viget

How To Create PDFs in Rails

Mike Ackerman, Developer

Posted in # C O D E

on November 20, 2013

We worked with The Bill of Rights Institute recently to create an interactive digital course for American History teachers. One of the interesting challenges, among many, stemmed from the fact that the project had large sections of readable content. One of our goals was to make it easy for students and teachers to print out their reading material if and when they're not able to read it on screen.

To make printing possible, I needed to create PDF files that were similar to the HTML content. These files needed to be both viewable in the browser and downloadable from the page the content lived on. In some cases, we wanted to selectively

remove some elements from the page or apply a slightly different stylesheet for printing the content.

After a bit of research, I found two possible approaches:

1. Generate a PDF “by hand” from source data using a tool like prawn
2. Take a source HTML document and transform that into a PDF

Taking the source HTML document and converting sounded ideal, because I wanted to keep similar CSS styling and layout of the page with minimal modifications. Since prawn is not an HTML to PDF generator, I investigated the following tools:

- Prince — A command line tool that can take an HTML source file from disk and turn it into a PDF. It can read from a local file or a URL. However, it’s pretty pricey; a server license carries a one-time fee of $3800.
- DocRaptor — Basically, this is Prince offered as a service.
- wkhtmltopdf — A free option that uses the WebKit rendering engine within QT.

wkhtmltopdf sounded like the best option to explore since it uses a browser engine to render the page and then save as a PDF. I found two Ruby gems that use this library: PDFKit & Wicked PDF.

I initially started using PDFKit and its included middleware, and I was able to very quickly get viewable and downloadable PDFs.

I enjoyed that the necessary binary files are included with the gem for a number of operating system environments, which saves you from having to install different packages in your respective application environments (OS X vs Ubuntu).

While PDFKit worked great at first, I eventually encountered a roadblock: I needed to be able to include different stylesheets & layouts for different "types" of PDF files,

which PDFKit didn't have any mention of supporting. I was also struggling to get asset paths working correctly on Heroku. The PDF generation actually happens in a separate process, so I somehow needed to use absolute URLs for paths to all assets.

After a bit of searching, I found the excellent Wicked PDF gem.

Wicked PDF

Wicked PDF doesn't package the binaries in the main gem, but it's simple to include the binaries that you need (you can grab from PDFKit gem) in your bin/ directory and set up Wicked PDF like:

```
platform = RUBY_PLATFORM

if platform.include?("darwin") # OS X machine
  binary_path = Rails.root.join('bin', 'wkhtmltopdf-0.9.9-OS-X-i386').to_s
elsif platform.include?("64-linux") # 64-bit linux machine
  binary_path = Rails.root.join('bin', 'wkhtmltopdf-amd64').to_s
end

WickedPdf.config = { :exe_path => binary_path }
```

Wicked PDF also has the optional middleware, but I decided to not use it so that I could have more fine-grained control over where PDF files can be accessed and specifying their layout and template for each "type."

Viewing PDFs in the browser:

```
respond_to do |format|
  format.pdf do
    render :pdf => "my_pdf_name.pdf",
      :disposition => "inline",
      :template => "controller_name/show.pdf.erb",
      :layout => "pdf_layout.html"
  end

  format.html
end
```

Downloading PDFs as a file:

```
def download
  html = render_to_string(:action => :show, :layout => "pdf_layout.html")
  pdf = WickedPdf.new.pdf_from_string(html)

  send_data(pdf,
    :filename => "my_pdf_name.pdf",
    :disposition => 'attachment')
end
```

Wicked PDF also includes examples and a handy helper method for specifying assets and substituting them inline into the HTML document: `<%=`

`wicked_pdf_stylesheet_link_tag "my_styles" %>` & `<%= wicked_pdf_javascript_include_tag "my_scripts" %>`

It also allows for easily debugging the PDF page by viewing it as an HTML page.

You can do this by using the described option:

`:show_as_html => params[:debug].present?`

This allows you to simply add a `?debug=true` **to the end of your path.**

Example: `http://example.com/my_pdf_name...`

Ultimately, I found Wicked PDF to be the best choice due to: ease of setup, ease of using different layouts and assets for PDFs, and excellent documentation and examples. Some of the examples included how to use assets on Heroku, assets from a CDN, using the asset helper methods, and how to generate and download files using `send_file`**.**

Have you worked on a similar project? Any input on best solutions? Let us know in the comments below.

Mike is a developer who uses his computer and electrical engineering degrees to craft complex but streamlined back-end systems for clients such as Shure and Privia Medical Group.

More posts by Mike

• 3 years

Claudiu ago

nice post. One question, do you have any knowledge for any ruby libraries that have this feature: "write to a pdf, that has form enabled?"

regards,

Share ›

• 3 years

Mike Ackerman → Claudiu ago

Hi Claudiu,
I'm not sure I understand what you mean by "write to a pdf, that has form enabled". Can you explain this a bit more?

1 Share ›

• 3 years

palisir → Mike Ackerman ago

I think he is referring to the form feature of PDF's (the ability to fill a form inside a rendered pdf).

And short answer is : no. It's a pain in the ass to build forms in pdf, even with desktop software.

Share ›

• 3 years

Tarellel → Claudiu ago

Great Post

No comprende...

Do you possibly mean to write the contents of a form directly to a PDF?
Of maybe do mean to use a ruby library, to create a PDF with form elements/input fields?

Share ›

• 3 years

MrChris ago

How do you get around the problem where wkhtmltopdf will just cut off lines, paragraphs, or headings half way (verticall) through the line and put the rest of the text on the next page?

Share ›

• 3 years

Mike Ackerman → MrChris ago

In our case we avoided page breaks in certain elements with the following CSS:
.article__section--figure {
page-break-inside: avoid;

page-break-inside: avoid;
}

There are a number of ways you can solve page break issues with CSS as well. Check out these links:
http://davidwalsh.name/css-pag...

https://developer.mozilla.org/...

Share ›

Let's get to work.

Have an unsolvable problem or audacious idea? We'd love to hear about it.

C O N T → A C T U

hello@viget.com

703.891.0670

Washington DC Metro

Durham, NC

Boulder, CO

The Viget Newsletter

A curated periodical featuring thoughts, opinions, and tools for building a better digital world.

C H E → C K I T O U T