User interface patterns

- Navigation
- Carousels
- Data tables
- Guidelines

A design viewed on a small screen shouldn't look like a shrunk-down version of a large-screen layout. Likewise, a design viewed on a large screen shouldn't look like a blown-up version of a small-screen layout. Instead, the design needs to be flexible enough to adapt to all screen sizes. A successful responsive design makes the most of every form factor.

This means that some interface elements may need to look quite different depending on the context they're viewed in. You might need to apply very different CSS to the same HTML codebase to make the most of different screen sizes. That can be quite a design challenge!

Here are some common challenges that you might face.

Navigation

Displaying a list of navigation links is quite straightforward on a large screen. There's plenty of space to accommodate those links.

On a small screen, space is at a premium. When you're designing for this situation, it's tempting to hide navigation behind a button. The problem with this solution is that users have to then take two steps to get anywhere: open the menu, then choose an option. Until the menu is opened, the user is left wondering "where can I go?”

Try to find a strategy that avoids hiding your navigation. If you have a relatively small number of items, you can style the navigation to look good on small screens.

The same website with five navigation links viewed in a mobile browser and viewed in a tablet browser. The navigation is visible on both devices.

That pattern won't scale if your navigation has lots of links. The navigation will look cluttered if the links wrap onto two or three lines on a small screen.

One possible solution is to keep links on one line but truncate the list at the edge of the screen. Users can swipe horizontally to reveal the links that aren't immediately visible. This is the overflow pattern.

The advantage of this technique is that it scales to any device width and any number of links. The disadvantage is that users might miss links that aren't initially visible. If you use this technique for your main navigation, make sure the first few links are the most important ones and visually indicate that there are more items in the list. The previous example uses a gradient for this indicator.

As a last resort you could choose to have your navigation hidden by default and provide a toggle mechanism for users to show and hide the contents. This is called progressive disclosure.

The same website with five navigation links viewed in a mobile browser and viewed in a tablet browser. The navigation is visible on the tablet, but hidden on the mobile device.

Make sure the button that toggles the display of the navigation is labeled. Don't rely on an icon to be understood.

Three unlabelled icons: the first is three horizontal lines; the second is three by three grid; the third is three circles arranged vertically.

An unlabelled icon is "mystery meat" navigation—users won't know what's in there until they bite into it. Provide a text label to let users know what the button will reveal.

Carousels

What's true of navigation is also true of other content: try not to hide anything anyway. A carousel is a common method of hiding content away. It can look quite neat, but there's a good chance that your users will never discover the hidden content. Carousels are better at solving organizational problems—like what content should feature on the homepage—than at serving users.

That said, when space is at a premium, carousels can prevent a page getting too long and cluttered. You could employ a hybrid approach: show content in a carousel for small screens, but display the same content in a grid for larger screens.

For narrow screens, display items in a row using flexbox. The row of items will extend beyond the edge of the screen. Use `overflow-x: auto` to allow horizontal swiping.

```
@media (max-width: 50em) {
  .cards {
    display: flex;
    flex-direction: row;
```

```
    overflow-x: auto;
    scroll-snap-type: inline mandatory;
    scroll-behavior: smooth;
  }
  .cards .card {
    flex-shrink: 0;
    flex-basis: 15em;
    scroll-snap-align: start;
  }
}
```

The logical version of `overflow-x` is `overflow-inline` which would better match with the value for `scroll-snap-type`. This example uses the physical version for better cross-browser support.

The `scroll-snap` properties ensure that the items can be swiped in a way that feels smooth. Thanks to `scroll-snap-type: inline mandatory`, the items snap into place.

When the screen is large enough—wider than `50em` in this case—switch over to grid and display the items in rows and columns, without hiding anything.

```
@media (min-width: 50em) {
  .cards {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(15em, 1fr));
  }
}
```

https://codepen.io/web-dot-dev/pen/XWeNPzY

Crucially, an item in the carousel view doesn't take up the full width. If it did, there would be no indication that more content is available beyond the edge of the viewport.

Carousels are another example of the overflow pattern in action. If you have many items that people can browse through, you could keep using the overflow pattern even on large screens, including televisions. This media scroller uses multiple carousels to manage a significant amount of options.

Again, the `scroll-snap` properties ensure that the interaction feels smooth. Also, notice that the images in the carousel have `loading="lazy"` applied to them. In this case, the images aren't below the fold, they're beyond the edge, but the same principle applies: if the user never swipes as far as that item, the image won't be downloaded, saving on bandwidth.

https://codepen.io/argyleink/pen/bGgyOGP

The previous example shows a more complete demo, you can read more about it in the article building a media scroller component.

With the addition of JavaScript, you can add interactive controls to a carousel. You could even make it automatically cycle through items. But think long and hard before doing that—autoplaying might work if the carousel is the only content on the page, but an autoplaying carousel is incredibly annoying if someone is trying to interact with other content (like reading text, for example). You can read more carousel best practices.

Data tables

The `table` element is perfect for structuring tabular data; rows and columns of related information. But if the table gets too large, it could break your small-screen layout.

You can apply the overflow pattern to tables. In this example, the table is wrapped in a `div` with a class of `table-container`.

```
.table-container {
  max-inline-size: 100%;
  overflow-x: auto;
  scroll-snap-type: inline mandatory;
  scroll-behavior: smooth;
}
.table-container th,
.table-container td {
  scroll-snap-align: start;
  padding: var(--metric-box-spacing);
}
```

https://codepen.io/web-dot-dev/pen/MWEbqQQ

Guidelines

The overflow pattern is a good compromise for small screens but make sure it's clear that the off-screen content is reachable. Consider placing a shadow or a gradient over the edge where content is truncated.

Progressive disclosure is a useful way to save space, but be careful about using it for very important content. It's better suited to secondary actions. Ensure the interface element that

triggers the disclosure is clearly labeled—don't rely on iconography alone.

Design for smaller screens first. It's easier to adapt small-screen designs to larger screens than the other way around. If you design for a large screen first, there's a danger that your small-screen design will feel like an afterthought.

For more layout and UI element patterns, explore the web.dev Patterns section.

When you're adapting interface elements to different screen sizes, media queries are very useful for figuring out the dimensions of the device. But media features like `min-width` and `min-height` are just the beginning. Next, you'll discover a whole host of other media features.

□□□ #3
□ □□□ □□□ 2026-10-31 22:49:09 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:16 UTC