Mastering Modern CSS Layouts

Grids, Flexbox & Responsive Design

Ohans Emmanuel

Mastering Modern CSS Layouts

Mastering Modern CSS Layouts

Everything you need to Master the Powerful Flexbox & Grid CSS Layout Models

Ohans Emmanuel

Preface

Like a lot of front-end developers, my face beamed with smiles as I found out I didn’t have to write a lot of hackish, rather messed up CSS anymore. There was this thing called Flexbox and I was determined to learn how it worked.

Did the Flexbox model fool me? Oh yes it did! It looked quite different from what I was used to and I got very confused many times. This book however tells that I found a way around my confusion, but that took lots of hours and on many different days.

And the CSS Grid model? If you’ve played around with this, then you already know it takes a complete re-working of how you’ve viewed CSS layouts. It’s perhaps the more confusing of the two - at least for me, when I first got started.

I hope to relieve you of the stress many of us go through while learning to use these incredibly powerful layout models that will re-shape the way we’ve all written CSS layout codes forever.

Every decent endeavor takes some time but It’s the sole aim of this book to guide to mastery, so you can get up and running in less the time it’d normally take and with many practical use cases for today!

Acknowledgment

It can be a pain in the neck to self publish a book. For that reason I must appreciate some really amazing people without whom this book wouldn’t be possible.

Big thanks to my family. Watching me take away so much time to write this book, without complains... how awesome.

Thank you guys!

About this book

Here we are! This books aims to teach you everything about Flexbox and Grids, the CSS layout models that makes life really easy. That's a bit much. Okay, it makes layout with CSS really easy, not to mention that this is most certainly going to be the new standard in the coming years.

This book assumes you know a bit of CSS to get along, but no previous knowledge of the Flexbox, Grid model or responsive design is assumed.

Pathway

This book is divided into three parts.

Part 1 is an introduction to the Flexbox and Grid model. Let's say it's a look at these powerful layout models first from a conceptual perspective. If you have been asking questions like what's Flexbox? How do I understand the syntax of the Grid layout? What can be done with Flexbox and Grids? All these questions (and more) will be answered in this section of the book. Part 1 aims to give you a strong understanding of both the Flexbox and Grid model.

On finishing part 1, you'd start to feel really confident and then we take things seriously by taking a look at some practical real world examples on how your newly found knowledge may be put to good use. I do love this section as it lets your knowledge loose into real world use cases.

With some practical use cases under your belt we will go on to lay out much more full-fledged applications. Chances are, at work you'll have to use more than one technology and certain requirements will be needed. Requirements that transcend simple strategies. You'd want to accommodate your layout to varying devices such as mobile, tablets etc. Very importantly, you will need to deal with those clients of yours on old browsers! How do we make sure that the relatively new Flexbox and Grid model work just fine on Mr Jones' PC which runs an old ,outdated browser?

We will equally take a look at CSS3 animations. A lot of people do not realize that the Flexbox model comes with some savor when dealing with CSS3 animations.
Did I mention that this chapter is my favorite? The reason is simple. Here we go pro and wrap up things like we would in the real world.

And after knowing all this, you may close the book. You'll be able to make your colleagues look clueless, at least when it comes to using the Flexbox & Grid models - I hope that's not such a bad thing

Code conventions

This book comes with a lot of copious examples that show in great detail the topics covered. The source code in listings or in text appears in a `fixed-width` font like `this` to separate it from ordinary text.

Getting the source code

The code for all examples in this book is available for download on Github at https://github.com/ohansemmanuel/mastering-modern-css-layouts where each chapter has a corresponding folder.

If you’re as lazy as me, you’d probably just copy and paste these codes to see them work. That’s not a very good idea you know? So, go grab the code but use it as a reference.

Type the codes by hand when you need to see how things work. Trust me, you’ll thank me later for this. You’ll probably spend more of your time at work writing codes not just copying and pasting off of the Internet.

About the Author

Ohans Emmanuel is a software engineer at codeHouseNg where he works a lot with new web technologies including Html5, Javascript - and CSS3 (gotta make things look pretty too). He is human, and crazy enough to think he will make the world a much more better place someday.

Contents

PART 3: GRID AND FLEXBOX IN CONTEXT

Part 2

Practical

Aye! You made it to the second art of the show and, like most intermediary scenes, It is packed full with creative punch lines and surprises. Perhaps, I've saved up the better for this section!

In the previous sections, we spent time laying a solid foundation of understanding the Flexbox and Grid model and even applying it to some real world scenario. It's show time!
In this section we'll walk through building more full-fledged layouts with a focus on responsive designs.

Chapter 6 shows what the Flexbox model brings to the responsive design table and we will explore in detail how well it's suited for this purpose.
With Flexbox, we get some responsiveness right off the bat, but you'll learn how to customize your layouts further for varying screen resolutions using media queries, you'll understand what breakpoints are and then move on to build a responsive layout with Flexbox.

In chapter 7, we will switch our focus to the Grid layout. Does this add anything to responsiveness? Doesn't the Flexbox model handle all of that? How well suited is the Grid layout for responsive designs? I hope to answer all these questions and more in this beautiful chapter.

It doesn't end there. In chapter 8 we'll wrap things up by building an advanced responsive design with Grids and Flexbox. That's ambiguous. Here is what we'll actually do: We will sum up all we've learned, and do something huge. Huge? Perhaps. We will build a Facebook clone! Oh yes.

After these chapters you should begin to feel like a Flexbox and Grid expert.

5

Responsive design with Flexbox

This chapter covers

- Using CSS media queries for custom styles on selected devices
- How certain Flexbox properties are well suited for dealing with mobile devices

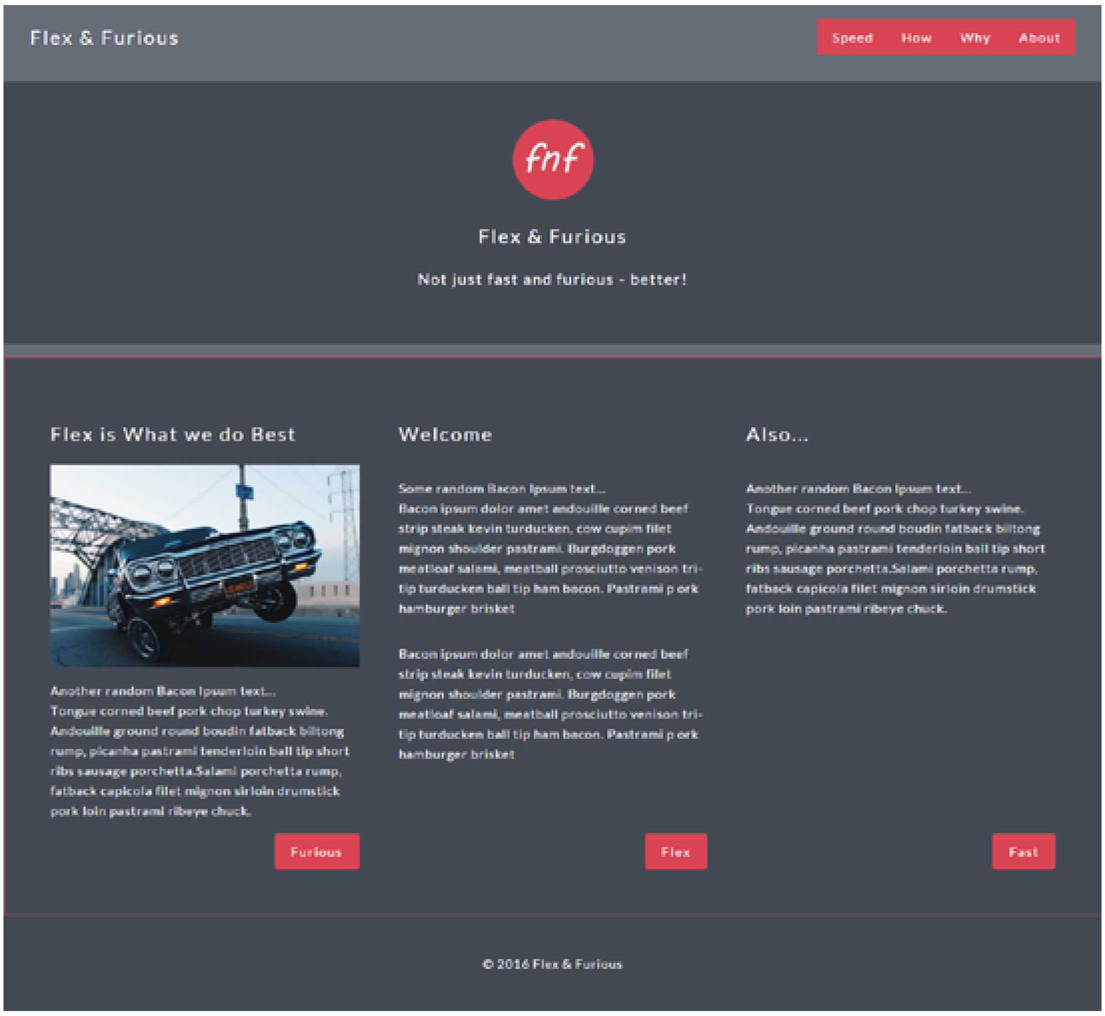
In previous chapters, you learned what the Flexbox model is, how to use it in your project, and now you already know why it's so popular and such a powerful tool to have in your responsive design arsenal.

Flexbox is awesome. I really have to say that again, now that you're starting to see reasons with me.

In this chapter we will be working through our first implementation of a responsive design with Flexbox. It's nothing fancy, but there are some core concepts to be learned. We'll build on these concepts as we face more challenging projects in the coming chapters.

Uh, so what are we going to be working on? We'll be building a couple responsive modules for a fictitious company called Flex and Furious. Sounds fascinating, huh?

Here's what the finished layout looks like. Nothing fancy, like I said. You may need to zoom in on this image though.



Flex & Furious

Speed How Why About

fnf

Flex & Furious

Not just fast and furious - better!

Flex is What we do Best

Another random Bacon Ipsum text...
Tongue corned beef pork chop turkey swine. Andouille ground round boudin fatback biltong rump, picanha pastrami tenderloin ball tip short ribs sausage porchetta.Salami porchetta rump, fatback capicola filet mignon sirloin drumstick pork loin pastrami ribeye chuck.

Furious

Welcome

Some random Bacon Ipsum text...
Bacon ipsum dolor amet andouille corned beef strip steak kevin turducken, cow cupim filet mignon shoulder pastrami. Burgdoggen pork meatloaf salami, meatball prosciutto venison tri-tip turducken ball tip ham bacon. Pastrami p ork hamburger brisket

Bacon ipsum dolor amet andouille corned beef strip steak kevin turducken, cow cupim filet mignon shoulder pastrami. Burgdoggen pork meatloaf salami, meatball prosciutto venison tri-tip turducken ball tip ham bacon. Pastrami p ork hamburger brisket

Flex

Also...

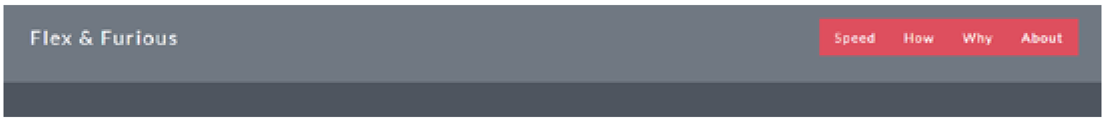
Another random Bacon Ipsum text...
Tongue corned beef pork chop turkey swine. Andouille ground round boudin fatback biltong rump, picanha pastrami tenderloin ball tip short ribs sausage porchetta.Salami porchetta rump, fatback capicola filet mignon sirloin drumstick pork loin pastrami ribeye chuck.

Fast

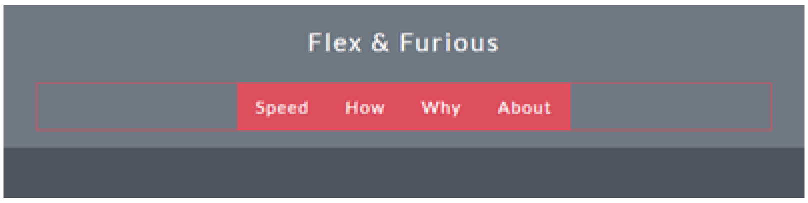
© 2016 Flex & Furious

You may not have noticed any responsiveness here but you may view this in your browser and try resizing to see the responsiveness in action. We will start off with building the responsive navigation for the website.

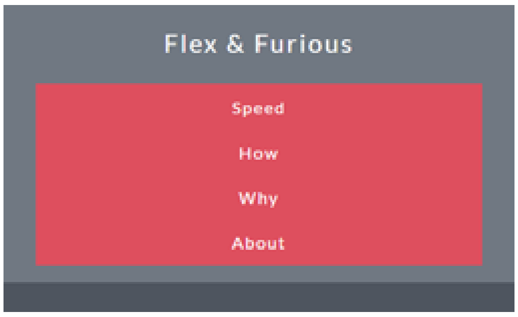
For clarity, I'd like to show here what we are looking to achieve. On laptops and bigger screens we want the navigation to be displayed like this:



and when on a tablet device (or on resizing your browser to a corresponding width), we want this:



and finally, we want mobile users to see this:



I know you could achieve this with whatever technique you've used in the past or are familiar with, however, we'll see how painless it is to employ this responsive layout with Flexbox and without dealing with complexities or hacks.

Before we delve into the styling, let's have a look at our html file for the Navigation bar.

Listing 7.1: Base html for the navigation bar

```
<nav class="fnf-nav">
   <h1 class="fnf-nav-brand">Flex &amp Furious</
h1>
        <ul class="fnf-nav-main">
             <li><a href="#">Speed</a></li>
             <li><a href="#">How</a></li>
             <li><a href="#">Why</a></li>
             <li><a href="#">About</a></li>
        </ul>
```

```
</nav>
```

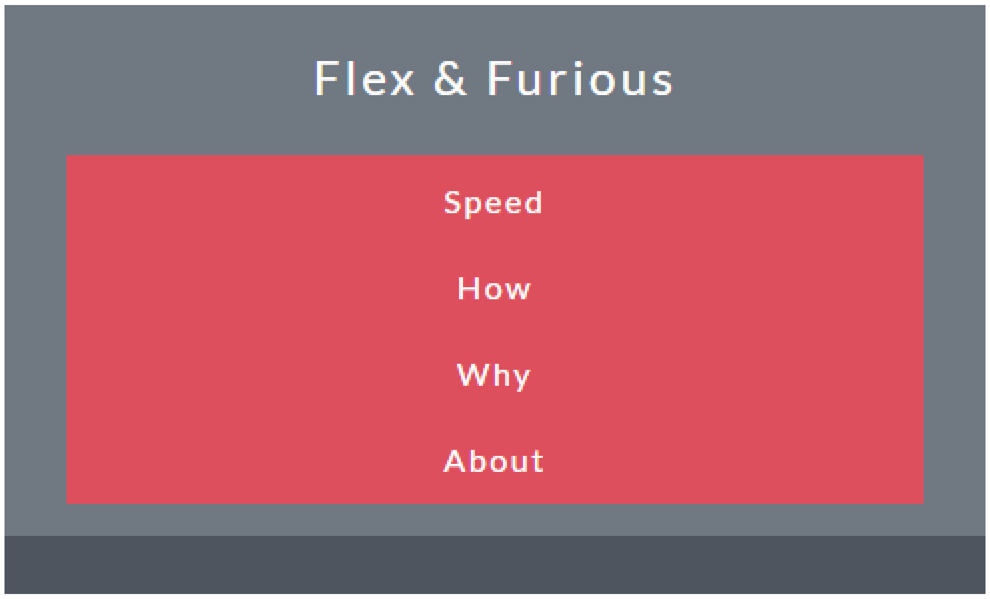
Listing 7.1 above is super basic and nothing magical is happening yet. The entire navigation is wrapped within a `nav` element, with an h1 heading and unordered list which houses the links. The links are dummy links - which would be enough this project. By dummy links, I mean the links lead nowhere with no 'real url' attached. You see the `href="#"` ??

Also note the convention I'm using here. The `nav` element has a class of "`fnf-nav`" where `fnf` stands for Flex & Furious. The class names for the `h1` element and unordered list are easy to wrap your head around, `.fnf-nav-brand` and `.fnf-nav-main`.

Now that we have our html file in place, it's a good time to begin styling the navigation. For now, I'd leave things simple. No Flexbox styles - just some basic initial styling to get us off the ground.

Before we do that let's follow some good practice. I'd be employing a mobile first approach in all the projects in this book. If you are not familiar with what that means, it just means we will be designing for mobile screens first! Users on mobile devices. Weird? Not really. It's a popular convention, one I advice you begin to employ if you don't already do so.

After dealing with how the navigation would be displayed on mobile devices - specifically phones, we would then progressively deal with larger screen sizes - tablets and desktops. So here is what we need to deal with first.



Remember that by default, the h1 heading and the unordered list, ul will stack vertically - with the ul below the h1 element. So what we need to do to achieve the mobile view above is relatively easy. The following listing shows that.

Listing 7.2: Mobile first navigation style

```
.fnf-nav {
   text-align: center;
   padding: 0 2em;
}
```

Align nav element texts to the center. Give a padding left and right of 2em.

```
ul.fnf-nav-main {
   border: 1px solid #da4453;
}

ul.fnf-nav-main li {
   padding: 0.5em 1em;
   border: 1px solid #da4453;
   background-color: #da4453;
}
```

Get all li elements that are children of the ul with a class of .fnf-nav-main and give some padding, border and bg-color.

```
ul.fnf-nav-main li:hover {
   background-color: transparent;
}
```

And *bravo!* We've got the navigation bar good to go. The listing above sets relevant spacing within the `nav` element, adds a reddish border to the unordered list and all list items. The list items which hold the links are then spaced appropriately too with `padding: 0.5em 1em`, and finally, when a user has their cursor over the list items (on hover), the list items take a transparent background. No colors, just plainly see through the lists. `background-color: transparent.`

I know this isn't particularly exciting yet - just hang on. Flexbox coming soon!

There is one more thing I forgot to state. I had styled the base elements earlier. The `body` element, `headers`, `links` etc. Let me show you what I mean since I don't want you getting confused

Listing 7.3 : Base element styles

```
* {
   box-sizing: border-box;
}

body {
   background-color: #656d78;
   margin: 0;
   font-family: Lato, sans-serif;
   font-weight: 300;
   line-height: 1.6;
   letter-spacing: .05em;
   color: #f5f7fa;
}

ul {
   margin: 0;
   padding: 0;
   list-style: none;
}

h1,
h2,
h3,a {
```

Stop all elements from increasing in width or height when extra spacing is added.

Take away the spacing outside and inside the unordered list plus No default styles on the list please.

```
    color: #fff;
    font-weight: 400;
    letter-spacing: .1em;
}


img {
    max-width: 100%;
}

a {
    text-decoration: none;
}

main,
aside {
    padding: 1.5em;
}
```

All image elements should fill up the entire space within their containers - this gives a responsive feel upon resizing

I have tried to explain the rationale behind some of the codes above in the sidebar. That's the way it's been throughout the book. However, if something else confuses you, something I didn't address in the sidebar, here is what to do:

Since you do have the source files, delete and edit the values around whatever you do not understand and see how it affects the page. I'm sure that will answer your questions coupled with a bit of a Google search - we all use Google! Please do a Google search, it sure saves me a lot of beating around the bush.

And here comes the fun part. We may now move on to cater for larger screens. If it's now fuzzy what we are trying to achieve, please see the exercise files or scroll to earlier pages and view the screenshots earlier discussed.

Let's face it, I'll still be here. There's no rush.

Now that we are both clear on what we are trying to achieve, let's write some code!

Media Queries

Ideally, the first intelligent question to ask is, "How do I specifically target different screen sizes?" Since we want the navigation to be displayed differently on varying devices, this becomes an important question - the foundation for doing anything we've hoped to achieve here.

The answer to that is: (drum-rolls please) by using something we call Media Queries. Media Queries are at the heart of responsive web design, and they are really very handy when working on any responsive design project. So what are media queries? Let me walk you through what they are and you'd be able to teach it to someone else in no time.

If you've ever written a line of code in any programming language then you're probably familiar with the concept of a conditional statement. Something like the if statement.

In Javascript, it looks like this

```
if (myName == "Ohans Emmanuel") {
   say ("You're awesome dude!");
}
```

We are communicating via our program and saying if a certain myName variable is equal to the string of characters "Ohans Emmanuel", which by the way happens to be my name, then say "You're awesome dude!"

If you're lost and don't understand any of that, this is what I'm trying to do with those lines of code: carry out a set of instructions only if the expression in the braces in this case, (myName == "Ohans Emmanuel") is true.
Got that? It isn't a difficult concept, I think reading this paragraph again may help if you still don't get it.

So, why do I bring up all of this?

We need to achieve the same effect with css. We want to say something like: “if the client’s screen size is 1000px, then do this”, or we can get more specific and say, “if the client is on a handheld device, then do this”, or, “if it’s a device with a width between 200px and 500px, do this!”

Okay, that’s not how we actually write it css, but that’s how the media query thing works. So how do we write this in css ?

The most popular form in which media queries are used is something called the @media rule and it looks like this:

```
  @media handheld and (max-width: 300px) {
     /*write your css in this code block*/
  }
```

Looking at it, you can almost guess what that does. For an handheld device with a maximum width of 300px ... do this and that (to be written in the code block - where the “write style here” comment exists). Any styles within the code block will only apply to devices that match the expression `handheld and (max-width: 300px)`

So we should just go ahead and use this, right? well, no we won’t. For some reasons, the Android and iOS aren’t matched by that keyword “`handheld`” in the `@media` rule written above. We need something more generic, so we’d use the keyword “`screen`”. For instance:

```
@media screen and (min-width: 500px) {
     /*write your css in this code block*/
}
```

Notice how there is an “and” specified after the “`screen`” keyword? You shouldn’t forget to have this written in your `@media` rules! I forget to write that sometimes.

That been said, we are almost set to finish up the responsive navigation bar. One more concept to be understood.

Breakpoints

Hold up 5 tablets or laptops from different manufacturers and you’d easily notice they most likely all have varying screen sizes. I said we’ll be using media queries, and we need some specification we can bank on to target a large share of all tablets and laptops. This spec is what we call a breakpoint.

We couldn’t possibly write @media rules for all screen sizes the world has seen, line after line, targeting just each screen size. That would be a waste of time - not something a modern developer particularly loves. So we do so by targeting a group of devices, based on their screen sizes.

Here is what I mean. If every tablet device in my community has a minimum width of 750px, then having 750px as my breakpoint for tablet devices, I may target all tablet devices in my community with this:

```
@media screen and (min-width: 750px) {
        /*do something*/
}
```

Pretty easy, and a whole community has been taken care of. However, my community doesn’t house everyone. So for the purposes of working through this project we’ll use the following breakpoints.

Tablet devices: at least 769px,
Laptops and bigger screens: above 1025px.

This means we assume that all tablet devices have a minimum width of 769px while laptops and other bigger screen devices are at least 1025px wide.

These aren't "my choices", but rather the popular standards among the responsive web design community and you may employ them in whatever projects you work on if you don't already do so.

Flexbox and Media Queries

Here is where we actually get to do all the cool stuffs I promised. So let's start with how the navigation will be displayed on tablet devices. Thanks to Flexbox and media queries, it's a breeze.

Thanks! and I’m super excited to share this book with you. I’m making a lot of incremental changes to this book everyday. So...

Watch the Github Repository.

- Entire book may be read on Github for free
- Contains extra chapters, more code snippets etc.