Extracted from:

Rails, Angular, Postgres, and Bootstrap, Second Edition

Powerful, Effective, Efficient, Full-Stack Web Development

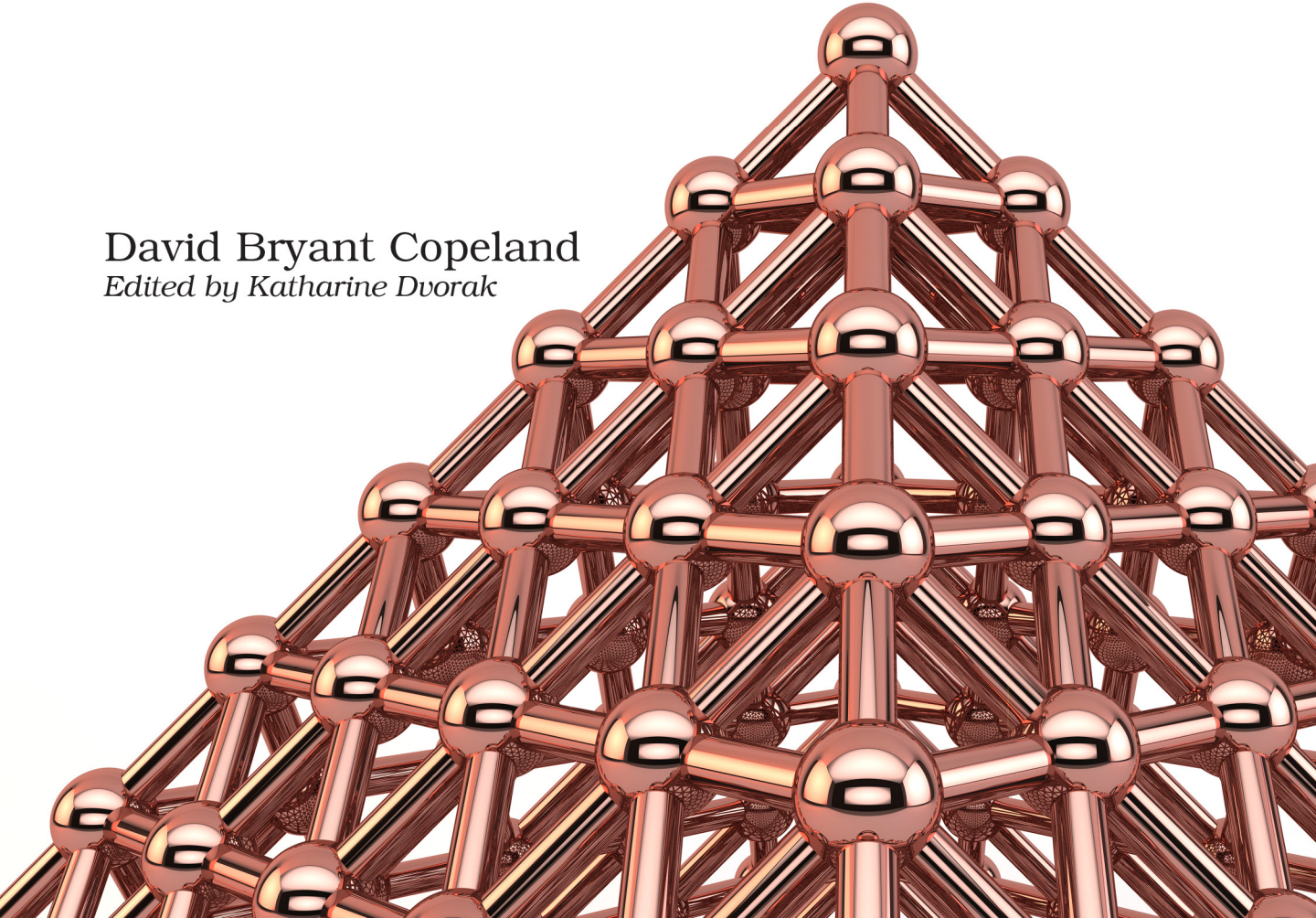
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Rails, Angular, Postgres, and Bootstrap
Second Edition

Powerful, Effective, Efficient, Full-Stack Web Development



David Bryant Copeland
Edited by Katharine Dvorak

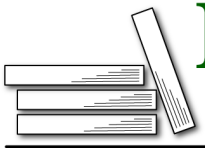
Rails, Angular, Postgres, and Bootstrap, Second Edition

Powerful, Effective, Efficient, Full-Stack Web Development

David Bryant Copeland

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Susannah Davidson Pfalzer
Development Editor: Katharine Dvorak
Indexing: Potomac Indexing, LLC
Copy Editor: Liz Welch
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-220-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2017

The Grid: The Cornerstone of a Web Design

I don't know about you, but looking at a complex layout like the one we're going to build gives me a bit of anxiety. It's not just that CSS can be difficult to use, but it's also not immediately clear how to wrangle all the parts of this design.

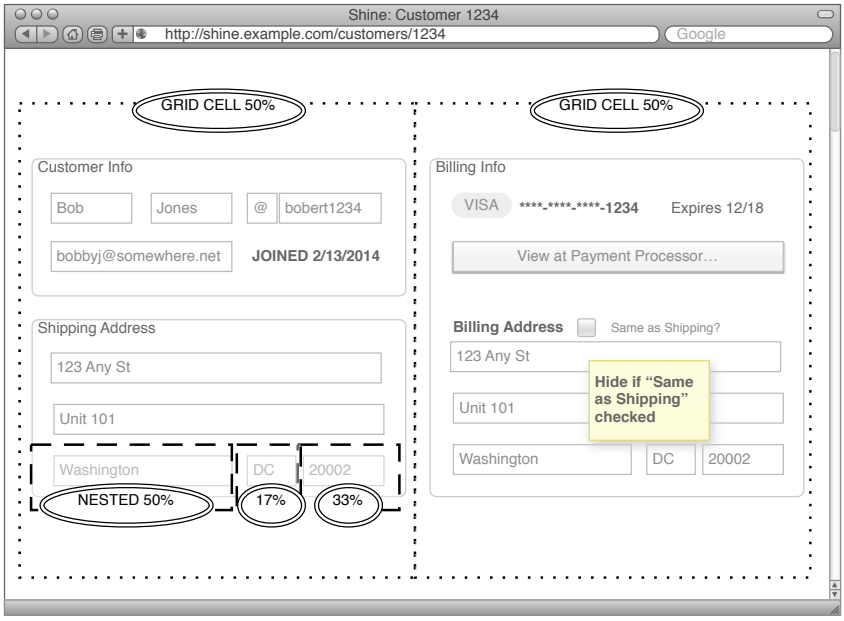
Like functional decomposition in programming, a *grid* is how you can break down a user interface into smaller parts. You can focus on each part of the design, and rely on the grid to keep everything looking visually cohesive.

A grid is more or less what it sounds like—a means of aligning elements along a fixed horizontal and/or vertical axis. You might not have realized it, but you've been using a grid already. By just using Bootstrap's default styles and form classes, the forms we created in Chapter 2, *Create a Great-Looking Login with Bootstrap and Devise*, on page ? (as well as the search results from Chapter 5, *Create Clean Search Results*, on page ?) use a *horizontal grid*. This means that each row of information is spaced in a particular way to make the text and other elements pleasing and orderly.

For the view here, we need a vertical grid, which allows us to place content into side-by-side columns. This is how we'll achieve most of the layout we want. Bootstrap provides a set of CSS classes that allow us to create a grid. Under the covers, it uses CSS floats, which can get messy quickly, but Bootstrap's grid abstracts that away.

Bootstrap's grid has 12 columns. You can combine columns in any way you like to make larger columns, without disrupting the flow and spacing of the grid. For example, you could have a two-column layout where the first column is 25% of the entire width, leaving the remaining 75% for the second column, or you could have three columns of equal size, each taking 33% of the available width.

If you think about your design in terms of rows and columns, you can start to see the grids pop out of our design. Take a look at the following figure.



You can see two grid cells, each taking 50% of the available space, for the main columns of our design, but you can also see a grid nested in each form. The city/state/zip part of the shipping address could be thought of as a grid where the city takes 50% (six grid cells), the state takes 17% (or two grid cells), and the zip code takes the remaining 33% (or four grid cells).

What this means is that, if we have sufficiently generic CSS classes that allow us to place content into grid cells, and to place those cells into rows, and to nest grids within each other, all with proper padding, spacing, and margins, we can break up any design into a series of grids.

This is exactly what Bootstrap's grid system will do.

Using Bootstrap's Grid

Bootstrap's grid is quite powerful, especially if you've never used one before. In this section, we'll build the layout for our view using Bootstrap's grid. As we saw in the previous section, our layout starts with two equal-sized grid cells: one that holds the customer information and shipping address, and the other that holds the billing information.

First, we'll create these cells, which will demonstrate the various CSS classes needed to enable Bootstrap's grid. Then, we'll see how the grid can nest within itself to lay out the customer information and shipping address as a grid-within-a-grid.

Lay Out the Two Main Columns

The most obvious grid in our design is one that holds the two main columns, each taking half the available space. To do this, we'll create two nested div tags inside a parent div, giving each the appropriate CSS class—provided by Bootstrap—to lay it all out in a grid (see on page 8 for some details on why we're using divs).

The outer div has the class row, which tells Bootstrap we're going to place columns inside it. The divs inside the row have class col-md-*X*, where *X* is the number of columns, out of 12, that this particular column should take up. Because we want two equal-sized columns, we want each of *our* columns to take up six of Bootstrap's. Thus, each div will get the class col-md-6 (see Chapter 13, *Dig Deeper*, on page ? for what the -md- means).

We can add this markup to app/javascript/CustomerDetailsComponent/template.html, replacing the bare-bones markup we had there from the last section.

```
<section class="customer-details" *ngIf="customer">
<form><div class="row">
  <div class="col-md-6">
    <h1>Customer</h1>
  </div>
  <div class="col-md-6">
    <h1>Billing Info</h1>
  </div>
</div></form>
</section>
```

If you bring this up in your browser, you'll see that our two headings are shown side by side:

Now, let's tackle the content *inside* these columns. As we saw earlier, we can think of each section of our page as having a nested grid inside this one. Bootstrap's grid works exactly this way.

Build Forms Using a Grid-Within-a-Grid

Bootstrap's grid is not a fixed width, so whenever you write , Bootstrap will divide up the grid in that row based on the available space. This is a powerful feature of the grid system. Much like how we decompose complex objects into smaller ones to make our code easier to understand, we can decompose larger views into smaller ones using the grid.

Why Do We Sometimes Use div and Sometimes Not?

Before HTML5, there weren't a lot of standard elements you could use to describe your content. As a result, the div element came into favor as the way to organize content, particularly for targeting by CSS styling. With the advent of HTML5, more meaningful elements are available, such as article, section, header, and footer.

Because of this, the W3C recommends that div be used only as a last resort,[a] when no other elements are available.

What this means is that you want to use the right tags when describing your content, regardless of the visualization you are going for. You can then use div tags to achieve the layouts you want. Because div is semantically meaningless, it allows anyone reading your view templates to see clearly what parts of the view are for styling and layout and what parts are for organizing the content.

So, the general rule of thumb is to use divs in cases where you need an element to style against, and *not* as a way to describe content.

a. http://www.w3.org/TR/html5/grouping-content.html#the-div-element

By thinking of each page's component as a grid, we can design that component without worrying about where it is on the page. Bootstrap's grid components will make sure it works.

Let's style the customer information section using the grid. We can see from our mock-up that we have three rows, and the first row has three columns. Since the second and third rows just have one column that takes up the entire row, we don't need to use the grid markup for them. So, we just need to create a grid for the first row.

We'll be using the form classes we saw in Chapter 2, *Create a Great-Looking Login with Bootstrap and Devise*, on page ?, so hopefully this will look familiar. The first name, last name, and username are all about the same size data-wise, so we can create three equal-sized columns for them. Because Bootstrap's grid is 12 columns, we want each of our columns to take up four of Bootstrap's columns, so we'll use the class col-md-4 on each div.

```
<section class="customer-details" *ngIf="customer">
<form><div class="row">
  <div class="col-md-6">
    <h1>Customer</h1>
➤   <div class="row">
➤     <div class="col-md-4">
        <div class="form-group">
          <label class="sr-only" for="first-name">First Name</label>
          <input type="text" class="form-control"
```

```
                name="first-name" value="Bob">
          </div>
        </div>
➤       <div class="col-md-4">
          <div class="form-group">
            <label class="sr-only" for="last-name">Last Name</label>
            <input type="text" class="form-control"
                   name="last-name" value="Jones">
          </div>
        </div>
➤       <div class="col-md-4">
          <div class="form-group">
            <label class="sr-only" for="username">Username</label>
            <input type="text" class="form-control"
                   name="username" value="bobert123">
          </div>
        </div>
➤     </div>
      <div class="form-group">
        <label class="sr-only" for="email">Email</label>
        <input type="text" class="form-control"
               name="email" value="bobbyj@somewhere.net">
      </div>
      <label for="joined">Joined</label> 12/13/2014
      <h2>Shipping Address</h2>
```

Note that we used form-group on a different element as col-md-4. This isn't technically required but is commonly done to separate concerns. Generally, you want classes used for your grid to be separate from classes used for styling so that you can be sure your grid doesn't get messed up by styling classes. Also, we can add more styling later without worrying about how the grid will affect it. Take a look at what we've done in our browser (see the following figure), and you can see that it looks pretty good!

Up to now, we've created grid cells that are all the same size. Let's lay out the shipping address part of our page, which requires that some of the grid cells be larger than others.