Django Pydenticon Documentation

Release 0.2-dev

Branko Majic

May 22, 2017

Contents

Django Pydenticon is a Django application that provides an identicon generator. It builds upon Pydenticon, a Python library for generating deterministic identicons. Pydenticon capabilities are merely exposed via a web interface (through HTTP).

CHAPTER 1

Support

In case of problems with the application, please do not hestitate to contact the author at **django-pydenticon (at) majic.rs**. The library itself is hosted on Github, and on author's own servers:

- https://github.com/azaghal/django-pydenticon
- https://code.majic.rs/django-pydenticon
- https://projects.majic.rs/django-pydenticon

CHAPTER 2

License

Django Pydenticon is released under terms of *BSD (3-Clause) License*:

```
Copyright (c) 2014, Branko Majic
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

  Redistributions in binary form must reproduce the above copyright notice, this
  list of conditions and the following disclaimer in the documentation and/or
  other materials provided with the distribution.

  Neither the name of Branko Majic nor the names of any other
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Contents:

About Django Pydenticon

Django Pydenticon is a Django application that provides an identicon generator. The implementation uses Pydenticon library for generating the identicons:

- https://github.com/azaghal/pydenticon/
- https://projects.majic.rs/pydenticon

Django Pydenticon comes with some pre-defined sane defaults for generating the identicons, but is configurable, letting the user generate identicons with custom parameters.

Why was this application created?

A number of web-based applications written in Python have a need for visually differentiating between users by using avatars for each one of them.

This functionality is particularly popular with comment-posting since it increases the readability of threads.

The problem is that lots of those applications need to allow anonymous users to post their comments as well. Since anonymous users cannot set the avatar for themselves, usually a random avatar is created for them instead.

There is a number of free (as in free beer) services out there that allow web application developers to create such avatars. Unfortunately, this usually means that the users visiting websites based on those applications are leaking information about their browsing habits etc to these third-party providers.

Django Pydenticon was written in order to resolve such an issue for one of the applications (Django Blog Zinnia, in particular), and to allow the author to set up his own avatar/identicon service. It was developed to be used in combination with Pydenticon library for generating identicons.

Features

Django Pydenticon has the following features:

- Uses Pydentcion library for generating the identicons.
- User data used for generating the identicons is read from URL.
- Passed user data can be pre-hashed in order to avoid leakage of important information.
- All aspects of Pydenticon generator can be configured via project settings.
- Some parameters for generated identicons can be overridden per-request.
- Comes with sane default configuration options. No special configuration is necessary beyond installing and enabling the application in project.

Installation

Django Pydenticon can be installed through one of the following methods:

- Using *pip*, which is the easiest and recommended way for production websites.

Requirements

Django Pydenticon depends on the following Python packages:

- Django web framework.
- Pydenticon library, which is used for generating the identicons.

Using pip

In order to install latest stable release of Django Pydenticon using *pip*, run the following command:

```
pip install django-pydenticon
```

In order to install the latest development version of Django Pydenticon from Github, use the following command:

```
pip install -e git+https://github.com/azaghal/django-pydenticon#egg=django_pydenticon
```

Warning: You will need to update the `pip` installation in your virtual environment if you get the following error while running the above command:

```
AttributeError: 'NoneType' object has no attribute 'skip_requirements_regex'
```

You can update `pip` to latest version with:

```
pip install -U pip
```

After this you should proceed to *configure your Django installation*.

Configuring your Django installation

Once Django Pydenticon has been installed, you need to perform the following steps in order to make it available inside of your Django project:

1. Edit your project's settings configuration file (`settings.py`), and update the `INSTALLED_APPS` to include application `django_pydenticon`.
2. Edit your project's URL configuration file (`urls.py`), and add the following line to top of the file:

   ```
   import django_pydenticon.urls
   ```

3. Edit your project's URL configuration file (`urls.py`), and add the following line to the `urlpatterns` setting:

   ```
   url(r'^identicon/', include(django_pydenticon.urls.get_patterns())),
   ```

Note: It is not mandatory to use `identicon/` as prefix. You can use any prefix as with any other Django application.

After this the Django Pydenticon application will be available under the `/identicon/` path (relative to your Django project's base URL), or under any custom prefix path you have selected for deploying the application.

Where to go next?

After Django Pydenticon has been installed, you should learn *how to use the application*, and may also be intersted to change one of default *configuration options*.

Warning: It is highly recommended to have a look at documentation covering *privacy* if you have not done so before. The chapter covers some common privacy issues when using personally-identifiable information for generating identicons (like e-mails or names).

Usage

Django Pydenticon is targeted at developers who wish to integrate an identicon service in their Django projects. This chapter covers details on how an identicon image is served, and how to integrate Django Pydenticon with other applications.

Requesting identicons

Identicon images are served through specially formatted URL. Whenever such URL is submitted to Django Pydenticon application, an identicon image is created on the fly.

The format of URL is `/image/USER_DATA` (relative to prefix URL assigned for the application), where **USER_DATA** can be either in hashed or raw format. For example, if Django Pydenticon application is reachable under `/identicon/`, identicon images can be requested using the following URLs:

- `/identicon/image/somedataforhashing` (raw data)
- `/identicon/image/55d207ea47247b375dc1f495517f1332` (pre-hashed data using *md5*)

Warning: Keep in mind that if user data is submitted in pre-hashed form, the digest used should match with the digest configured for Django Pydenticon application. If digest does not match, the user data will be treated as any other user data, and it will be hashed once again.

URL instance namespaces

When resolving Django Pydenticon URLs, you should always use the URL names in conjunction with application instance namespace.

Default application instance namespace is `django_pydenticon`. Alternative instance namespace can be specified by passing an (optional) argument to `django_pydenticons.urls.get_patterns` function.

For example, if default namespace is in use, the `image` URL would be referenced as `django_pydenticon:image` in template tag `url` or function call `reverse`.

Generating identicon URLs in templates

If the data (whether raw or hashed) is available in template's context, an identicon URL can be easily generated from within the template itself. This can be achieved via `url` tag.

The URL for serving the identicons is named `image`. It should always be referenced in conjunction with an application instance namespace. The application namespace defaults to `django_pydenticon`, unless custom instance namespace is passed when including the application URLs via `django_pydenticon.urls.get_patterns`. In case of default namespace, that means the URL would be referenced to as `django_pydenticon:image`.

For example, let's say that it's necessary to show an identicon based on username next to every comment. Related template snippet could look something similar to the following:

```
<ul>
{% for comment in comments %}
  <li><img src="{% url 'django_pydenticon:image' comment.user.username %}"/>{{␣
↪comment.text }}</li>
{% endfor %}
</ul>
```

Generating identicon URLs programatically

The URLs can be generated programtically, using Python code. Afterwards those URLs can be either passed into template's rendering context, or used inside of code for whatever other purposes. This is achieved by using the `reverse` URL resolver (from `django.core.urlresolvers`).

The URL for serving the identicons is named `image`. It should always be referenced in conjunction with an application instance namespace. The application namespace defaults to `django_pydenticon`, unless custom instance namespace is passed when including the application URLs via `django_pydenticon.urls.get_patterns`. In case of default namespace, that means the URL would be referenced to as `django_pydenticon:image`.

For example, let's say that it's necessary to show an identicon based on username next to every comment. A special context variable could be passed into template that would contain a list of comments, where each comment consists out of identicon URL and comment itself. The Python code could look something similar to:

```
comments_context = []

for comment in comments:
    identicon_url = reverse("django_pydenticon:image",
                            kwargs={"data": comment.user.username})
    comments_context.append({"text": comment.text,
                             "identicon_url": identicon_url})

return render_to_response('myapp/comments.html',
                          {"comments": comments_context})
```

With the above context set-up, the `myapp/comments.html` template could contain a snippet similar to:

```
<ul>
{% for comment in comments %}
  <li><img src="{{ comment.identicon_url }}"/>{{ comment.text }}</li>
{% endfor %}
</ul>
```

Overriding identicon parameters

By default, the identicon generator will use parameters from project settings for each request, falling back to application defaults if none were defined. In addition to this static configuration, some parameters can be overridden per request.

Per-request identicon generator parameters are passed via *GET* parameters. The following *GET* parameters are available:

w Specifies the width of generated identicon image in pixels. Overrides the `PYDENTICON_WIDTH` configuration option.

h Specifies the height of generated identicon image in pixels. Overrides the `PYDENTICON_HEIGHT` configuration option.

f Specifies the format of generated identicon. Overrides the `PYDENTICON_FORMAT` configuration option.

p Specifies the padding that will be added to the generated identicon image. The value should be provided as 4 comma-separated positive integers.

i Specifies whether the background and foreground colour in generated identicon should be inverted (swapped) or not. The value passed for this parameter should be `true` or `false`.

Passing an invalid parameter value via *GET* parameter will result in a `SuspiciousOperation` exception being raised.

For example, the following request would generate an identicon with width of `320`, height of `240`, format `PNG`, padding (top, bottom, left, right) of `10, 10, 20, 20`, and with inverted foreground and background colours:

```
/identicon/image/somedata?w=320&h=240&f=png&p=10,10,20,20&i=true
```

Privacy

Generating identicons thorugh Django Pydenticon using raw user data may have undesirable consequences on privacy if the data used is meant to be ketp as a secret.

This privacy issue can in particular arise if using data like usernames, e-mails, or real names of users for generating avatars in publicly-accessible websites.

As a rule-of-thumb, you should **never**, **ever** pass such data raw into the identicon URL. This approach would leak the confidential information in plain text to any interested parties. Instead, calculate a digest of the raw data, and pass the hex digest as part of the URL instead.

Note: In some cases you may opt to pass raw data. For example, if usernames are visible as part of posted comments, they're probably already scrapeable, and having them as part of identicon URL won't hide them anyway.

Additionally, the default digest algorithm (*MD5*) may not be safe enough for such data. Even in case where a stronger digest algorithm is used, an attacker might attempt to generate rainbow tables, and scrape the web pages hashed data contained within identicon URLs.

There's two feasible approaches to resolve this:

- Always apply *salt* to user-identifiable data before calculating a hex digest. This can hugely reduce the efficiency of brute force attacks based on rainbow tables (although it will not mitigate it completely).
- Instead of hashing the user-identifiable data itself, every time you need to do so, create some random data instead, hash that random data, and store it for future use (cache it), linking it to the original data that it was generated for. This way the hex digest being put as part of an image link into HTML pages is not derived in any way from the original data, and can therefore not be used to reveal what the original data was.

Keep in mind that using identicons will inevitably still allow people to track someone's posts across your website. Identicons will effectively automatically create pseudonyms for people posting on your website. If that may pose a problem, it might be better not to use identicons at all.

Finally, small summary of the points explained above:

- Always use hex digests in identicon URLs.
- Instead of using privately identifiable data for generating the hex digest, use randmoly generated data, and associate it with privately identifiable data. This way hex digest cannot be traced back to the original data through brute force or rainbow tables.
- If unwilling to generate and store random data, at least make sure to use salt when hashing privately identifiable data.

Configuration

A number of configuration options can be set in Django project that affect the identicon generation. Each configuration option comes with a default value that's used if it's not specified explicitly in project settings.

The application will verify configuration options, and raise an `ImproperlyConfigured` exception in case of a problem.

`PYDENTICON_ROWS`

Specifies how many *block* rows a generated identicon should have. The value should be a positive integer.

Default value: `5`

`PYDENTICON_COLUMNS`

Specifies how many *block* columns a generated identicon should have. The value should be a positive integer.

Default value: `5`

`PYDENTICON_WIDTH`

Specifies the width of generated identicon images in pixels (without padding). The value should be a positive integer.

Default value: `200`

`PYDENTICON_HEIGHT`

Specifies the height of generated identicon images in pixels (without padding). The value should be a positive integer.

Default value: `200`

`PYDENTICON_PADDING`

Specifies the padding that will be added to the generated identicon image. The padding is specified as tuple containing 4 elements, where each element is a positive integer.

Each element of the tuple is used for padding the identicon image along one of the edges. The order is: *top*, *bottom*, *left*, *right*.

Default value: `(20, 20, 20, 20)`

`PYDENTICON_FORMAT`

Specifies the default format of the generated identicons. The value should be a string. Supported values are:

- `"png"` (for PNG images)
- `"ascii"` (for ASCII/textual representation of identicon)

Default value: `"png"`

`PYDENTICON_FOREGROUND`

Specifies a list or tuple of foreground colours that should be used when generating the identicons. Each element of list/tuple should be a string conformant to colour specification from the Pillow library.

In addition to regular RGB, you can also use the RGBA (RGB with alpha channel) scheme to introduce partial or complete transparency when specyfing colour. Alpha channel value ranges between 0 (full transparency) to 255 (no transparency). For example `rgba(224,224,224,0)` will result in fully transparent background, while `rgba(224,224,244,128)` will result in approximatelly 50% transparency.

Default value: `("rgb(45,79,255)", "rgb(254,180,44)", "rgb(226,121,234)", "rgb(30,179,253)", "rgb(232,77,65)", "rgb(49,203,115)", "rgb(141,69,170)")`

`PYDENTICON_BACKGROUND`

Specifies a (single) background colour that should be used when generating the identicons. This should be a string conformant to colour specification from the Pillow library. The value should be a string.

In addition to regular RGB, you can also use the RGBA (RGB with alpha channel) scheme to introduce partial or complete transparency when specyfing colour. Alpha channel value ranges between 0 (full transparency) to 255 (no transparency). For example `rgba(224,224,224,0)` will result in fully transparent foreground, while `rgba(224,224,244,128)` will result in approximatelly 50% transparency. Different foreground colours can have different values for alpha channel.

Default value: `"rgb(224,224,224)"`

`PYDENTICON_DIGEST`

Specifies digest class that should be used for generating the identicons. Digest class should support accepting a single constructor argument for passing the data on which the digest will be run. Instances of the class should also support a single hexdigest() method that should return a digest of passed data as a hex string. The value should be a callable.

Default value: `hashlib.md5`

`PYDENTICON_INVERT`

Specifies whether the background and foreground colour in generated identicons should be inverted (swapped) or not. The value should be a boolean (`True` or `False`).

Default value: `False`

API Reference

Views

This section lists documentation for all views available in Django Pydenticon.

`django_pydenticon.views.`**`image`**`(`*request*, *data*`)`

Generates identicon image based on passed data.

Arguments:

> data - Data which should be used for generating an identicon. This data will be used in order to create a digest which is used for generating the identicon. If the data passed is a hex digest already, the digest will be used as-is.

Returns:

> Identicon image in raw format.

URL

This section lists documentation for all URL-related functions available in Django Pydenticon.

`django_pydenticon.urls.`**`get_patterns`**`(`*instance='django_pydenticon'*`)`

Generates URL patterns for Django Pydenticon application. The return value of this function can be used directly by the django.conf.urls.include function.

Arguments:

> instance - Instance namespace that should be assigned to generated URL patterns.

Returns:

> Tuple consisting out of URL patterns, instance namespace, and application namespace.

Release notes

0.2

Update introduces support for Django 1.8+ and some documentation improvements.

New features:

- DJPYD-7: Update application for use in Django 1.8, 1.9, and Django 1.10

  Minimum requirement for Django is now 1.8.x. Fixes are compatible with Django 1.9.x and 1.10.x as well.

Enhancements:

- DJPYD-8: Update Pydenticon requirement to version 0.3

  Introduced explicit dependency on Pydenticon 0.3, which also introduces ability to handle transparency in PNGs.

0.1

Initial release of Django Pydenticon. Implemented features:

- Serving of identicons through designated URL.
- User data for generating identicons passed via URL.
- Sane configuration defaults for identicon generator for zero-configuration.
- Ability to set parameters of generated identicons.
- Ability to override some of the identicon generation attributes per-request via *GET* parameters.
- Full documentation covering installation, usage, privacy. API reference included as well.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index