rwd
Responsive Web Design

Layout

Rearranging content into different layouts may be the first thing you think of when you picture responsive design, and with good reason. The layout helps form our first impression of a site’s content and usability.

Responsive design is based on fluid layouts that expand and contract to fill the available space in the viewport. One **fluid layout** is usually not enough, however, to serve all screen sizes. More often, two or three layouts are produced to meet requirements across devices, with small adjustments between layout shifts.

Designers typically start with a one-column layout that fits well on small handheld devices and rearrange elements into columns as more space is available. They may also have the design for the widest screens worked out early on so there is an end-point in mind. The design process may involve a certain amount of switching between views and making decisions about what happens along the way.

Responsive layout patterns

The manner in which a site transitions from a small-screen layout to a wide-screen layout must make sense for that particular site, but there are a few patterns (common and repeated approaches) that have emerged over the years. We can thank Luke Wroblewski (known for his "Mobile First" approach to web design, which has become the standard) for doing a survey of how responsive sites handle layout. Following are the top patterns Luke named in his article:

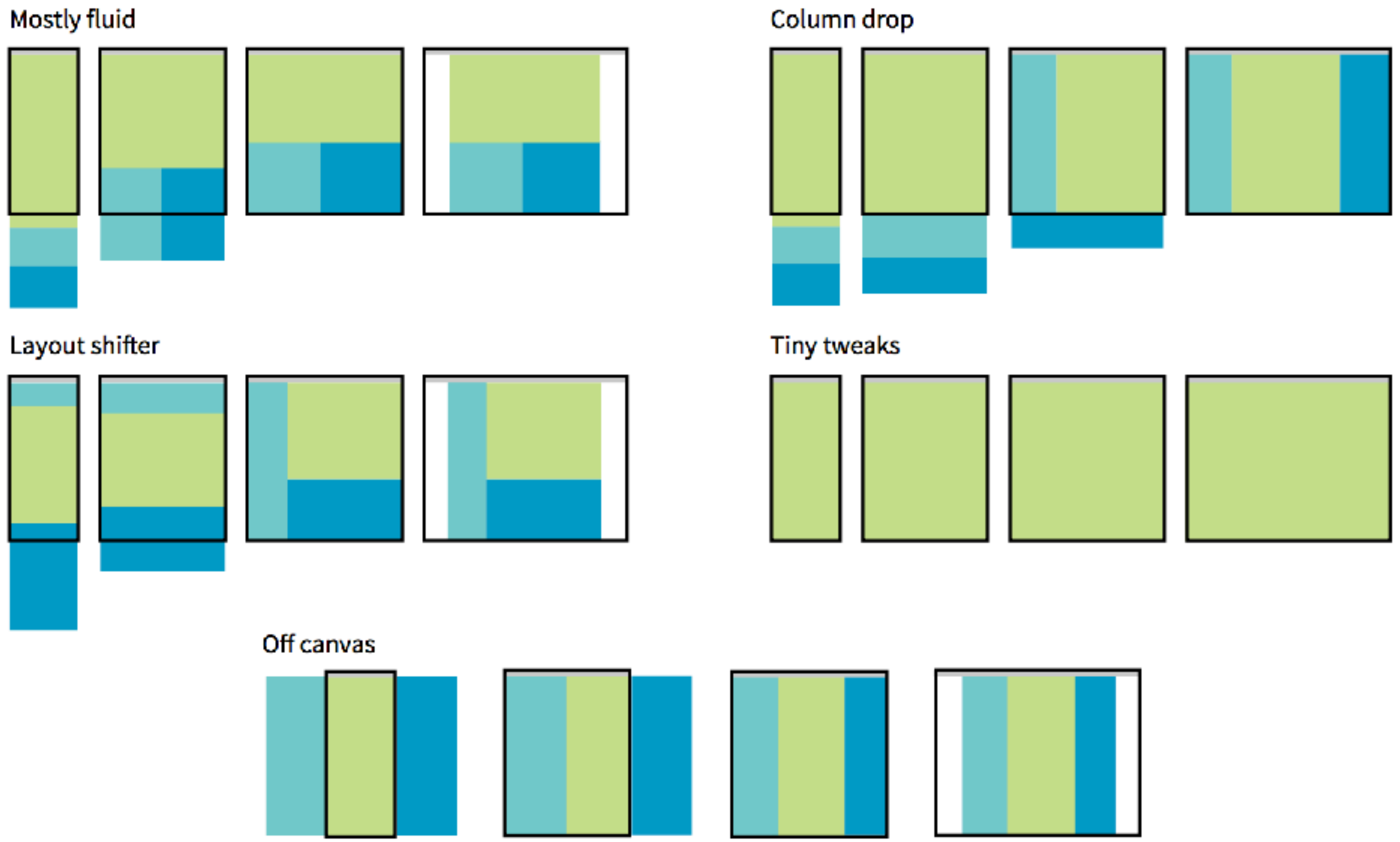


FIGURE 17-9. Examples of the responsive layout patterns identified by Luke Wroblewski.

Mostly fluid

This pattern uses a single-column layout for small screens, and another fluid layout that covers medium and large screens, with a maximum width set to prevent it from becoming too wide. It generally requires less work than other solutions.

Column drop

This solution shifts between one-, two-, and three-column layouts based on available space. When there isn't room for extra columns, the sidebar columns drop below the other columns until everything is stacked verti- cally in the one-column view.

Layout shifter

If you want to get really fancy, you can completely reinvent the layout for a variety of screen sizes. Although expressive and potentially cool, it is not necessary. In general, you can solve the problem of fitting your content to multiple environments without going overboard.

Tiny tweaks

Some sites use a single-column layout and make tweaks to type, spacing, and images to make it work across a range of device sizes.

Off canvas

As an alternative to stacking content vertically on small screens, you may choose to use an “off-canvas” solution. In this pattern, a page component is located just out of sight on the left or right of the screen and flies into view when requested. A bit of the main content screen remains visible on the edge to orient users as to the relationship of moving parts. This was made popular by Facebook, wherein Favorites and Settings were placed on a panel that slid in from the left when users clicked a menu icon

Navigation

Navigation feels a little like the Holy Grail of Responsive Web Design. It is critical to get it right. Because navigation at desktop widths has pretty much been conquered, the real challenges come in re-creating our navigation options on small screens. A number of successful patterns have emerged for small screens, which I will briefly summarize here

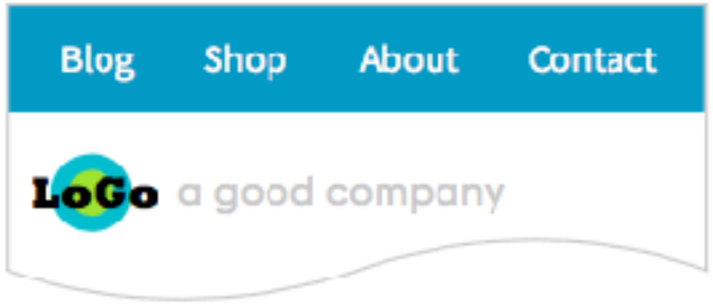
Top navigation

If your site has just a few navigation links, they may fit just fine in one or two rows at the top of the screen.

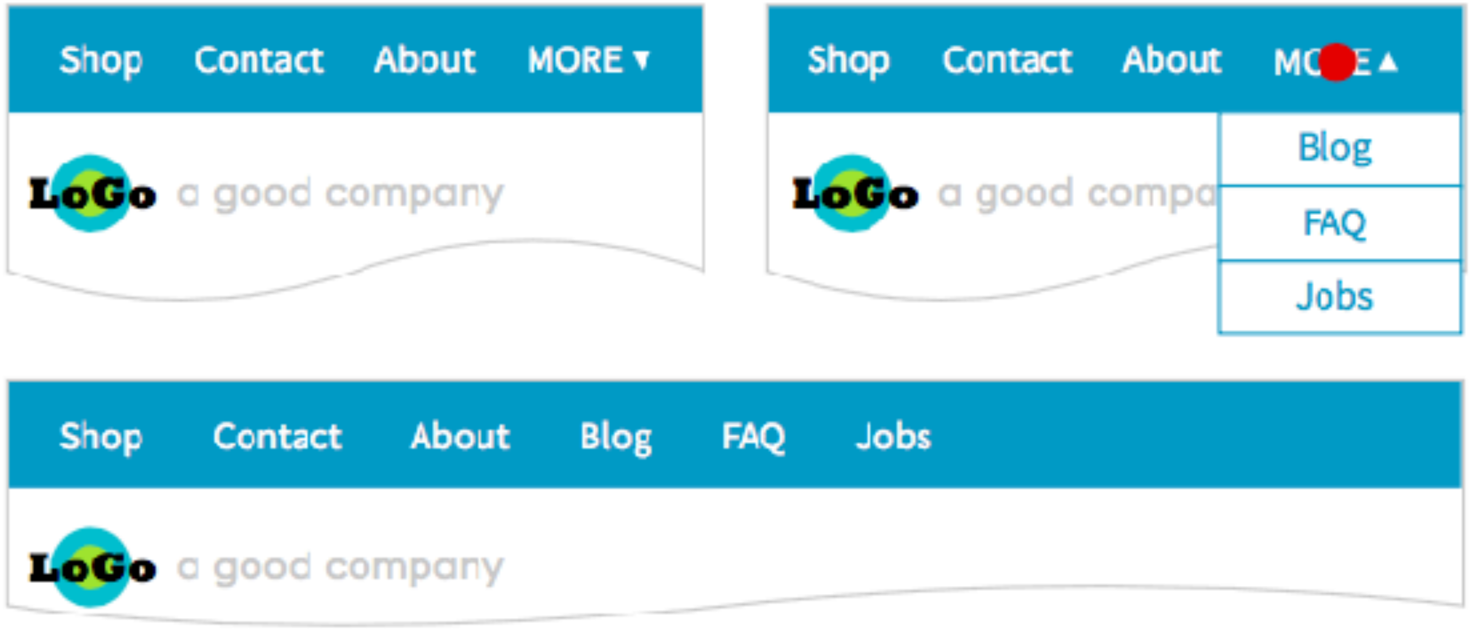
Priority +

In this pattern, the most important navigation links appear in a line across the top of the screen alongside a More link that exposes additional options. The pros are that the primary links are in plain view, and the number of links shown can increase as the device width increases. The cons include the difficulty of determining which links are worthy of the prime small-screen real estate.

Top navigation



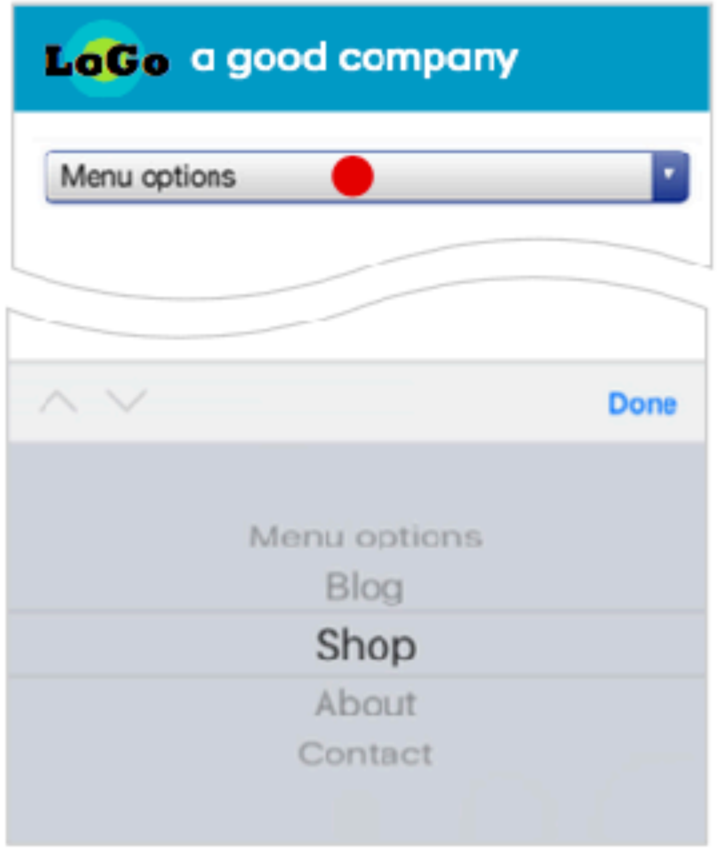
Priority +



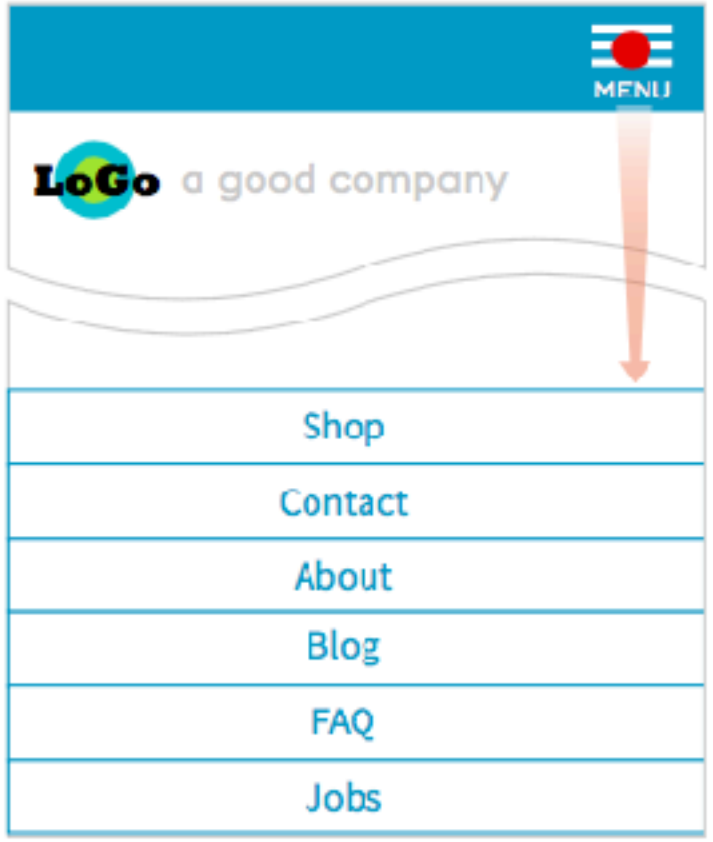
KEY

● = tap

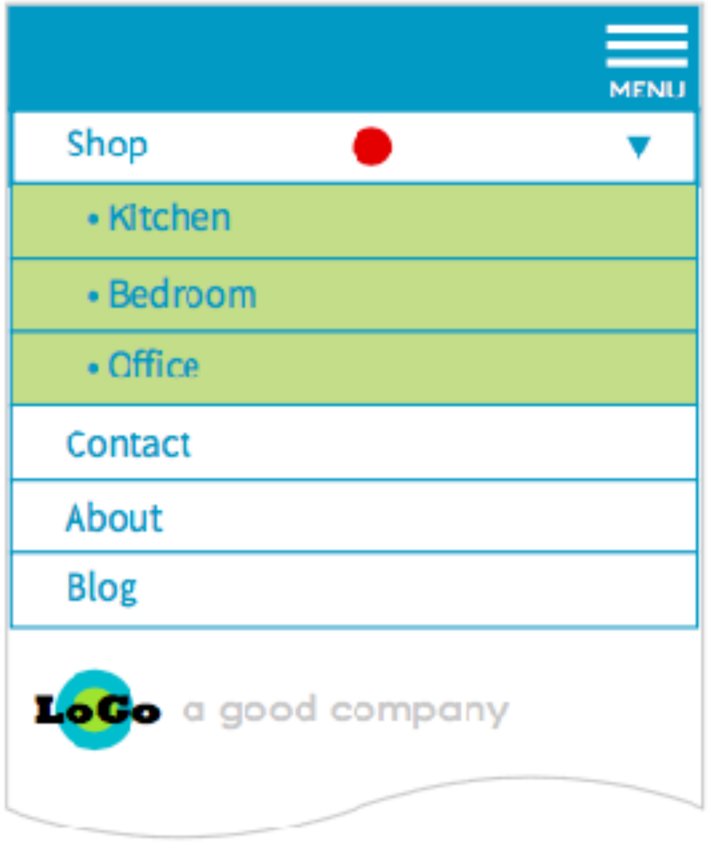
Select menu



Link to footer menu



Accordion sub-navigation



Select menu

For a medium list of links, some sites use a select input form element. Tapping the menu opens the list of options using the select menu UI of the operating system, such as a scrolling list of links at the bottom of the screen or on an overlay. The advantage is that it is compact, but on the downside, forms aren't typically used for navigation, and the menu may be overlooked.

Link to footer menu

One straightforward approach places a Menu link at the top of the page that links to the full navigation located at the bottom of the page. The risk with this pattern is that it may be disorienting to users who suddenly find themselves at the bottom of the scroll.

Accordion sub-navigation

When there are a lot of navigation choices with sub-navigation menus, the small-screen solution becomes more challenging, particularly when you can't hover to get more options as you can with a mouse. Accordions that expand when you tap a small arrow icon are commonly used to reveal and hide sub-navigation. They may even be nested several levels deep. To avoid nesting navigation in accordion submenus, some sites simply link to separate landing pages that contain a list of the sub-navigation for that section.

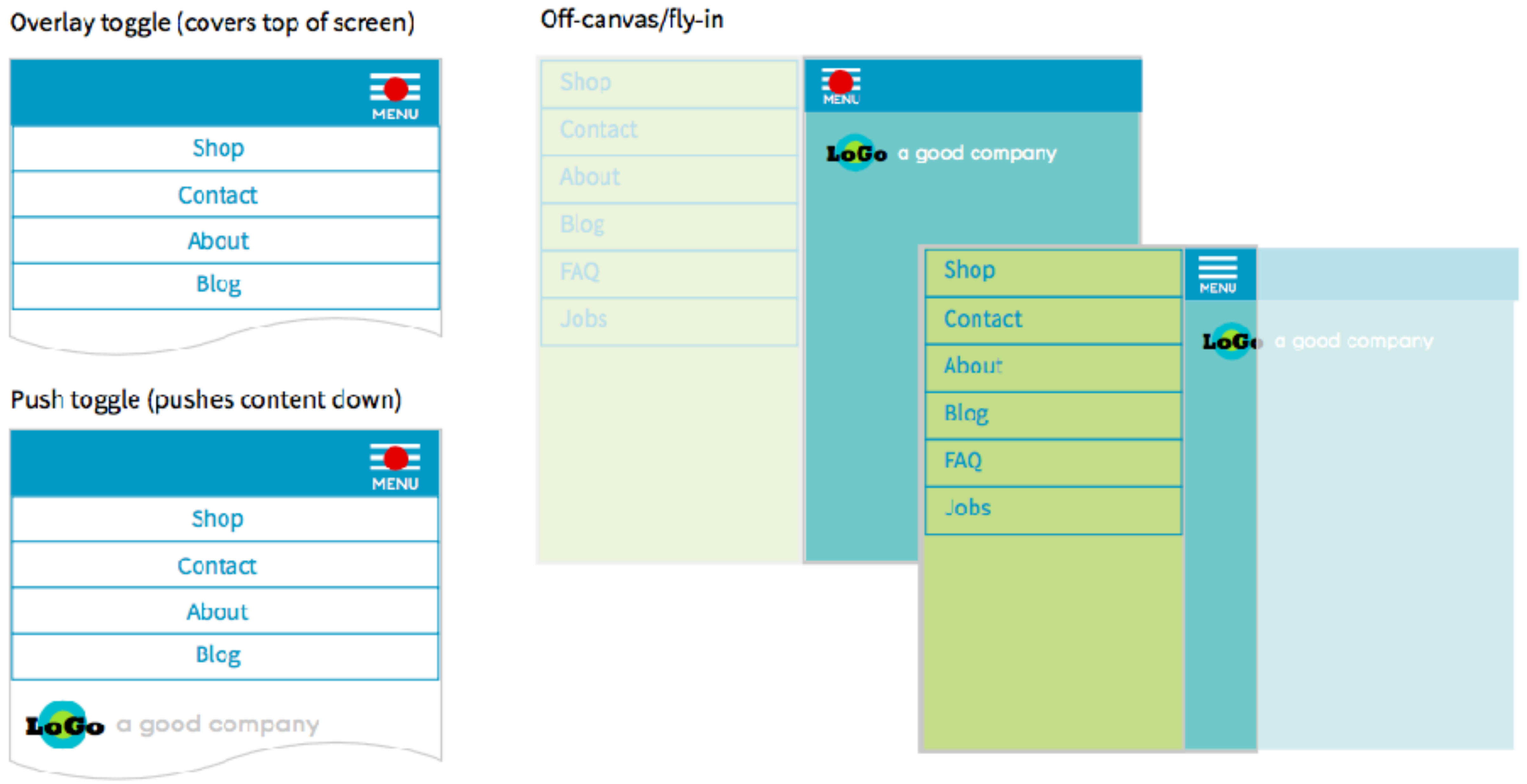


FIGURE 17-11. Responsive navigation patterns.

Display Property

The display property specifies if/how an element is displayed. Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline. A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can). An inline element does not start on a new line and only takes up as much width as necessary.

```
        display: none;
```

commonly used with JavaScript to hide/show elements without deleting and recreating them.

Overriding Default Display

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

```
li {
    display: inline;
}

span {
    display: block;
}
```

Note: Setting the display property of an element only changes how the element is displayed, NOT what kind of element it is. So, an inline element with display: block; is not allowed to have other block elements inside it.

Overflow Property

The CSS overflow property controls what happens to content that is too big to fit into an area. The overflow property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area. The overflow property only works for block elements with a specified height. The overflow property has the following values:

- **visible** - Default. The overflow is not clipped. It renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, but a scrollbar is added to see the rest of the content
- **auto** - If overflow is clipped, a scrollbar should be added to see the rest of the content

Properties for left and right

overflow-x

specifies what to do with the left/right edges of the content.

overflow-y

specifies what to do with the top/bottom edges of the content.

```
div.theExample1 {
    background-color: lightblue;
    height: 40px;
    width: 200px;
    overflow-y: scroll;
}

div.theExample2 {
    background-color: lightblue;
    height: 40px;
    width: 200px;
    overflow-y: hidden;
}
```

code from W3

Media Queries

the @media rule tells the browser to include a block of CSS properties only if a certain condition is true.

So this:

```
@media only screen and (max-width: 500px)  {
    body {
      background-color: light blue;
    }
}
```

Translates to:

```
if (the maximum width of the web page is 500 pixels) {
                then do this stuff
}
```

Breakpoint

```
/* For mobile phones: */
[class*="col-"] {
    width: 100%;
}
@media only screen and (min-width: 768px) {
    /* For desktop: */
    .col-1 {width: 8.33%;}
    .col-2 {width: 16.66%;}
    .col-3 {width: 25%;}
    .col-4 {width: 33.33%;}
    .col-5 {width: 41.66%;}
    .col-6 {width: 50%;}
    .col-7 {width: 58.33%;}
    .col-8 {width: 66.66%;}
    .col-9 {width: 75%;}
    .col-10 {width: 83.33%;}
    .col-11 {width: 91.66%;}
    .col-12 {width: 100%;
}
```

add a **breakpoint** where certain parts of the design will behave differently on each side of the breakpoint

many examples: https://www.w3schools.com/Css/css_rwd_mediaqueries.asp

Flexbox & CSS Grid

By Flexbox and CSS Grid are designed to be responsive and eliminate the need for media queries. However, they both offer many options as to the design while remaining responsive.

- + Example of Responsive Image Gallery
- + CSS Flexbox
- + CSS Grid + Responsive Layout

Flexbox & CSS Grid

"The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time. The two specifications share some common features, however, and if you have already learned how to use flexbox, the similarities should help you get to grips with Grid."

MDN

The flex-direction property defines in which direction the container wants to stack the flex items - either flex-direction: row or flex-direction: column. However, by using flex-wrap property. Read all about CSS Flexbox @ W3.

CSS Grid

A grid is an intersecting set of horizontal and vertical lines - one set defining columns and the other rows. Elements can be placed onto the grid, respecting these column and row lines.

How Grid Layout Works

The process for using the CSS Grid Layout Module is fundamentally simple:

+ Use the display property to turn an element into a grid container. The element's children automatically become grid items.
+ Set up the columns and rows for the grid. You can set them up explicitly and/or provide directions for how rows and columns should get created on the fly (the css grid is very flexible).
+ Assign each grid item to an area on the grid. If you don't assign them explicitly, they flow into the cells sequentially.

The element that has the display: **grid property** applied to it becomes the grid container and defines the context for grid formatting. All of its direct child elements automatically become grid items that end up positioned in the grid. You can define an explicit grid with grid layout but the specification also deals with the content added outside of a declared grid, which adds additional rows and columns when needed. Features such as adding "as many columns that will fit into a container" are included.

Grid line

The horizontal and vertical dividing lines of the grid are called grid lines.

Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it.

Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells.

Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis.

The structure established for the grid is independent from the number of grid items in the container. You could place 4 grid items in a grid with 12 cells, leaving 8 of the cells as 'whitespace.' That's the flexibility of grids. You can also set up a grid with fewer cells than grid items, and the browser adds cells to the grid to accommodate them.

CSS TRANSITIONS

ThCSS Transitions smooth out otherwise abrupt changes to property values between two states over time by filling in the frames in between. Animators call that tweening. When used with reserve, CSS Transitions can add sophistication and polish to your interfaces and even improve usability.

When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

- Which CSS property to change (**transition-property**) (Required)
- How long it should take (**transition-duration**) (Required)
- The manner in which the transition accelerates (**transition-timing- function**)
- Whether there should be a pause before it starts (**transition-delay**)

Transitions require a **beginning state** and an **end state**. The element as it appears when it first loads is the beginning state. The end state needs to be triggered by a state change such as :hover, :focus, or :active…

transition-property

identifies the CSS property that is changing and that you want to transition smoothly. In our example, it’s the background-color. You can also change the foreground color, borders, dimensions, font- and text-related attributes, and many more. TABLE 18-1 lists the animatab CSS properties as of this writing. The general rule is that if its value is a color, length, or number, that property can be a transition property.

Midterm

Your midterm is to create a website using HTML + CSS. It must contain at least **three layers** of HTML file parenting, **two different** CSS styling, clean + heavily **commented code**, interactive CSS + thoughtful concept + appealing design and the project must have a **title**.

You must come up with a project proposal. A slide deck that illustrates the concept and purpose of your site as well as a sitemap and wireframes for the different pages. This is due **Next Week - October 24**. We will look at + discuss everyone's proposal on those days.

Projects should be hosted on your FM server in it's unique directory + url. URLs should posted to the wiki before **4pm** on **Wednesday October 31**. We will spend most (if not all of the class) critiquing your sites.

The Midterm is both the project + critique on the 31st **AS WELL AS** the project proposal and process that leads to the final outcome. Your ideation is as important as your execution - so next week should be considered as the 31st.

Midterm Prompts

1. Create a hyper text narrative or net art piece
2. Invent a fictional or futuristic product (a machine that records your dreams, a shoe that plays music, etc) + create a website for it. Inspiration
3. Create a website for something / someone in your life. Your band, your Mom's ceramics hobby, etc. Inspiration

How can we think of creating content for the web that is NOT a portfolio? You will be graded in equal parts on technical expertise, creativity + concept and a clear indication of how much you've learned over the last six weeks. Code must be responsive, commented + full of links to the resources you found that helped you create your project