django-easy-pdf Documentation

Release 0.1.0

Filip Wasilewski

February 24, 2016

Contents

CHAPTER 1

django-easy-pdf

Django PDF rendering, the easy way. Developed at en.ig.ma software shop.

1.1 Overview

This app makes rendering PDF files in Django really easy. It can be used to create invoices, bills and other documents from simple HTML markup and CSS styles. You can even embed images and use custom fonts.

The library provides both Class-Based View that is almost a drop-in replacement for Django's `TemplateView` as well as helper functions to render PDFs in the backend outside the request scope (i.e. using Celery workers).

1.2 Quickstart

1. Include `django-easy-pdf`, `xhtml2pdf>=0.0.6` and `reportlab>=2.7,<3` in your `requirements.txt` file.
2. Add `easy_pdf` to `INSTALLED_APPS`.
3. Create HTML template for PDF document and add a view that will render it:

```
{% extends "easy_pdf/base.html" %}

{% block content %}
    <div id="content">
        <h1>Hi there!</h1>
    </div>
{% endblock %}
```

```
from easy_pdf.views import PDFTemplateView

class HelloPDFView(PDFTemplateView):
    template_name = "hello.html"
```

1.3 Documentation

The full documentation is at django-easy-pdf.rtfd.org.

A live demo is at easy-pdf.herokuapp.com. You can run it locally after installing dependencies by running `python demo.py` script from the cloned repository.

1.4 Dependencies

`django-easy-pdf` depends on:

- `django>=1.5.1`
- `xhtml2pdf>=0.0.6`
- `reportlab>=2.7,<3`

1.5 License

`django-easy-pdf` is released under the MIT license.

1.6 Other Resources

- GitHub repository - https://github.com/nigma/django-easy-pdf
- PyPi Package site - http://pypi.python.org/pypi/django-easy-pdf

1.7 Commercial Support

This app and many other help us build better software and focus on delivering quality projects faster. We would love to help you with your next project so get in touch by dropping an email at en@ig.ma.

1.8 Content

1.8.1 Installation

Add `django-easy-pdf=<version>` and `git+https://github.com/chrisglass/xhtml2pdf.git` to your `requirements.txt` file or install it directly from the command line by invoking:

```
$ pip install django-easy-pdf
$ pip install "xhtml2pdf>=0.0.6" "reportlab>=2.7,<3"
```

1.8.2 Usage

Prepare HTML Templates

Create a Django HTML template with embedded CSS style. You can use special style attributes to format the PDF output.

For more information on the supported HTML and CSS rules see docs at https://github.com/chrisglass/xhtml2pdf/blob/master/doc/usage.rst

You can also use custom embeddable resources like images and fonts. Put them inside Django's `STATIC_ROOT` directory and make sure they are available locally on the server even if you are serving your static files from S3 or other CDN.

For now only local resources are supported.

```
{% extends "easy_pdf/base.html" %}

{% block extra_style %}
    <style type="text/css">
        @font-face { font-family: Lato; src: url(fonts/Lato-Reg.ttf); }
        body {
            font-family: "Lato", "Helvetica", "sans-serif";
            color: #333333;
        }
    </style>
{% endblock %}

{% block content %}
    <div id="content">
        <div class="main">
            <h1>Hi there!</h1>
        </div>
    </div>
{% endblock %}
```

Create PDF rendering views

This part is easy. The PDF rendering view inherits from `TemplateResponseMixin` so it works in the same way as Django's `TemplateView`. Just point it to a HTML template and define `get_context_data()` method to pass any extra variables to the template:

```
from easy_pdf.views import PDFTemplateView

class HelloPDFView(PDFTemplateView):
    template_name = "hello.html"

    def get_context_data(self, **kwargs):
        return super(HelloPDFView, self).get_context_data(
            pagesize="A4",
            title="Hi there!",
            **kwargs
        )
```

Then add the view to your url config and start serving PDF files rendered from the HTML template.

```
urlpatterns = patterns("",
    url(r"^hello.pdf$", HelloPDFView.as_view())
)
```

Rendering PDF outside Django views

See *PDF rendering functions*.

1.8.3 API Overview

Views

PDFTemplateResponseMixin

class `easy_pdf.views.`**`PDFTemplateResponseMixin`**
Bases: `django.views.generic.base.TemplateResponseMixin`

A mixin class that implements PDF rendering and Django response construction.

`pdf_filename` = None
Optional name of the PDF file for download. Leave blank for display in browser.

`pdf_kwargs` = None
Additional params passed to `render_to_pdf_response()`

`get_pdf_filename`()
Returns *`pdf_filename`* value by default.

If left blank the browser will display the PDF inline. Otherwise it will pop up the "Save as.." dialog.

Return type `str()`

`get_pdf_kwargs`()
Returns *`pdf_kwargs`* by default.

The kwargs are passed to `render_to_pdf_response()` and `xhtml2pdf.pisa.pisaDocument()`.

Return type `dict`

`get_pdf_response`(*context*, ***response_kwargs*)
Renders PDF document and prepares response.

Returns Django HTTP response

Return type `django.http.HttpResponse`

`render_to_response`(*context*, ***response_kwargs*)

PDFTemplateView

class `easy_pdf.views.`**`PDFTemplateView`**(***kwargs*)
Bases: *`easy_pdf.views.PDFTemplateResponseMixin`*, `django.views.generic.base.ContextMixin`, `django.views.generic.base.View`

Concrete view for serving PDF files.

```
class HelloPDFView(PDFTemplateView):
    template_name = "hello.html"
```

`get`(*request*, **args*, ***kwargs*)
Handles GET request and returns HTTP response.

PDF rendering functions

`easy_pdf.rendering.`**`render_to_pdf`**(*template*, *context*, *encoding=u'utf-8'*, ***kwargs*)
Create PDF document from Django html template.

Parameters

- **template** (*str*) – path to Django template
- **context** (dict or django.template.Context) – template context

Returns rendered PDF

Return type bytes

Raises *PDFRenderingError*, *UnsupportedMediaPathException*

easy_pdf.rendering.**render_to_pdf_response**(*request*, *template*, *context*, *filename=None*, *encoding=u'utf-8'*, ***kwargs*)

Renders a PDF response using given request, template and context.

If filename param is specified then the response Content-Disposition header will be set to attachment making the browser display a "Save as.." dialog.

Parameters

- **request** (django.http.HttpRequest) – Django request
- **template** (*str*) – path to Django template
- **context** (dict or django.template.Context) – template context

Return type django.http.HttpResponse

Other lower-level helpers

easy_pdf.rendering.**html_to_pdf**(*content*, *dest*, *encoding="utf-8"*, *link_callback=fetch_resources*, ***kwargs*)

Converts html content into PDF document.

Parameters **content** (*unicode*) – html content

Returns PDF content

Return type bytes

Raises *PDFRenderingError*

easy_pdf.rendering.**fetch_resources**(*uri*, *rel*)

Retrieves embeddable resource from given uri.

For now only local resources (images, fonts) are supported.

Parameters **uri** (*str*) – path or url to image or font resource

Returns path to local resource file.

Return type str

Raises *UnsupportedMediaPathException*

easy_pdf.rendering.**make_response**(*content*, *filename=None*, *content_type=u'application/pdf'*)

Wraps content into HTTP response.

If filename is specified then Content-Disposition: attachment header is added to the response.

Default Content-Type is application/pdf.

Parameters

- **content** (*bytes*) – response content

- **filename** (*str*) – optional filename for file download
- **content_type** (*str*) – response content type

Return type django.http.HttpResponse

easy_pdf.rendering.**encode_filename**(*filename*)
Encodes filename part for Content-Disposition: attachment.

```
>>> print(encode_filename("abc.pdf"))
filename=abc.pdf
>>> print(encode_filename("aa bb.pdf"))
filename*=UTF-8''aa%20bb.pdf
>>> print(encode_filename(u"zażółć.pdf"))
filename*=UTF-8''za%C5%BC%C3%B3%C5%82%C4%87.pdf
```

Exceptions

exception easy_pdf.exceptions.**EasyPDFError**
Bases: exceptions.Exception

Base error class

exception easy_pdf.exceptions.**UnsupportedMediaPathException**
Bases: *easy_pdf.exceptions.EasyPDFError*

Resource not found or unavailable

exception easy_pdf.exceptions.**PDFRenderingError**(*message*, *content*, *log*, **args*, ***kwargs*)
Bases: *easy_pdf.exceptions.EasyPDFError*

PDF Rendering error. Check HTML template for errors.

1.8.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/nigma/django-easy-pdf/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

django-easy-pdf could always use more documentation, whether as part of the official django-easy-pdf docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nigma/django-easy-pdf/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-easy-pdf* for local development.

1. Fork the *django-easy-pdf* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-easy-pdf.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-easy-pdf
$ cd django-easy-pdf/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8:

```
$ flake8 easy_pdf
```

To get flake8, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
2. The pull request should work for Python 2.7, and 3.3 (if there are compatible 3rd party packages available). Check https://travis-ci.org/nigma/django-easy-pdf/pull_requests and make sure that the tests pass for all supported Python versions.

1.8.5 Credits

Development Lead

- Filip Wasilewski <en@ig.ma>

Contributors

- Jon Bolt (@epicbagel)
- @msaizar
- @SaeX

1.8.6 History

0.1.0 (2014-01-24)

- First release

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

e

r

v

E

F

G

H

M

P

R

U