Print pages to PDF files

For offline reading or printing, the document should be exported to PDF format. A manual method is to print each post by the user browsers. This post guides to configure a plugin to automatically export all site's posts to PDF during the build time, and add a download button to download generated PDF files.

#python #mkdocs #pdf

Last update: 2021-06-03 14:04:45

Table of Content

1. The cover page

When printing to a PDF file, the first page should show the post title and its short description. This page is called the cover page which will be created only in printing mode.

Create an element with class `cover` in the `main.html` template to wrap the cover section. In print mode, this element should cover the full height (100%) of the first paper and align its content vertically. After the line of tags, the updated date will be shown to easily check the latest version of the document:

overrides\main.html

```
<style>
@media print {
    .md-typeset .cover {
    height: 100vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
    }
}
</style>
<div class="cover">
    <h1 class="page-title">{{ title | d(config.site_name, true) }}</h1>
    <p class="page-description">{{ description }}</p>
    {% if tags is defined %}
    <p class="page-tags">
        {% for tag in tags %}
        <a class="tag" href="{{ config.site_url }}tags/#{{tag}}">
            <span class="tag-name">
                #{{ tag }}
            </span>
        </a>
        {% endfor %}
    </p>
    {% endif %}
    {% if page.meta.git_revision_date_localized %}
        {% import "partials/language.html" as lang with context %}
            <p class="md-source-date">
                <hr style="margin-bottom: .5em;">
                <small>
                    {{ lang.t("source.revision.date") }}:
                    {{ page.meta.git_revision_date_localized }}
                </small>
            </p>
    {% endif %}
</div>
```

2. The Table of Content page

When displaying on a screen, the Table of Content is displayed in the right sidebar. In printed pages, there should be a page to display the table of content too. This page is also only visible in printing.

The base Material for MkDocs theme has a partial block for Table of Content section, so I just need to include it in the modified `main.html` template, between the cover page and the main content.

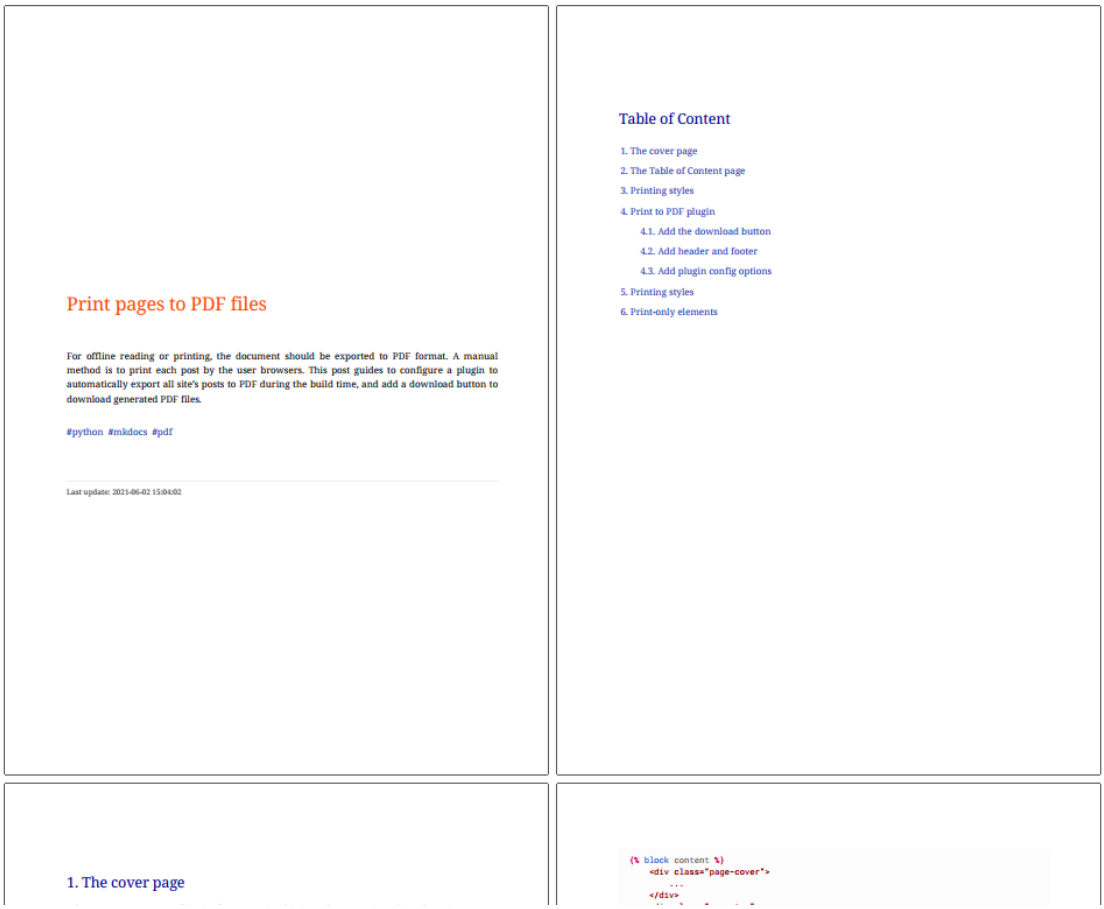
```
{% block content %}
    <div class="cover">
        ...
    </div>
    <div class="toc">
        <h2>Table of Content</h2>
        {% include "partials/toc.html" %}
    </div>
    {{ page.content }}
{% endblock%}
```

There are some styles applied for this section:

- Hide the default label and add a new `<h2>` header
- Remove list-style to make a clear list
- When printing, remove color effect on link items

```
.md-typeset .toc {
    display: none;
}
.md-typeset .toc label {
    display: none;
}
.md-typeset .toc .md-nav {
    font-size: unset;
    line-height: 1.6;
}
.md-typeset .toc .md-nav--secondary {
    margin-left: -2em;
}
.md-typeset .toc .md-nav__list {
    margin: 0;
}
.md-typeset .toc ul {
    list-style: none;
}
@media print {
    .md-typeset .toc {
        display: block;
        page-break-after: always;
```

```
    }
    .md-typeset .toc .md-nav__link {
        color: var(--md-typeset-a-color);
    }
    .md-typeset .toc .md-nav__link.md-nav__link--active {
        font-weight: unset;
    }
}
```



Preview of the printed document

3. Printing styles

There are some more additional styles need to be applied on the page when printing. I preview the printed version using **Save to PDF** option in the Chrome browser.

Set the paper size and printing margins:

```
@page {
    size: a4 portrait;
    margin: 25mm 15mm 25mm 20mm;
}
```

Some elements only show in printing version, add media query type to display them:

```
.md-typeset .print-only {
    display: none;
}
@media print {
    .md-typeset .print-only {
        display: block;
    }
    .md-typeset .screen-only {
        display: none;
    }
}
```

Tabs labels should be marked in printing as they are selected:

```
.md-typeset .tabbed-set > label {
    border-color: var(--md-accent-fg-color);
    color: var(--md-accent-fg-color);
}
```

The disqus section also needs hidden in printing:

```
@media print {
    .md-typeset #__comments,
    .md-typeset #disqus_recommendations,
    .md-typeset #disqus_thread {
        display: none;
    }
}
```

Class `.np` is used for force break page in printing:

```
.md-typeset .np {
    page-break-after: always;
}
```

it is good to used with a custom block:

```
::: np
```

Some elements do not keep the top margin in printing:

```
@media print {
    .md-typeset .np + *,
    .md-typeset .toc + *,
    .md-typeset .btn-actions + * {
        margin-top: 0;
```

```
    }
}
```

Image and its caption should be displayed in the same page:

```
@media print {
    .md-typeset figure {
        page-break-inside: avoid;
    }
}
```

Admonition can be printed on multiple pages:

```
@media print {
    .md-typeset .admonition,
    .md-typeset details {
        page-break-inside: auto;
    }
}
```

4. Print to PDF plugin

The MkDocs PDF with JS Plugin[1] exports documentation in PDF format with rendered JavaScript content. This is very useful if documents have mermaid diagrams. A download button will be added to the top of the page, and it is hidden in the PDF files.

For executing the JavaScript code, ChromeDriver is used, so it is nesseccary to:

1. Install Chrome
2. Download ChromeDriver
3. Add the ChromeDriver to OS user's `PATH` environment

After that, install the plugin:

```
pip install -U git+https://github.com/vuquangtrong/mkdocs-pdf-with-js-plugin.git
```

⚠ Install the original plugin with `pip install mkdocs-pdf-with-js-plugin` if don't need a customized version. The following features are not implemented in the original version.

Enable the plugin:

```
plugins:
    - search # built-in search must be always activated
    - pdf-with-js
```

While building `mkdocs build` or serving `mkdocs serve` the documentation, the PDF files will be generated. They are stored in the `site\pdfs` folder.

4.1. Add the download button

Create an element to contain the download button at the beginning of the document content in the `base.html` template. This element should be hidden in printing mode.

The plugin will find the `<div class="btn-actions">` element to insert a button. If there is no such existing element, the plugin will create a new element and insert to the page content.

```
def _add_link(self, soup, page_paths):

    icon = BeautifulSoup(''
        '<span class="twemoji">'
            '<svg viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">'
                '<path d="M5 20h14v-2H5m14-9h-4V3H9v6H5l7 7 7-7z"></path>'
            '</svg>'
        '</span>',
        'html.parser')
    text = "PDF"

    btn = soup.new_tag("a", href=page_paths["relpath"])
    btn.append(icon)
    btn.append(text)
    btn['class'] = 'md-button'

    bar = soup.find("div", {"class" : "btn-actions"})
    if bar:
        bar.p.insert(0, btn)
    else:
        toc = soup.find("div", {"class" : "toc"})
        if toc:
            div = BeautifulSoup(''
                '<div class="btn-actions screen-only">'
                    '<p></p>'
                '</div>',
                'html.parser')
            div.p.insert(0, btn)
            toc.insert_after(div)

    return soup
```

4.2. Add header and footer

The command sent to ChromeDriver to print a page is `Page.printToPDF` , read more at Chrome DevTools Protocol - printToPDF.

This command needs some params to control the printing, which include:

landscape : boolean

Paper orientation. Defaults to false.

displayHeaderFooter : boolean

Display header and footer. Defaults to false.

headerTemplate: string

HTML template for the print header. Should be valid HTML markup with following classes used to inject printing values into them:

- *date*: formatted print date
- *title*: document title
- *url*: document location
- *pageNumber*: current page number
- *totalPages*: total pages in the document

For example, `<span class=title></span>` would generate a span containing the *title*.

footerTemplate : string

HTML template for the print footer. Should use the same format as the *headerTemplate*.

Those params are initialized in the `__init__` function:

```
def __init__(self):
    self.displayHeaderFooter = True
    self.headerTemplate = \
        '<div style="font-size:8px; margin:auto;">' \
        '<span class=title></span>' \
        '</div>'
    self.footerTemplate= \
        '<div style="font-size:8px; margin:auto;">' \
        'Page <span class="pageNumber"></span> of ' \
        '<span class="totalPages"></span>' \
        '</div>'
```

and they are used to create print options in in a dictionary variable:

```
def _get_print_options(self):
    return {
        'landscape': False,
        'displayHeaderFooter': self.displayHeaderFooter,
        'footerTemplate': self.footerTemplate,
        'headerTemplate': self.headerTemplate,
        'printBackground': True,
        'preferCSSPageSize': True,
    }
```

Finally, the print options are used in the print command:

```
def print_to_pdf(self, driver, page):
    driver.get(page["url"])
    result = self._send_devtools_command(
        driver, "Page.printToPDF",
        self._get_print_options()
    )
    self._write_file(result['data'], page["pdf_file"])
```

4.3. Add plugin config options

To allow user to change the print options in the project config file `mkdocs.yml`, add the config fields into the `plugin.py` file.

```
class PdfWithJS(BasePlugin):
    config_scheme = (
        ('enable', config_options.Type(bool, default=True)),
        ('display_header_footer', config_options.Type(bool, default=False)),
        ('header_template', config_options.Type(str, default='')),
        ('footer_template', config_options.Type(str, default='')),
    )
```

When the MkDocs engine calls to `on_config()` function in this plugin, save the user's configs as below:

```
def on_config(self, config, **kwargs):
    self.enabled = self.config['enable']
    self.printer.set_config (
        self.config['display_header_footer'],
        self.config['header_template'],
        self.config['footer_template']
        )
    return config
```

By doing this, users can add their params to the `pdf-with-js` entry under the `plugins` field in the config file `mkdocs.yml`:

```
plugins:
    - search # built-in search must be always activated
    - pdf-with-js:
          enable: true
          display_header_footer: true
          header_template: >-
              <div style="font-size:8px; margin:auto; color:lightgray;">
                  <span class=title></span>
              </div>
          footer_template: >-
              <div style="font-size:8px; margin:auto; color:lightgray;">
```

```
                Page <span class="pageNumber"></span> of
                <span class="totalPages"></span>
            </div>
```

That's it. When all blog posts now have a download button for users to get a PDF version.

---

1. originally developed by smaxtec ↩