django-critical Documentation

Release 0.1.1

Martín Blech

September 21, 2016

Contents

Contents:

CHAPTER 1

django-critical

Inlines critical path CSS and defers loading full CSS asynchronously.

1.1 Documentation

The full documentation is at https://django-critical.readthedocs.org.

1.2 Quickstart

- Install django-critical:

```
pip install django-critical
```

- Add `critical` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # other apps
    "critical",
)
```

- Add `critical.middleware.CriticalCssMiddleware` to your `MIDDLEWARE_CLASSES` setting:

```
MIDDLEWARE_CLASSES = (
    # other middlewares
    "critical.middleware.CriticalCssMiddleware",
)
```

- Point your `CRITICAL_PENTHOUSE_PATH` setting to the correct path, e.g.:

```
CRITICAL_PENTHOUSE_PATH = os.path.join(
    BASE_DIR, 'node_modules/penthouse/penthouse.js')
```

- If phantomjs is not in your PATH, you have to set `CRITICAL_PHANTOMJS_PATH` and point it to your phantomjs binary:

```
CRITICAL_PHANTOMJS_PATH = os.path.join(
    BASE_DIR, 'node_modules/phantomjs/bin/phantomjs')
```

1.3 How it works

`django-critical` is activated by adding `{% load critical %}` to your template.

`django-critical` then parses the HTML between `{% critical %}` and `{% endcritical %}` and searches for CSS.

```
{% critical %}
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<style>
  .jumbotron h1 {
    color: red;
  }
</style>
{% endcritical %}
```

It downloads and concatenates all that CSS and uses penthouse to extract the critical path CSS rules from it, which end up inlined and minified in place of the original CSS.

```
<style>html{font-family:sans-serif;-webkit-text-size-adjust:100%...</style>
```

Later in the HTML, before the closing `</body>` tag, the `{% critical_async %}` template tag takes care of loading the rest of the CSS using FilamentGroup's loadCSS.

1.4 Caching

`django-critical` calculates the critical path CSS for the first request, caches the result and reuses this CSS verbatim for further requests. Most web applications have different critical path CSS for different groups of pages, though. This can be solved using the `{% critical_key "<key>" %}` template tag, so that different templates can have different caching keys.

1.5 Is this stable and ready for production use?

No. `django-critical` is in a very early stage of development, so you should use it at your own risk. Bug reports and contributions are welcome, though!

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-critical
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-critical
$ pip install django-critical
```

CHAPTER 3

Usage

To use django-critical in a project:

```
import critical
```

CHAPTER 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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/martinblech/django-critical/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/martinblech/django-critical/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-critical.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-critical
$ cd django-critical/
$ 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 and the tests, including testing other Python versions with tox:

```
$ flake8 critical tests
$ python setup.py test
$ tox
```

To get flake8 and tox, 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.

4.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. 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.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/martinblech/django-critical/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_critical
```

CHAPTER 5

Credits

5.1 Development Lead

- Martín Blech <martinblech@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2014-08-02)

- First release on PyPI.