

Extracted from:

Rails, Angular, Postgres, and Bootstrap

Powerful, Effective, and Efficient
Full-Stack Web Development

This PDF file contains pages extracted from *Rails, Angular, Postgres, and Bootstrap*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2015 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

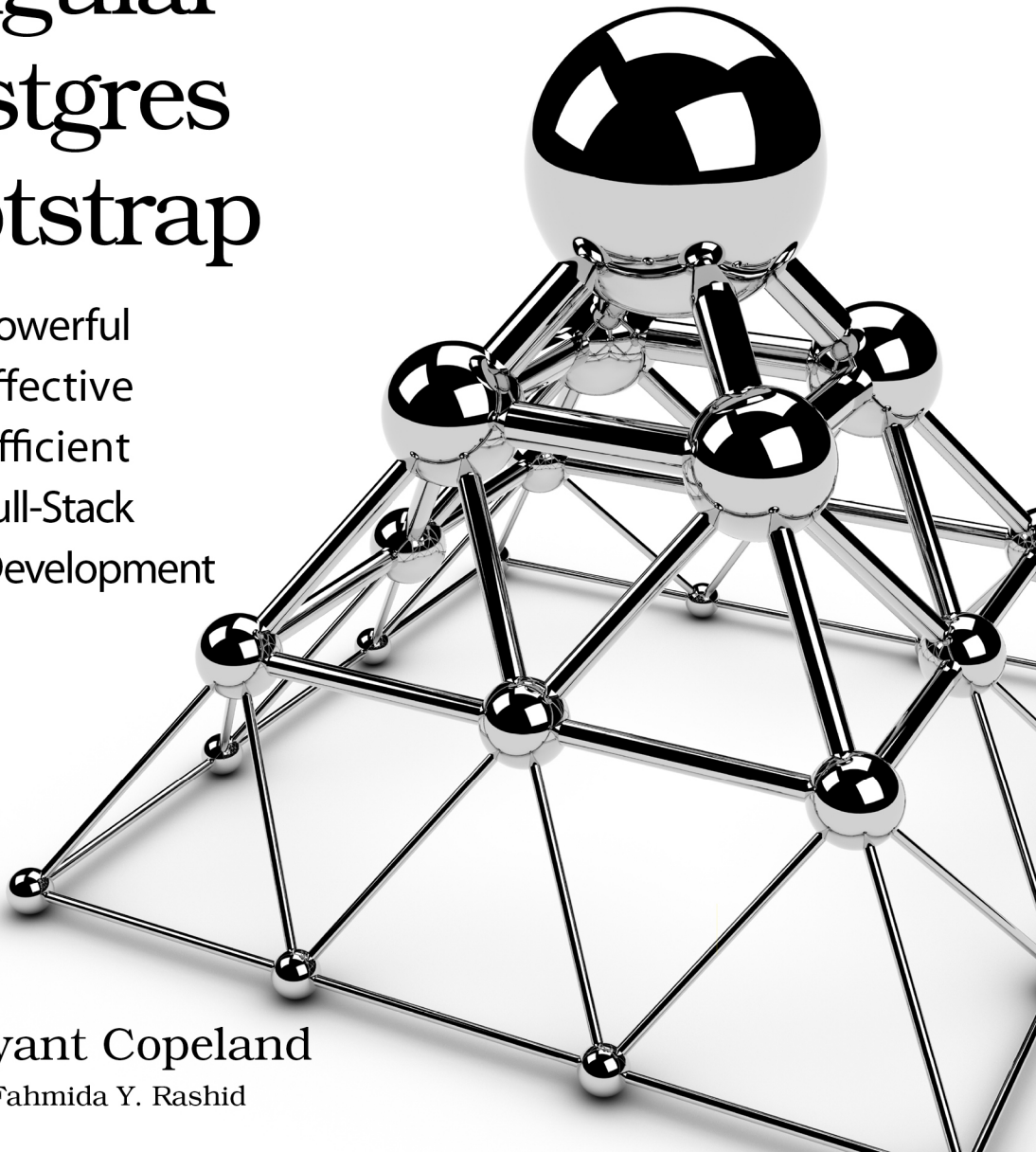
The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The
Pragmatic
Programmers

Rails Angular Postgres Bootstrap

Powerful
Effective
Efficient
Full-Stack
Web Development



David Bryant Copeland

edited by Fahmida Y. Rashid

Rails, Angular, Postgres, and Bootstrap

Powerful, Effective, and Efficient
Full-Stack Web Development

David Bryant Copeland

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <https://pragprog.com>.

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

Copyright © 2015 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-13: 978-1-68050-126-1

Encoded using the finest acid-free high-entropy binary digits.

Book version: B1.0—July 29, 2015

Paginating the Results using Bootstrap's Components

Adding pagination can be done in just two steps: adjusting the query to find the right “page”, and adding pagination controls to the view. There are several RubyGems out there that can help us, but it's actually not that much code to just do it ourselves. Since we'll be porting our view over to Angular in the next chapter, there's little benefit to integrating a gem at this point.

We'll take it one step at a time. First, we'll adjust the controller to handle pagination.

Handling Pagination in the Controller

For simplicity, we'll hard-code the size of a page to 10 results, and look for a new parameter, :page that indicates which page the user wants, with a default of 0.

```
search/pagination/shine/app/controllers/customers_controller.rb
class CustomersController < ApplicationController
  PAGE_SIZE = 10

  def index
    @page = (params[:page] || 0).to_i

    # ...

  end
end
```

Next, we'll use both PAGE_SIZE and @page to construct parameters to ActiveRecord's offset and limit methods. Since our results are sorted, we can rely on these two methods to allow us to reliably page through the results without the order changing between pages.

```
search/pagination/shine/app/controllers/customers_controller.rb
@customers = Customer.where(
  customer_search_term.where_clause,
  customer_search_term.where_args).
  order(customer_search_term.order).
  ➤ offset(PAGE_SIZE * @page).limit(PAGE_SIZE)
```

That's all there is to our controller. Now, we'll adjust the view to allow paging.

Adding Pagination Controls to the View

To keep things simple, we'll go with a previous/next style of pagination. This means we'll need two links on the page, which we can create by adding or

subtracting 1 to @page and passing that to the Rails-provided `customers_path` helper.

To style the links, Bootstrap provides a component we can use, called a *pager*. Let's set it up in a partial, which we'll then use to place the pager before *and* after the results list (this allows the user to always have the pager handy). We've highlighted the markup and classes Bootstrap requires to style the pager. Pay special attention to `disabled`, which will give our Previous button a disabled look if we're on the first page.

search/pagination/shine/app/views/customers/_pager.html.erb

```
<nav>
>   <ul class="pager">
>     <li class="previous" <%= page == 0 ? 'disabled' : '' %>>
      <%= link_to "&larr; Previous".html_safe,
        customers_path(keywords: keywords, page: page - 1) %>
      </li>
>     <li class="next">
      <%= link_to "Next &rarr;".html_safe,
        customers_path(keywords: keywords, page: page + 1) %>
      </li>
    </ul>
  </nav>
```

Now, we'll include the partial in `app/views/customers/index.html.erb`.

search/pagination/shine/app/views/customers/index.html.erb

```
<section class="search-results">
  <header>
    <h1 class="h3">Results</h1>
  </header>
  <%= render partial: "pager",
    locals: { keywords: @keywords, page: @page } %>

  <ol class="list-group">

    <!-- ... -->

  </ol>
  <%= render partial: "pager",
    locals: { keywords: @keywords, page: @page } %>
</section>
```

If we start our server and search, we'll now only see 10 results, and our pager controllers work to allow us to step through them.

Customer Search

Results

← Previous
Next →

Robert Jones bobby_nellie.skiles bob123@somewhere.net	JOINED 2015-03-14
Winston Bob reuben_hilpert0 boyd0@dibbertgoyette.biz	JOINED 2015-03-14
Emery Bob daphne1 ivy1@boylekohler.net	JOINED 2015-03-14
Vada Bob cleta2 maye_lind2@leschfeeney.biz	JOINED 2015-03-14
Humberto Bob stephan.gulgowski3 greg.sanford3@dickensmorierte.info	JOINED 2015-03-14
Godfrey Bobby albertha.cole5 cristina5@pouros.net	JOINED 2015-03-14
Bob Little gerard.jaskolski0 linda0@borerpouros.org	JOINED 2015-03-14
Bob Mayert kathlyn.wuckert2 kelley2@welch.net	JOINED 2015-03-14
Bob Mosciski alberta3 jodie_mccullough3@stark.org	JOINED 2015-03-14
Bob Rosenbaum naomi1 florian_stehr1@macgyverdamore.org	JOINED 2015-03-14

← Previous
Next →

Because of our indexes, Postgres' powerful implementation of order by, and Bootstrap's pre-made components, we were able to add performant pagination in just a few lines of code.