Telerik®
A PROGRESS COMPANY

THE ANATOMY OF RESPONSIVE ASP.NET APPS

CONTENTS

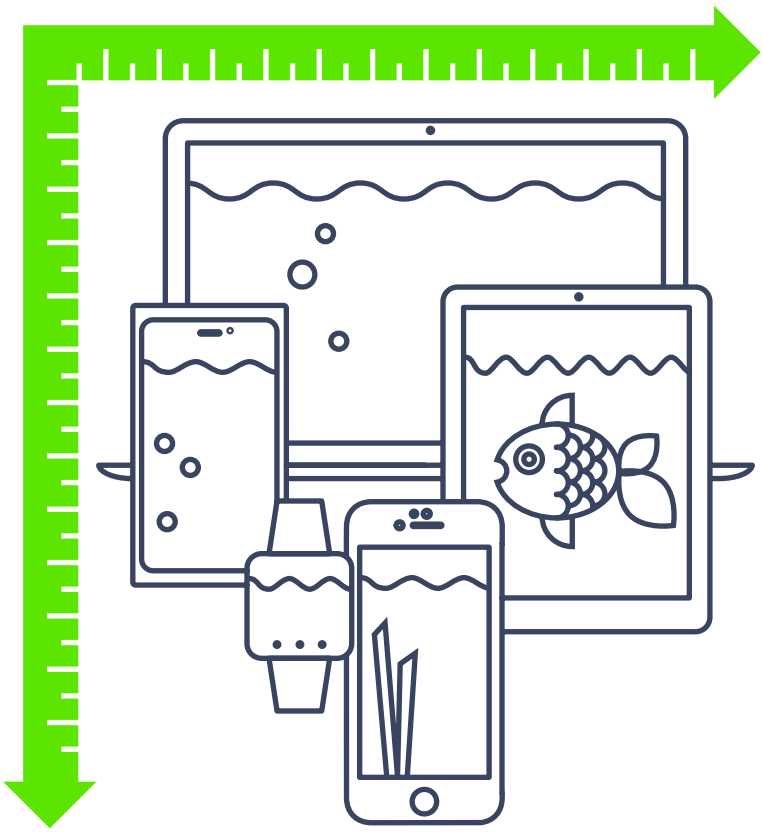
OVERVIEW

There are many reasons why responsive web design has gone from being called a trend to being synonymous with web best practices. It's a fast and cost-effective way to serve a tailored experience to desktop and mobile web users. You can use your existing skill set, have one code base, one set of URLs and one design language.

If you're an ASP.NET Web Forms or MVC developer, maybe you have already jumped in on a responsive design project and would like to learn more. If not, responsive design will probably be a requirement in the near future. This whitepaper will walk you through the must-know responsive web practices to help you succeed in building apps for any screen size. You'll learn how to:

- Leverage the basic building blocks of responsive web design—grid systems, media queries and flexible content—to create responsive layouts
- Choose a responsive web design framework (Bootstrap, Zurb Foundation, Telerik Page Layout)
- Use advanced Bootstrap features, such as fluid containers, offsets, push/pull and more, to achieve even the most complex responsive web scenarios

THE DNA OF RESPONSIVE WEB DESIGN



Responsive design is defined as fluid grids, flexible content and media queries. In general, responsive web design (RWD) means the layout of the project needs to adapt to the browser's viewport size. The content of the application should make efficient use of the available space on the screen, generally by being able to shrink or grow with varying container size. Let's take a look at what techniques can be used to achieve RWD.

MEDIA QUERIES, THE ESSENTIAL CORNERSTONE OF RWD

Media queries are a feature of CSS3. Without it responsive design would not be possible. A media query allows you to write expressions capable of detecting media related features. Media queries can contain a media type and/or media feature expressions. When all parts of the media query return true, the styles within the media query block will be applied to the style sheet. A media query is similar to an "if" statement of most major programming languages, except that it is designed specifically for media detection.

Let's look at an example and break down the various parts of a media query.

```
/* For screens > 768px */

/*  If the device has a screen
    And the device width is greater than or equal to 768px
    then color property = red */

@media only screen and (min-width: 768px) {
    .on-large { color: red; }
}
```

There are a few techniques for using media queries to transform layouts. These techniques are generally referred to as mobile first or desktop first RWD. In either scenario, a base layout is created and then device-specific CSS styles are conditionally added to append or override the existing style.

Let's look at an example of mobile first responsive design. In this example we'll define a base font size and then increase the font size for large screens.

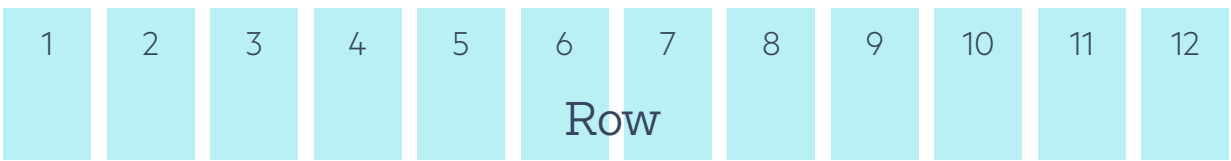
```
/*/*/
/* Base style for small devices small and up */

.hero-text { font-size: 22px; }

/* For large and up */
@media only screen and (min-width: 768px) {
    .hero-text { font-size: 48px; }
}
```

Media queries are essential to developing responsive apps and websites. Popular frameworks like Bootstrap, Foundation and Telerik RadPageLayout make extensive use of media queries to create their powerful grid systems. While each grid system varies slightly from one framework to another, there are many common components and techniques used among them. Let's peer at the source code of a few popular frameworks to understand how they work.

Grids, rows, columns and breakpoints

Fluid grid systems are the tool of choice for creating responsive layouts. Grids consist of multiple components working together to compose an infinite number of designs. For the most part, grids are made up of rows which are divided into columns. The columns are not physical columns, but a unit of measure for an element's width. Rows are typically divided into 12 columns because 12 is a number that is easily divided equally into whole parts. Columns also abstract away the math for dividing up the layout into percentages.

Column classes abstract away lengthy percentage values like .col-xs-5 { width: 41.66666667%; }

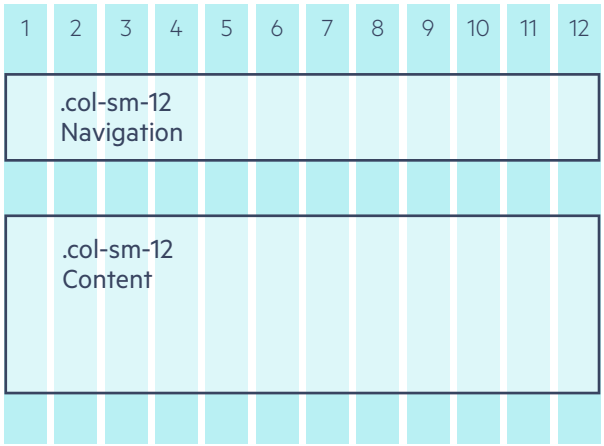


Grid systems support multiple layouts by specifying the device's size and width as part of a column. The syntax may vary between frameworks but the general concept is the same. Each device size has a corresponding media query and style properties that create the desired layout effect. These media queries are referred to as break points. Bootstrap identifies its break points as xs (extra small), sm (small), md (med), lg (large).

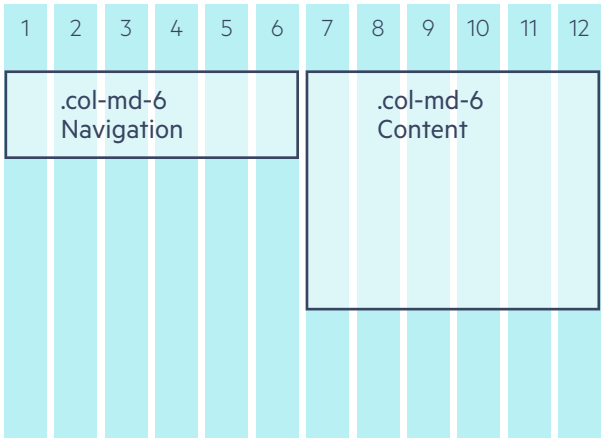
```
<!-- Bootstrap syntax col-[device size: xs, sm, md, lg]-[width: 1-12] -->
<div class="col-sm-12"> ... </div>
```

By combining multiple column declarations the layouts can take on different widths or stack vertically when an element occupies all 12 columns.

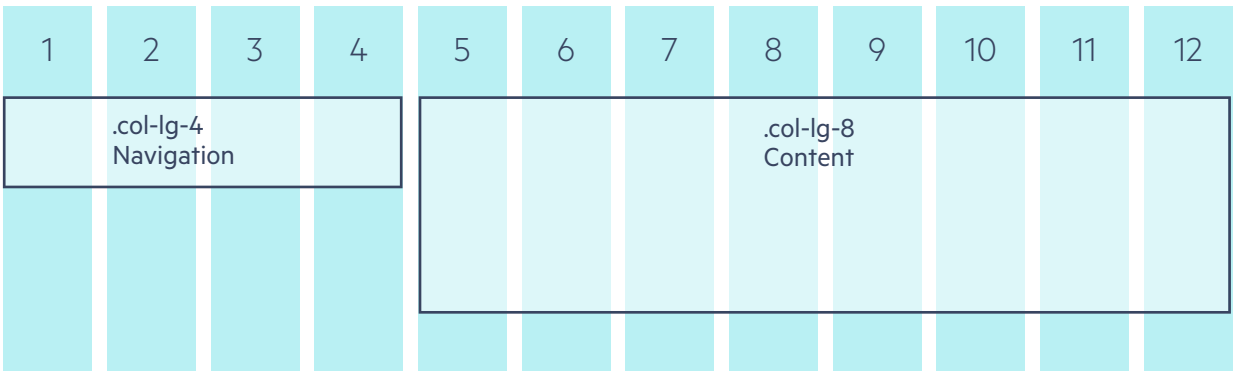
```
<div class="col-sm-12 col-md-6 col-lg-4">
    <!-- some navigation -->
</div>
<div class="col-sm-12 col-md-6 col-lg-8">
    <!-- some content -->
</div>
```



Small devices, tablets (≥768 px)



Medium devices, tablets (≥768 px)



Large devices, Desctops (≥1200px)

CSS media queries make this possible, just as we saw with the mobile first example. Since the size specific styles are contained within a media query, they are only applied by the browser when the media query expression is satisfied. Additionally, because of the CSS source order, the styles defined later in the style sheet will take priority over those above.

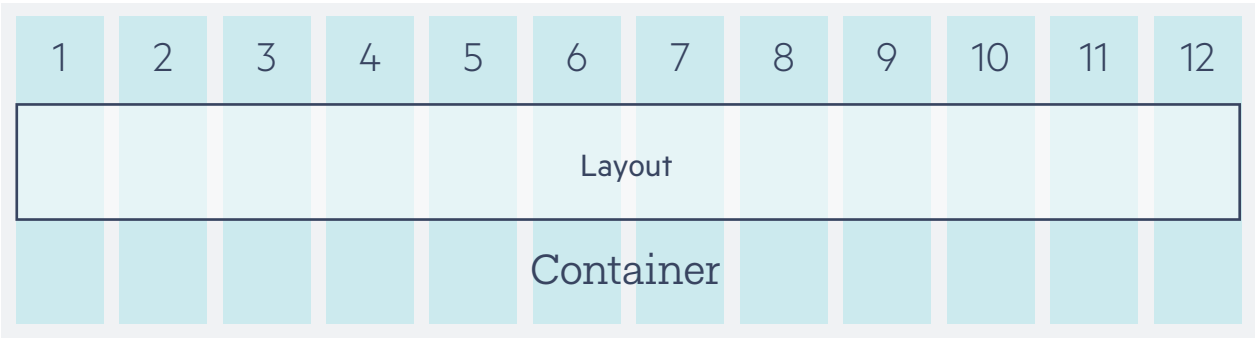
```
/* bootstrap source code */
@media (min-width: 768px) {
    .col-sm-12 { width: 100% }
}

@media (min-width: 992px) {
    .col-md-6 { width: 50% };
}

@media (min-width: 1200px) {
    .col-lg-8 { width: 66.66666666666666%; }
```

1200 px and up

```
@media (min-width: 768px) {
  .col-sm-12 { width: 100% };
}

@media (min-width: 992px) {
  .col-md-6 { width: 50% };
}

@media (min-width: 1200px) {
  .col-lg-8 { width: 66.66%; }
}
```

Computed styles

Some grids have the concept of a container. The container element is the outermost element of the layout. Its purpose is to create white space between the layout and the edge of the browser window.



If you notice your content is directly touching the edge of the browser or device's display, make sure you didn't forget to include a container element.

```
<div class="container">
    <!-- begin rows here -->
</div>
```

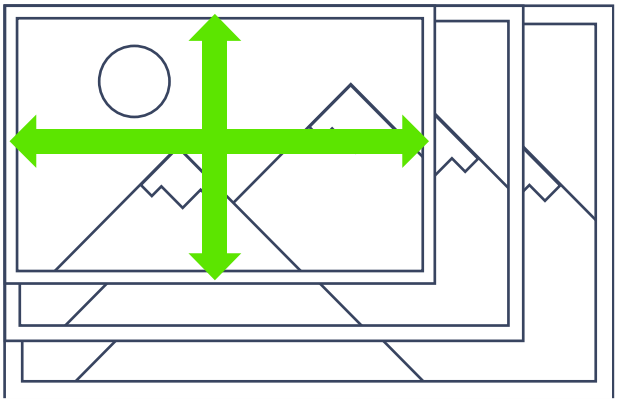
Let's dig into Bootstrap's source code and see how it works.

```
.container {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
@media (min-width: 768px) {
  .container {
    width: 750px;
  }
}
@media (min-width: 992px) {
  .container {
    width: 970px;
  }
}
@media (min-width: 1200px) {
  .container {
    width: 1170px;
  }
}
```

As you can see, a base padding of 15px is set for all browsers. As larger screens are detected, the container element's width is increased, making the layout expand to accommodate the available space. If more or less white space is desired around your layout, the width values can be changed to fit your design needs.

Working with Content

A responsive app or site is only as flexible as its content. When building responsive it is important to make sure images, video and other media make efficient use of the available space on the screen. Additionally, the content should be easily readable with appropriate font and image sizes.



A common problem occurs when a small screen device accesses a website where images don't fit appropriately. Images may be too large for the layout, extending past the view port, or may overlap other elements. The Foundation framework addresses the problem by setting the max-width of all images to 100%. Using the max-width property ensures that an image occupies 100% of its containers width but never exceeds its natural width.

When using max-width, if an image's natural width is 500px, it will only grow to 500px wide. However, when the CSS width property is used, the image will grow indefinitely to fill the parent container.

```
/* Foundation Source Code */
img {
    height: auto;
    max-width: 100%;
}
```

Bootstrap employs a similar technique using the class .img-responsive which must be applied individually to images that need this functionality.

```
<img class="img-responsive".../>
```

More often than not images will need to be flexible. In my opinion it is easier to set the value for all images as Foundation does, and then add a class to disable the functionality when needed.

```
/* disable flexible images */
img.no-resize { max-width: none; }
```

Content is king and can make or break a layout. Media elements like embed, video, iframe and object need a little help to guarantee they behave correctly on all screen sizes. Responsive frameworks have special classes just for these items.

Classes like Boostrap's.embed-responsive-item and Foundation's .flex-video guide browsers make sure embedded media dimensions are based on the width of their containing block by creating an intrinsic ratio that will properly scale on any device.

```
<!-- Bootstrap -->
<!-- 16:9 aspect ratio -->
<div class="embed-responsive embed-responsive-16by9">
  <iframe class="embed-responsive-item" src="..."></iframe>
</div>

<!-- 4:3 aspect ratio -->
<div class="embed-responsive embed-responsive-4by3">
  <iframe class="embed-responsive-item" src="..."></iframe>
</div>

<!-- Foundation -->
<!-- 16:9 aspect ratio -->
<div class="flex-video widescreen">
    <iframe src="..."></iframe>
</div>

<!-- 4:3 aspect ratio -->
<div class="flex-video">
    <iframe src="..."></iframe>
</div>
```

Single pieces of content like images and video are easily made responsive thanks to these helpful classes. Complex items like carousels, galleries, charts, graphs and more require responsive user interfaces.

Responsive UI

RWD frameworks are a great starting point and include UI components to get your project started. The components usually contain the basic features needed to operate and build prototypes. Feature complete applications require robust UI features. It is important to choose a UI suite that supports responsiveness out-of-the-box. Telerik® Kendo UI®, UI for ASP.NET MVC and UI for ASP.NET AJAX have responsive charts and controls that work seamlessly with any RWD framework you're using.

Summary

RWD is no longer a buzzword or trend, it's a design pattern that is a necessity for modern web applications. Through fluid grids, flexible content and media queries, apps can be created for any screen size. CSS media queries are a simple but powerful feature that makes responsive design possible. By learning how media queries are used in popular frameworks, projects can be built using common patterns and practices for responsive web design.

SELECTING A RESPONSIVE FRAMEWORK FOR YOUR NEXT .NET APP

There are a multitude of responsive web design frameworks that aim to solve the problem of creating one UI code base and delivering to any screen size. Responsive design frameworks like Bootstrap, Foundation and Telerik RadPageLayout are popular options, but choosing a framework isn't all about popularity, it's about using the right tool for the job. These frameworks have evolved over time and have strengths and weakness that fit different development needs. In this section we'll consider how project requirements, team experience and IE compatibility factor into deciding which framework is right for your next .NET app.

Bootstrap

Bootstrap is arguably the most popular and widely used RWD framework on the web. Its popularity may have stemmed from the fact that it was initially released as "Twitter Bootstrap." The Twitter name gave prominence to Bootstrap, which was perfect for developers looking for best practices and guidance. Since version 3.0, Bootstrap no longer carries the Twitter name, but remains the dominant RWD framework in the industry.

Bootstrap offers best in class features like a solid responsive grid, mobile first design, CSS helper classes, adaptive JavaScript components and much more. The grid is, by default, a standard 12 column grid with a simple syntax for creating layouts that support multiple screen sizes.

```
<div class="container">
  <div class="row">
    <div class="col-[size]-[width]">
      <!-- content -->
    </div>
  </div>
</div>
```

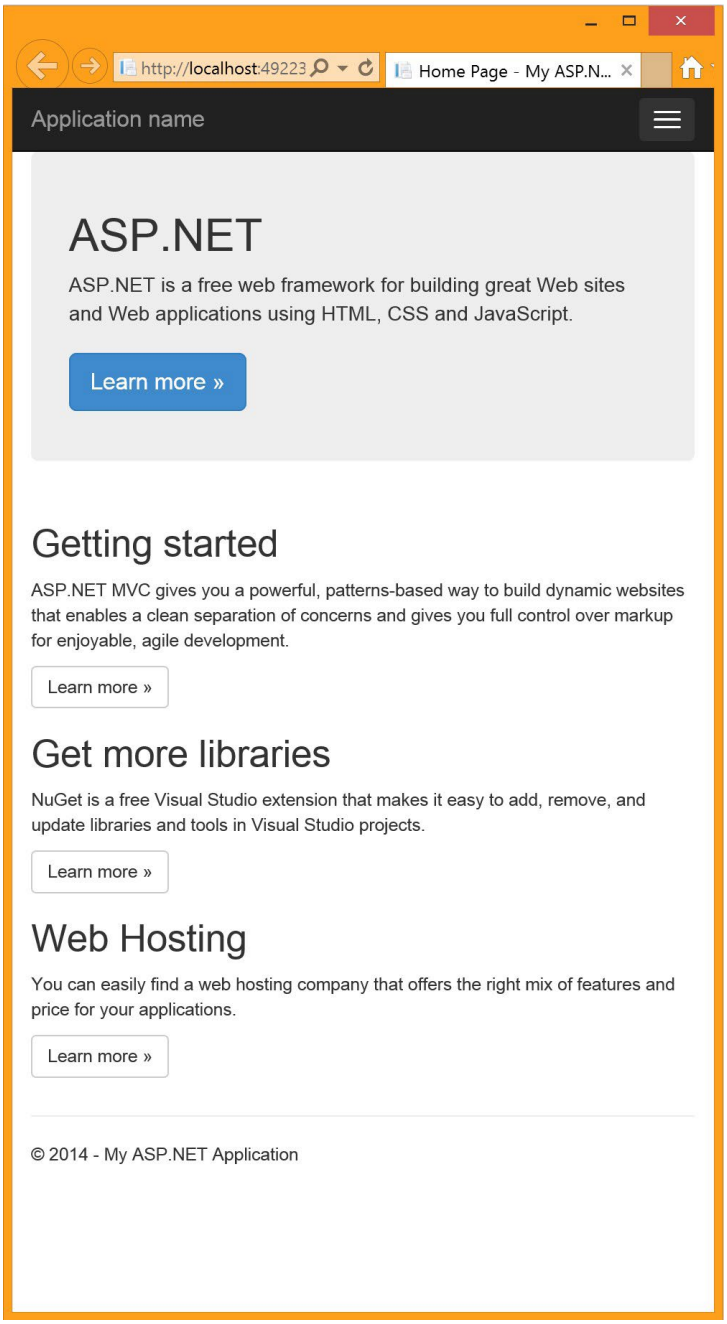
Additionally, the grid supports nested rows and advanced features like push and pull classes.

Bootstrap: Requirements & Compatibility

Developing RWD for the enterprise can carry strict requirements for supporting older IE browsers. While Bootstrap fully supports IE 10 and greater, some minor features are not compatible with IE 9. Support for IE 8 is much more difficult, requiring the addition of Respond.js, which comes with its own set of caveats.

Bootstrap: Key Considerations

Bootstrap's market share has helped the framework earn its position as the default UI framework for new ASP.NET MVC projects; i.e. if you use file > new project in Visual Studio, you are already using Bootstrap.



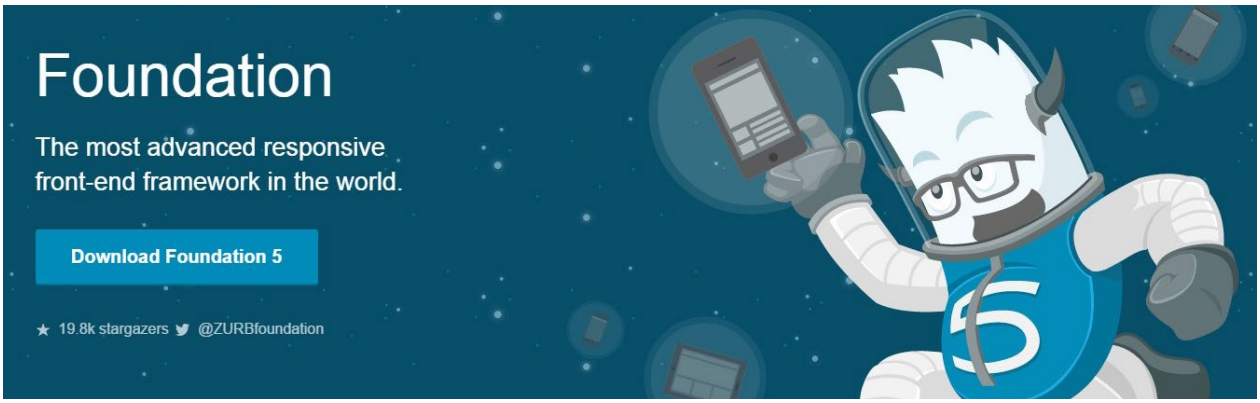
Documentation is another factor that makes Bootstrap a great choice. If your team is new to RWD, finding example code, blog posts and forum help is readily available.

For developers looking for advanced features, Bootstrap has preprocessor support for both LESS and Sass. The project source is written using LESS, which is important to consider if you plan on working with the source or contributing back to the project.

If you prefer to use Sass source rather than LESS, there is an official port called Bootstrap for Sass.

Foundation

Foundation was born from the same base code as Twitter Bootstrap prior to either frameworks public release. Foundation is an open source project created and maintained by Zurb, a product design and interaction company. Foundation was first to market with a mobile first design perspective with the release of Foundation 4.x.



Like other RWD frameworks, Foundation has a flexible grid and is packed with responsive and adaptive components. The grid arguably has the most legible syntax of RWD grids with its use of meaningful class names.

```
<div class="row">
  <div class="small-2 large-4 columns">...</div>
  <div class="small-4 large-4 columns">...</div>
  <div class="small-6 large-4 columns">...</div>
</div>
```

In addition to the standard grid system, Foundation supports a block-grid. The block-grid is excellent for evenly splitting contents of a list within the grid. This is useful with data-driven scenarios when a large series is rendered and the quantity of elements is not known until runtime.

```
<ul class="small-block-grid-3">
  <li><!-- Your content goes here --></li>
  <li><!-- Your content goes here --></li>
  <li><!-- Your content goes here --></li>
  <li><!-- Your content goes here -->/li>
  <li><!-- Your content goes here --></li>
</ul>
```

Foundation: Requirements & Compatibility

While Foundation is not the de facto standard for new .NET projects, it is available on NuGet. The installation process only requires adding the package to the project and choosing to overwrite the default layout and index files. Additionally, a package for users looking for Sass support is also available.

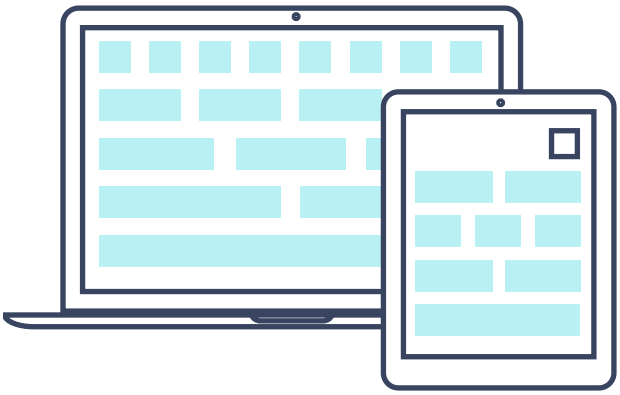
Developing for legacy browsers can be a tough requirement to fill, especially when trying to use modern techniques like RWD. If your requirements specify versions of IE 8 or below, then consider using prior versions of Foundation. Foundation 3, which supports IE 8 and Version 2 which supports IE 7, are also available on NuGet.

Foundation: Key Considerations

Foundation is packed with advanced features suitable for teams with experienced front end developers, particularly those who use Sass. Foundation includes an extensive system of Sass mixins which can be extremely useful in a skilled CSS developer's hands. The Sass mixin system is not only capable of customizing the look of the application, but can also be used to create semantic CSS selectors and domain specific languages.

Some of Foundation's adaptive components are unique to the framework as well. This includes Interchange, a utility for dynamically loading content based on screen size, and Joyride, a system for creating product feature tours.

Telerik RadPageLayout for ASP.NET Web Forms



The Telerik RadPageLayout fills a need for ASP.NET web forms developers. Unlike other layout frameworks such as Bootstrap and Zurb, you can configure RadPageLayout completely on the server side, letting RadPageLayout do the HTML generation for you. The RadPageLayout grid follows standard RWD grid conventions like using a row and column system to create layouts. Markup for the RadPageLayout control can be written manually or can be created via the property editor window.

```
<telerik:RadPageLayout ID="RadPageLayout2" runat="server" GridType="Fluid" >
 <Rows>
   <telerik:LayoutRow>
     <Columns>
       <telerik:LayoutColumn Span="8">
           Main content here
       </telerik:LayoutColumn>
       <telerik:CompositeLayoutColumn Span="4">
         <Rows>
           <telerik:LayoutRow>
             <Content>additional content 1</Content>
           </telerik:LayoutRow>
           <telerik:LayoutRow>
             <Content>additional content 2</Content>
           </telerik:LayoutRow>
         </Rows>
       </telerik:CompositeLayoutColumn>
     </Columns>
   </telerik:LayoutRow>
 </Rows>
</telerik:RadPageLayout>
```

Telerik RadPageLayout: Requirements & Compatibility

Given the longevity of enterprise applications, RadPageLayout is an excellent choice for adding modern web capabilities like RWD to your application. Because RadPageLayout is part of Telerik UI for ASP.NET AJAX, it can meet the requirements where dedicated technical support is desired.
IE 9 is the minimum requirement for implementing RadPageLayout. In addition, RadPageLayout works on all non-IE browsers and mobile devices.

RadPageLayout: Key Considerations

Software development teams who are experienced in writing ASP.NET web forms applications will be comfortable with the RadPageLayout. The server side markup style of development makes RadPageLayout a natural fit for developers looking to bring RWD to their application without the need to learn advanced CSS or write their own HTML. Because RadPageLayout is a server side control, all of the properties are displayed in the property window, further reducing the learning curve.

In addition to the RadPageLayout, Telerik UI for ASP.NET AJAX also features the Device Detection Framework. The Device Detection Framework provides server side access to the client's screen size information. Utilizing both RadPageLayout with the Device Detection Framework you can optimize your app using a technique called RESS, or Responsive Design with Server Side components. RESS allows you to choose not to display the heavy parts of your site or app on mobile devices, reducing network usage and optimizing the user experience.

Choosing What Works for You

When choosing a responsive framework, instead of looking at specific features look for properties that fit naturally with your teams or your personal style of development. Each framework has its own specialty, learning curve and user base.

Framework	Strengths	Team Dynamics
Bootstrap	Popular, Well Documented	Developer Friendly
Foundation	Highly Customizable	Front End Developer / Designer Friendly
RadPageLayout	Server Side Syntax, Ease of Use	ASP.NET Web Forms & AJAX Developers

Let's Review

Bootstrap is ideal for developers who need solid documentation and have lighter requirements in regards to feature customization. While Foundation offers less in terms of market share, it makes up for it in advanced customization features. Evening out the pack is RadPageLayout, which is optimal for teams who specialize in ASP.NET web forms development. Each framework has a lot to offer and the features can be overwhelming, but rest assured that all choices will lead to a successful project.

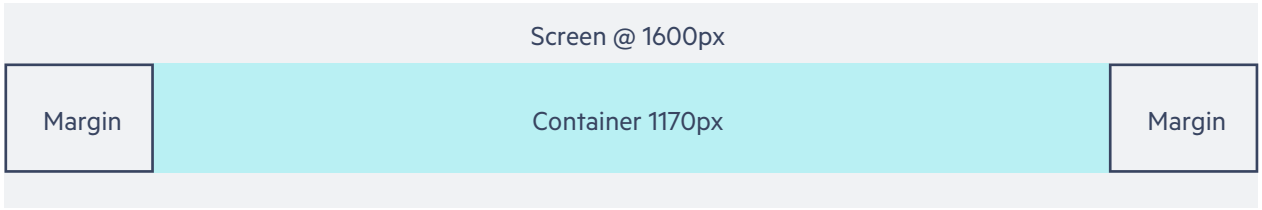
PERFECTING YOUR APP WITH ADVANCED BOOTSTRAP FEATURES

Responsive design changes the way we think about displaying content on our web pages. As the complexity of the user interface increases, so does the difficulty of building the interface. Because these complex interfaces require creative solutions, Bootstrap includes advanced features that can make short work out of even the toughest scenario.

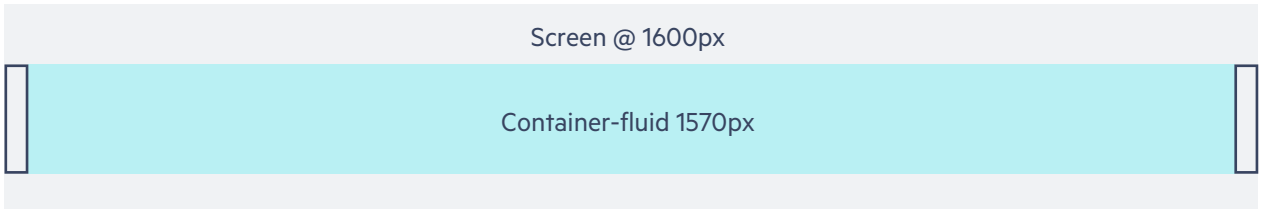
It's time to take your Bootstrap skills to the next level. Let's learn about powerful features like fluid containers, visibility, offsets, push/pull and more. By learning these features you'll be able to recognize when they are needed and know how to implement them. In addition, although the syntax may vary, these features are similarly available in other frameworks.

Fluid Containers

If you have used Bootstrap, you may already know about the container class. The container is a class which is applied to the outermost grid element. The whitespace can vary depending on device size, resulting in a maximum layout width at each breakpoint. Most of the time the default values are good enough for a given project, however if your project requires more customization, you may need a fluid container.



Bootstrap has a second container class, container-fluid. The container-fluid class works very differently than the basic container. Instead of having break points that determine the amount of whitespace to use, container-fluid creates a layout that fills the entire width of the screen.

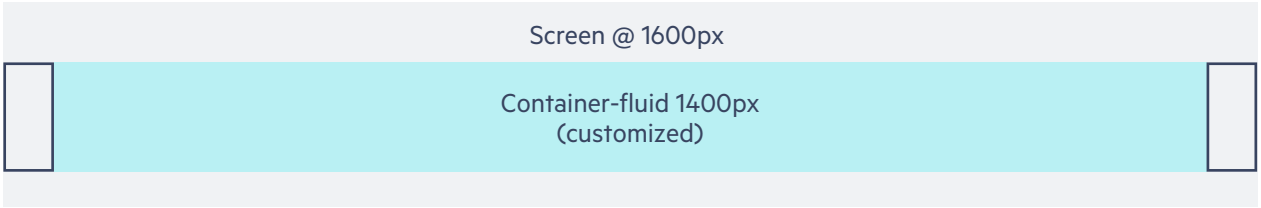


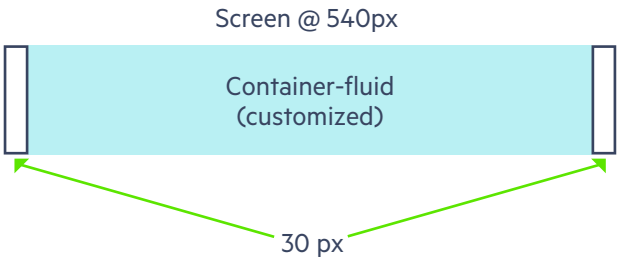
A fluid container is useful for full screen applications, but it also can be used to create a custom page width for your layout. Having a custom page width can be especially useful when you want to set a maximum value that your layout will grow to, but not grow past. Let's see how this can be done in just a few easy steps.

First, start by changing an existing container to container-fluid.

Next, we'll specify a max-width for container-fluid. The value will be the largest size the layout will expand to. Once the layout fills this space it will stop growing, no matter how large the screen is. The CSS should be placed so that it overrides any default values from bootstrap.css.

```
.container-fluid {
    max-width: 1400px; // Custom max width for this layout
}
```





Finally, we can further customize the container by adding whitespace. By adding additional padding to the container, smaller screens will have a noticeable buffer between the edge of the screen and the application elements.

```
.container-fluid {
    max-width: 1400px;
    padding: 30px; // Whitespace or
outside gutter
}
```

The basic container is a good starting point for most apps, however using the fluid container puts you in control of the page size. Build the application to your requirements and user needs instead of conforming to the framework's default values.

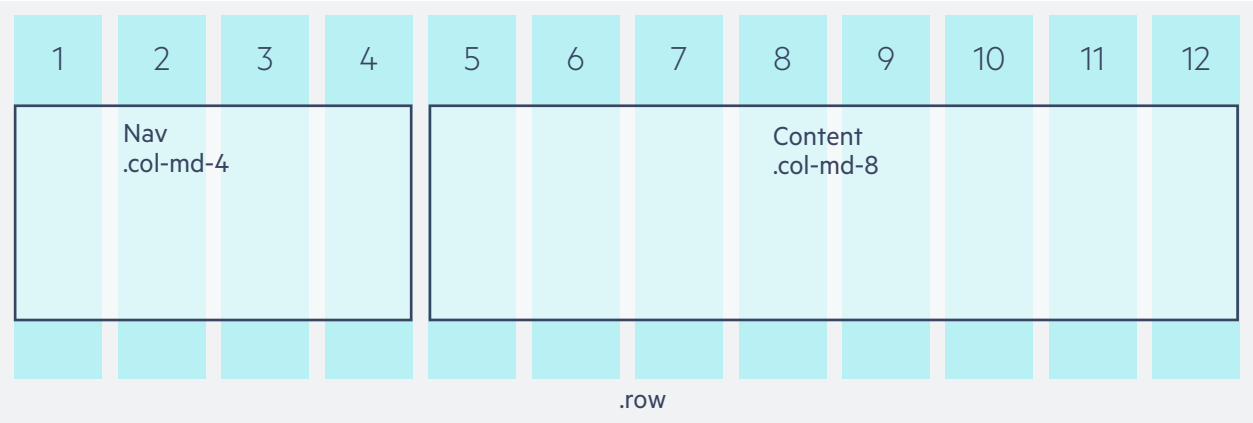
Nested Rows

Rows and columns are the fundamental building blocks of a Bootstrap grid. Its common practice to divide up a layout into pieces like navigation and content, but the grid is capable of so much more. Rows in Bootstrap can be infinitely nested, which means we have limitless control over how the grid is divided up. Once outer elements like navigation and content are defined, inner elements can be created that contain UIs that have their own grids.

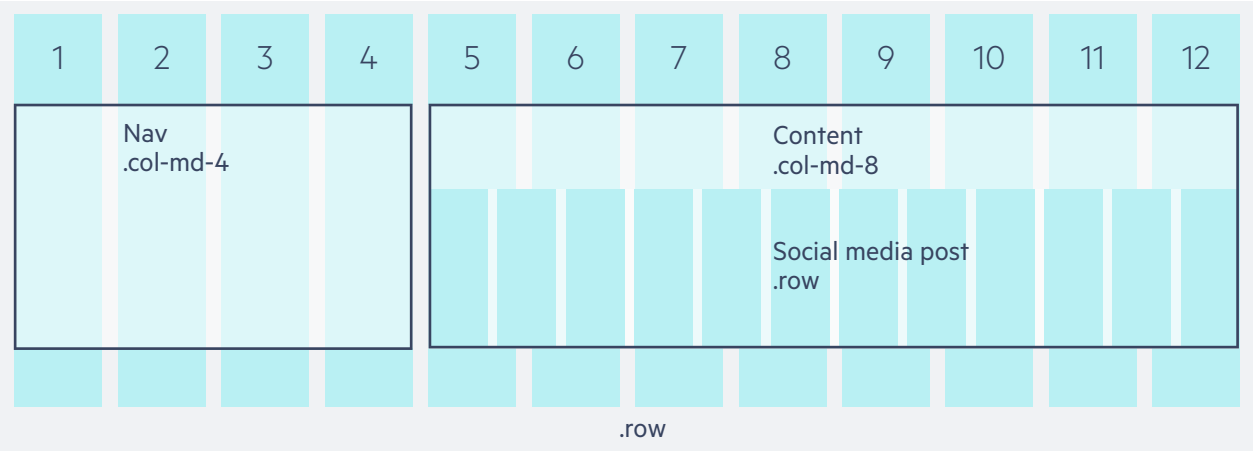
Each column can contain a row which can then be divided up again into 12 units, these units are relative to the container.

Let's look at building a social media UI as an example of nested grids. We start with the outer most layout elements. These layout elements will contain a navigation and content section.

```
<div class="row">
    <nav class="col-md-4">
        <!-- Navigation -->
    </nav>
    <section class="col-md-8">
        <!-- Content -->
    </section>
</div>
```



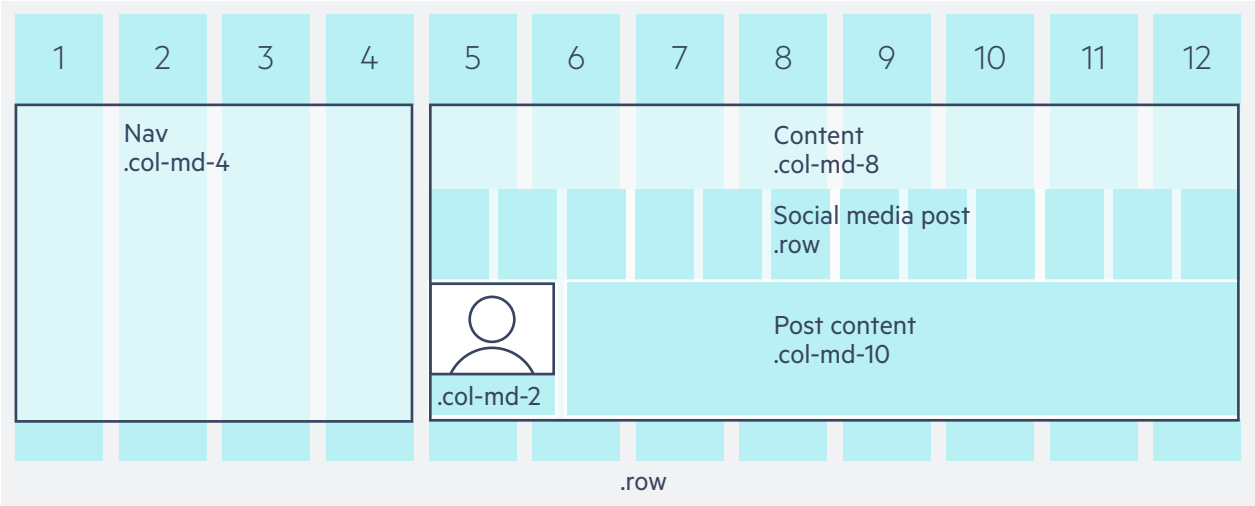
Next, we add a new .row to the content section. The new row will contain the UI for social media posts.



```
<div class="row">
    <nav class="col-md-4">
        <!-- Navigation -->
    </nav>
    <section class="col-md-8">
        <!-- Content -->
        <div class="row">
            <!-- Social Media Posts -->
        </div>
    </section>
</div>
```

The new row is divisible into 12 more columns. This row will be a social media post, so we'll make space for an avatar that's two columns wide and the post content will use the remaining 10 columns.

```
<div class="row">
    <nav class="col-md-4">
        <!-- Navigation -->
    </nav>
    <section class="col-md-8">
        <!-- Content -->
        <div class="row">
            <!-- Social Media Post -->
            <div class="col-md-2">
                <!-- Avatar -->
            </div>
            <div class="col-md-10">
                <!-- Post Content -->
            </div>
        </div>
    </section>
</div>
```



Offsets

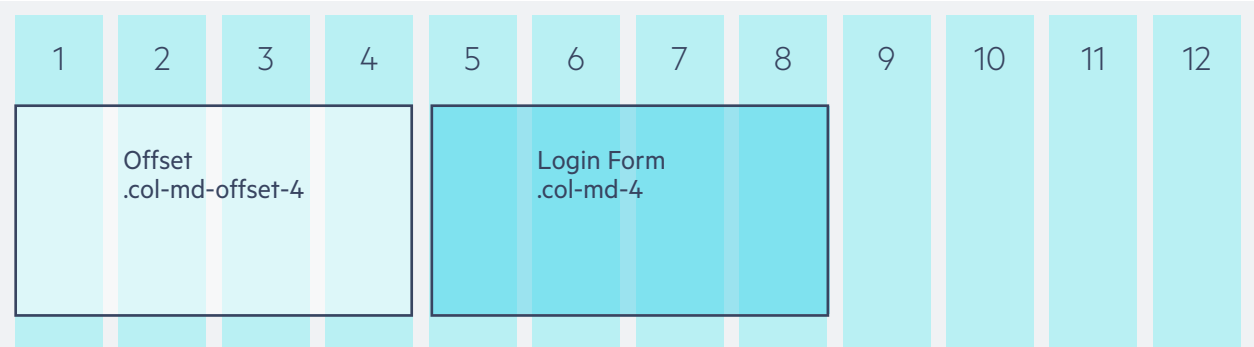
Sometimes UIs need a little help adapting to different screen sizes. Offsets are a great tool when elements need to create additional whitespace or horizontal positioning. Offsets can be used to horizontally center an element without creating unnecessary columns to achieve the effect. Offsets are created by adding the offset class .col-[screen size]-offset-[width] to an element.

In this example, we'll use offsets to center a login box on the screen. The login panel will be four columns wide; this means we will need four columns of white space on either side. First instinct may be to use three elements, each four columns wide and leave the outer two empty.

```
<div class="row">
  <div class="col-md-4">
    <!-- empty -->
  </div>
  <div class="col-md-4">
    <!-- login form -->
  </div>
  <div class="col-md-4">
    <!-- empty -->
  </div>
</div>
```

This approach would work, however the additional markup will increase page size and create needless clutter when editing the page. Instead, let's use the offset class to achieve the same effect using less markup.

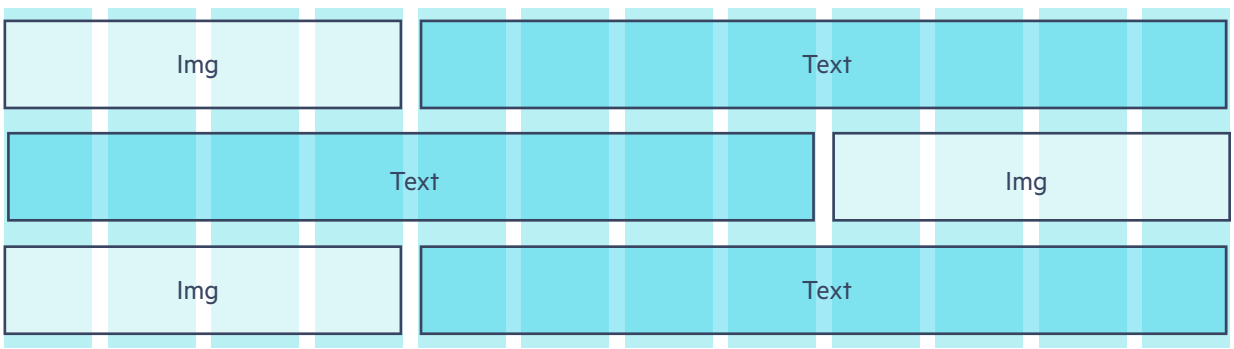
```
<div class="row">
  <div class="col-md-4 col-md-offset-4">
    <!-- login form -->
  </div>
</ddiv>
```



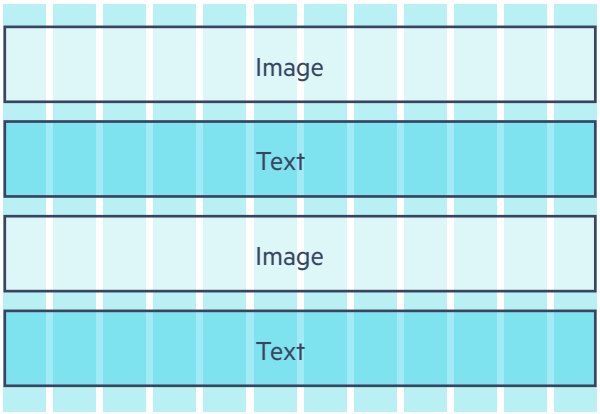
Push/pull

Content that uses a mix of text and media can be difficult to manage for different screen sizes. Creating one layout that can adapt appropriately can mean displaying elements on the screen in a different order than they appear in the markup. When you encounter this problem, the first instinct might be to reach for a JavaScript solution, however the issue can be resolved with a few simple CSS classes.

Let's use an alternating split view layout as an example. In this example, the large layout will alternate between image | text then text | image and so on.

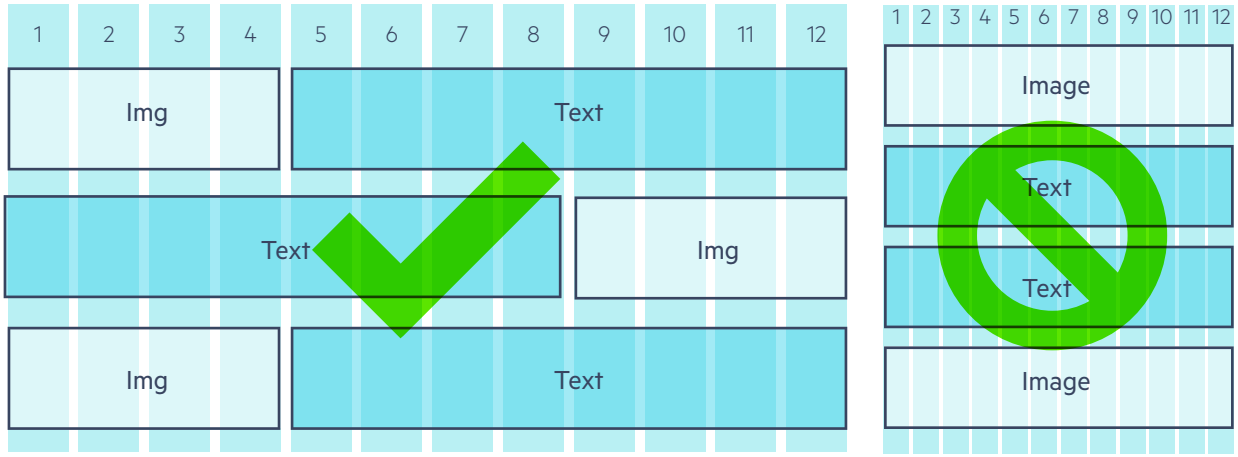


The layout for small screens should also alternate between image and text, except it should do so vertically.



If the markup is created with a desktop-first approach, then alternating the layout in the markup might seem like the right idea. However, the order in which the elements are defined in the markup will change both the large and small layouts, resulting in a large layout which behaves correctly, but a small layout that does not.

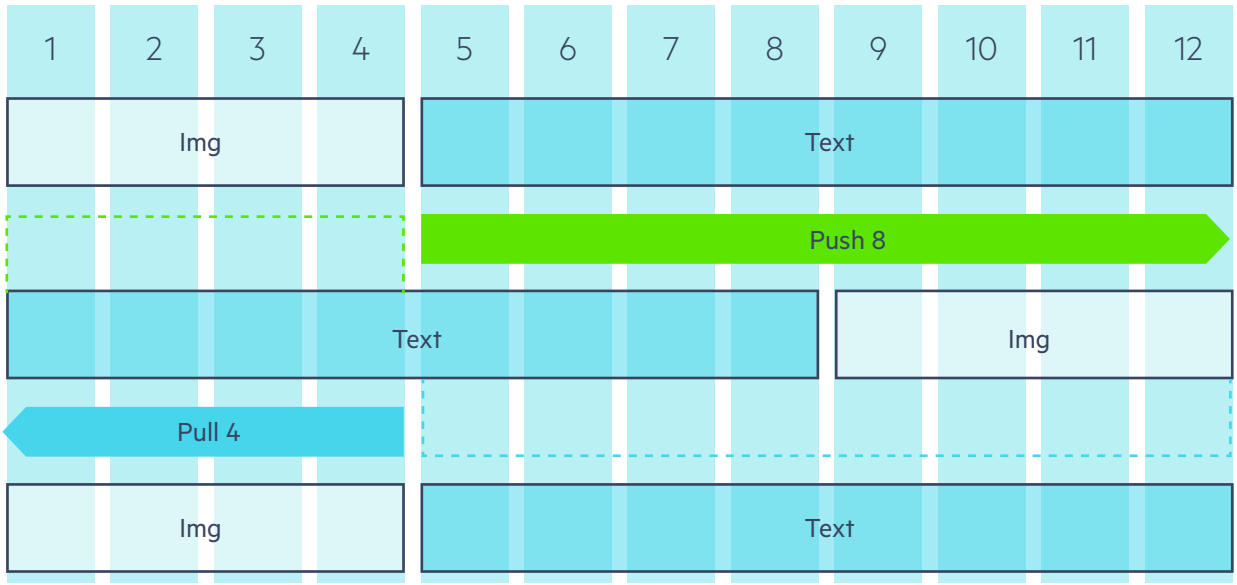
```
<div class="row">
    <div class="col-lg-4">
        <!-- img -->
    </div>
    <div class="col-lg-8">
        <!-- text -->
    </div>
</div>

<div class="row">
    <div class="col-lg-8">
        <!-- text -->
    </div>
    <div class="col-lg-4">
        <!-- img -->
    </div>
</div>
```



Instead of changing the source code order, build the layout starting mobile first. Next, use the push/pull classes to move the elements horizontally around the grid. This is done by pushing the left elements to the right, and pulling the right elements to the left, effectively swapping their positions.

```
<div class="row">
    <div class="col-lg-4">
        <!-- img -->
    </div>
    <div class="col-lg-8">
        <!-- text -->
    </div>
</div>

<div class="row">
    <div class="col-lg-4 col-lg-push-8">
        <!-- img -->
    </div>
    <div class="col-lg-8 col-lg-pull-4">
        <!-- text -->
    </div>
</div>
```

Adding col-lg-push-8 and col-lg-pull-4 to the elements causes them to swap positions on a large screen.



Push/pull classes can be used with multiple columns in a row and at different screen sizes.

SUMMARY

The Bootstrap grid system is a powerful tool for creating layouts. Understanding Bootstrap's advanced features puts you in full control, allowing you to build exactly what's needed for your project. Custom containers with container-fluid allow you to fine tune the overall width of a layout. When complex UIs are needed, nested rows can solve even the most complex app layouts. Finally, offsets and push/pull classes let you determine when and where elements move about the grid. Be a layout master and combine Bootstrap's advanced features in your next project.

CONCLUSION

We can no longer make assumptions about the size of the user's screen. Web applications must be designed to dynamically make use of screen capacity. Using responsive web design, a broad range of devices can be used to view an application built from a single code base. Understanding how RWD works, selecting the right framework for your project and mastering advanced techniques will ensure your next project is a huge success.

About the Author

Ed Charbeneau is a web enthusiast, speaker, writer, design admirer and Developer Advocate for Telerik. He has designed and developed web-based applications for business, manufacturing, systems integration as well as customer facing websites. Ed enjoys geeking out to cool new tech, brainstorming about future technology and admiring great design. You can find out more at http://edcharbeneau.com.