django-responsive2 Documentation

Release 0.1.3

Mishbah Razzaque

Sep 27, 2017

Contents

Contents:

CHAPTER 1

django-responsive2

django-responsive2 is an experimental Django app that gives web designers tools for building responsive websites. It can dynamically swap content based on breakpoints.

Why would you use django-responsive2?

This project was inspired by Twitter Bootstrap's Responsive Utilities. Bootstrap provides some handful helper classes, for faster mobile-friendly development. These can be used for showing and hiding content by device via media query combined with large, small, and medium devices.

Similarly `django-responsive2` can be used to render different content based on device screen sizes and pixel ratios. However, while it is very useful to show/hide content using css display property, Bootstrap Responsive Utilities does not actually prevent the content from being loaded on to the page. It is best explained through examples.

Sample example template using django-responsive2:

```
<div class="container">
    <div class="row">
        {% if device.is_xsmall or device.is_small %}
            <div class="col-sm">
                {# Rendered for x-small/small screen devices #}
                <img src="images/myimage_sm.jpg" alt="Descriptive alt tag" />
            </div>
        {% elif device.is_medium %}
            <div class="col-md">
                {# Rendered for medium screen devices #}
                <img src="images/myimage_md.jpg" alt="Descriptive alt tag" />
            </div>
        {% else %}
            <div class="col-lg">
                {# Rendered for large/xlarge screen devices #}
                <img src="images/myimage_lg.jpg" alt="Descriptive alt tag" />
            </div>
        {% endif %}
    </div>
</div>
```

In this very simple example, using the Bootstrap Responsive Utilities, all 3 images would have been loaded on to the page, wasting precious bandwidth, together with increase in page load time.

In comparison, using `django-responsive2`, only `col-sm` will be rendered for small screen devices (e.g. an iPhone), `col-m` will be displayed for medium screen devices (e.g. an iPad) and lastly `col-lg` will be visible for large screen devices or any devices that do not match the rules for small/medium devices.

Using django-responsive2 in your views

You can also use the `django-responsive2` in your Django views to do particular things based on the matched media queries for the visitors device.

The `ResponsiveMiddleware` middleware sets the `device` attribute on every request object, so you can use `request.device` to get the device information for your visitors:

```
MIDDLEWARE_CLASSES=(
    ...
    'responsive.middleware.ResponsiveMiddleware'
    ...
)
```

Here's an (verbose) example of what the a view could look like, `request.device.matched` returns a list of matched media queries for the visitors device.

e.g. `['small', 'retina']`

```
def home(request):

if 'retina' in request.device.matched:
    thumbnail_high_resolution = True
else:
    thumbnail_high_resolution = False

if request.device.is_small:
    hide_ads = True
else:
    hide_ads = False

...
context = {
    'thumbnail_high_resolution': thumbnail_high_resolution,
    'hide_ads': hide_ads
}

...
```

Quickstart

1. Install django-responsive2:

```
pip install django-responsive2
```

2. Add `responsive` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'responsive',
    ...
)
```

3. Add `django.core.context_processors.request` and `responsive.context_processors.device` to your `TEMPLATE_CONTEXT_PROCESSORS`:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'django.core.context_processors.request',
    'responsive.context_processors.device',
    ...
)
```

4. Add the `ResponsiveMiddleware` to MIDDLEWARE_CLASSES:

```
MIDDLEWARE_CLASSES = (
    ...
    'responsive.middleware.ResponsiveMiddleware',
    ...
)
```

Configuration

`django-responsive2` lets you to define the breakpoints at which your layout will change, adapting to different screen sizes. Here's the default breakpoints:

```
RESPONSIVE_MEDIA_QUERIES = {
    'small': {
        'verbose_name': _('Small screens'),
        'min_width': None,
        'max_width': 640,
    },
    'medium': {
        'verbose_name': _('Medium screens'),
        'min_width': 641,
        'max_width': 1024,
    },
    'large': {
        'verbose_name': _('Large screens'),
        'min_width': 1025,
        'max_width': 1440,
    },
    'xlarge': {
        'verbose_name': _('XLarge screens'),
        'min_width': 1441,
        'max_width': 1920,
    },
    'xxlarge': {
        'verbose_name': _('XXLarge screens'),
        'min_width': 1921,
        'max_width': None,
    }
}
```

** Borrowed from ZURB Foundation framework, see http://foundation.zurb.com/docs/media-queries.html

While there are several different items we can query on, the ones used for django-responsive2 are min-width, max-width, min-height and max-height.

- min_width — Rules applied for any device width over the value defined in the config.
- max_width — Rules applied for any device width under the value defined in the config.
- min_height — Rules applied for any device height over the value defined in the config.
- max_height — Rules applied for any device height under the value defined in the config.
- pixel_ratio — Rules applied for any device with devicePixelRatio defined in the config.

You can override the default media queries by defining own in your `RESPONSIVE_MEDIA_QUERIES` in your `settings.py`. For example:

```
RESPONSIVE_MEDIA_QUERIES = {
    'iphone': {
        'verbose_name': _('iPhone Retina'),
        'min_width': 320,    # mobile first queries
        'pixel_ratio': 2
    },
    ...
}
```

For every media queries, the `device` object will have a `is_FOO` attribute, where FOO is the name of the media query. This attribute returns `True/False`.

Continuing with the example `RESPONSIVE_MEDIA_QUERIES` settings above, here's a simple corresponding template:

```
<div class="container">
    <div class="row">
        {% if device.is_iphone %}
            {# this snippet will only be rendered for retina devices with minimum␣
↪device width 320 #}
            <div class="app-store">
                <a href="#">Available on the App Store</a>
            </div>
        {% endif %}
    </div>
</div>
```

Documentation

The full documentation is at https://django-responsive2.readthedocs.org.

Credits

This app started as a clone of `django-responsive` with some minor modifications to fit my own project requirements. So a big thank you to @mlavin for his hard work.

Shout out to @jezdez for the awesome `django-appconf` — used by this project to handle default configurations.

CHAPTER 2

Installation

1. Install django-responsive2:

```
pip install django-responsive2
```

2. Add `responsive` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'responsive',
    ...
)
```

3. Add `django.core.context_processors.request` and `responsive.context_processors.device` to your `TEMPLATE_CONTEXT_PROCESSORS`:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'django.core.context_processors.request',
    'responsive.context_processors.device',
    ...
)
```

4. Add the `ResponsiveMiddleware` to MIDDLEWARE_CLASSES:

```
MIDDLEWARE_CLASSES = (
    ...
    'responsive.middleware.ResponsiveMiddleware',
    ...
)
```

CHAPTER 3

Usage

`django-responsive2` lets you to define the breakpoints at which your layout will change, adapting to different screen sizes. Here's the default breakpoints:

```
RESPONSIVE_MEDIA_QUERIES = {
    'small': {
        'verbose_name': _('Small screens'),
        'min_width': None,
        'max_width': 640,
    },
    'medium': {
        'verbose_name': _('Medium screens'),
        'min_width': 641,
        'max_width': 1024,
    },
    'large': {
        'verbose_name': _('Large screens'),
        'min_width': 1025,
        'max_width': 1440,
    },
    'xlarge': {
        'verbose_name': _('XLarge screens'),
        'min_width': 1441,
        'max_width': 1920,
    },
    'xxlarge': {
        'verbose_name': _('XXLarge screens'),
        'min_width': 1921,
        'max_width': None,
    }
}
```

** Borrowed from ZURB Foundation framework, see http://foundation.zurb.com/docs/media-queries.html

While there are several different items we can query on, the ones used for django-responsive2 are min-width, max-width, min-height and max-height.

- min_width — Rules applied for any device width over the value defined in the config.
- max_width — Rules applied for any device width under the value defined in the config.
- min_height — Rules applied for any device height over the value defined in the config.
- max_height — Rules applied for any device height under the value defined in the config.
- pixel_ratio — Rules applied for any device with devicePixelRatio defined in the config.

You can override the default media queries by defining own in your `RESPONSIVE_MEDIA_QUERIES` in your `settings.py`. For example:

```
RESPONSIVE_MEDIA_QUERIES = {
    'iphone': {
        'verbose_name': _('iPhone Retina'),
        'min_width': 320,
        'pixel_ratio': 2
    },
    ...
}
```

For every media queries, the `device` object will have a `is_FOO` attribute, where FOO is the name of the media query. This attribute returns `True/False`.

Continuing with the example `RESPONSIVE_MEDIA_QUERIES` settings above, here's a simple corresponding template:

```
<div class="container">
    <div class="row">
        {% if device.is_iphone %}
            {# this snippet will only be rendered for retina devices with min_width =␣
↪320 #}
            <div class="app-store">
                <a href="#">Available on the App Store</a>
            </div>
        {% endif %}
    </div>
</div>
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/mishbahr/django-responsive2/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-responsive2 could always use more documentation, whether as part of the official django-responsive2 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/mishbahr/django-responsive2/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-responsive2* for local development.

1. Fork the *django-responsive2* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-responsive2.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-responsive2
$ cd django-responsive2/
$ 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 responsive 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.

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/mishbahr/django-responsive2/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

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

CHAPTER 5

Credits

This app started as a clone of `django-responsive` with some minor modifications to fit my own project requirements. So a big thank you to @mlavin for his hard work.

Shout out to @jezdez for the awesome `django-appconf` — used by this project to handle default configurations.

Development Lead

- Mishbah Razzaque <mishbahx@gmail.com>

Contributors

- Ashley Wilson <scifilem@gmail.com>

CHAPTER 6

History

0.1.0 (2014-10-15)

- First release on PyPI.