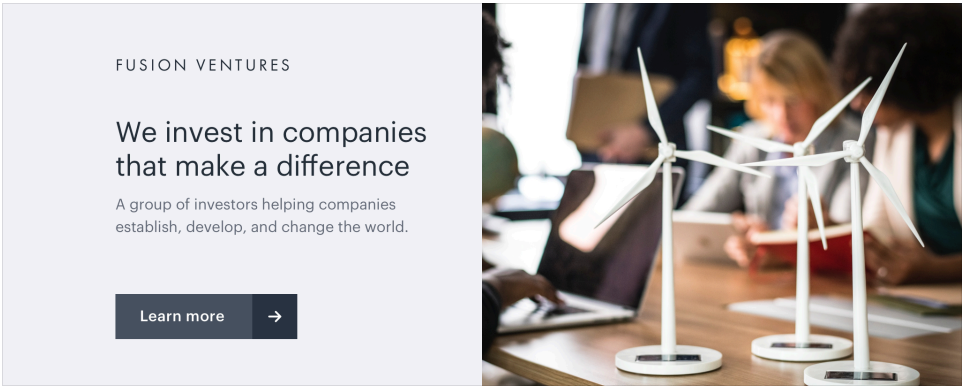
A small border radius is pretty neutral, and doesn't really communicate much of a personality on its own:



A large border radius starts to feel more playful:



...while no border radius at all feels a lot more serious or formal:

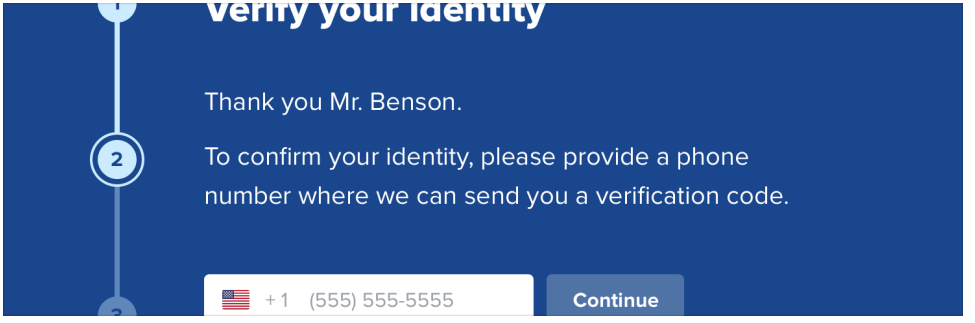


Whatever you choose, it's important to stay consistent. Mixing square corners with rounded corners in the same interface almost always looks worse than sticking with one or the other.

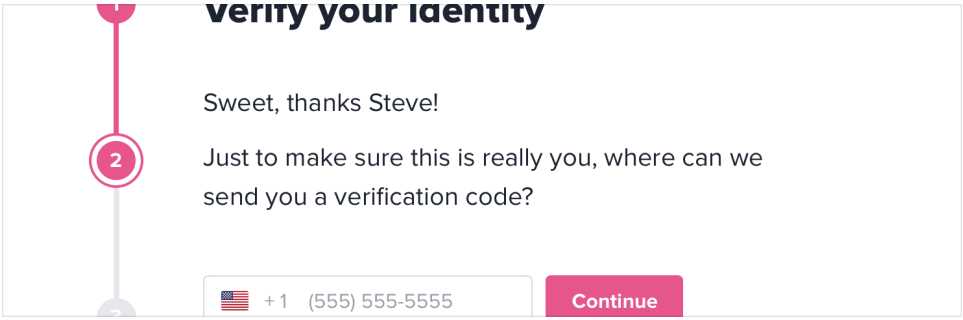
Language

While not a visual design technique per se, the words you use in an interface have a massive influence on the overall personality.

Using a less personal tone might feel more official or professional:



...while using friendlier, more casual language makes a site feel, well, friendlier:



Words are everywhere in a user interface, and choosing the right ones is just as (if not more) important than choosing the right color or typeface.

Deciding what you actually want

A lot of the time you'll probably just have a gut feeling for the personality you're going for. But if you don't, a great way to simplify the decision is to take a look at other sites used by the people who want to reach.

If they are mostly pretty "serious business", maybe that's how your site should look too. If they are more playful with a bit of humor, maybe that's a better direction to take.

Just try not to borrow too much from direct competitors, you don't want to look like a second-rate version of something else.

Limit your choices

Having millions of colors and thousands of fonts to choose from might sound nice in theory, but in practice it's usually a paralyzing curse.

And it's not just fonts and colors, either — you can easily waste time agonizing over almost any minor design decision.

Should this text be 12px or 13px?

Should this box shadow have a 10% opacity or a 15% opacity?

Should this avatar be 24px or 25px tall?

Should I use a medium font weight for this button or semibold?

Should this headline have a bottom margin of 18px or 20px?

When you're designing without constraints, decision-making is torture because there's always going to be more than one right choice.

For example, these buttons all have different background colors, but it's almost impossible to tell the difference between them by just looking at them.

How are you supposed to make a confident decision if none of these would really be bad choices?

Define systems in advance

Instead of hand-picking values from a limitless pool any time you need to make a decision, *start with a smaller set of options*.

Don't reach for the color picker every time you need to pick a new shade of blue — choose from a set of 8-10 shades picked out ahead of time.

Similarly, don't tweak a font size one pixel at a time until it looks perfect. Define a restrictive type scale in advance and use that to make any future font size decisions.

12px The quick brown fox jumps over the lazy dog

14px The quick brown fox jumps over the lazy dog

16px The quick brown fox jumps over the lazy dog

18px The quick brown fox jumps over the lazy dog

20px The quick brown fox jumps over the lazy dog

24px The quick brown fox jumps over the lazy dog

32px The quick brown fox jumps over the lazy dog

48px The quick brown fox jumps over th

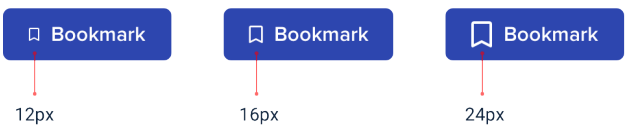
When you build systems like this, you only have to do the hard work of picking the initial values *once* instead of every time you're designing a new piece of UI. It's a bit more work up front, but it's worth it — it'll save you a ton of decision fatigue down the road.

Designing by process of elimination

When you're designing using a constrained set of values, decision-making is a lot easier because there are a lot fewer "right" choices.

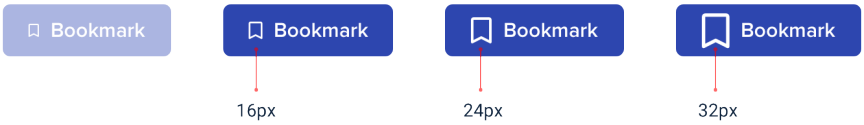
For example, say you're trying to choose a size for an icon. You've defined a sizing scale in advance where your only small-to-medium sized options are 12px, 16px, 24px, and 32px.

To pick the best option, start by taking a guess at which one will look best, maybe 16px. Then try the values on either side (12px and 24px) for comparison.



Chances are, two of those options will seem like *obviously* bad choices. If it's the options on the outside, you're done — the middle option is the only good choice.

If one of the outer options looks best, do another comparison using that option as the “middle” value and make sure there’s not a better choice.



This approach works for anything where you’ve defined a system. When you’re limited to a set of options that all look noticeably different, picking the best one is a piece of cake.

Systematize everything

The more systems you have in place, the faster you’ll be able to work and the less you’ll second guess your own decisions.

You’ll want systems for things like:

- Font size
- Font weight
- Line height
- Color
- Margin
- Padding
- Width
- Height
- Box shadows

- Border radius
- Border width
- Opacity

...and anything else you run into where it feels like you're laboring over a low-level design decision.

You don't have to define all of this stuff ahead of time, just make sure you're approaching design with a system-focused mindset. Look for opportunities to introduce new systems as you make new decisions, and try to avoid having to make the same minor decision twice.

Designing with systems is going to be a recurring theme throughout this book, and in later chapters we'll talk about building a lot of these systems in finer detail.

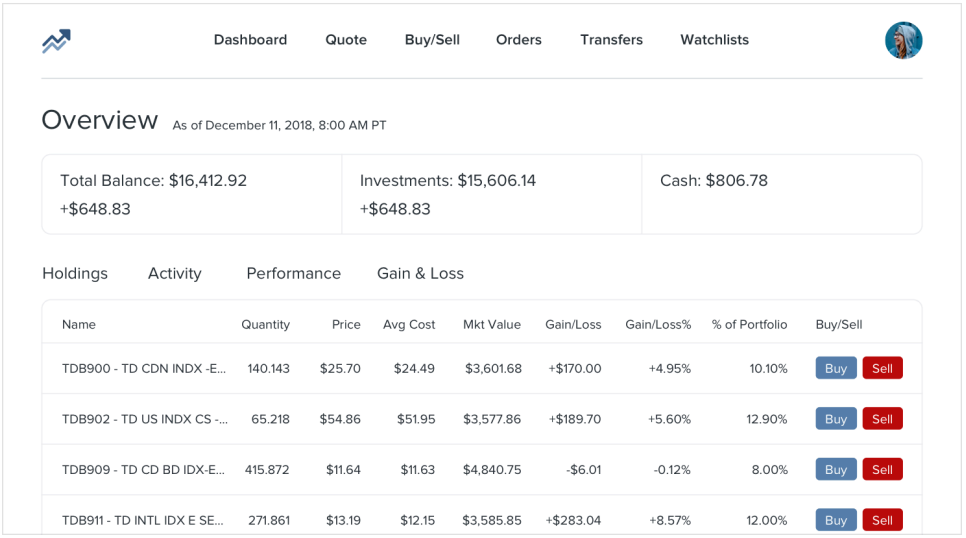
Hierarchy is Everything

Not all elements are equal

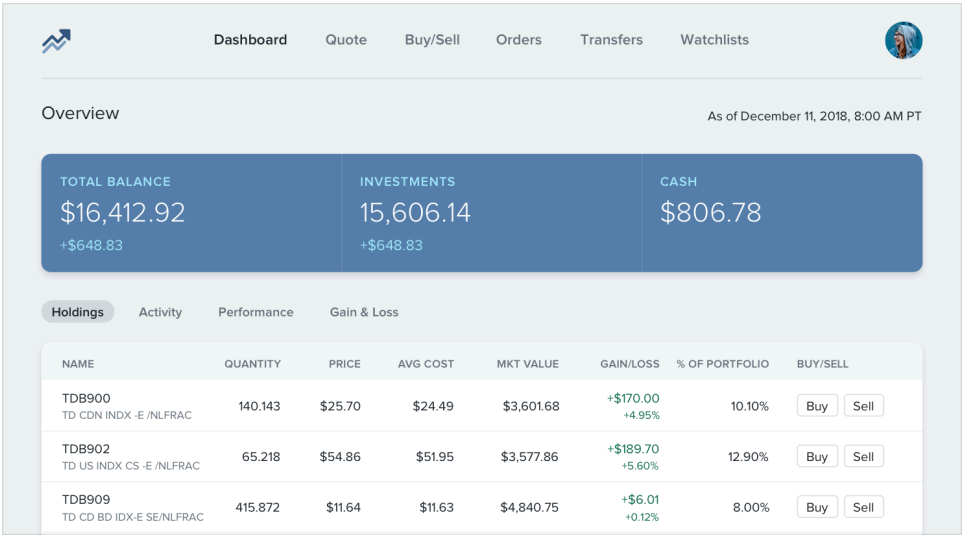
When you think of visual design as “styling things so they look good”, it’s easy to see why it might feel hard to achieve without innate artistic talent. But it turns out that one of the biggest factors in making something “look good” has nothing to do with superficial styling at all.

Visual hierarchy refers to how important the elements in an interface appear in relation to one another, and it’s the most effective tool you have for making something feel “designed”.

When everything in an interface is competing for attention, it feels noisy and chaotic, like one big wall of content where it’s not clear what actually matters:



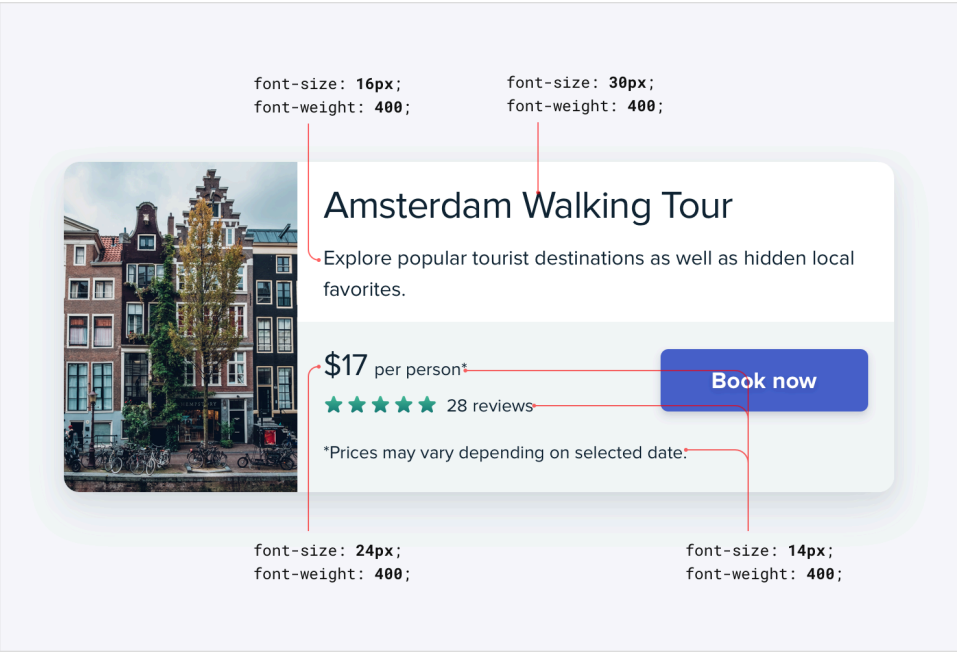
When you deliberately de-emphasize secondary and tertiary information, and make an effort to highlight the elements that are most important, the result is immediately more pleasing, even though the color scheme, font choice, and layout haven't changed:



So how do you actually make this happen? In the following chapters, we'll cover a number of specific strategies you can use to introduce hierarchy into your designs.

Size isn’t everything

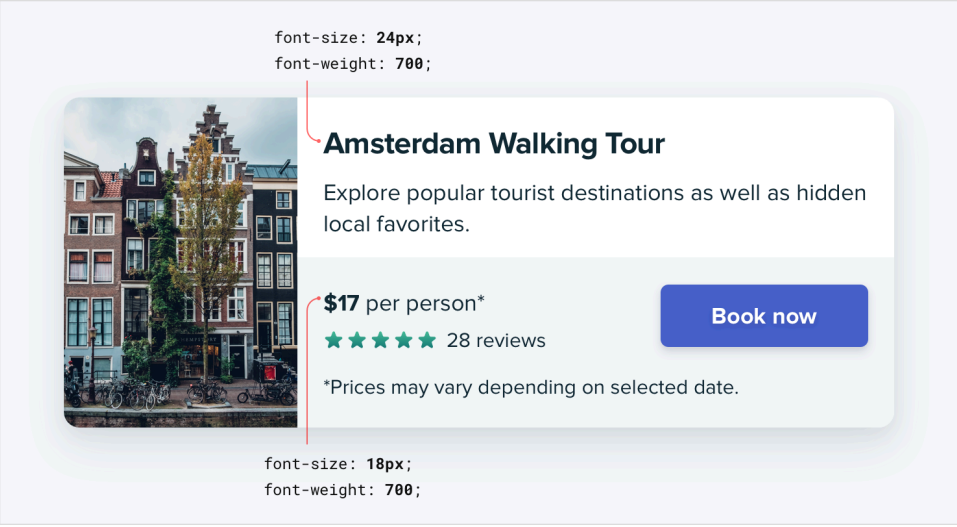
Relying too much on font size to control your hierarchy is a mistake — it often leads to primary content that’s too large, and secondary content that’s too small.



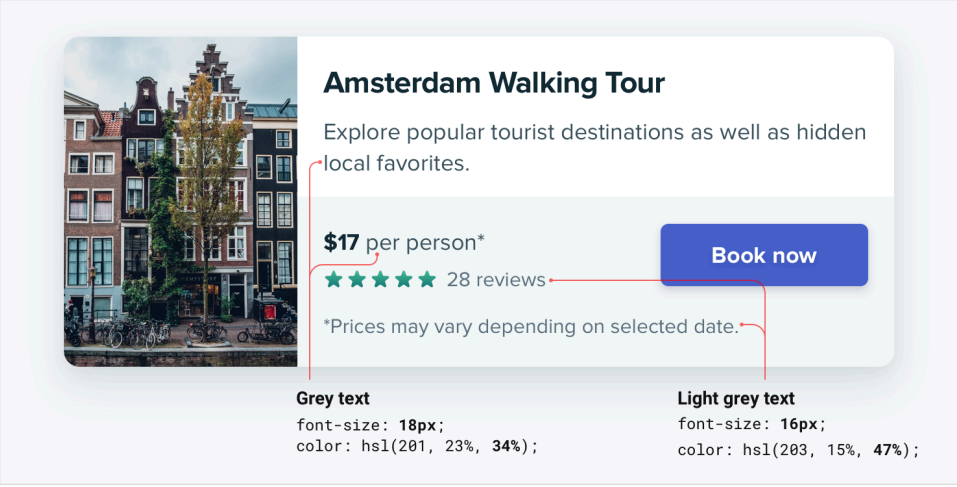
Instead of leaving all of the heavy lifting to font size alone, try using font weight or color to do the same job.

For example, making a primary element bolder lets you use a more

reasonable font size, and often does a better job at communicating its importance anyways:



Similarly, using a softer color for supporting text instead of a tiny font size makes it clear that the text is secondary while sacrificing less on readability:

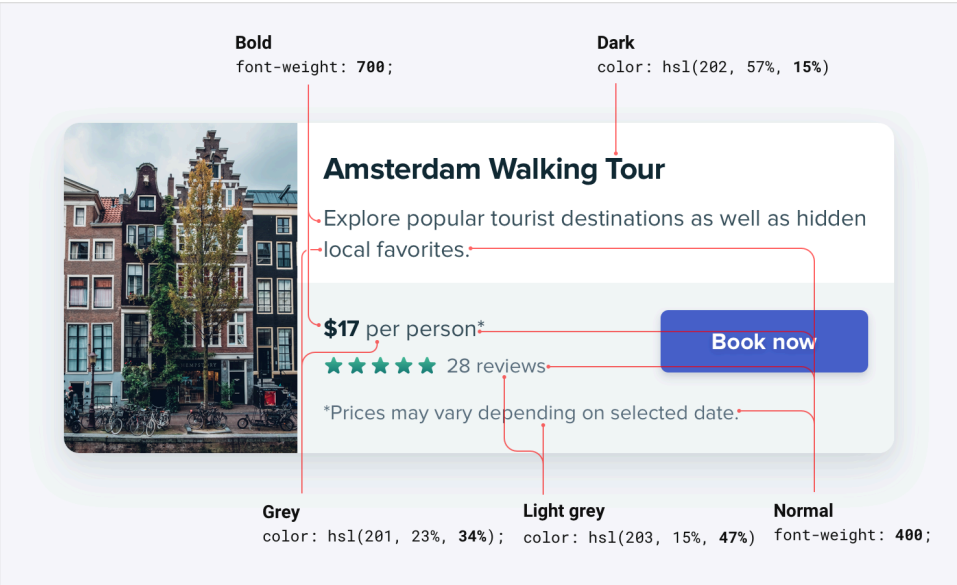


Try and stick to two or three colors:

- A dark color for primary content (like the headline of an article)
- A grey for secondary content (like the date an article was published)
- A lighter grey for tertiary content (maybe the copyright notice in a footer)

Similarly, two font weights are usually enough for UI work:

- A normal font weight (400 or 500 depending on the font) for most text
- A heavier font weight (600 or 700) for text you want to emphasize



Stay away from font weights under 400 for UI work — they can work for large headings but are too hard to read at smaller sizes. If you're considering

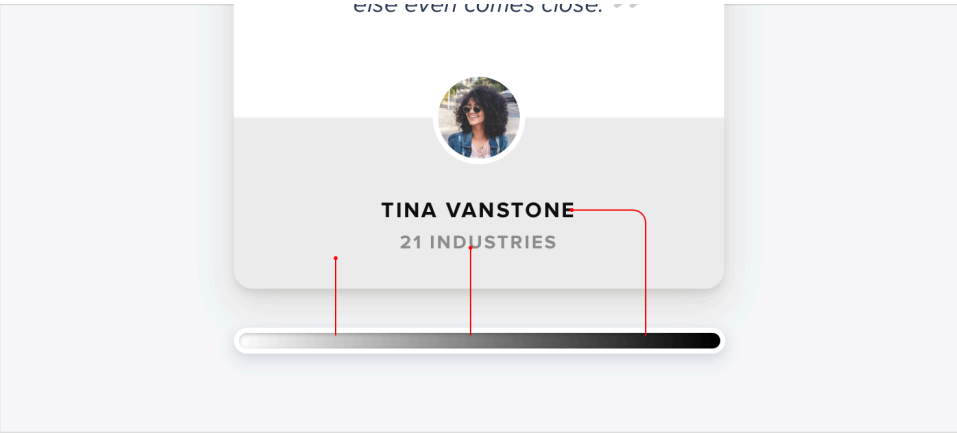
using a lighter weight to de-emphasize some text, use a lighter color or smaller font size instead.

Don’t use grey text on colored backgrounds

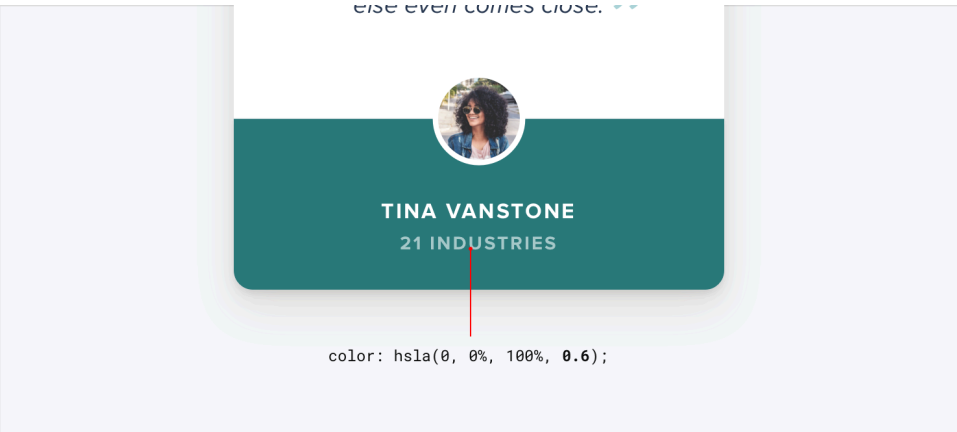
Making text a lighter grey is a great way to de-emphasize it on white backgrounds, but it doesn’t look so great on colored backgrounds.

That’s because the effect we’re actually seeing with grey on white is *reduced contrast*.

Making the text closer to the background color is what actually helps create hierarchy, not making it light grey.



You might think that the easiest way to achieve this is to use white text and reduce the opacity:

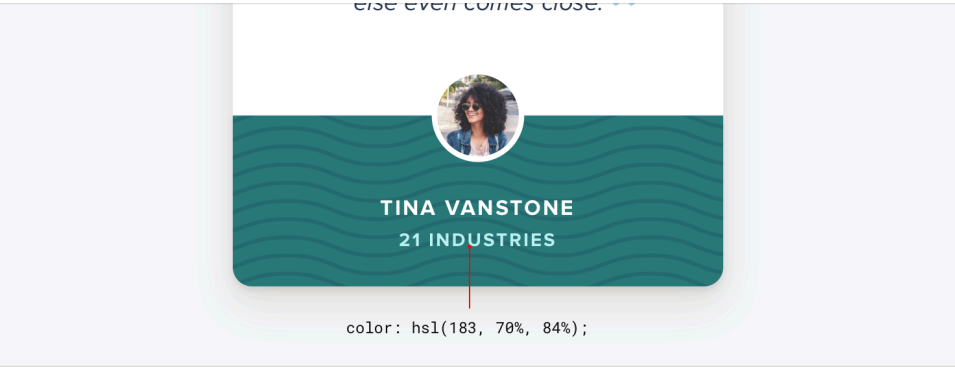


While this *does* reduce the contrast, it often results in text that looks dull, washed out, and sometimes even disabled.

Even worse, using this approach on top of an image or pattern means the background will show through the text:

A better approach is to *hand-pick a new color*, based on the background color.

Choose a color with the same hue, and adjust the saturation and lightness until it looks right to you:

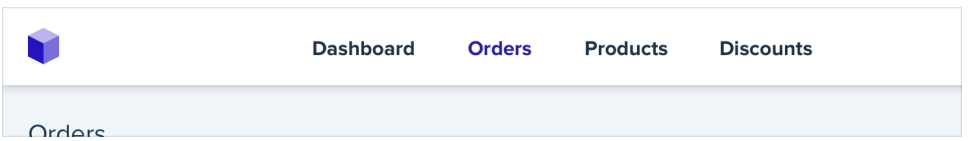


Hand-picking a color this way makes it easy to reduce the contrast without the text looking faded.

Emphasize by de-emphasizing

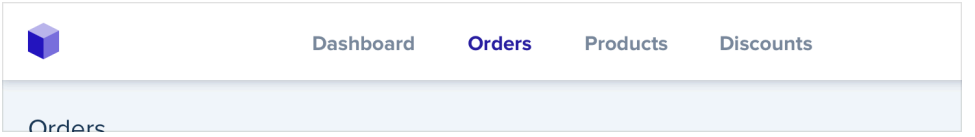
Sometimes you'll run into a situation where the main element of an interface isn't standing out enough, but there's nothing you can add to it to give it the emphasis it needs.

For example, despite trying to make this active nav item "pop" by giving it a different color, it still doesn't really stand out compared to the inactive items:



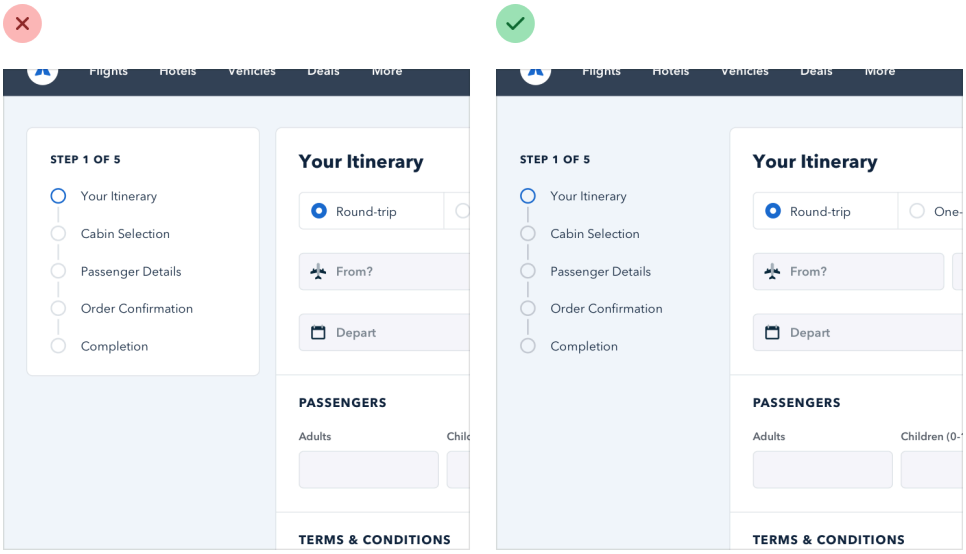
When you run into situations like this, instead of trying to further emphasize the element you want to draw attention to, figure out how you can *de-emphasize* the elements that are competing with it.

In this example, you could do that by giving the inactive items a softer color so they sit more in the background:



You can apply this thinking to bigger pieces of an interface as well. For

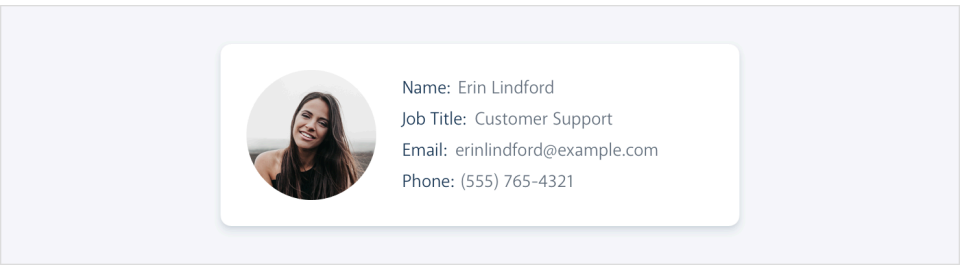
example, if a sidebar feels like it's competing with your main content area, don't give it a background color — let the content sit directly on the page background instead:



Labels are a last resort

Put down the accessibility pitchfork — this isn't about forms.

When presenting data to the user (especially data from the database), it's easy to fall into the trap of displaying it using a naive *label: value* format.



The problem with this approach is that it makes it difficult to present the data with any sort of hierarchy; every piece of data is given equal emphasis.

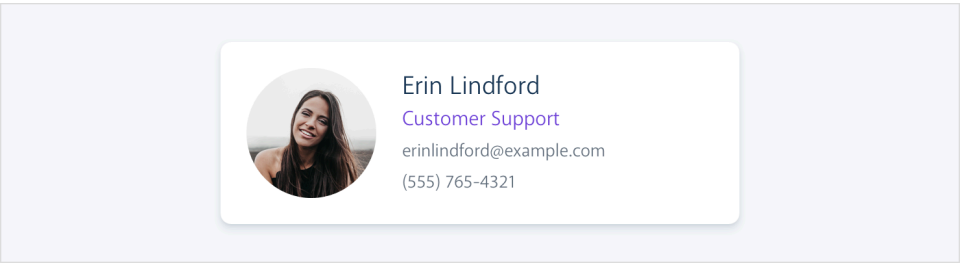
You might not need a label at all

In a lot of situations, you can tell what a piece of data is just by looking at the format.

For example, *janedoe@example.com* is an email address, *(555) 765-4321* is a phone number and *$19.99* is a price.

When the format isn't enough, the context often is. When you see the phrase *"Customer Support"* listed below someone's name in an employee directory,

you don't need a label to make the connection that that is the department the person works in.

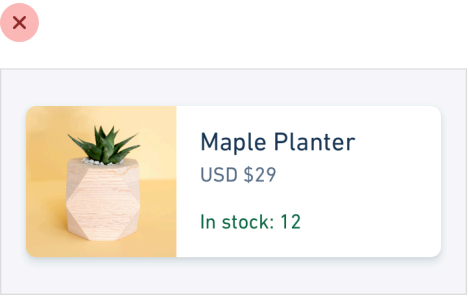


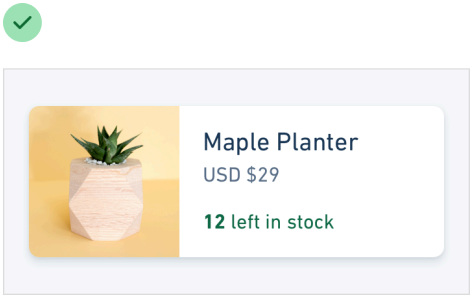
When you're able to present data without labels, it's much easier to emphasize important or identifying information, making the interface easier to use while at the same time making it feel more "designed".

Combine labels and values

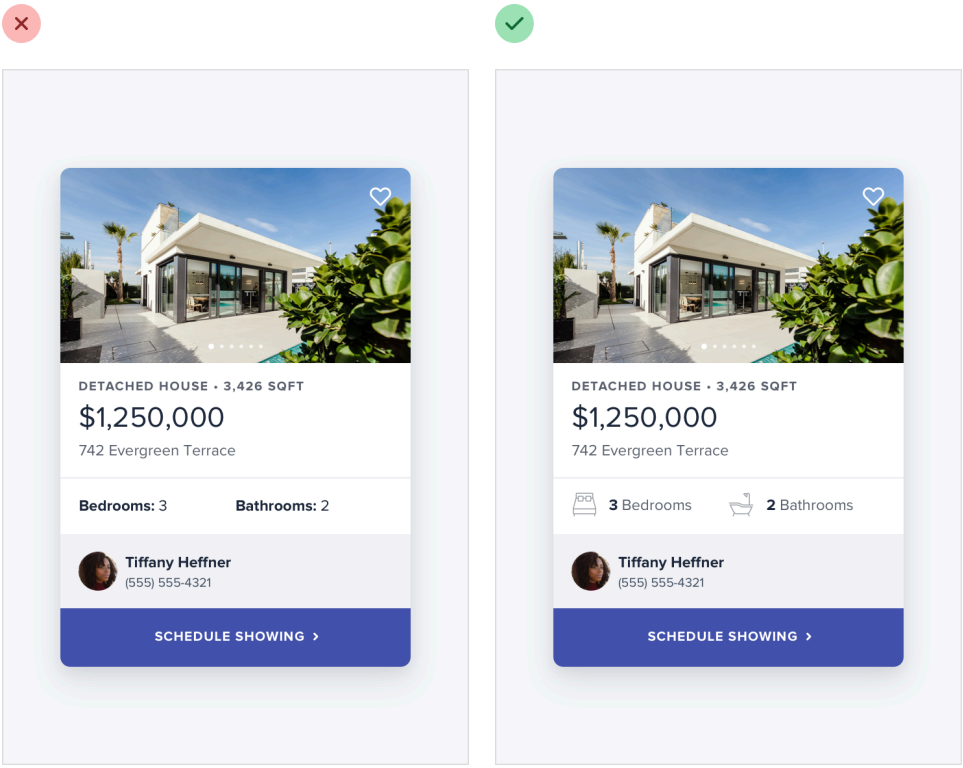
Even when a piece of data isn't completely clear without a label, you can often avoid adding a label by adding clarifying text to the value.

For example, if you need to display inventory in an e-commerce interface, instead of "In stock: 12", try something like "12 left in stock".





If you're building a real estate app, something like "Bedrooms: 3" could simply become "3 bedrooms".

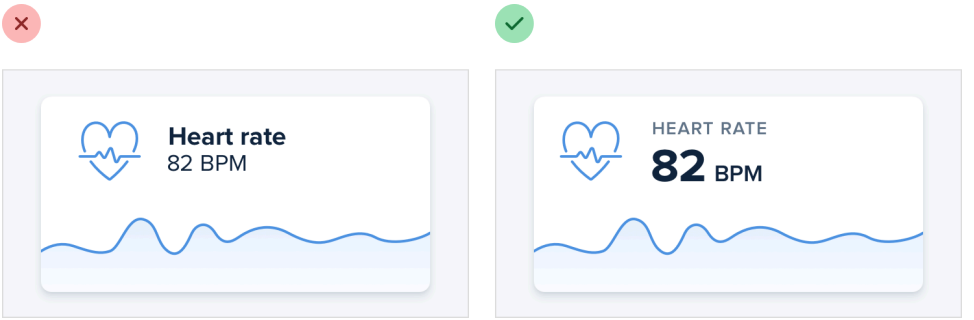


When you're able to combine labels and values into a single unit, it's much easier to give each piece of data meaningful styling without sacrificing on clarity.

Labels are secondary

Sometimes you really *do* need a label; for example when you're displaying multiple pieces of similar data and they need to be easily scannable, like on a dashboard.

In these situations, add the label, but treat it as supporting content. The data itself is what matters, the label is just there for clarity.



De-emphasize the label by making it smaller, reducing the contrast, using a lighter font weight, or some combination of all three.

When to emphasize a label

If you're designing an interface where you know the user will be *looking for* the label, it might make sense to the emphasize the label instead of the data.

This is often the case on information-dense pages, like the technical specifications of a product.

If a user is trying to find out the dimensions of a smartphone, they're probably scanning the page for words like "depth", not "7.6mm".

Dimensions	**Height:** 2.31 inches (58.6 mm) **Width:** 4.87 inches (123.8 mm) **Depth:** 0.30 inch (7.6 mm)
Power and Battery	**Talk time:** Up to 8 hours on 3G

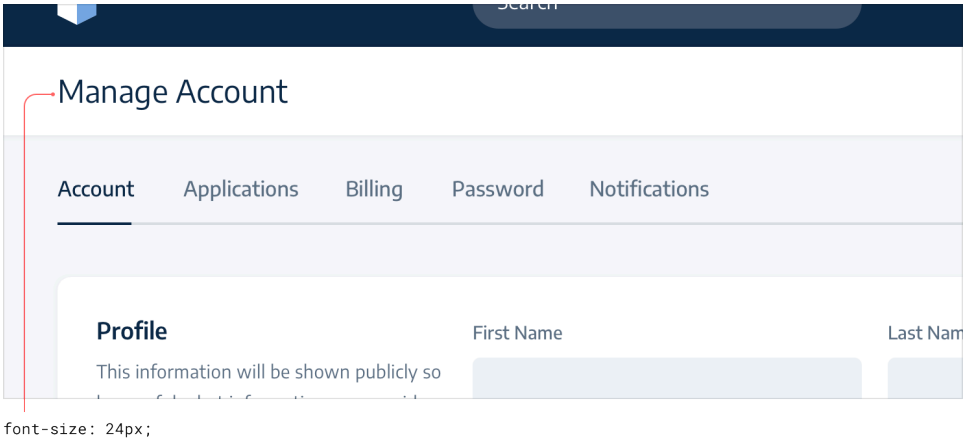
Don’t de-emphasize the data *too* much in these scenarios; it’s still important information. Simply using a darker color for the label and a slightly lighter color for the value is often enough.

Separate visual hierarchy from document hierarchy

It's important to use semantic markup when building for the web, which means you'll often be using heading tags like `h1`, `h2`, or `h3` if you decide to add a title to part of an interface.

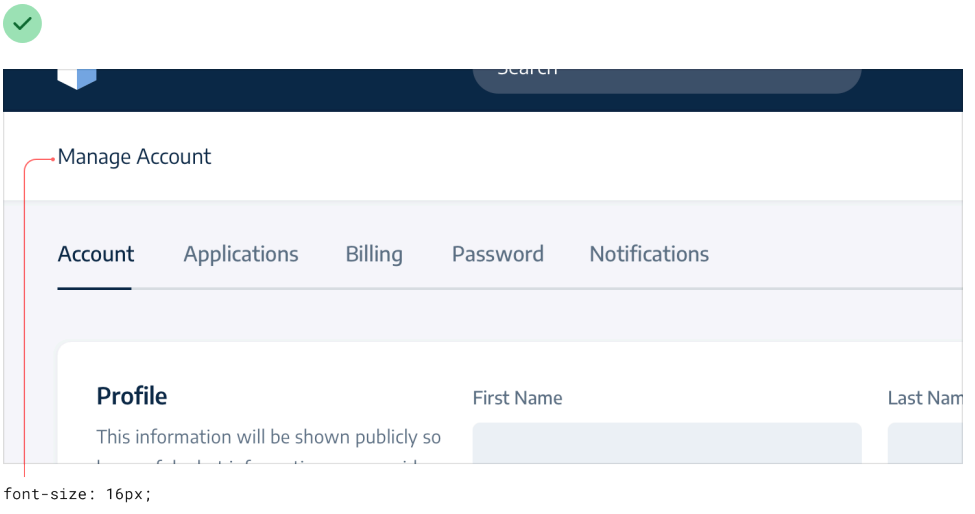
By default, web browsers assign progressively smaller font sizes to heading elements, so an `h1` is pretty large, and an `h6` is pretty small. This can be helpful for document-style content like articles or documentation, but it can encourage some bad decisions in application UIs.

Using an `h1` tag to add a title like *Manage Account* to a page makes perfect sense semantically, but because we're trained to believe that `h1` elements should be big, it's easy to fall into the trap of making those titles bigger than they really need to be.



A lot of the time, section titles act more like *labels* than headings — they are supportive content, they shouldn't be stealing all the attention.

Usually the *content* in that section should be the focus, not the title. That means that a lot of the time, titles should actually be pretty small:

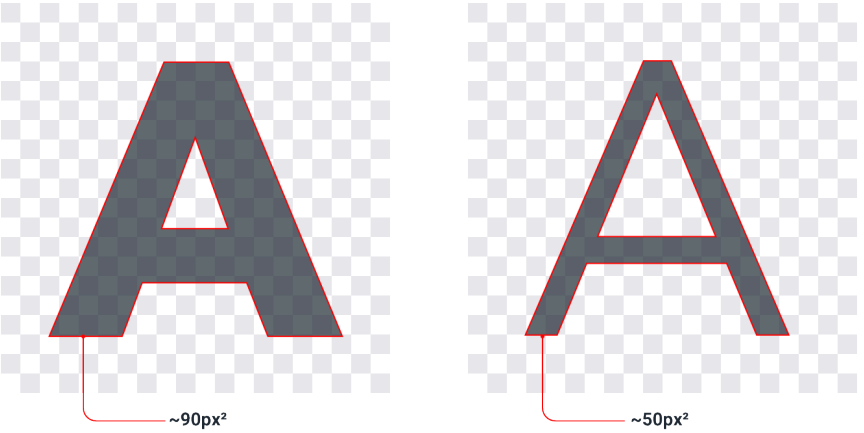


Taken to the extreme, you might even include section titles in your markup for accessibility reasons but *completely hide* them visually because the content speaks for itself.

Don't let the element you're using influence how you choose to style it — pick elements for semantic purposes and style them however you need to create the best visual hierarchy.

Balance weight and contrast

The reason bold text feels emphasized compared to regular text is that bold text covers more surface area — in the same amount of space, more pixels are used for text than for the background.

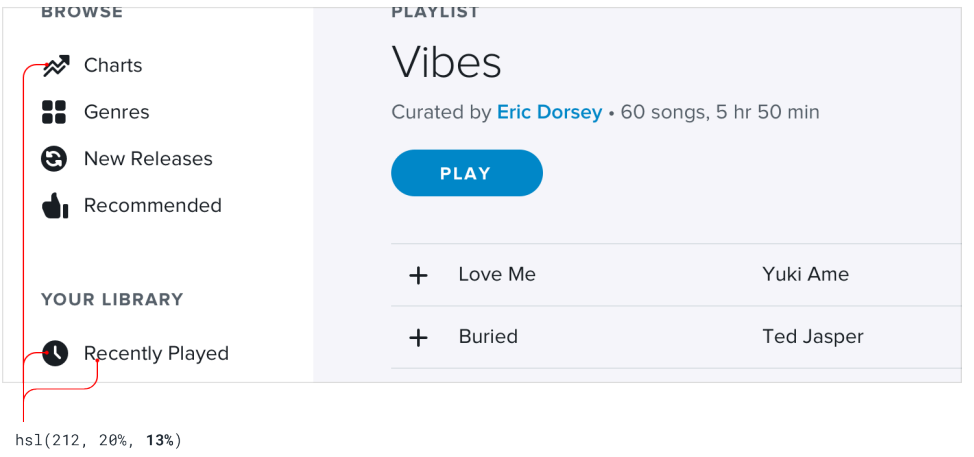


So why is this interesting? Well it turns out that the relationship between surface area and hierarchy has implications on other elements in a UI as well.

Using contrast to compensate for weight

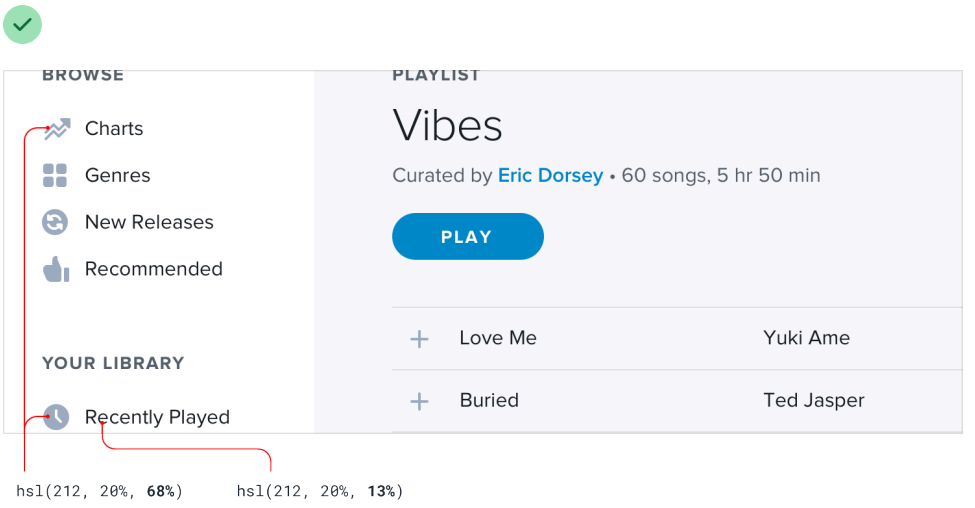
One of the places understanding this relationship becomes important is when working with icons.

Just like bold text, icons *(especially solid ones)* are generally pretty "heavy" and cover a lot of surface area. As a result, when you put an icon next to some text, the icon tends to feel emphasized.



Unlike text, there's no way to change the "weight" of an icon, so to create balance it needs to be de-emphasized in some other way.

A simple and effective way to do this is to lower the contrast of the icon by giving it a softer color.

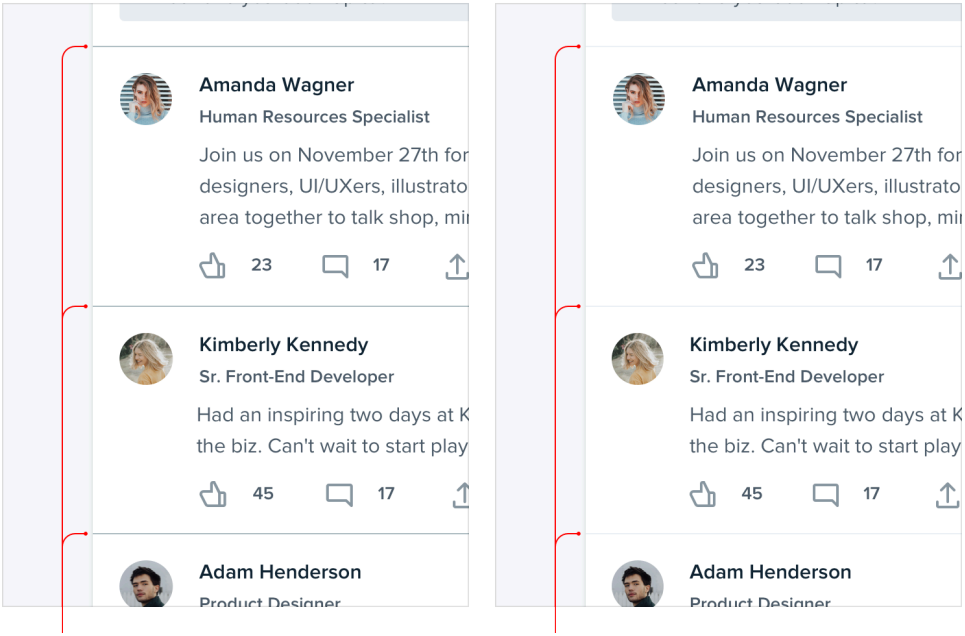


This works anywhere you need to balance elements that have different weights. Reducing the contrast works like a counterbalance, making heavier elements feel lighter even though the weight hasn't changed.

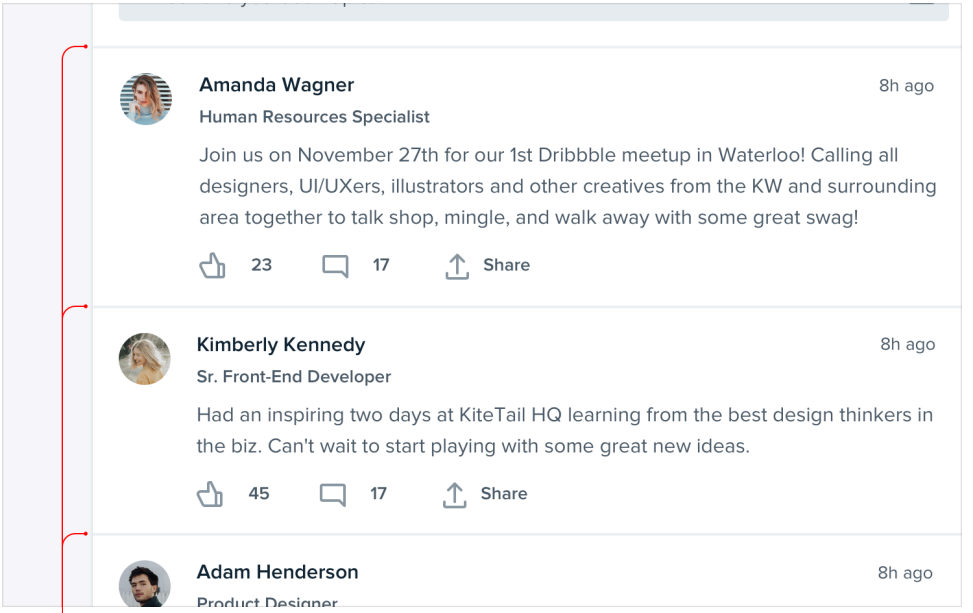
Using weight to compensate for contrast

Just like how reducing contrast helps to de-emphasize heavy elements, increasing weight is a great way to add a bit of emphasis to low contrast elements.

This is useful when things like thin 1px borders are too subtle using a soft color, but darkening the color makes the design feel harsh and noisy.



Making the border a bit heavier by increasing the width helps to emphasize it without losing the softer look:

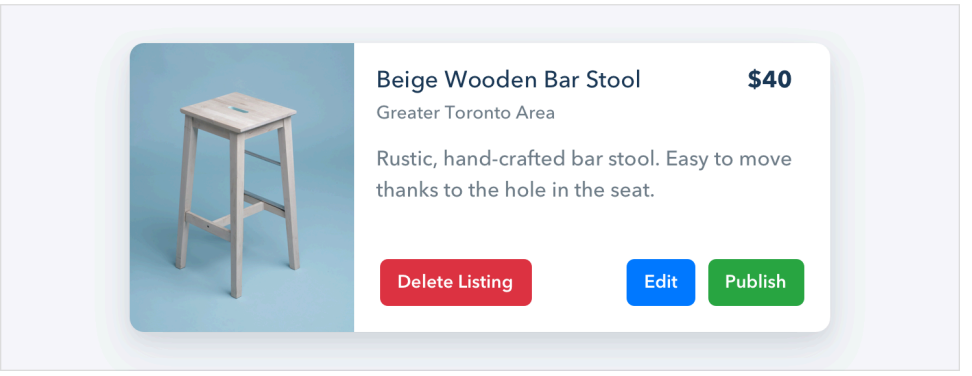


Balanced

```
border: 2px solid hsl(210, 23%, 95%);
```

Semantics are secondary

When there are multiple actions a user can take on a page, it's easy to fall into the trap of designing those actions based purely on semantics.



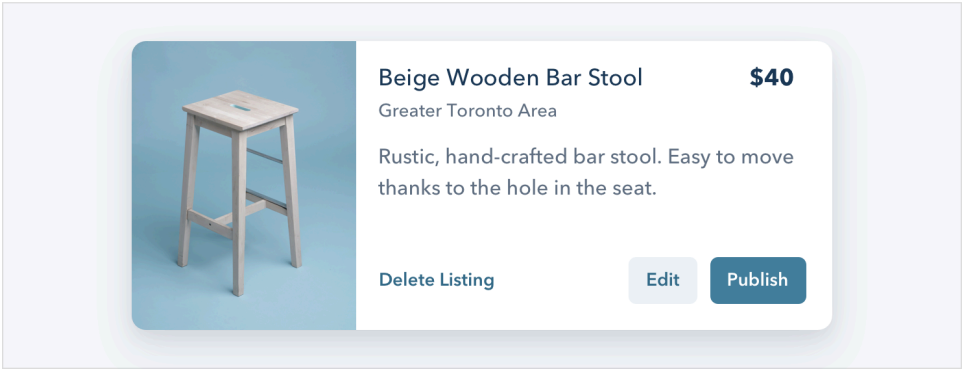
Semantics are an important part of button design, but that doesn't mean you can forget about hierarchy.

Every action on a page sits somewhere in a pyramid of importance. Most pages only have one true primary action, a couple of less important secondary actions, and a few seldom used tertiary actions.

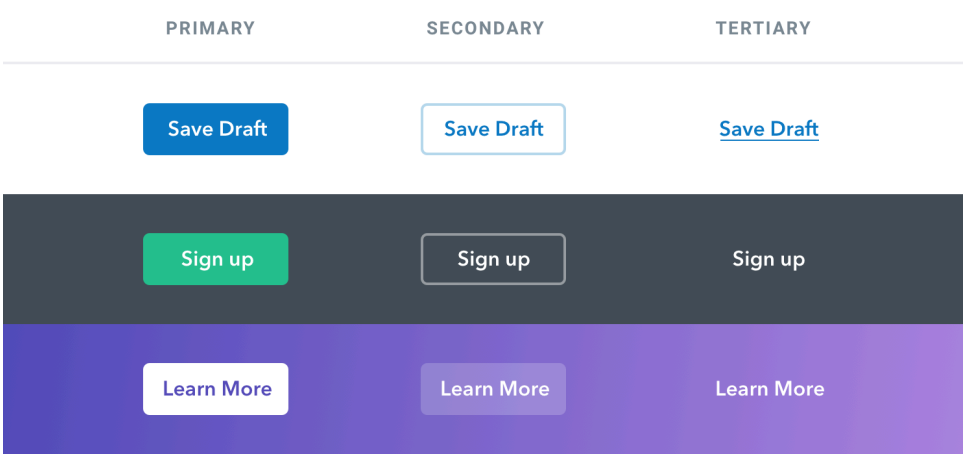
When designing these actions, it's important to communicate their place in the hierarchy.

- **Primary actions should be obvious.** Solid, high contrast background colors work great here.

- **Secondary actions should be clear but not prominent.** Outline styles or lower contrast background colors are great options.
- **Tertiary actions should be discoverable but unobtrusive.** Styling these actions like links is usually the best approach.



When you take a hierarchy-first approach to designing the actions on page, the result is a much less busy UI that communicates more clearly:



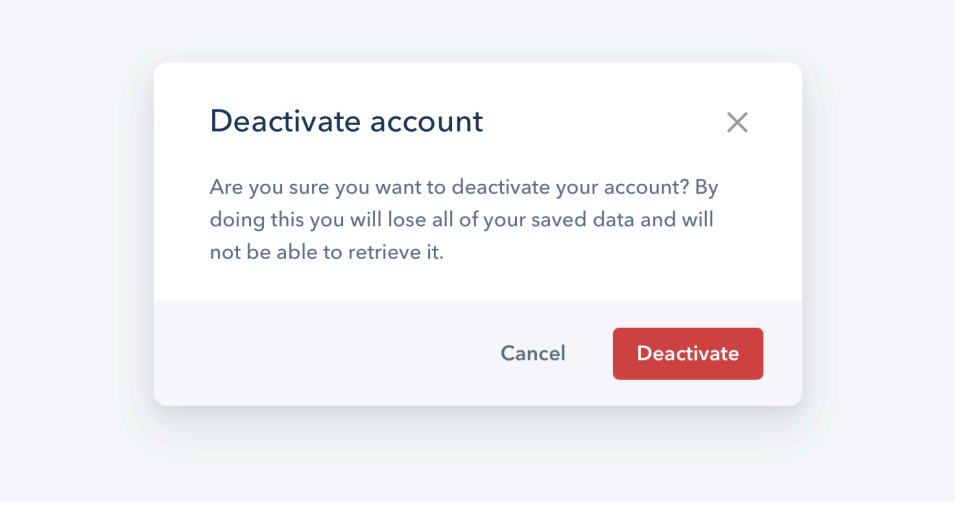
Destructive actions

Being destructive or high severity doesn't automatically mean a button should be big, red, and bold.

If a destructive action isn't the primary action on the page, it might be better to give it a secondary or tertiary button treatment.

PRIMARY	SECONDARY	TERTIARY
Unpublish	Unpublish	Unpublish

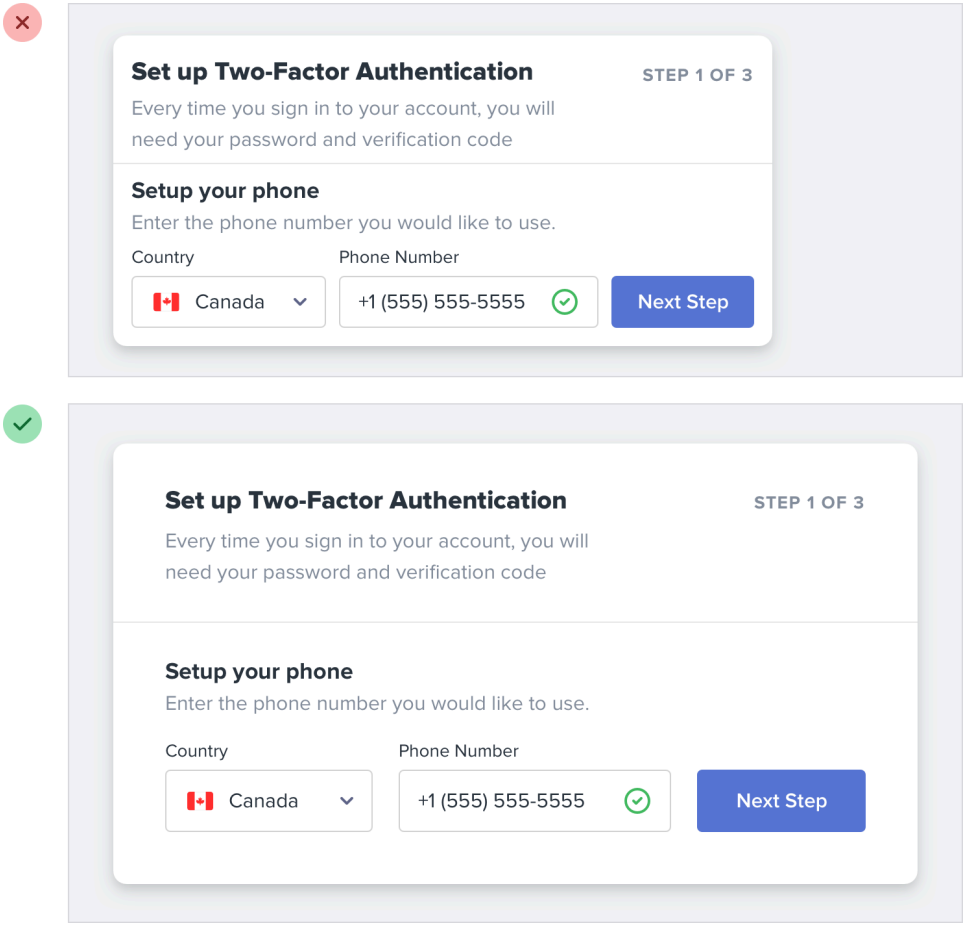
Combine this with a confirmation step where the destructive action actually is the primary action, and apply the big, red, bold styling there.



Layout and Spacing

Start with too much white space

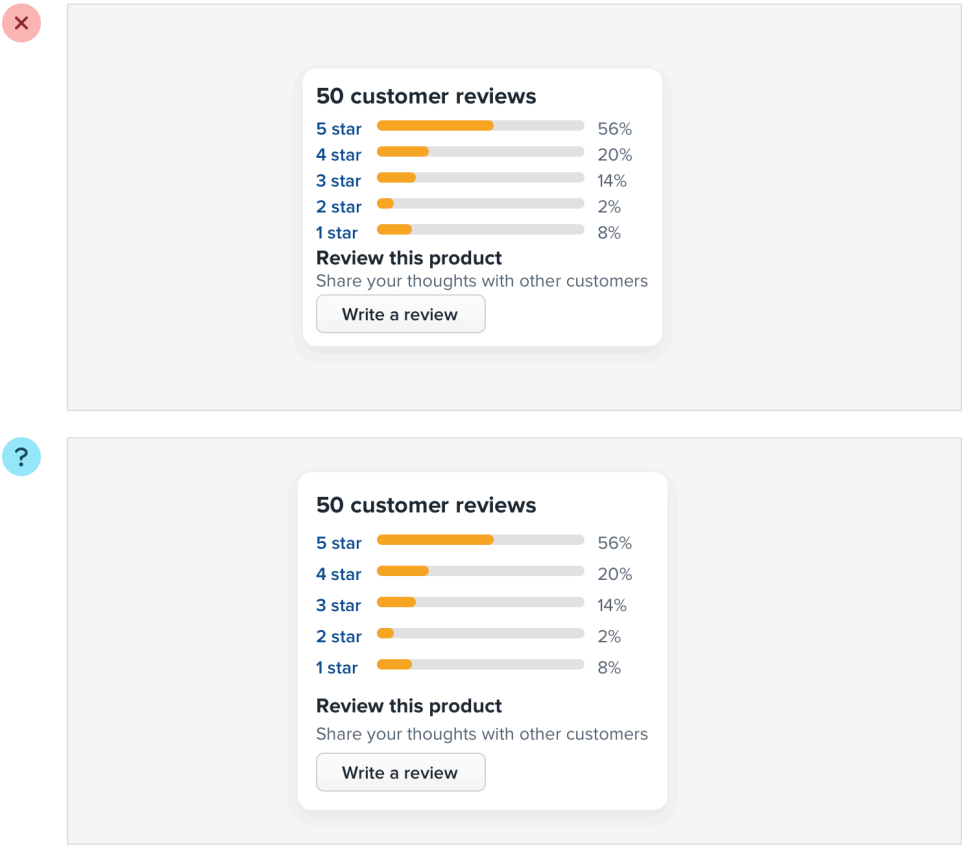
One of the easiest ways to clean up a design is to simply give every element a little more room to breathe.



Sounds simple enough, right? So how come we don't usually do it?

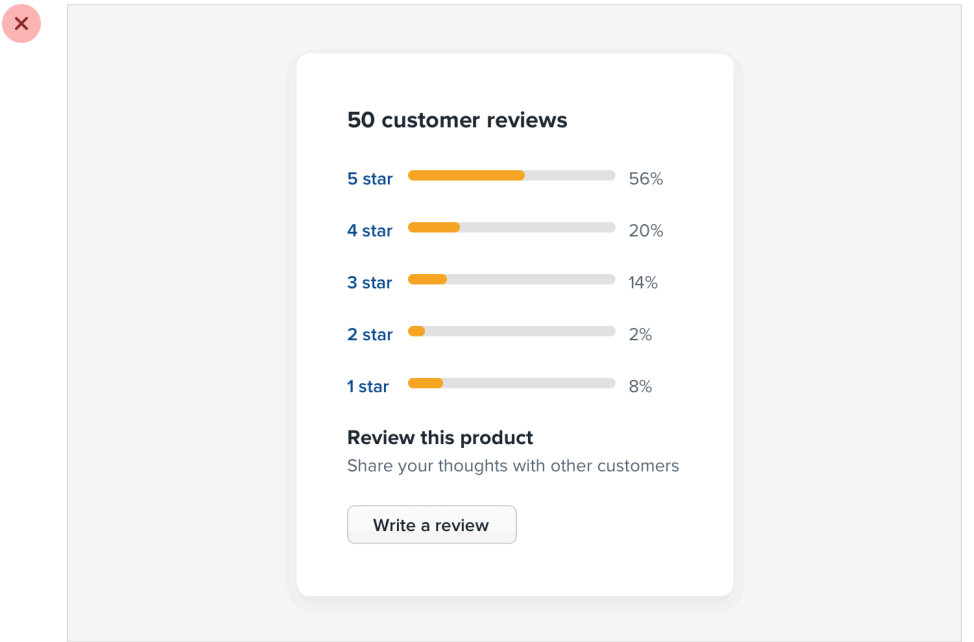
White space should be removed, not added

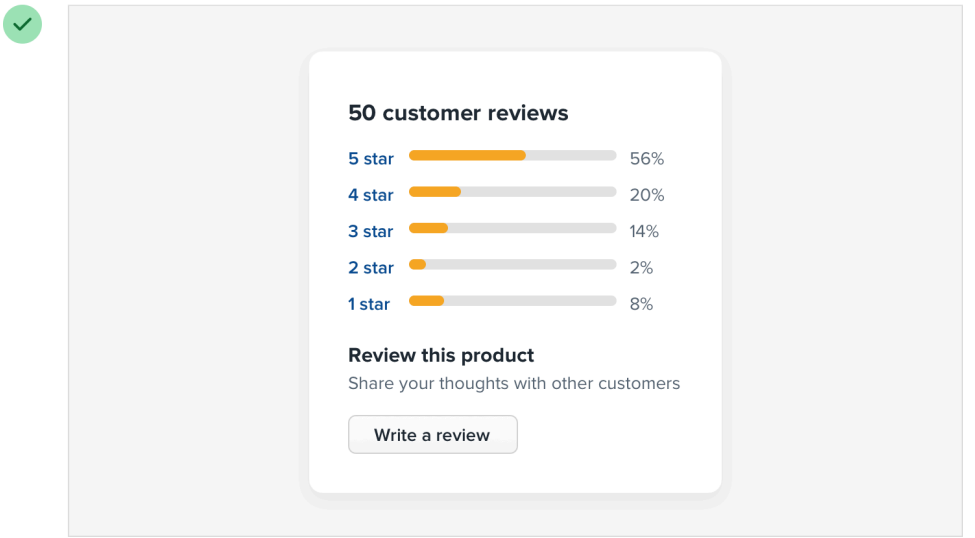
When designing for the web, white space is almost always *added* to a design — if something looks little too cramped, you add a bit of margin or padding until things look better.



The problem with this approach is that elements are only given the minimum amount of breathing room necessary to not look *actively bad*. To make something actually look *great*, you usually need more white space.

A better approach is to start by giving something *way too much* space, then remove it until it you're happy with the result.



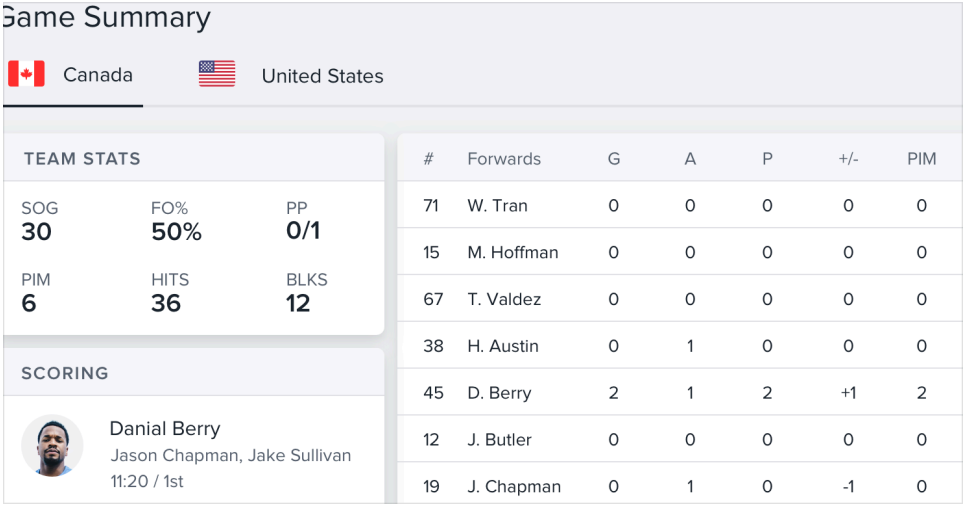


You might think you'd end up with too much white space this way, but in practice, what might seem like "a little too much" when focused on an individual element ends up being closer to "just enough" in the context of a complete UI.

Dense UIs have their place

While interfaces with a lot of breathing room almost always feel cleaner and simpler, there are certainly situations where it makes sense for a design to be much more compact.

For example, if you're designing some sort of dashboard where a lot of information needs to be visible at once, packing that information together so it all fits on one screen might be worth making the design feel more busy.

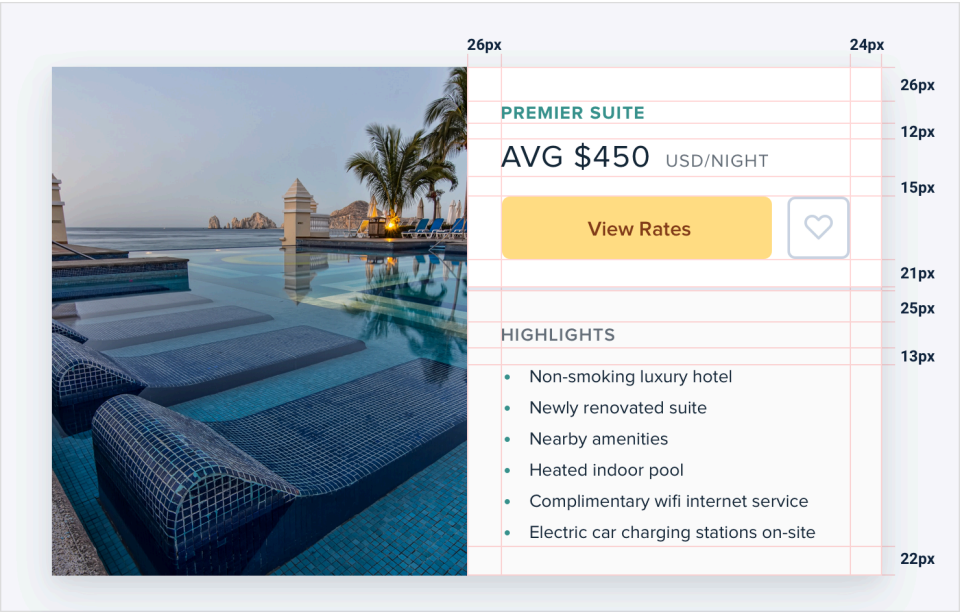


The important thing is to make this a deliberate decision instead of just being the default. It's a lot more obvious when you need to remove white space than it is when you need to add it.

Establish a spacing and sizing system

You shouldn't be nitpicking between 120px and 125px when trying to decide on the perfect size for an element in your UI.

Painfully trialing arbitrary values one pixel at a time will drastically slow you down at best, and create ugly, inconsistent designs at worst.



Instead, limit yourself to a constrained set of values, defined in advance.

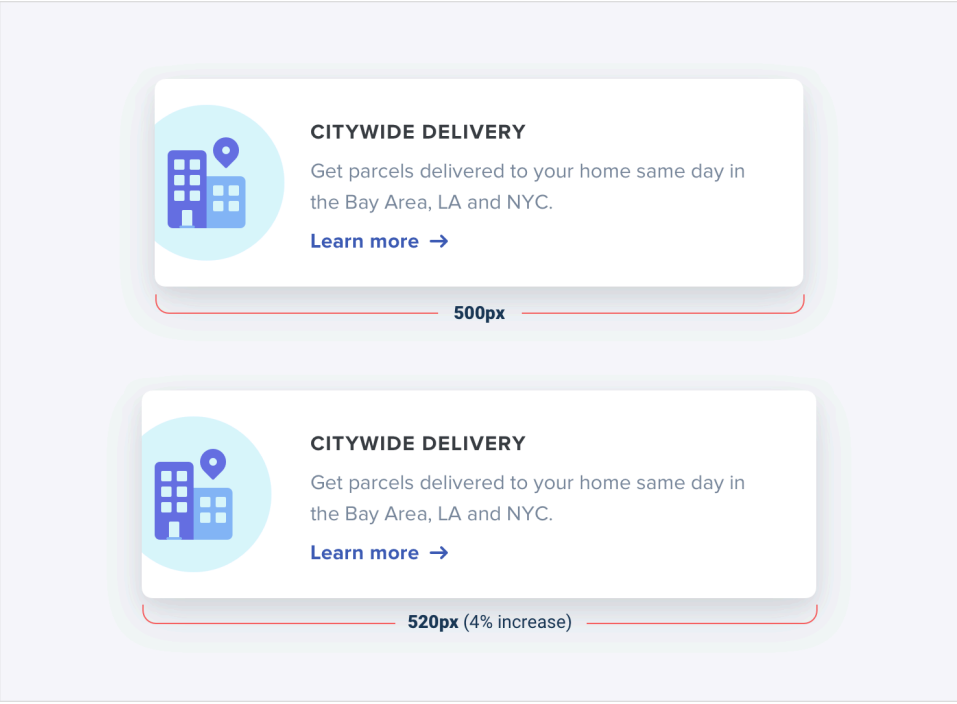
A linear scale won't work

Creating a spacing and sizing system isn't quite as simple as something like *"make sure everything is a multiple of 4px"* — a naive approach like that doesn't make it any easier to choose between 120px and 125px.

For a system to be truly useful, it needs to take into consideration the *relative* difference between adjacent values.

At the small end of the scale *(like the size of an icon, or the padding inside a button)*, a couple of pixels can make a big difference. Jumping from 12px to 16px is an increase of 33%!

But at the large end *(the width of a card, or the vertical spacing in a landing page hero)*, a couple of pixels is basically imperceivable. Even increasing the width of a card from 500px to 520px is only a difference of 4%, which is *eight times* less significant than the jump from 12px to 16px.



If you want your system to make sizing decisions easy, make sure no two values in your scale are ever closer than about 25%.

Defining the system

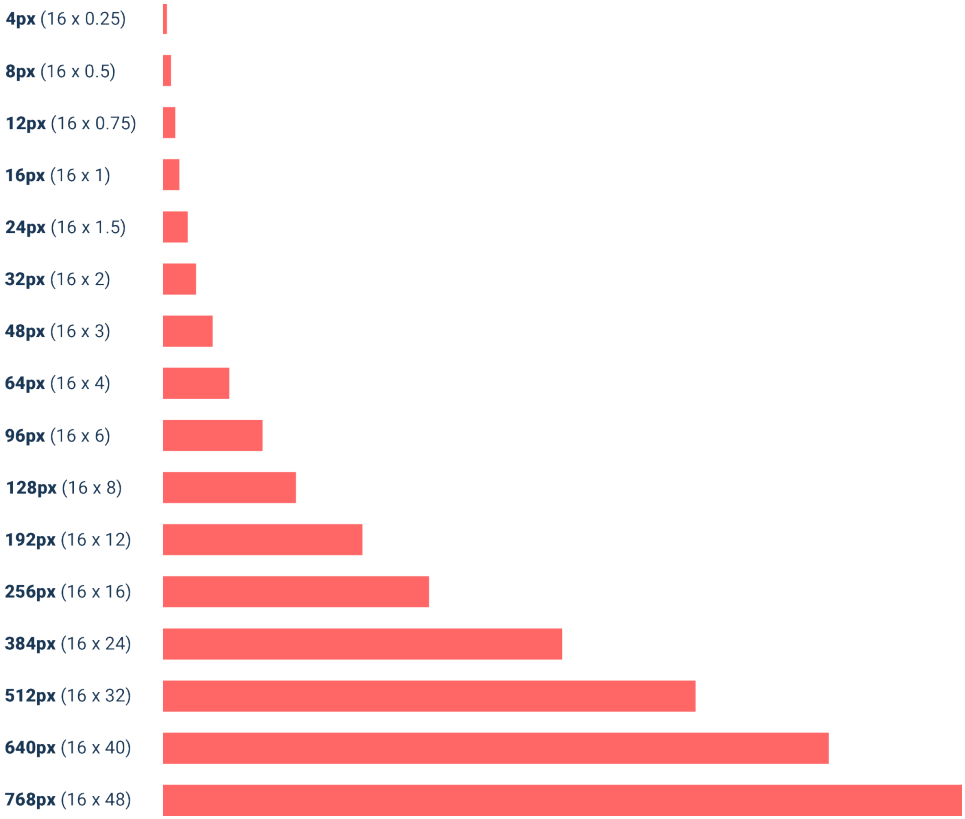
Just like you don't want to toil over arbitrary values when sizing an element or fine-tuning the space between elements, you don't want to build your spacing and sizing scale from arbitrary values either.

A simple approach is to start with a sensible *base* value, then build a scale using factors and multiples of that value.

16px is a great number to start with because it divides nicely, and also happens to be the default font size in every major web browser.

The values at the small end of the scale should start pretty packed together, and get progressively more spaced apart as you get further up the scale.

Here's an example of a fairly practical scale built using this approach:

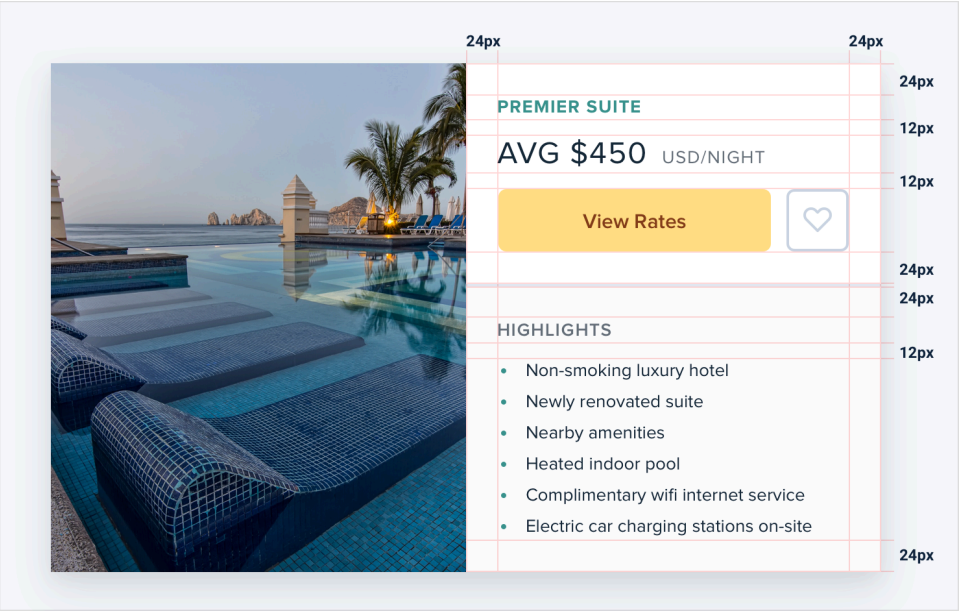


Using the system

Once you've defined your spacing and sizing system, you'll find that you're able to design a hell of a lot faster, especially if you design in the browser *(sticking to a system is easier when you're typing in numbers than when you're dragging with the mouse.)*

Need to add some space under an element? Grab a value from your scale and try it out. Not quite enough? The next value is probably perfect.

While the workflow improvements are probably the biggest benefit, you'll also start to notice a subtle consistency in your designs that wasn't there before, and things will look just a little bit cleaner.

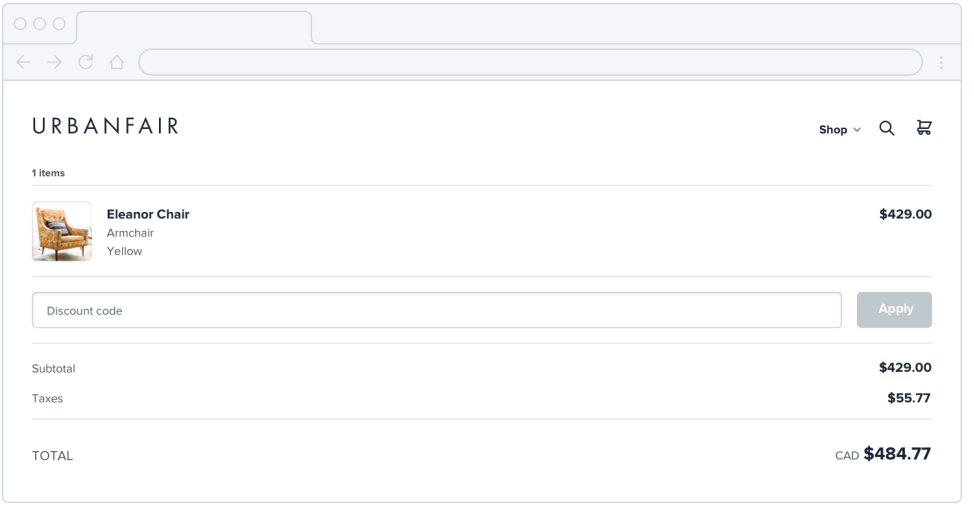


A spacing and sizing system will help you create better designs, with less effort, in less time. Design advice doesn't get much more valuable than that.

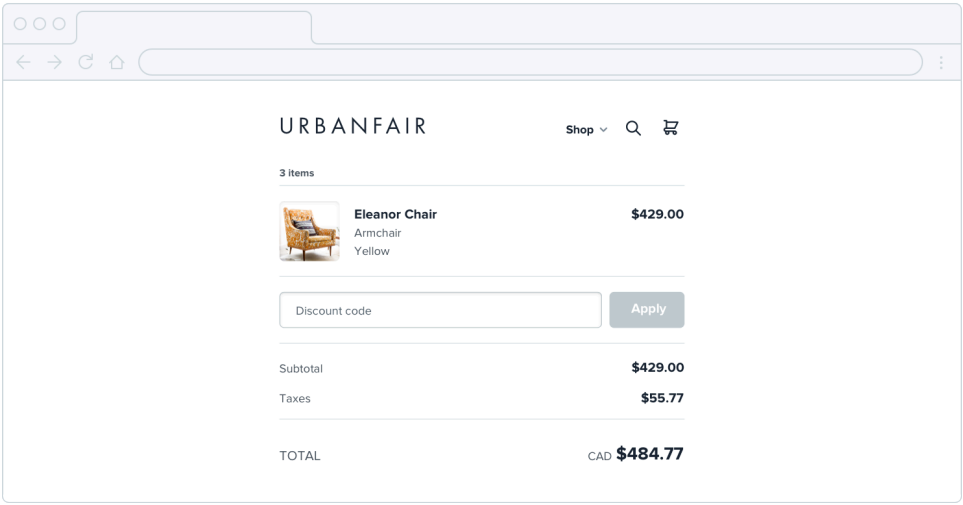
You don’t have to fill the whole screen

Remember when 960px was the de facto layout width for desktop-size designs? These days you’d be hard-pressed to find a *phone* with a resolution that low.

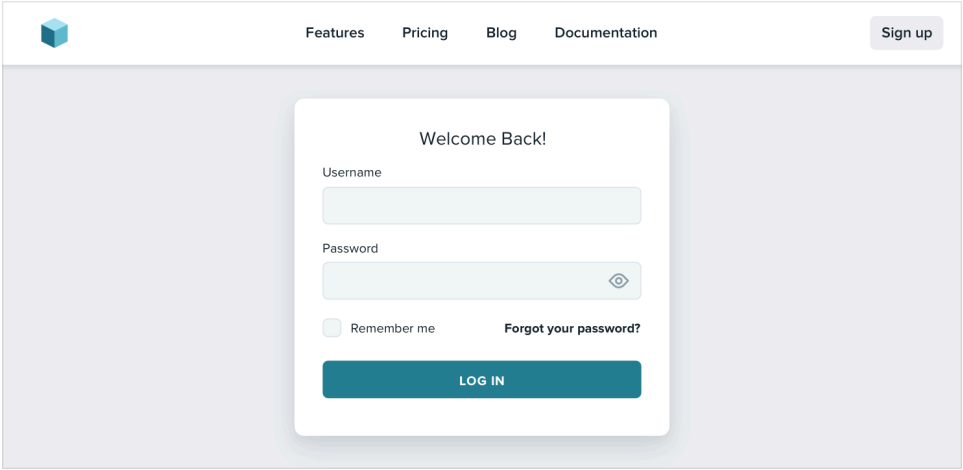
So it’s no surprise that when most of us open our design tool of choice on our high resolution displays, we give ourselves *at least* 1200-1400px of space to fill. But just because you have the space, doesn’t mean you need to use it.



If you only need 600px, use 600px. Spreading things out or making things unnecessarily wide just makes an interface harder to interpret, while a little extra space around the edges never hurt anyone.



This is just as applicable to individual sections of an interface, too. You don’t need to make *everything* full-width just because something else (like your navigation) is full-width.

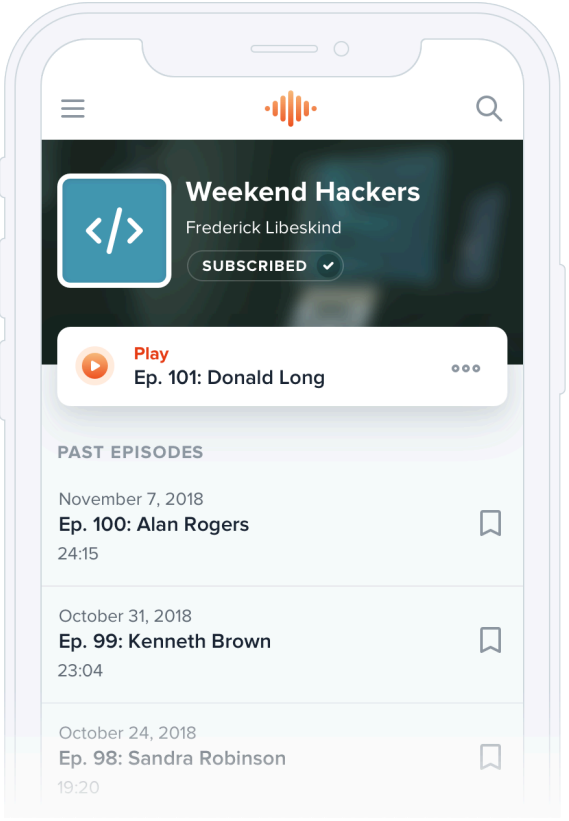


Give each element just the space it needs — don’t make something worse just to make it match something else.

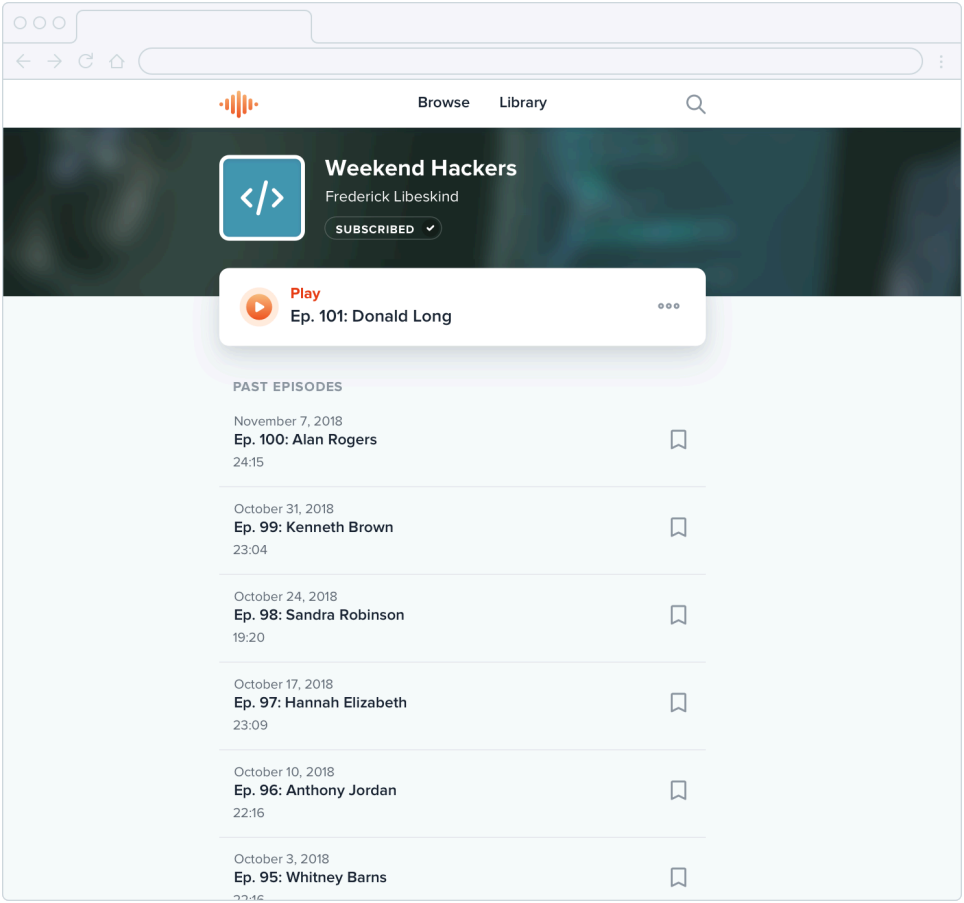
Shrink the canvas

If you’re having a hard time designing a small interface on a large canvas, shrink the canvas! A lot of the time it’s easier to design something small when the constraints are real.

If you’re building a responsive web application, try starting with a ~400px canvas and designing the mobile layout first.



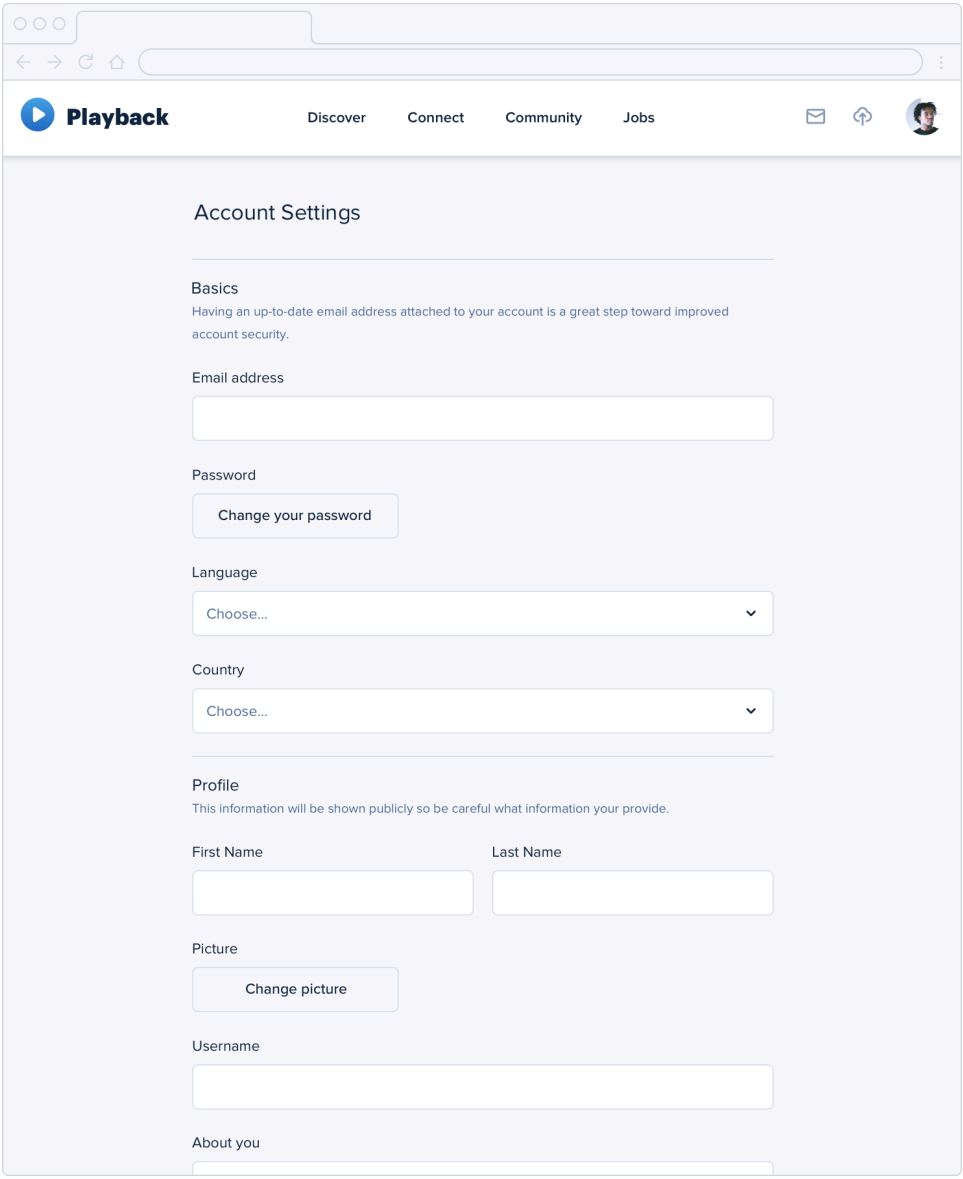
Once you have a mobile design you’re happy with, bring it over to a larger size screen and adjust anything that felt like a compromise on smaller screens. Odds are you won’t have to change as much as you think.



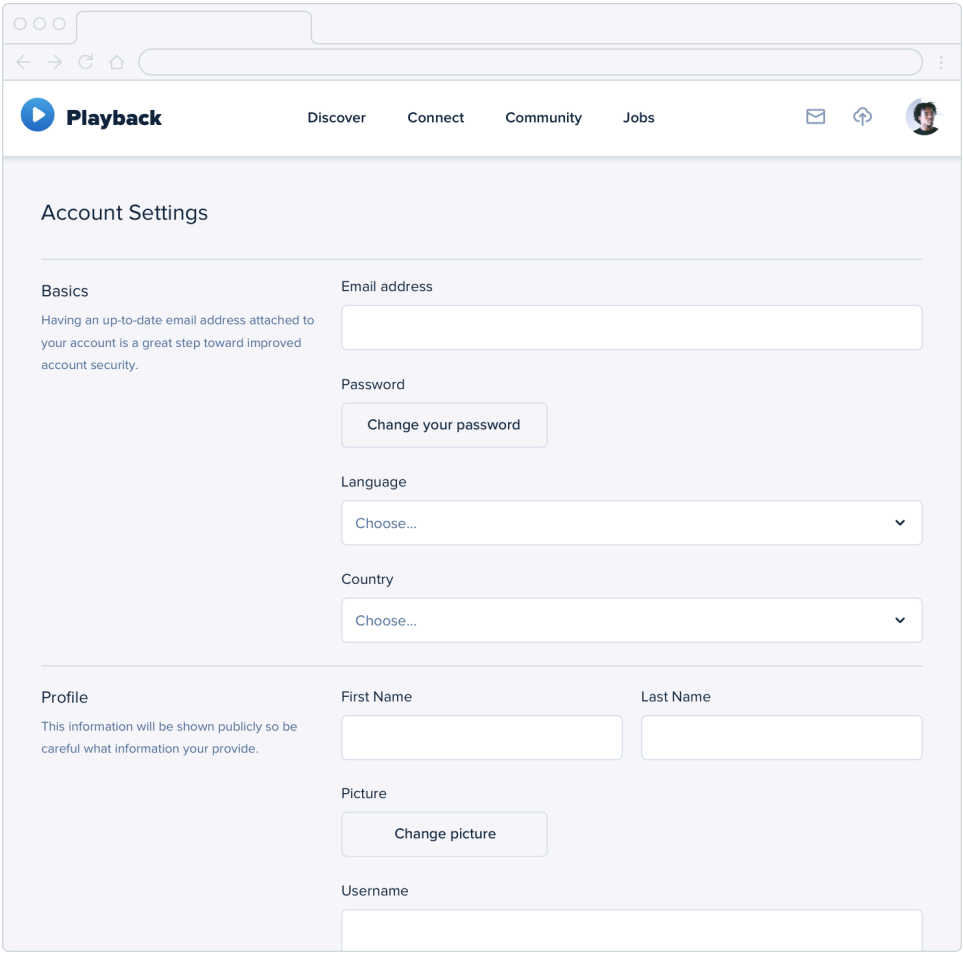
Thinking in columns

If you’re designing something that works best at a narrower width but feels unbalanced in the context of an otherwise wide UI, see if you can split it into columns instead of just making it wider.

For example, take this narrow form layout:



If you wanted to make better use of the available space without making the form harder to use, you could break the supporting text out into a separate column:



This makes the design feel more balanced and consistent without compromising on the optimal width for the form itself.

Don’t force it

Just like you shouldn’t worry about filling the whole screen, you shouldn’t try to cram everything into a small area unnecessarily either.

If you need a lot of space, go for it! Just don’t feel obligated to fill it if you don’t have to.

Grids are overrated

Using a system like a 12-column grid is a great way to simplify layout decisions, and can bring a satisfying sense of order to your designs.

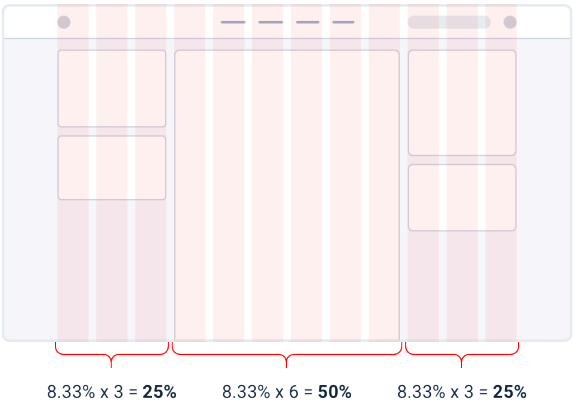
But even though grids can be useful, outsourcing *all* of your layout decisions to a grid can do more harm than good.

Not all elements should be fluid

Fundamentally, a grid system is just about giving elements fluid, percentage-based widths, where you're choosing from a constrained set of percentages.

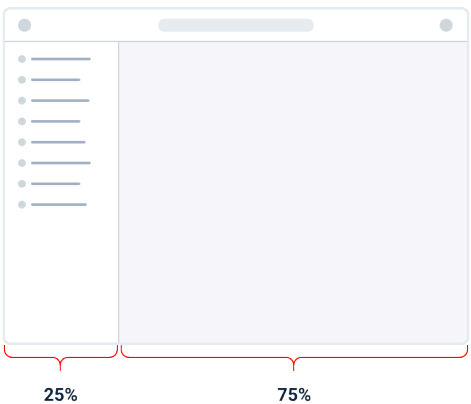
For example, in a 12-column grid each column is 8.33% wide. As long as an

element's width is some multiple of 8.33% *(including any gutters)*, that element is "on the grid".



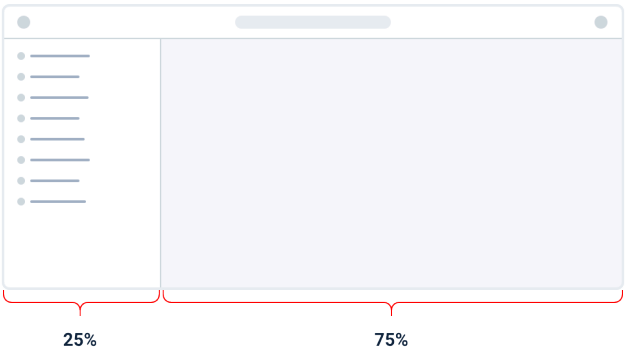
The problem with treating grid systems like a religion is that there are a lot of situations where it makes much more sense for an element to have a *fixed* width instead of a relative width.

For example, consider a traditional sidebar layout. Using a 12-column grid system, you might give the sidebar a width of three columns (25%) and the main content area a width of nine columns (75%).

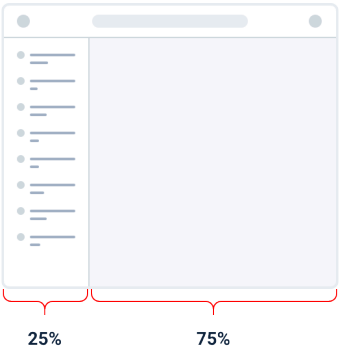


This might seem fine at first, but think about what happens when you resize the screen.

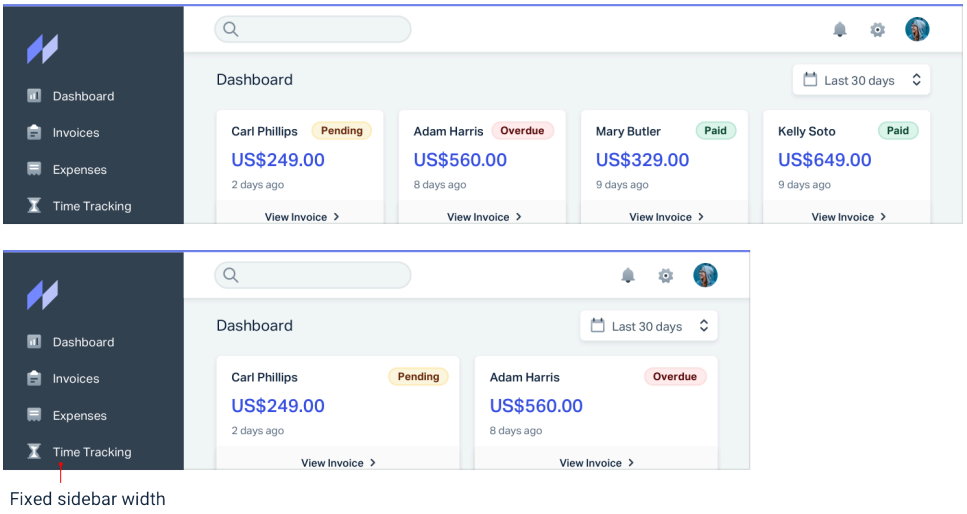
If you make the screen wider the sidebar gets wider too, taking up space that could've been put to better use by the main content area.



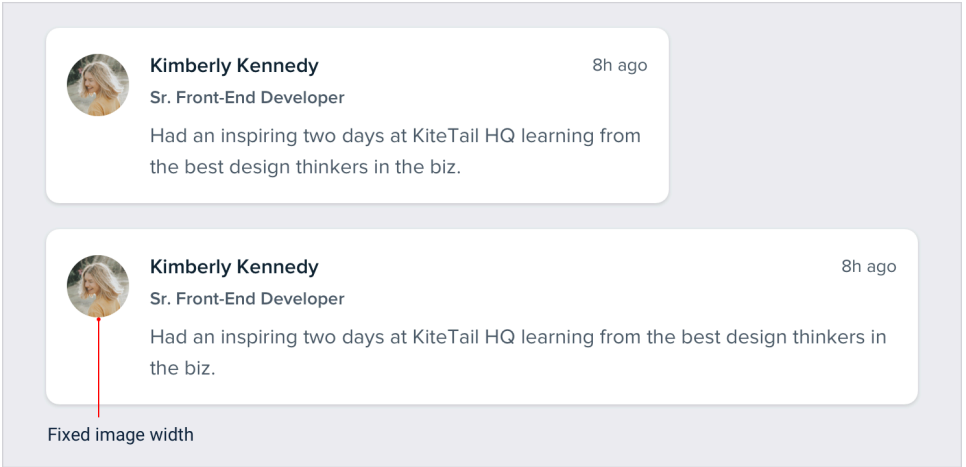
Similarly, if you make the screen narrower, the sidebar can shrink below its minimum reasonable width, causing awkward text wrapping or truncation.



In this situation, it makes much more sense to give the sidebar a fixed width that's optimized for its contents. The main content area can then flex to fill the remaining space, using its own *internal* grid to lay out its children.

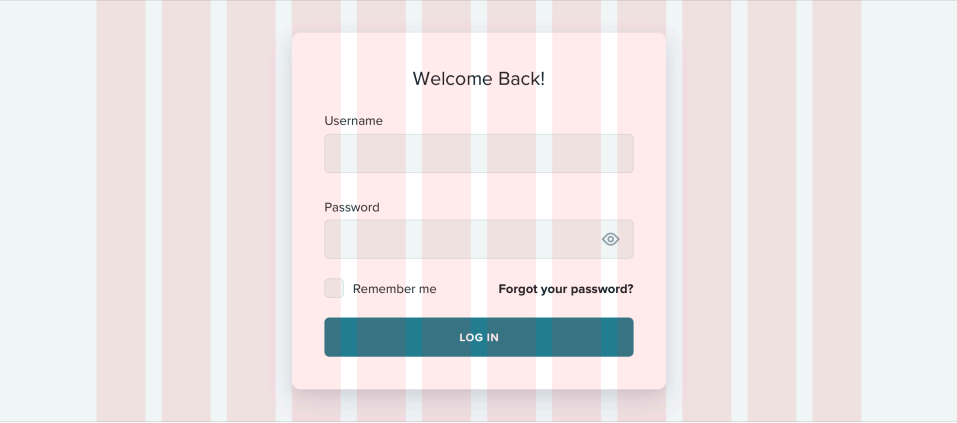


This applies within components, too — don't use percentages to size something unless you actually want it to scale.

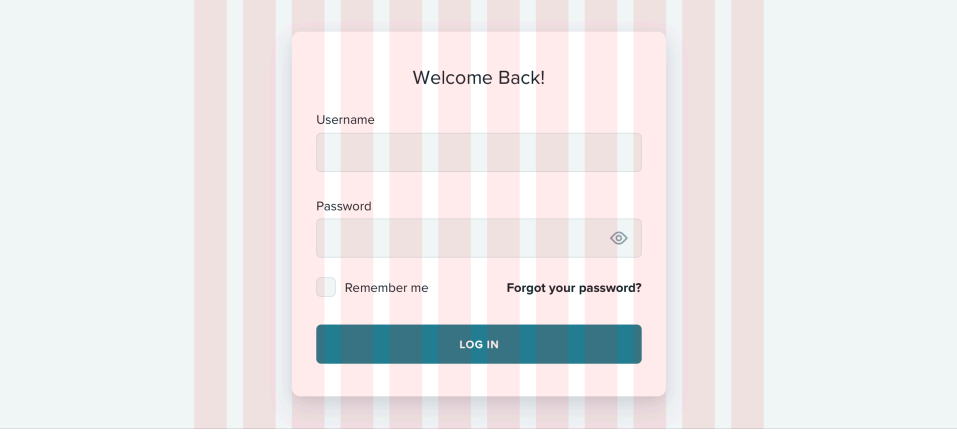


Don’t shrink an element until you need to

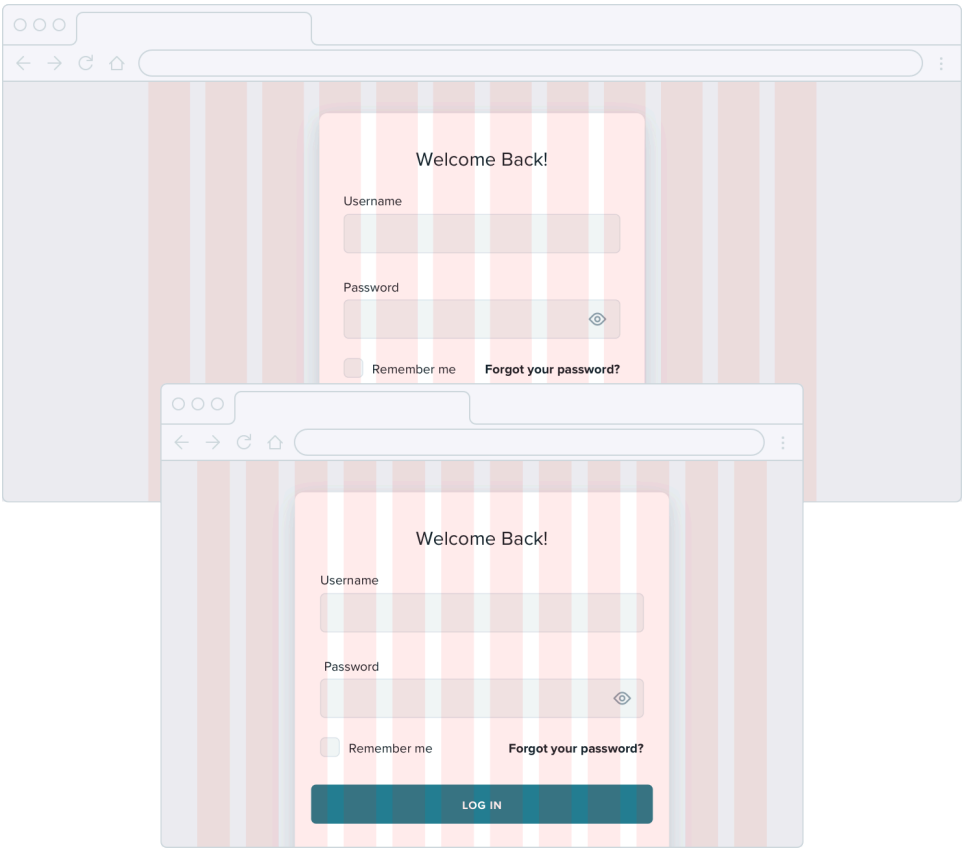
Say you’re designing a login card. Using the full screen width would look ugly, so you give it a width of 6 columns (50%) with a 3-column offset on each side.



On medium-sized screens you realize the card is a little narrow even though you have the space to make it bigger, so at that screen size you switch it to a width of 8 columns, with two empty columns on each side.



The silly thing about this approach is that because column widths are fluid, there's a range in screen sizes where the login card is *wider* on medium screens than it is on large screens:



If you know that say 500px is the optimal size for the card, why should it ever get smaller than that if you have the space for it?

Instead of sizing elements like this based on a grid, give them a max-width so they don't get too large, and only force them to shrink when the screen gets smaller than that max-width.