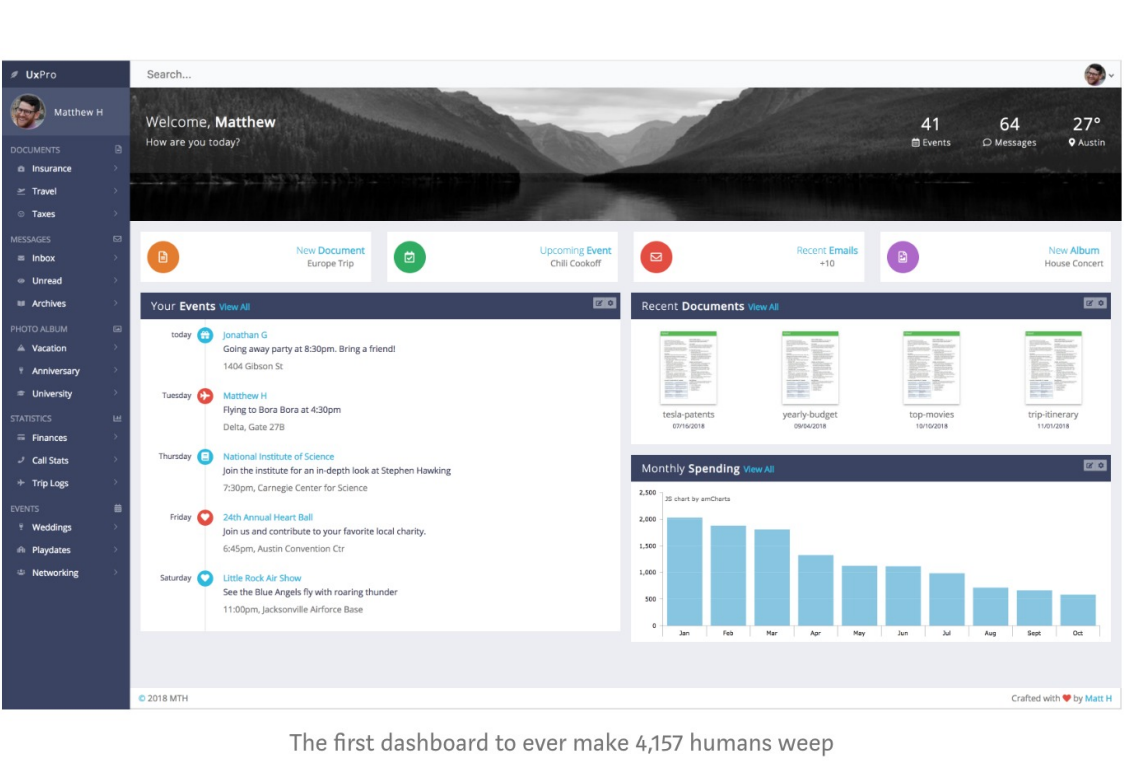
Build a Responsive, Modern Dashboard Layout With CSS Grid and Flexbox

Create a beautiful, responsive dashboard page without any framework and including a sliding side nav

Matt Holland Follow
Nov 14, 2018 · 8 min read ★



The first dashboard to ever make 4,157 humans weep

Have you every been mystified by those gorgeous online dashboard demos that have you wondering, "Will I ever be able to build that…?" Well, I am here to tell you that you *can build it!*

. . .

How Do We Go About This?

We will first introduce some CSS Grid basics. We'll then use those to build our base dashboard layout. After that, we'll explore the setup and structure of our inner content blocks, including some flexbox. Finally, we'll talk briefly about responsible mobile-responsiveness, while integrating mobile-friendly sliding functionality into our side nav. What will you walk away with? A boom-goes-the-dynamite dashboard that will make people like you a lot.

Note: We will be building a simplified version of the dashboard above, which uses the same concepts as the full version. The full version would make this piece silly long…

First things first: our basic grid layout

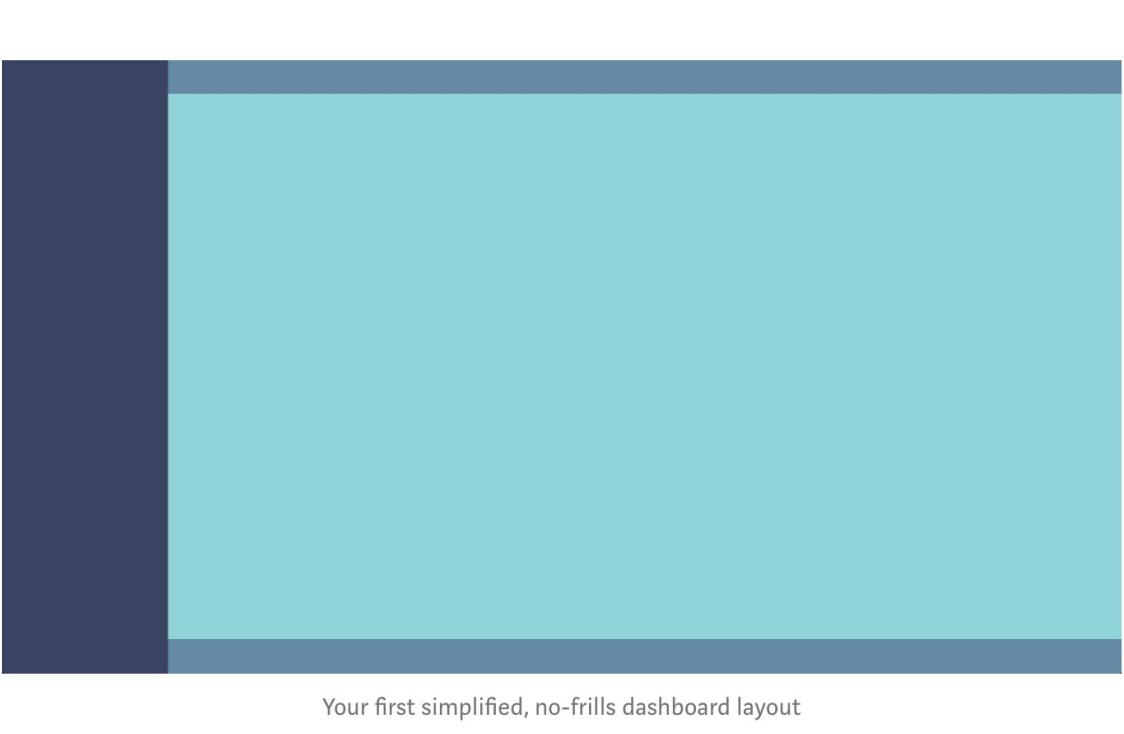
I'm going to break down CSS Grid for you, short and sweet. We need a main grid container, and then we need a `div` (or semantic element) for each element within the grid container:

```
// Simple dashboard grid layout
<div class="grid-container">
  <header class="header"></header>
  <aside class="sidenav"></aside>
  <main class="main"></main>
  <footer class="footer"></footer>
</div>
```

dashboard.html hosted with ❤ by GitHub view raw

Our basic overall dashboard page structure

Pretty simple structure, right? Our layout will help produce this immediately-beautiful canvas below. Don't worry, we will be adding more content.



Your first simplified, no-frills dashboard layout

Next, let's lay down the law…I mean CSS

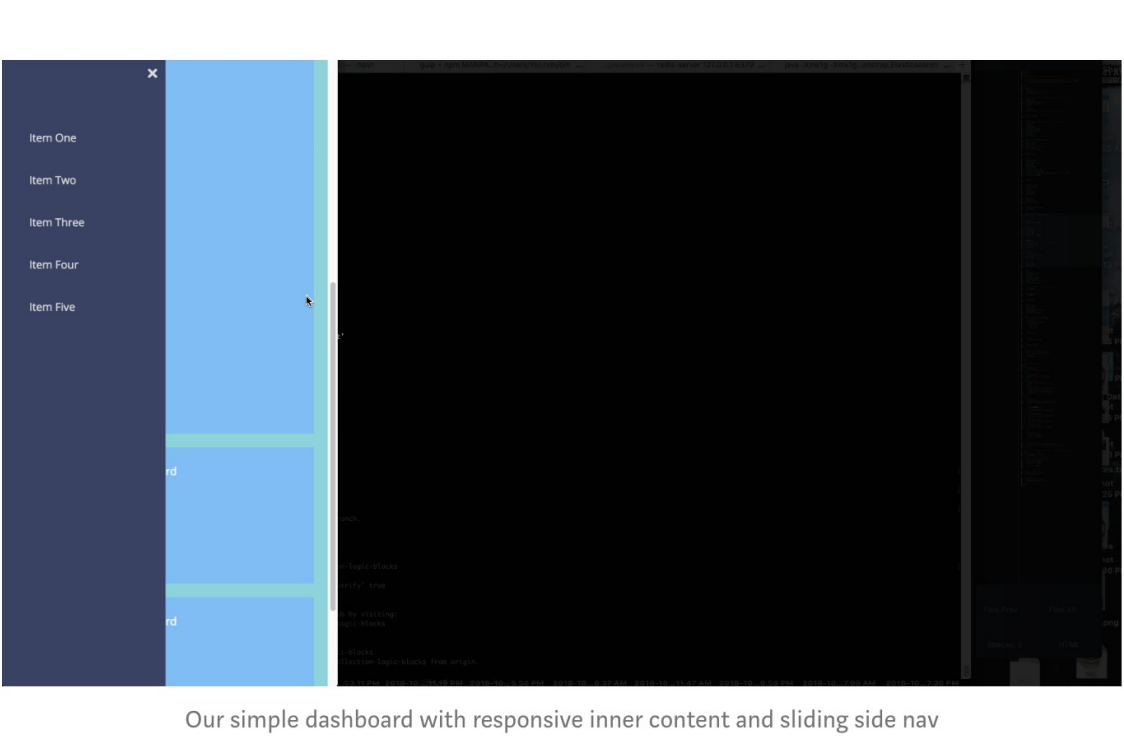
Your main container must be defined as `display: grid;` for any grid functionality to actually work. We also give it a height of 100% to tell it that we want our dashboard to fill the entire page. And, for each child container, we will assign a name so that we can tell the grid what to do with it. Then we will use the names to create the page structure in a spreadsheet-like declaration using `grid-template-areas`:

```
/* Simple dashboard grid CSS */

/* Assign grid instructions to our parent grid container */
.grid-container {
  display: grid;
  grid-template-columns: 240px 1fr;
  grid-template-rows: 50px 1fr 50px;
  grid-template-areas:
    "sidenav header"
    "sidenav main"
    "sidenav footer";
  height: 100vh;
}

/* Give every child element its grid name */
.header {
  grid-area: header;
  background-color: #648ca6;
}

.sidenav {
  grid-area: sidenav;
  background-color: #394263;
}

.main {
  grid-area: main;
  background-color: #8fd4d9;
}

.footer {
  grid-area: footer;
  background-color: #648ca6;
}
```

dashboard.css hosted with ❤ by GitHub view raw

Our basic dashboard grid layout styles

Our final, responsive dashboard will look like this:



Our simple dashboard with responsive inner content and sliding side nav

Grid-template-areas explained

We basically assigned each of our child containers a name, and then threw them into a spreadsheet-like format via `grid-template-areas`. Dead simple.

We have two columns total from left to right. The first column is 250px wide (the side nav), and the second one is 1fr, or fraction. That means that it will take up the remaining container space after the first column has been drawn.

Next, we declare three total rows. Rows flow from top to bottom. So, starting at the top, we have a `<header>` element that is 50px tall. Then we declare the `<main>` content area, which is given a height of 1fr. This means that it will vertically stretch to fill the remaining window space after explicitly-declared heights have been drawn. Finally, we declare our `<footer>`, also at a height of 50px.

Adding the header and footer

Both the `<header>` and the `<footer>` will be flex containers, with flex spacing and alignment:

```
// Create the styles and structure for our header and footer elements; grid-area declare
<style>
  .header, .footer {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 16px;
    background-color: #648ca6;
  }
</style>

<header class="header">
  <div class="header__search">Search...</div>
  <div class="header__avatar">Your face</div>
</header>

<footer class="footer">
  <div class="footer__copyright">&copy; 2018 MTH</div>
  <div class="footer__signature">Made with love by pure genius</div>
</footer>
```

dashboard-header-footer.html hosted with ❤ by GitHub view raw

Our dashboard header and footer setup

We use `justify-content: space-between;` to spread out the first and last elements so that they stretch to both ends of their container. This is so easy compared to old-school `float` instructions. And, thanks to `align-items: center;`, we have our items perfectly aligned without having to rely on padding, etc., for centering.

A brief note on our weird-looking CSS class syntax: we are using BEM-style (block-element-modifier) CSS, which I recommend for scalability and readability. It is recommended to avoid raw html tag selectors.

Adding the side nav element

For the side nav content, we use traditional `<ul>` and `<li>` elements. I recommend this over `<div>` or any other elements for good html semantics and human readability. We will be adding mobile-friendly sliding functionality into our side nav a little bit further down:

```
// Styles and structure for our simplified dashboard side nav content
<style>
  .sidenav {
    display: flex; /* Will be hidden on mobile */
    flex-direction: column;
    grid-area: sidenav;
    background-color: #394263;
  }

  .sidenav__list {
    padding: 0;
    margin-top: 85px;
    list-style-type: none;
  }

  .sidenav__list-item {
    padding: 20px 20px 20px 40px;
    color: #ddd;
  }

  .sidenav__list-item:hover {
    background-color: rgba(255, 255, 255, 0.2);
    cursor: pointer;
  }
</style>


<aside class="sidenav">
  <ul class="sidenav__list">
    <li class="sidenav__list-item">Item One</li>
    <li class="sidenav__list-item">Item Two</li>
    <li class="sidenav__list-item">Item Three</li>
    <li class="sidenav__list-item">Item Four</li>
    <li class="sidenav__list-item">Item Five</li>
  </ul>
</aside>
```

dashboard-sidenav.html hosted with ❤ by GitHub view raw

Our dashboard side nav basic structure

Adding the first `<main>` section element

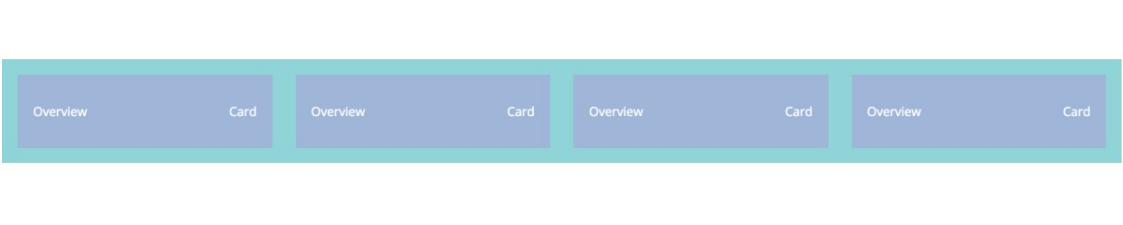
This one is straightforward and simple. Another flex container:

```
// The very first element inside of the <main> parent container
<style>
  .main-header {
    display: flex;
    justify-content: space-between;
    margin: 20px;
    padding: 20px;
    height: 150px; /* Force our height since we don't have actual content yet */
    background-color: #e3e4e6;
    color: slategray;
  }
</style>

<div class="main-header">
  <div class="main-header__heading">Hello User</div>
  <div class="main-header__updates">Recent Items</div>
</div>
```

main-header.html hosted with ❤ by GitHub view raw

Our first <main> section element

Now things get interesting… the responsive grid introduction cards



This is one of my favorite sections of our dashboard, because we get to utilize a super effective, elegant grid solution. Watch the dashboard animation provided previously again. Notice how these gray cards behave as the screen changes? Our cards amazingly keep a consistent gutter width between them, they have consistent wrapping, and when we can't fill up an entire row, our wrapped card matches exactly the height and width of the card above it, while remaining lined up with it. This is *very* challenging and tedious to accomplish without the method that I'm about to show you:

```
// Main section responsive grid intro cards
<style>
  .main-overview {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(265px, 1fr)); /* Where the magic happ
    grid-auto-rows: 94px;
    grid-gap: 20px;
    margin: 20px;
  }

  .overviewcard {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 20px;
    background-color: #d3d3;
  }
</style>

<div class="main-overview">
  <div class="overviewcard">
    <div class="overviewcard__icon">Overview</div>
    <div class="overviewcard__info">Card</div>
  </div>
  <div class="overviewcard">
    <div class="overviewcard__icon">Overview</div>
    <div class="overviewcard__info">Card</div>
  </div>
  <div class="overviewcard">
    <div class="overviewcard__icon">Overview</div>
    <div class="overviewcard__info">Card</div>
  </div>
  <div class="overviewcard">
    <div class="overviewcard__icon">Overview</div>
    <div class="overviewcard__info">Card</div>
  </div>
</div>
```

dashboard-intro-cards.html hosted with ❤ by GitHub view raw

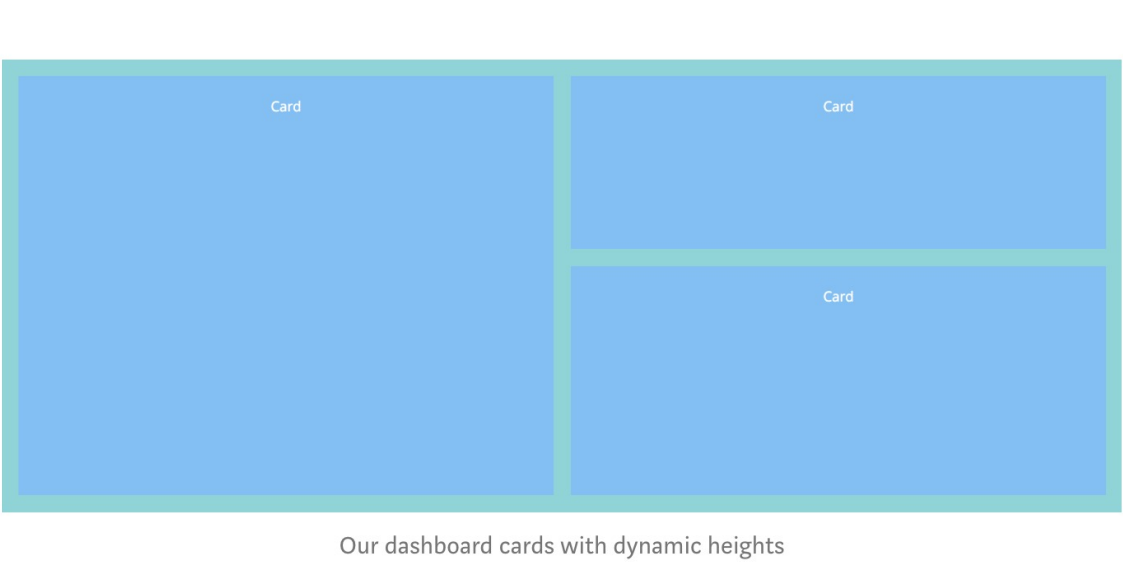
The very nice, responsive grid overview cards

Neat how we used a grid container for these, which is inside of our main page grid container, right? We did this because it's the most straightforward, elegant solution to use for the needed responsiveness of the card items. Our instruction `repeat(auto-fit, minmax(265px, 1fr)` takes care of a couple of major hurdles:

1. If cards go below 265px in width, they will wrap to another row.
2. If cards go above 265px in width, they will stretch to take up the available remaining container width.
3. When cards wrap to a new row (`auto-fit`), they will line up from left to right with the cards above them, matching their widths! *And* you also get built-in responsiveness without any media queries!

Using the column `repeat` method is also a *fantastic* way to build beautiful, responsive image galleries, even with images of differing sizes. You even have access to dynamic packing algorithms with the `grid-auto-flow: dense;` instruction. This will prevent any empty row spaces due to different image heights. They must scale with each other in relative `fr` units for this method to work, however, which is why we won't use it for our cards shown below.

Adding the main content containers



Our dashboard cards with dynamic heights

This section also has an interesting twist. These cards will contain your dashboard's main content items, and they will vary in height from each other due to their dynamic content. On most typical interfaces, you would want like-minded cards on the same row to have the same uniform height and width. And you would accomplish that by assigning each card a `flex: 1;` value so that they would grow to match the tallest card.

However, in our case, we don't want to force these cards to match each other's height because their content subject matter will vary. To get them flowing naturally in two columns, we will use a special CSS property for this, `column-count`:

```
// Build the main section content cards
<style>
  .main-cards {
    column-count: 2;
    column-gap: 20px;
    margin: 20px;
  }

  .card {
    display: flex;
    flex-direction: column;
    align-items: center;
    width: 100%;
    background-color: #82bef6;
    margin-bottom: 20px;
    -webkit-column-break-inside: avoid;
    padding: 24px;
    box-sizing: border-box;
  }

  /* Force varying heights to simulate dynamic content */
  .card:first-child {
    height: 485px;
  }

  .card:nth-child(2) {
    height: 200px;
  }

  .card:nth-child(3) {
    height: 265px;
  }
</style>

<div class="main-cards">
  <div class="card">Card</div>
  <div class="card">Card</div>
  <div class="card">Card</div>
</div>
```

dashboard-main-containers.html hosted with ❤ by GitHub view raw

The dashboard main content cards

Using `column-count` will ensure that our content inside of the `main-cards` section gets split into two columns. We also apply a gap between the cards with `column-gap`. This is very similar to our overview cards, where we used `grid-gap`.

The reason we didn't use `display: grid;` for this section is that our heights for each card are dynamic. We want them to flow naturally into two columns, while observing their varying heights. This method also prevents us from having to use a traditional float grid, which would have us calculating percentage widths, gutters, and special margin rules for first and last child elements.

We also used `column-break-inside: avoid;` to make sure that each card does not get its content split. `column-count` will actually break up the content of each element to make both rows the same height, which we do not want.

A brief note on responsible mobile-responsiveness

Watch the responsive video at the top of the piece again. See how the `main-cards` section turns into one column as you approach tablet-sized screens? And see how the side nav disappears on mobile-sized screens? For this responsiveness, we should really write our CSS in a mobile-first format.

That means that our initial CSS should be mobile-friendly. Then, as our screen size increases, we observe larger screen styles using graduating `min-width` media queries. This is better practice than overriding desktop styles with `max-width` media queries, as that approach can lead to some headaches:

```
// Make our dashboard responsive in a mobile-first format
<style>
  .grid-container {
    display: grid;
    grid-template-columns: 1fr; /* Side nav is hidden on mobile */
    grid-template-rows: 50px 1fr 50px;
    grid-template-areas:
      'header'
      'main'
      'footer';
    height: 100vh;
  }

  .sidenav {
    display: none;
    grid-area: sidenav;
    background-color: #394263;
  }

  .main-cards {
    column-count: 1;
    column-gap: 20px;
    margin: 20px;
  }

  /* Non-mobile styles, 750px breakpoint */
  @media only screen and (min-width: 46.875em) {
    /* Show the sidenav */
    .grid-container {
      grid-template-columns: 240px 1fr; /* Show the side nav for non-mobile screens */
      grid-template-areas:
        "sidenav header"
        "sidenav main"
        "sidenav footer";
    }

    .sidenav {
      display: flex;
      flex-direction: column;
    }
  }

  /* Medium-sized screen breakpoint (tablet, 1050px) */
  @media only screen and (min-width: 65.625em) {
    /* Break out main cards into two columns */
    .main-cards {
      column-count: 2;
    }
  }

</style>
```

dashboard-responsive.html hosted with ❤ by GitHub view raw

Graduating min-width media queries

Making our side nav slidable on mobile devices

Our job is not really complete until we also make the side nav useable on mobile devices. We need to:

1. Add our menu icon and close icon.
2. Add some responsive transitions for the sliding action.
3. Write some JavasSript to make our clicks activate the side nav.

We'll use the Font Awesome library for our icons, and bring in jQuery for some easy DOM manipulation (See the codepen for reference).

```
<!-- Dashboard main menu icon and side nav close icon -->

<div class="grid-container">
  <div class="menu-icon">
    <i class="fas fa-bars"></i>
  </div>
</div>

<aside class="sidenav">
  <div class="sidenav__close-icon">
    <i class="fas fa-times"></i>
  </div>
</aside>
```

dashboard-sliding-side-nav.html hosted with ❤ by GitHub view raw

The main menu icon and side nav close icon for mobile

Next, let's update our CSS to include the new icons and to give our side nav some sliding transitions. Once again, we will use *graduating* media queries:

```

/* Hamburger menu icon, stays fixed on mobile for any possible scrolling */
.menu-icon {
  position: fixed;
  display: flex;
  top: 5px;
  left: 10px;
  align-items: center;
  justify-content: center;
  border-radius: 50%;
  z-index: 1;
  cursor: pointer;
  padding: 12px;
  background-color: #DADAE3;
}

/* Make room for the menu icon on mobile */
.header__search {
  margin-left: 42px;
}

/* Mobile-first side nav styles */
.sidenav {
  grid-area: sidenav;
  display: flex;
  flex-direction: column;
  height: 100%;
  width: 240px;
  position: fixed;
  overflow-y: auto;
  box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.16), 0 0 0 1px rgba(0, 0, 0, 0.08);
  z-index: 2; /* Needs to sit above the hamburger menu icon */
  background-color: #394263;
  transform: translateX(-245px);
  transition: all .6s ease-in-out;
}

/* The active class is toggled on hamburger and close icon clicks */
.sidenav.active {
  transform: translateX(0);
}

/* Only visible on mobile screens */
.sidenav__close-icon {
  position: absolute;
  visibility: visible;
  top: 8px;
  right: 12px;
  cursor: pointer;
  font-size: 20px;
  color: #ddd;
}

/* Non-mobile styles for side nav responsiveness, 750px breakpoint */
@media only screen and (min-width: 46.875em) {
  .sidenav {
    position: relative; /* Fixed position on mobile */
    transform: translateX(0);
  }

  .sidenav__close-icon {
    visibility: hidden;
  }
}
```

dashboard-sliding-nav.css hosted with ❤ by GitHub view raw

Mobile-responsive styles for our side nav and menu icons

Finally, we need to write some JavaScript to get our clicks working. The sliding functionality is accomplished by toggling the `.active` class name, which updates the `transform: translateX();` instruction. Don't forget to add the jQuery CDN link before your ending `</body>` tag:

```
<body>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></scrip
  <script>
    const menuIconEl = $('.menu-icon');
    const sidenavEl = $('.sidenav');
    const sidenavCloseEl = $('.sidenav__close-icon');

    // Add and remove provided class names
    function toggleClassName(el, className) {
      if (el.hasClass(className)) {
        el.removeClass(className);
      } else {
        el.addClass(className);
      }
    }

    // Open the side nav on click
    menuIconEl.on('click', function() {
      toggleClassName(sidenavEl, 'active');
    });

    // Close the side nav on click
    sidenavCloseEl.on('click', function() {
      toggleClassName(sidenavEl, 'active');
    });
  </script>
</body>
```

dashboard-sidenav.js hosted with ❤ by GitHub view raw

The JavaScript that makes our side nav slide

You should now have a fully responsive side nav. Scale down your window until it is mobile-sized. It should hide, and the menu icon should appear. Click it, and the side nav should slide into view. Click the close icon, and the side nav should close again. Pretty slick, eh? And you thought you couldn't do it…

. . .

In Summary…

What did you just learn?

1. How to build a dashboard layout using CSS Grid & Flexbox.
2. Special rules for dynamic content blocks using `repeat`, `auto-fit`, `min-max`, `column-count`, `grid-gap`, and `column-gap`.
3. The *right* way to write mobile-first, responsive CSS using graduating `min-width` media queries.
4. How to build a sliding, mobile-responsive side nav.

Please check out the full dashboard codepen, which includes a clickable, sliding side nav with accordion list items, a nice dropdown user menu, and some slick `transition` hover action.

I hope you gained some new knowledge from this piece and that dashboard layouts will never mystify you ever again. Please comment if you have any questions!

Css Grid · Flexbox · Dashboard · Responsive Design · Programming

3.3K claps