Extracted from:

Rails, Angular, Postgres, and Bootstrap, Second Edition

Powerful, Effective, Efficient, Full-Stack Web Development

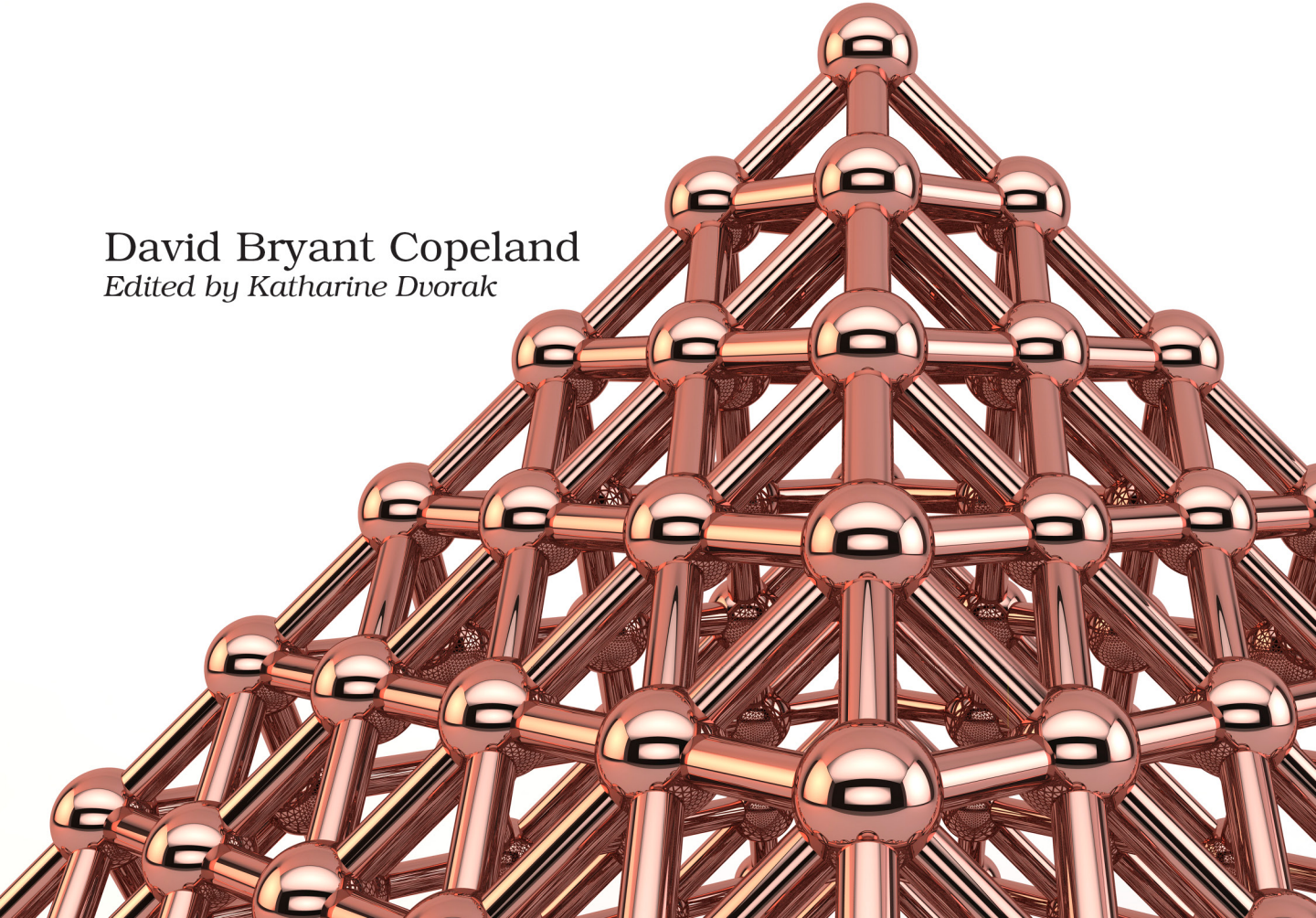
The Pragmatic Bookshelf

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

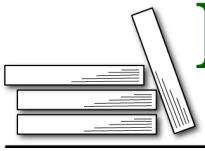
Rails, Angular, Postgres, and Bootstrap, Second Edition

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

Changing Our Search to Use Typeahead

Given everything you've done up to this point, changing the search from one where you must click a button to one where the search happens as you type will actually be fairly straightforward. Because Angular has allowed us to separate our concerns, you have all the code you need in place. You'll just need to connect it to the user interface in a different way.

Currently, when the user modifies the contents of the text field, Angular updates the value of keywords in our CustomerSearchComponent class. You can't directly see it, but it does it as the user types. If you can hook into that behavior, you can perform your search as the user is typing.

Angular provides a way to do this by binding to the ngModelChange property. The code you have now is

```
<input bindon-ngModel="keywords" ... >
```

which is equivalent to this:

```
<input bind-ngModel="keywords"
         on-ngModelChange="keywords=$event" ... >
```

Recall that on- creates a one-way binding from the view to our code (you used this for the click event earlier in *Respond to Click Events*, on page ?). As part of sending the event back to our code for ngModelChange, Angular sets the global variable $event. This means that instead of assigning it to keywords, as happens by default, you can send $event to our search function:

```
<input type="text" id="keywords" name="keywords" \
       placeholder="First Name, Last Name, or Email Address"\
       class="form-control input-lg" \
       bind-ngModel="keywords" \
       on-ngModelChange="search($event)"> \
```

Note that because you've replaced Angular's default behavior, you need to set this.keywords inside search yourself. Right after you do that, however, you can use the updated value to perform the search just as before (though you're only going to do a search for three or more characters so you don't do too broad a search):

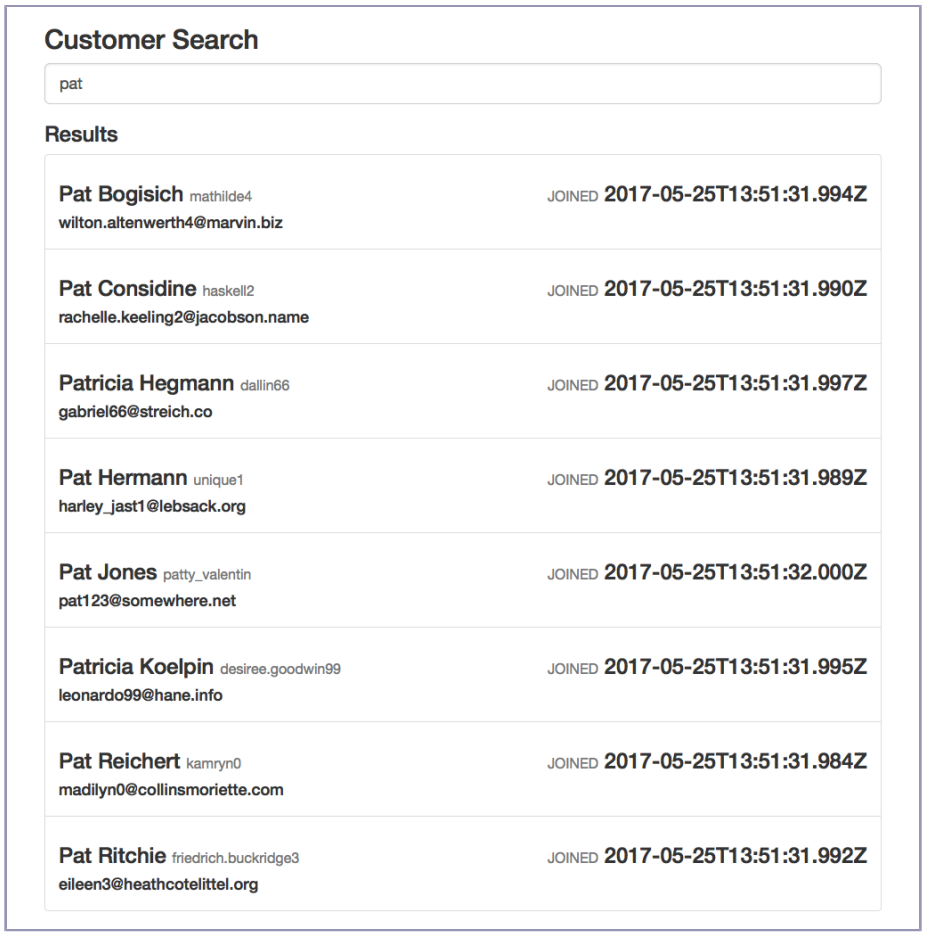
6_angular/50-actual-typeahead/shine/app/javascript/packs/customers.js

```
search: function($event) {
  var self = this;
  self.keywords = $event;
  if (self.keywords.length < 3) {
    return;
  }
  self.http.get(
    "/customers.json?keywords=" + self.keywords
  ).subscribe(
    function(response) {
      self.customers = response.json().customers;
    },
    function(response) {
      window.alert(response);
    }
  );
}
```

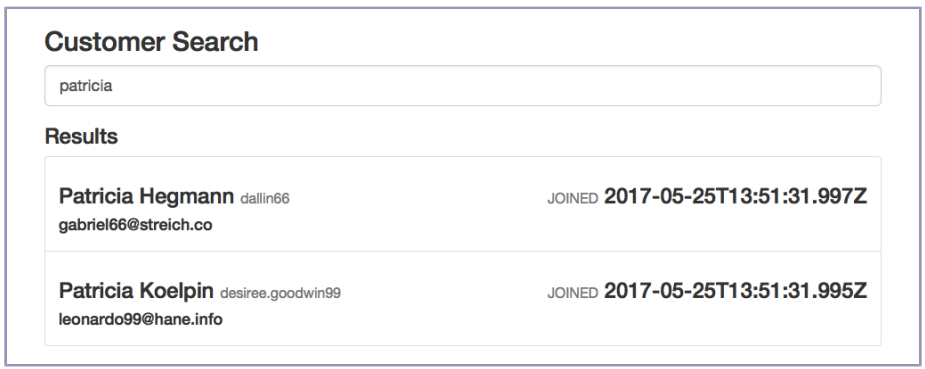
Finally, let's remove the "Find Customers" button since it's no longer needed. Removing this means you can remove the span surrounding the button as well as the div you used to make the button group. Our search form now looks like so:

```
<section class="search-form"> \
  <form> \
    <label for="keywords" class="sr-only">Keywords></label> \
    <input type="text" id="keywords" name="keywords" \
           placeholder="First Name, Last Name, or Email Address"\
           bind-ngModel="keywords" \
           on-ngModelChange="search($event)" \
           class="form-control input-lg">\
  </form> \
</section> \
```

Now, reload the page and type in "pat." You'll see some search results like those shown in the figure on page 7.



If you keep typing out “patricia,” the results automatically reduce to only those that match, as shown in the next figure.



The typeahead works! The entire feature required little code (once you installed and configured Angular—a one-time cost), and instead of implementing typeahead with a special-purpose library, you have set up a framework for implementing any user interface you might need. Because of how Angular works, you aren't wrestling with how to attach our JavaScript to our DOM elements or how to interact with the back end. Because of how Rails works, our back end is almost identical to the original back end.

In other words, by using what Rails gives us, and using what Angular gives us, you were able to create a fairly sophisticated feature quickly and without a lot of code. And it's fast, thanks to Postgres's sophisticated indexing and ordering features.