Documentation for django-analytical

Release 3.1.0

Joost Cassee

May 02, 2023

Contents

The django-analytical application integrates analytics services into a Django project.

Package https://pypi.python.org/pypi/django-analytical/

Source https://github.com/jazzband/django-analytical

CHAPTER 1

Overview

Using an analytics service with a Django project means adding Javascript tracking code to the project templates. Of course, every service has its own specific installation instructions. Furthermore, you need to include your unique identifiers, which then end up in the templates. Not very nice.

This application hides the details of the different analytics services behind a generic interface, and keeps personal information and configuration out of the templates. Its goal is to make the basic set-up very simple, while allowing advanced users to customize tracking. Each service is set up as recommended by the services themselves, using an asynchronous version of the Javascript code if possible.

To get a feel of how django-analytical works, check out the *Tutorial*.

CHAPTER 2

Contents

2.1 Tutorial

This tutorial will show you how to install and configure django-analytical for basic tracking, and then briefly touch on two common customization issues: visitor identification and custom data tracking.

Suppose your Django website provides information about the IPv4 to IPv6 transition. Visitors can discuss their problems and help each other make the necessary changes to their network infrastructure. You want to use two different analytics services:

- *Clicky* for detailed traffic analysis
- *Crazy Egg* to see where visitors click on your pages

At the end of this tutorial, the project will track visitors on both Clicky and Crazy Egg, identify authenticated users and add extra tracking data to segment mouse clicks on Crazy Egg based on whether visitors are using IPv4 or IPv6.

2.1.1 Setting up basic tracking

To get started with django-analytical, the package must first be installed. You can download and install the latest stable package from the Python Package Index automatically by using `easy_install`:

```
$ easy_install django-analytical
```

For more ways to install django-analytical, see *Installing the Python package*.

After you install django-analytical, you need to add it to the list of installed applications in the `settings.py` file of your project:

```
INSTALLED_APPS = [
    ...
    'analytical',
    ...
]
```

Then you have to add the general-purpose django-analytical template tags to your base template:

```
{% load analytical %}
<!DOCTYPE ... >
<html>
    <head>
        {% analytical_head_top %}

        ...

        {% analytical_head_bottom %}
    </head>
    <body>
        {% analytical_body_top %}

        ...

        {% analytical_body_bottom %}
    </body>
</html>
```

Finally, you need to configure the Clicky Site ID and the Crazy Egg account number. Add the following to your project `settings.py` file (replacing the `x`'s with your own codes):

```
CLICKY_SITE_ID = 'xxxxxxxx'
CRAZY_EGG_ACCOUNT_NUMBER = 'xxxxxxxx'
```

The analytics services are now installed. If you run Django with these changes, both Clicky and Crazy Egg will be tracking your visitors.

2.1.2 Identifying authenticated users

Suppose that when your visitors post questions on IPv6 or tell others about their experience with the transition, they first log in through the standard Django authentication system. Clicky can identify and track individual visitors and you want to use this feature.

If django-analytical template tags detect that the current user is authenticated, they will automatically include code to send the username to services that support this feature. This only works if the template context has the current user in the `user` or `request.user` context variable. If you use a `RequestContext` to render templates (which is recommended anyway) and have the `django.contrib.auth.context_processors.auth` context processor in the `TEMPLATE_CONTEXT_PROCESSORS` setting (which is default), then this identification works without having to make any changes.

For more detailed information on automatic identification, and how to disable or override it, see *Identifying authenticated users*.

2.1.3 Adding custom tracking data

Suppose that you think that visitors who already have IPv6 use the website in a different way from those still on IPv4. You want to test this hypothesis by segmenting the Crazy Egg heatmaps based on the IP protocol version.

In order to filter on protocol version in Crazy Egg, you need to include the visitor IP protocol version in the Crazy Egg tracking code. The easiest way to do this is by using a context processor:

```
def track_ip_proto(request):
    addr = request.META.get('HTTP_X_FORWARDED_FOR', '')
    if not addr:
        addr = request.META.get('REMOTE_ADDR', '')
    if ':' in addr:
        proto = 'ipv6'
    else:
        proto = 'ipv4'  # assume IPv4 if no information
    return {'crazy_egg_var1': proto}
```

Use a `RequestContext` when rendering templates and add the `'track_ip_proto'` to `TEMPLATE_CONTEXT_PROCESSORS`. In Crazy Egg, you can now select *User Var1* in the overlay or confetti views to see whether visitors using IPv4 behave differently from those using IPv6.

This concludes the tutorial. For information about setting up, configuring and customizing the different analytics services, see *Features and customization* and *Services*.

2.2 Installation and configuration

Integration of your analytics service is very simple. There are four steps: installing the package, adding it to the list of installed Django applications, adding the template tags to your base template, and configuring the services you use in the project settings.

1. *Installing the Python package*
2. *Installing the Django application*
3. *Adding the template tags to the base template*
4. *Enabling the services*

2.2.1 Installing the Python package

To install django-analytical the `analytical` package must be added to the Python path. You can install it directly from PyPI using `easy_install`:

```
$ easy_install django-analytical
```

You can also install directly from source. Download either the latest stable version from PyPI or any release from GitHub, or use Git to get the development code:

```
$ git clone https://github.com/jazzband/django-analytical.git
```

Then install the package by running the setup script:

```
$ cd django-analytical
$ python setup.py install
```

2.2.2 Installing the Django application

After you installed django-analytical, add the `analytical` Django application to the list of installed applications in the `settings.py` file of your project:

```
INSTALLED_APPS = [
    ...
    'analytical',
    ...
]
```

2.2.3 Adding the template tags to the base template

Because every analytics service uses own specific Javascript code that should be added to the top or bottom of either the head or body of the HTML page, django-analytical provides four general-purpose template tags that will render the code needed for the services you are using. Your base template should look like this:

```
{% load analytical %}
<!DOCTYPE ... >
<html>
    <head>
        {% analytical_head_top %}

        ...

        {% analytical_head_bottom %}
    </head>
    <body>
        {% analytical_body_top %}

        ...

        {% analytical_body_bottom %}
    </body>
</html>
```

Instead of using the generic tags, you can also just use tags specific for the analytics service(s) you are using. See *Services* for documentation on using individual services.

2.2.4 Enabling the services

Without configuration, the template tags all render the empty string. Services are configured in the project `settings.py` file. The settings required to enable each service are listed here:

- *Chartbeat*:

  ```
  CHARTBEAT_USER_ID = '12345'
  ```

- *Clickmap*:

  ```
  CLICKMAP_TRACKER_CODE = '12345678....912'
  ```

- *Clicky*:

  ```
  CLICKY_SITE_ID = '12345678'
  ```

- *Crazy Egg*:

  ```
  CRAZY_EGG_ACCOUNT_NUMBER = '12345678'
  ```

- *Facebook Pixel*:

```
FACEBOOK_PIXEL_ID = '1234567890'
```

- *Gaug.es*:

```
GAUGES_SITE_ID = '0123456789abcdef0123456789abcdef'
```

- *Google Analytics (legacy)*:

```
GOOGLE_ANALYTICS_PROPERTY_ID = 'UA-1234567-8'
```

- *Google Analytics (gtag.js)*:

```
GOOGLE_ANALYTICS_GTAG_PROPERTY_ID = 'UA-1234567-8'
```

- *Google Analytics (analytics.js)*:

```
GOOGLE_ANALYTICS_JS_PROPERTY_ID = 'UA-12345678-9'
```

- *HubSpot*:

```
HUBSPOT_PORTAL_ID = '1234'
HUBSPOT_DOMAIN = 'somedomain.web101.hubspot.com'
```

- *Intercom*:

```
INTERCOM_APP_ID = '0123456789abcdef0123456789abcdef01234567'
```

- *KISSinsights*:

```
KISS_INSIGHTS_ACCOUNT_NUMBER = '12345'
KISS_INSIGHTS_SITE_CODE = 'abc'
```

- *KISSmetrics*:

```
KISS_METRICS_API_KEY = '0123456789abcdef0123456789abcdef01234567'
```

- *Lucky Orange*:

```
LUCKYORANGE_SITE_ID = '123456'
```

- *Matomo (formerly Piwik)*:

```
MATOMO_DOMAIN_PATH = 'your.matomo.server/optional/path'
MATOMO_SITE_ID = '123'
```

- *Mixpanel*:

```
MIXPANEL_API_TOKEN = '0123456789abcdef0123456789abcdef'
```

- *Olark*:

```
OLARK_SITE_ID = '1234-567-89-0123'
```

- *Optimizely*:

```
OPTIMIZELY_ACCOUNT_NUMBER = '1234567'
```

- *Performable*:

```
PERFORMABLE_API_KEY = '123abc'
```

- *Rating@Mail.ru*:

```
RATING_MAILRU_COUNTER_ID = '1234567'
```

- *SnapEngage*:

```
SNAPENGAGE_WIDGET_ID = 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX'
```

- *Woopra*:

```
WOOPRA_DOMAIN = 'abcde.com'
```

- *Yandex.Metrica*:

```
YANDEX_METRICA_COUNTER_ID = '12345678'
```

The django-analytical application is now set-up to track visitors. For information about identifying users, further configuration and customization, see *Features and customization*.

2.3 Features and customization

The django-analytical application sets up basic tracking without any further configuration. This page describes extra features and ways in which behavior can be customized.

2.3.1 Internal IP addresses

Visits by the website developers or internal users are usually not interesting. The django-analytical will comment out the service initialization code if the client IP address is detected as one from the *ANALYTICAL_INTERNAL_IPS* setting. The default value for this setting is `INTERNAL_IPS`.

Example:

```
ANALYTICAL_INTERNAL_IPS = ['192.168.1.45', '192.168.1.57']
```

Note: The template tags can only access the visitor IP address if the HTTP request is present in the template context as the `request` variable. For this reason, the *ANALYTICAL_INTERNAL_IPS* setting only works if you add this variable to the context yourself when you render the template, or you use the `RequestContext` and add `'django.core.context_processors.request'` to the list of context processors in the `TEMPLATE_CONTEXT_PROCESSORS` setting.

2.3.2 Identifying authenticated users

Some analytics services can track individual users. If the visitor is logged in through the standard Django authentication system and the current user is accessible in the template context, the username can be passed to the analytics services that support identifying users. This feature is configured by the *ANALYTICAL_AUTO_IDENTIFY* setting and is enabled by default. To disable:

```
ANALYTICAL_AUTO_IDENTIFY = False
```

Note: The template tags can only access the visitor username if the Django user is present in the template context either as the `user` variable, or as an attribute on the HTTP request in the `request` variable. Use a `RequestContext` to render your templates and add `'django.contrib.auth.context_processors.auth'` or `'django.core.context_processors.request'` to the list of context processors in the `TEMPLATE_CONTEXT_PROCESSORS` setting. (The first of these is added by default.) Alternatively, add one of the variables to the context yourself when you render the template.

Changing the identity

If you want to override the identity of the logged-in user that the various providers send you can do it by setting the `analytical_identity` context variable in your view code:

```
context = RequestContext({'analytical_identity': user.uuid})
return some_template.render(context)
```

or in the template:

```
{% with analytical_identity=request.user.uuid|default:None %}
    {% analytical_head_top %}
{% endwith %}
```

or by implementing a context processor, e.g.

```
# FILE: myproject/context_processors.py
from django.conf import settings

def get_identity(request):
    return {
        'analytical_identity': 'some-value-here',
    }

# FILE: myproject/settings.py
TEMPLATES = [
    {
        'OPTIONS': {
            'context_processors': [
                'myproject.context_processors.get_identity',
            ],
        },
    },
]
```

That allows you as a developer to leave your view code untouched and make sure that the variable is injected for all templates.

If you want to change the identity only for specific provider use the `*_identity` context variable, where the `*` prefix is the module name of the specific provider.

2.4 Services

This section describes what features are supported by the different analytics services. To start using a service, you can either use the generic installation instructions (see *Installation and configuration*), or add service-specific tags to your templates.

If you would like to have another analytics service supported by django-analytical, please create an issue on the project issue tracker. See also *Helping out*.

Currently supported services:

2.4.1 Chartbeat – traffic analysis

Chartbeat provides real-time analytics to websites and blogs. It shows visitors, load times, and referring sites on a minute-by-minute basis. The service also provides alerts the second your website crashes or slows to a crawl.

Installation

To start using the Chartbeat integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Chartbeat template tags to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Chartbeat tracking code is inserted into templates using template tags. At the top of the template, load the `chartbeat` template tag library. Then insert the `chartbeat_top` tag at the top of the head section, and the `chartbeat_bottom` tag at the bottom of the body section:

```
{% load chartbeat %}
<html>
<head>
{% chartbeat_top %}

...

{% chartbeat_bottom %}
</body>
</html>
```

Because these tags are used to measure page loading time, it is important to place them as close as possible to the start and end of the document.

Configuration

Before you can use the Chartbeat integration, you must first set your User ID.

Setting the User ID

Your Chartbeat account has a unique User ID. You can find your User ID by visiting the Chartbeat Add New Site page. The second code snippet contains a line that looks like this:

```
var _sf_async_config={uid:XXXXX,domain:"YYYYYYYYYY"};
```

Here, `XXXXX` is your User ID. Set `CHARTBEAT_USER_ID` in the project `settings.py` file:

```
CHARTBEAT_USER_ID = 'XXXXX'
```

If you do not set a User ID, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `CHARTBEAT_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Setting the domain

The Javascript tracking code can send the website domain to Chartbeat. If you use multiple subdomains this enables you to treat them as one website in Chartbeat. If your project uses the sites framework, the domain name of the current `Site` will be passed to Chartbeat automatically. You can modify this behavior using the `CHARTBEAT_AUTO_DOMAIN` setting:

```
CHARTBEAT_AUTO_DOMAIN = False
```

Alternatively, you set the domain through the `chartbeat_domain` context variable when you render the template:

```
context = RequestContext({'chartbeat_domain': 'example.com'})
return some_template.render(context)
```

It is annoying to do this for every view, so you may want to set it in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def chartbeat(request):
    return {'chartbeat_domain': 'example.com'}
```

The context domain overrides the domain from the current site. If no domain is set, either explicitly or implicitly through the sites framework, then no domain is sent, and Chartbeat will detect the domain name from the URL. If your website uses just one domain, this will work just fine.

Thanks go to Chartbeat for their support with the development of this application.

2.4.2 Clickmap – visual click tracking

Clickmap is a real-time heatmap tool to track mouse clicks and scroll paths of your website visitors. Gain intelligence about what's hot and what's not, and optimize your conversion with Clickmap.

Installation

To start using the Clickmap integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Clickmap template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Clickmap Javascript code is inserted into templates using a template tag. Load the `clickmap` template tag library and insert the `clickmap` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load clickmap %}
...
{% clickmap %}
</body>
</html>
```

Configuration

Before you can use the Clickmap integration, you must first set your Clickmap Tracker ID. If you don't have a Clickmap account yet, sign up to get your Tracker ID.

Setting the Tracker ID

Clickmap gives you a unique Tracker ID, and the `clickmap` tag will include it in the rendered Javascript code. You can find your Tracker ID clicking the link named "Tracker" in the dashboard of your Clickmap account. Set `CLICKMAP_TRACKER_ID` in the project `settings.py` file:

```
CLICKMAP_TRACKER_ID = 'XXXXXXXX'
```

If you do not set an Tracker ID, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the *`ANALYTICAL_INTERNAL_IPS`* setting (which is `INTERNAL_IPS` by default,) the tracking code is commented out. See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.3 Clicky – traffic analysis

Clicky is an online web analytics tool. It is similar to Google Analytics in that it provides statistics on who is visiting your website and what they are doing. Clicky provides its data in real time and is designed to be very easy to use.

Installation

To start using the Clicky integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Clicky template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Clicky tracking code is inserted into templates using a template tag. Load the `clicky` template tag library and insert the `clicky` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load clicky %}
...
{% clicky %}
</body>
</html>
```

Configuration

Before you can use the Clicky integration, you must first set your website Site ID. You can also customize the data that Clicky tracks.

Setting the Site ID

Every website you track with Clicky gets its own Site ID, and the `clicky` tag will include it in the rendered Javascript code. You can find the Site ID in the *Info* tab of the website *Preferences* page, in your Clicky account. Set `CLICKY_SITE_ID` in the project `settings.py` file:

```
CLICKY_SITE_ID = 'XXXXXXXX'
```

If you do not set a Site ID, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `CLICKY_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Custom data

As described in the Clicky customized tracking documentation page, the data that is tracked by Clicky can be customized by setting the `clicky_custom` Javascript variable before loading the tracking code. Using template context variables, you can let the `clicky` tag pass custom data to Clicky automatically. You can set the context variables in your view when you render a template containing the tracking code:

```
context = RequestContext({'clicky_title': 'A better page title'})
return some_template.render(context)
```

It is annoying to do this for every view, so you may want to set custom properties in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def clicky_global_properties(request):
    return {'clicky_timeout': 10}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Here is a table with the most important variables. All variables listed on the customized tracking documentation page can be set by replacing `clicky_custom.` with `clicky_`.

Context variable	Clicky property	Description
`clicky_session`	session	Session data. A dictionary containing `username` and/or `group` keys.
`clicky_goal`	goal	A succeeded goal. A dictionary containing `id` and optionally `revenue` keys.
`clicky_split`	split	Split testing page version. A dictionary containing `name`, `version` and optionally `goal` keys.
`clicky_href`	href	The URL as tracked by Clicky. Default is the page URL.
`clicky_title`	title	The page title as tracked by Clicky. Default is the HTML title.

Identifying authenticated users

If you have not set the session property explicitly, the username of an authenticated user is passed to Clicky automatically. See *Identifying authenticated users*.

Thanks go to Clicky for their support with the development of this application.

2.4.4 Crazy Egg – visual click tracking

Crazy Egg is an easy to use hosted web application that visualizes website clicks using heatmaps. It allows you to discover the areas of web pages that are most important to your visitors.

Installation

To start using the Crazy Egg integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Crazy Egg template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Crazy Egg tracking code is inserted into templates using a template tag. Load the `crazy_egg` template tag library and insert the `crazy_egg` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load crazy_egg %}
...
{% crazy_egg %}
</body>
</html>
```

Configuration

Before you can use the Crazy Egg integration, you must first set your account number. You can also segment the click analysis through user variables.

Setting the account number

Crazy Egg gives you a unique account number, and the `crazy egg` tag will include it in the rendered Javascript code. You can find your account number by clicking the link named "What's my code?" in the dashboard of your Crazy Egg account. Set `CRAZY_EGG_ACCOUNT_NUMBER` in the project `settings.py` file:

```
CRAZY_EGG_ACCOUNT_NUMBER = 'XXXXXXXX'
```

If you do not set an account number, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `CRAZY_EGG_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

User variables

Crazy Egg can segment clicks based on user variables. If you want to set a user variable, use the context variables `crazy_egg_var1` through `crazy_egg_var5` when you render your template:

```
context = RequestContext({'crazy_egg_var1': 'red',
                          'crazy_egg_var2': 'male'})
return some_template.render(context)
```

If you use the same user variables in different views and its value can be computed from the HTTP request, you can also set them in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def track_admin_role(request):
    if request.user.is_staff():
        role = 'staff'
    else:
        role = 'visitor'
    return {'crazy_egg_var3': role}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

The work on Crazy Egg was made possible by Bateau Knowledge. Thanks go to Crazy Egg for their support with the development of this application.

2.4.5 Facebook Pixel – advertising analytics

Facebook Pixel is Facebook's tool for conversion tracking, optimisation and remarketing.

Installation

To start using the Facebook Pixel integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Facebook Pixel template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Facebook Pixel code is inserted into templates using template tags. Because every page that you want to track must have the tag, it is useful to add it to your base template. At the top of the template, load the `facebook_pixel` template tag library. Then insert the `facebook_pixel_head` tag at the bottom of the head section, and optionally insert the `facebook_pixel_body` tag at the bottom of the body section:

```
{% load facebook_pixel %}
<html>
<head>
...
{% facebook_pixel_head %}
</head>
<body>
...
{% facebook_pixel_body %}
</body>
</html>
```

Note: The `facebook_pixel_body` tag code will only be used for browsers with JavaScript disabled. It can be omitted if you don't need to support them.

Configuration

Before you can use the Facebook Pixel integration, you must first set your Pixel ID.

Setting the Pixel ID

Each Facebook Adverts account you have can have a Pixel ID, and the `facebook_pixel` tags will include it in the rendered page. You can find the Pixel ID on the "Pixels" section of your Facebook Adverts account. Set `FACEBOOK_PIXEL_ID` in the project `settings.py` file:

```
FACEBOOK_PIXEL_ID = 'XXXXXXXXXX'
```

If you do not set a Pixel ID, the code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `FACEBOOK_PIXEL_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.6 Gaug.es – Real-time tracking

Gaug.es is an easy way to implement real-time tracking for multiple websites.

Installation

To start using the Gaug.es integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Gaug.es template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Gaug.es Javascript code is inserted into templates using a template tag. Load the `gauges` template tag library and insert the `gauges` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the top of the HTML head:

```
{% load gauges %}
<html>
<head>
{% gauges %}
...
```

Configuration

Before you can use the Gaug.es integration, you must first set your site id.

Setting the site id

Gaug.es gives you a unique site id, and the `gauges` tag will include it in the rendered Javascript code. You can find your site id by clicking the *Tracking Code* link when logged into the on the gaug.es website. A page will display containing HTML code looking like this:

```
<script type="text/javascript">
  var _gauges = _gauges || [];
  (function() {
    var t   = document.createElement('script');
    t.type  = 'text/javascript';
    t.async = true;
    t.id    = 'gauges-tracker';
    t.setAttribute('data-site-id', 'XXXXXXXXXXXXXXXXXXXXXXX');
    t.src = '//secure.gaug.es/track.js';
    var s = document.getElementsByTagName('script')[0];
    s.parentNode.insertBefore(t, s);
  })();
</script>
```

The code `XXXXXXXXXXXXXXXXXXXXXXX` is your site id. Set `GAUGES_SITE_ID` in the project `settings.py` file:

```
GAUGES_SITE_ID = 'XXXXXXXXXXXXXXXXXXXXXXX'
```

If you do not set an site id, the Javascript code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the *`ANALYTICAL_INTERNAL_IPS`* setting (which is `INTERNAL_IPS`

by default,) the tracking code is commented out. See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.7 Google Analytics (legacy) – traffic analysis

Google Analytics is the well-known web analytics service from Google. The product is aimed more at marketers than webmasters or technologists, supporting integration with AdWords and other e-commence features.

Installation

To start using the Google Analytics (legacy) integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Google Analytics template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Google Analytics tracking code is inserted into templates using a template tag. Load the `google_analytics` template tag library and insert the `google_analytics` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load google_analytics %}
<html>
<head>
...
{% google_analytics %}
</head>
...
```

Configuration

Before you can use the Google Analytics integration, you must first set your website property ID. If you track multiple domains with the same code, you also need to set-up the domain. Finally, you can add custom segments for Google Analytics to track.

Setting the property ID

Every website you track with Google Analytics gets its own property ID, and the `google_analytics` tag will include it in the rendered Javascript code. You can find the web property ID on the overview page of your account. Set `GOOGLE_ANALYTICS_PROPERTY_ID` in the project `settings.py` file:

```
GOOGLE_ANALYTICS_PROPERTY_ID = 'UA-XXXXXX-X'
```

If you do not set a property ID, the tracking code will not be rendered.

Tracking multiple domains

The default code is suitable for tracking a single domain. If you track multiple domains, set the `GOOGLE_ANALYTICS_TRACKING_STYLE` setting to one of the `analytical.templatetags.google_analytics.TRACK_*` constants:

Constant	Value	Description
TRACK_SINGLE_DOMAIN	1	Track one domain.
TRACK_MULTIPLE_SUBDOMAINS	2	Track multiple subdomains of the same top domain (e.g. *fr.example.com* and *nl.example.com*).
TRACK_MULTIPLE_DOMAINS	3	Track multiple top domains (e.g. *example.fr* and *example.nl*).

As noted, the default tracking style is TRACK_SINGLE_DOMAIN.

When you track multiple (sub)domains, django-analytical needs to know what domain name to pass to Google Analytics. If you use the contrib sites app, the domain is automatically picked up from the current Site instance. Otherwise, you may either pass the domain to the template tag through the context variable google_analytics_domain (fallback: analytical_domain) or set it in the project settings.py file using GOOGLE_ANALYTICS_DOMAIN (fallback: ANALYTICAL_DOMAIN).

Display Advertising

Display Advertising allows you to view Demographics and Interests reports, add Remarketing Lists and support DoubleClick Campain Manager integration.

You can enable Display Advertising features by setting the GOOGLE_ANALYTICS_DISPLAY_ADVERTISING configuration setting:

```
GOOGLE_ANALYTICS_DISPLAY_ADVERTISING = True
```

By default, display advertising features are disabled.

Tracking site speed

You can view page load times in the Site Speed report by setting the GOOGLE_ANALYTICS_SITE_SPEED configuration setting:

```
GOOGLE_ANALYTICS_SITE_SPEED = True
```

By default, page load times are not tracked.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the GOOGLE_ANALYTICS_INTERNAL_IPS setting, the tracking code is commented out. It takes the value of *ANALYTICAL_INTERNAL_IPS* by default (which in turn is INTERNAL_IPS by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Custom variables

As described in the Google Analytics custom variables documentation page, you can define custom segments. Using template context variables google_analytics_var1 through google_analytics_var5, you can let the google_analytics tag pass custom variables to Google Analytics automatically. You can set the context variables in your view when your render a template containing the tracking code:

```
context = RequestContext({'google_analytics_var1': ('gender', 'female'),
                          'google_analytics_var2': ('visit', '1', SCOPE_SESSION)})
return some_template.render(context)
```

The value of the context variable is a tuple *(name, value, [scope])*. The scope parameter is one of the `analytical.templatetags.google_analytics.SCOPE_*` constants:

Constant	Value	Description
SCOPE_VISITOR	1	Distinguishes categories of visitors across multiple sessions.
SCOPE_SESSION	2	Distinguishes different visitor experiences across sessions.
SCOPE_PAGE	3	Defines page-level activity.

The default scope is `SCOPE_PAGE`.

You may want to set custom variables in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def google_analytics_segment_language(request):
    try:
        return {'google_analytics_var3': request.LANGUAGE_CODE}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Anonymize IPs

You can enable the IP anonymization feature by setting the `GOOGLE_ANALYTICS_ANONYMIZE_IP` configuration setting:

```
GOOGLE_ANALYTICS_ANONYMIZE_IP = True
```

This may be mandatory for deployments in countries that have a firm policies concerning data privacy (e.g. Germany).

By default, IPs are not anonymized.

Sample Rate

You can configure the Sample Rate feature by setting the `GOOGLE_ANALYTICS_SAMPLE_RATE` configuration setting:

```
GOOGLE_ANALYTICS_SAMPLE_RATE = 10
```

The value is a percentage and can be between 0 and 100 and can be a string or decimal value of with up to two decimal places.

Site Speed Sample Rate

You can configure the Site Speed Sample Rate feature by setting the `GOOGLE_ANALYTICS_SITE_SPEED_SAMPLE_RATE` configuration setting:

```
GOOGLE_ANALYTICS_SITE_SPEED_SAMPLE_RATE = 10
```

The value is a percentage and can be between 0 and 100 and can be a string or decimal value of with up to two decimal places.

Session Cookie Timeout

You can configure the Session Cookie Timeout feature by setting the `GOOGLE_ANALYTICS_SESSION_COOKIE_TIMEOUT` configuration setting:

```
GOOGLE_ANALYTICS_SESSION_COOKIE_TIMEOUT = 3600000
```

The value is the session cookie timeout in milliseconds or 0 to delete the cookie when the browser is closed.

Visitor Cookie Timeout

You can configure the Visitor Cookie Timeout feature by setting the `GOOGLE_ANALYTICS_VISITOR_COOKIE_TIMEOUT` configuration setting:

```
GOOGLE_ANALYTICS_VISITOR_COOKIE_TIMEOUT = 3600000
```

The value is the visitor cookie timeout in milliseconds or 0 to delete the cookie when the browser is closed.

2.4.8 Google Analytics (gtag.js) – traffic analysis

Google Analytics is the well-known web analytics service from Google. The product is aimed more at marketers than webmasters or technologists, supporting integration with AdWords and other e-commence features. The global site tag (gtag.js) is a JavaScript tagging framework and API that allows you to send event data to Google Analytics, Google Ads, and Google Marketing Platform.

Installation

To start using the Google Analytics integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Google Analytics template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Google Analytics tracking code is inserted into templates using a template tag. Load the `google_analytics_gtag` template tag library and insert the `google_analytics_gtag` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load google_analytics_gtag %}
<html>
<head>
{% google_analytics_gtag %}
...
</head>
...
```

Configuration

Before you can use the Google Analytics integration, you must first set your website property ID. If you track multiple domains with the same code, you also need to set-up the domain. Finally, you can add custom segments for Google Analytics to track.

Setting the property ID

Every website you track with Google Analytics gets its own property ID, and the `google_analytics_gtag` tag will include it in the rendered Javascript code. You can find the web property ID on the overview page of your account. Set `GOOGLE_ANALYTICS_GTAG_PROPERTY_ID` in the project `settings.py` file:

```
GOOGLE_ANALYTICS_GTAG_PROPERTY_ID = 'UA-XXXXXX-X'
```

If you do not set a property ID, the tracking code will not be rendered.

Please note that the accepted Property IDs should be one of the following formats:

- 'UA-XXXXXX-Y'
- 'AW-XXXXXXXXXX'
- 'G-XXXXXXXX'
- 'DC-XXXXXXXX'

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `GOOGLE_ANALYTICS_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Identifying authenticated users

The username of an authenticated user is passed to Google Analytics automatically as the `user_id`. See *Identifying authenticated users*.

According to Google Analytics conditions you should avoid sending Personally Identifiable Information. Using `username` as `user_id` might not be the best option. To avoid that, you can change the identity by setting `google_analytics_gtag_identity` (or `analytical_identity` to affect all providers) context variable:

```
context = RequestContext({'google_analytics_gtag_identity': user.uuid})
return some_template.render(context)
```

or in the template:

```
{% with google_analytics_gtag_identity=request.user.uuid|default:None %}
    {% analytical_head_top %}
{% endwith %}
```

2.4.9 Google Analytics (analytics.js) – traffic analysis

Google Analytics is the well-known web analytics service from Google. The product is aimed more at marketers than webmasters or technologists, supporting integration with AdWords and other e-commence features. The analytics.js library (also known as "the Google Analytics tag") is a JavaScript library for measuring how users interact with your website.

Installation

To start using the Google Analytics integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Google Analytics template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Google Analytics tracking code is inserted into templates using a template tag. Load the `google_analytics_js` template tag library and insert the `google_analytics_js` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load google_analytics_js %}
<html>
<head>
...
{% google_analytics_js %}
</head>
...
```

Configuration

Before you can use the Google Analytics integration, you must first set your website property ID. If you track multiple domains with the same code, you also need to set-up the domain. Finally, you can add custom segments for Google Analytics to track.

Setting the property ID

Every website you track with Google Analytics gets its own property ID, and the `google_analytics_js` tag will include it in the rendered Javascript code. You can find the web property ID on the overview page of your account. Set `GOOGLE_ANALYTICS_JS_PROPERTY_ID` in the project `settings.py` file:

```
GOOGLE_ANALYTICS_JS_PROPERTY_ID = 'UA-XXXXXXXX-X'
```

If you do not set a property ID, the tracking code will not be rendered.

Tracking multiple domains

The default code is suitable for tracking a single domain. If you track multiple domains, set the `GOOGLE_ANALYTICS_TRACKING_STYLE` setting to one of the `analytical.templatetags.google_analytics_js.TRACK_*` constants:

Constant	Value	Description
TRACK_SINGLE_DOMAIN	1	Track one domain.
TRACK_MULTIPLE_SUBDOMAINS	2	Track multiple subdomains of the same top domain (e.g. *fr.example.com* and *nl.example.com*).
TRACK_MULTIPLE_DOMAINS	3	Track multiple top domains (e.g. *example.fr* and *example.nl*).

As noted, the default tracking style is TRACK_SINGLE_DOMAIN.

When you track multiple (sub)domains, django-analytical needs to know what domain name to pass to Google Analytics. If you use the contrib sites app, the domain is automatically picked up from the current Site instance. Otherwise, you may either pass the domain to the template tag through the context variable google_analytics_domain (fallback: analytical_domain) or set it in the project settings.py file using GOOGLE_ANALYTICS_DOMAIN (fallback: ANALYTICAL_DOMAIN).

Display Advertising

Display Advertising allows you to view Demographics and Interests reports, add Remarketing Lists and support DoubleClick Campain Manager integration.

You can enable Display Advertising features by setting the GOOGLE_ANALYTICS_DISPLAY_ADVERTISING configuration setting:

```
GOOGLE_ANALYTICS_DISPLAY_ADVERTISING = True
```

By default, display advertising features are disabled.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the GOOGLE_ANALYTICS_INTERNAL_IPS setting, the tracking code is commented out. It takes the value of *ANALYTICAL_INTERNAL_IPS* by default (which in turn is INTERNAL_IPS by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Custom variables

As described in the Google Analytics custom variables documentation page, you can define custom segments. Using template context variables google_analytics_var1 through google_analytics_var5, you can let the google_analytics_js tag pass custom variables to Google Analytics automatically. You can set the context variables in your view when your render a template containing the tracking code:

```
context = RequestContext({'google_analytics_var1': ('gender', 'female'),
                          'google_analytics_var2': ('visit', 1)})
return some_template.render(context)
```

The value of the context variable is a tuple *(name, value)*.

You may want to set custom variables in a context processor that you add to the TEMPLATE_CONTEXT_PROCESSORS list in settings.py:

```
def google_analytics_segment_language(request):
    try:
        return {'google_analytics_var3': request.LANGUAGE_CODE}
```

(continues on next page)

(continued from previous page)

```
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Anonymize IPs

You can enable the IP anonymization feature by setting the `GOOGLE_ANALYTICS_ANONYMIZE_IP` configuration setting:

```
GOOGLE_ANALYTICS_ANONYMIZE_IP = True
```

This may be mandatory for deployments in countries that have a firm policies concerning data privacy (e.g. Germany).

By default, IPs are not anonymized.

Sample Rate

You can configure the Sample Rate feature by setting the `GOOGLE_ANALYTICS_SAMPLE_RATE` configuration setting:

```
GOOGLE_ANALYTICS_SAMPLE_RATE = 10
```

The value is a percentage and can be between 0 and 100 and can be a string or integer value.

Site Speed Sample Rate

You can configure the Site Speed Sample Rate feature by setting the `GOOGLE_ANALYTICS_SITE_SPEED_SAMPLE_RATE` configuration setting:

```
GOOGLE_ANALYTICS_SITE_SPEED_SAMPLE_RATE = 10
```

The value is a percentage and can be between 0 and 100 and can be a string or integer value.

Cookie Expiration

You can configure the Cookie Expiration feature by setting the `GOOGLE_ANALYTICS_COOKIE_EXPIRATION` configuration setting:

```
GOOGLE_ANALYTICS_COOKIE_EXPIRATION = 3600000
```

The value is the cookie expiration in seconds or 0 to delete the cookie when the browser is closed.

Custom Javascript Source

You can configure a custom URL for the javascript file by setting the `GOOGLE_ANALYTICS_JS_SOURCE` configuration setting:

```
GOOGLE_ANALYTICS_JS_SOURCE = 'https://www.example.com/analytics.js'
```

2.4.10 GoSquared – traffic monitoring

GoSquared provides both real-time traffic monitoring and and trends. It tells you what is currently happening at your website, what is popular, locate and identify visitors and track twitter.

Installation

To start using the GoSquared integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the GoSquared template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The GoSquared tracking code is inserted into templates using a template tag. Load the `gosquared` template tag library and insert the `gosquared` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load gosquared %}
...
{% gosquared %}
</body>
</html>
```

Configuration

When you set up a website to be tracked by GoSquared, it assigns the site a token. You can find the token on the *Tracking Code* tab of your website settings page. Set `GOSQUARED_SITE_TOKEN` in the project `settings.py` file:

```
GOSQUARED_SITE_TOKEN = 'XXX-XXXXXX-X'
```

If you do not set a site token, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `GOSQUARED_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Identifying authenticated users

If your websites identifies visitors, you can pass this information on to GoSquared to display on the LiveStats dashboard. By default, the name of an authenticated user is passed to GoSquared automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `gosquared_identity` or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'gosquared_identity': identity})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'gosquared_identity': request.user.username}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Thanks go to GoSquared for their support with the development of this application.

2.4.11 Heap – analytics and events tracking

Heap automatically captures all user interactions on your site, from the moment of installation forward.

Installation

To start using the Heap integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Configuration

Before you can use the Heap integration, you must first get your Heap Tracker ID. If you don't have a Heap account yet, sign up to get your Tracker ID.

Setting the Tracker ID

Heap gives you a unique ID. You can find this ID on the Projects page of your Heap account. Set `HEAP_TRACKER_ID` in the project `settings.py` file:

```
HEAP_TRACKER_ID = 'XXXXXXXX'
```

If you do not set an Tracker ID, the tracking code will not be rendered.

The tracking code will be added just before the closing head tag.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the *ANALYTICAL_INTERNAL_IPS* setting (which is INTERNAL_IPS by default,) the tracking code is commented out. See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.12 Hotjar – analytics and user feedback

Hotjar is a website analytics and user feedback tool.

Installation

To start using the Hotjar integration, you must have installed the django-analytical package and have added the analytical application to INSTALLED_APPS in your project settings.py file. See *Installation and configuration* for details.

Next you need to add the Hotjar template tag to your templates. This step is only needed if you are not using the generic analytical.* tags. If you are, skip to *Configuration*.

The Hotjar code is inserted into templates using template tags. Because every page that you want to track must have the tag, it is useful to add it to your base template. At the top of the template, load the hotjar template tag library. Then insert the hotjar tag at the bottom of the head section:

```
{% load hotjar %}
<html>
<head>
...
{% hotjar %}
</head>
...
</html>
```

Configuration

Before you can use the Hotjar integration, you must first set your Site ID.

Setting the Hotjar Site ID

You can find the Hotjar Site ID in the "Sites & Organizations" section of your Hotjar account. Set HOTJAR_SITE_ID in the project settings.py file:

```
HOTJAR_SITE_ID = 'XXXXXXXXX'
```

If you do not set a Hotjar Site ID, the code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the HOTJAR_INTERNAL_IPS setting, the tracking code is commented out.

It takes the value of *ANALYTICAL_INTERNAL_IPS* by default (which in turn is INTERNAL_IPS by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.13 HubSpot – inbound marketing

HubSpot helps you get found by customers. It provides tools for content creation, conversion and marketing analysis. HubSpot uses tracking on your website to measure effect of your marketing efforts.

Installation

To start using the HubSpot integration, you must have installed the django-analytical package and have added the `analytical` application to INSTALLED_APPS in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the HubSpot template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The HubSpot tracking code is inserted into templates using a template tag. Load the `hubspot` template tag library and insert the `hubspot` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load hubspot %}
...
{% hubspot %}
</body>
</html>
```

Configuration

Before you can use the HubSpot integration, you must first set your portal ID, also known as your Hub ID.

Setting the portal ID

Your HubSpot account has its own portal ID, the `hubspot` tag will include them in the rendered JavaScript code. You can find the portal ID by accessing your dashboard. Alternatively, read this Quick Answer page. Set HUBSPOT_PORTAL_ID in the project `settings.py` file:

```
HUBSPOT_PORTAL_ID = 'XXXX'
```

If you do not set the portal ID, the tracking code will not be rendered.

Deprecated since version 0.18.0: *HUBSPOT_DOMAIN* is no longer required.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the HUBSPOT_INTERNAL_IPS setting, the tracking code is commented out. It takes the value of *ANALYTICAL_INTERNAL_IPS* by default (which in turn is INTERNAL_IPS by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.14 Intercom.io – Real-time tracking

Intercom.io is an easy way to implement real-chat and individual support for a website

Installation

To start using the Intercom.io integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Intercom.io template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Intercom.io Javascript code is inserted into templates using a template tag. Load the `intercom` template tag library and insert the `intercom` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load intercom %}
<html>
<head></head>
<body>
<!-- Your page -->
{% intercom %}
</body>
</html>
...
```

Configuration

Before you can use the Intercom.io integration, you must first set your app id.

Setting the app id

Intercom.io gives you a unique app id, and the `intercom` tag will include it in the rendered Javascript code. You can find your app id by clicking the *Tracking Code* link when logged into the on the intercom.io website. A page will display containing HTML code looking like this:

```
<script id="IntercomSettingsScriptTag">
    window.intercomSettings = { name: "Jill Doe", email: "jill@example.com", created_
↪at: 1234567890, app_id: "XXXXXXXXXXXXXXXXXXXXXXXX" };
</script>
<script>(function(){var w=window;var ic=w.Intercom;if(typeof ic==="function"){ic(
↪'reattach_activator');ic('update',intercomSettings);}else{var d=document;var␣
↪i=function(){i.c(arguments)};i.q=[];i.c=function(args){i.q.push(args)};w.Intercom=i;
↪function l(){var s=d.createElement('script');s.type='text/javascript';s.async=true;
↪s.src='https://static.intercomcdn.com/intercom.v1.js';var x=d.getElementsByTagName(
↪'script')[0];x.parentNode.insertBefore(s,x);}if(w.attachEvent){w.attachEvent('onload
↪',l);}else{w.addEventListener('load',l,false);}})()</script>
```

The code `XXXXXXXXXXXXXXXXXXXXXXXX` is your app id. Set `INTERCOM_APP_ID` in the project `settings.py` file:

```
INTERCOM_APP_ID = 'XXXXXXXXXXXXXXXXXXXXXXXX'
```

If you do not set an app id, the Javascript code will not be rendered.

Custom data

As described in the Intercom documentation on custom visitor data, the data that is tracked by Intercom can be customized. Using template context variables, you can let the `intercom` tag pass custom data to Intercom automatically. You can set the context variables in your view when your render a template containing the tracking code:

```
context = RequestContext({'intercom_cart_value': cart.total_price})
return some_template.render(context)
```

For some data, it is annoying to do this for every view, so you may want to set variables in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
from django.utils.hashcompat import md5_constructor as md5

GRAVATAR_URL = 'http://www.gravatar.com/avatar/'

def intercom_custom_data(request):
    try:
        email = request.user.email
    except AttributeError:
        return {}
    email_hash = md5(email).hexdigest()
    avatar_url = "%s%s" % (GRAVATAR_URL, email_hash)
    return {'intercom_avatar': avatar_url}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Standard variables that will be displayed in the Intercom live visitor data are listed in the table below, but you can define any `intercom_*` variable you like and have that detail passed from within the visitor live stream data when viewing Intercom.

Context variable	Description
`intercom_name`	The visitor's full name.
`intercom_email`	The visitor's email address.
`intercom_user_id`	The visitor's user id.
`created_at`	The date the visitor created an account

Identifying authenticated users

If you have not set the `intercom_name`, `intercom_email`, or `intercom_user_id` variables explicitly, the username and email address of an authenticated user are passed to Intercom automatically. See *Identifying authenticated users*.

Verifying identified users

Intercom supports HMAC authentication of users identified by user ID or email, in order to prevent impersonation. For more information, see Enable identity verification on your web product in the Intercom documentation.

To enable this, configure your Intercom account's HMAC secret key:

```
INTERCOM_HMAC_SECRET_KEY = 'XXXXXXXXXXXXXXXXXXXXXXX'
```

(You can find this secret key under the "Identity verification" section of your Intercom account settings page.)

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the *ANALYTICAL_INTERNAL_IPS* setting (which is INTERNAL_IPS by default,) the tracking code is commented out. See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.15 KISSinsights – feedback surveys

KISSinsights provides unobtrusive surveys that pop up from the bottom right-hand corner of your website. Asking specific questions gets you the targeted, actionable feedback you need to make your site better.

Installation

To start using the KISSinsights integration, you must have installed the django-analytical package and have added the analytical application to INSTALLED_APPS in your project settings.py file. See *Installation and configuration* for details.

Next you need to add the KISSinsights template tag to your templates. This step is only needed if you are not using the generic analytical.* tags. If you are, skip to *Configuration*.

The KISSinsights survey code is inserted into templates using a template tag. Load the kiss_insights template tag library and insert the kiss_insights tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the top of the HTML body:

```
{% load kiss_insights %}
...
</head>
<body>
{% kiss_insights %}
...
```

Configuration

Before you can use the KISSinsights integration, you must first set your account number and site code.

Setting the account number and site code

In order to install the survey code, you need to set your KISSinsights account number and website code. The kiss_insights tag will include it in the rendered Javascript code. You can find the account number and website code by visiting the code installation page of the website you want to place the surveys on. You will see some HTML code with a Javascript tag with a src attribute containing //s3.amazonaws.com/ki.js/XXXXX/YYY.js. Here XXXXX is the account number and YYY the website code. Set KISS_INSIGHTS_ACCOUNT_NUMBER and KISS_INSIGHTS_WEBSITE_CODE in the project settings.py file:

```
KISSINSIGHTS_ACCOUNT_NUMBER = 'XXXXX'
KISSINSIGHTS_SITE_CODE = 'XXX'
```

If you do not set the account number and website code, the survey code will not be rendered.

Identifying authenticated users

If your websites identifies visitors, you can pass this information on to KISSinsights so that you can tie survey submissions to customers. By default, the username of an authenticated user is passed to KISSinsights automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `kiss_insights_identity` or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'kiss_insights_identity': identity})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'kiss_insights_identity': request.user.email}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Showing a specific survey

KISSinsights can also be told to show a specific survey. You can let the `kiss_insights` tag include the code to select a survey by passing the survey ID to the template in the `kiss_insights_show_survey` context variable:

```
context = RequestContext({'kiss_insights_show_survey': 1234})
return some_template.render(context)
```

For information about how to find the survey ID, see the explanation on "How can I show a survey after a custom trigger condition?" on the KISSinsights help page.

2.4.16 KISSmetrics – funnel analysis

KISSmetrics is an easy to implement analytics solution that provides a powerful visual representation of your customer lifecycle. Discover how many visitors go from your landing page to pricing to sign up, and how many drop out at each stage.

Installation

To start using the KISSmetrics integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the KISSmetrics template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The KISSmetrics Javascript code is inserted into templates using a template tag. Load the `kiss_metrics` template tag library and insert the `kiss_metrics` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the top of the HTML head:

```
{% load kiss_metrics %}
<html>
<head>
{% kiss_metrics %}
...
```

Configuration

Before you can use the KISSmetrics integration, you must first set your API key.

Setting the API key

Every website you track events for with KISSmetrics gets its own API key, and the `kiss_metrics` tag will include it in the rendered Javascript code. You can find the website API key by visiting the website *Product center* on your KISSmetrics dashboard. Set `KISS_METRICS_API_KEY` in the project `settings.py` file:

```
KISS_METRICS_API_KEY = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```

If you do not set an API key, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `KISS_METRICS_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Identifying users

If your websites identifies visitors, you can pass this information on to KISSmetrics so that you can tie events to users. By default, the username of an authenticated user is passed to KISSmetrics automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `kiss_metrics_identity` or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'kiss_metrics_identity': identity})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'kiss_metrics_identity': request.user.email}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Alias

Alias is used to associate one identity with another. This most likely will occur if a user is not signed in yet, you assign them an anonymous identity and record activity for them and they later sign in and you get a named identity.

For example:

```
context = RequestContext({
    'kiss_metrics_alias': {'my_registered@email' : 'my_user_id'},
})
return some_template.render(context)
```

The output script tag will then include the corresponding properties as documented in the KISSmetrics alias API docs.

Recording events

You may tell KISSmetrics about an event by setting a variable in the context.

For example:

```
context = RequestContext({
    'kiss_metrics_event': ['Signed Up', {'Plan' : 'Pro', 'Amount' : 9.99}],
})
return some_template.render(context)
```

The output script tag will then include the corresponding Javascript event as documented in the KISSmetrics record API docs.

Recording properties

You may also set KISSmetrics properties without a corresponding event.

For example:

```
context = RequestContext({
    'kiss_metrics_properties': {'gender': 'Male'},
})
return some_template.render(context)
```

The output script tag will then include the corresponding properties as documented in the KISSmetrics set API docs.

2.4.17 Lucky Orange – All-in-one conversion optimization

Lucky Orange is a website analytics and user feedback tool.

Installation

To start using the Lucky Orange integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Lucky Orange template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Lucky Orange tracking code is inserted into templates using template tags. Because every page that you want to track must have the tag, it is useful to add it to your base template. At the top of the template, load the `luckyorange` template tag library. Then insert the `luckyorange` tag at the bottom of the head section:

```
{% load luckyorange %}
<html>
<head>
...
{% luckyorange %}
</head>
...
</html>
```

Configuration

Before you can use the Lucky Orange integration, you must first set your Site ID.

Setting the Lucky Orange Site ID

You can find the Lucky Orange Site ID in the "Settings" of your Lucky Orange account, reachable via the gear icon on the top right corner. Set `LUCKYORANGE_SITE_ID` in the project `settings.py` file:

```
LUCKYORANGE_SITE_ID = 'XXXXXX'
```

If you do not set a Lucky Orange Site ID, the code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `LUCKYORANGE_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.18 Matomo (formerly Piwik) – open source web analytics

Matomo is an open analytics platform currently used by individuals, companies and governments all over the world.

Installation

To start using the Matomo integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Matomo template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Matomo tracking code is inserted into templates using a template tag. Load the `matomo` template tag library and insert the `matomo` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body as recommended by the Matomo best practice for Integration Plugins:

```
{% load matomo %}
...
{% matomo %}
</body>
</html>
```

Configuration

Before you can use the Matomo integration, you must first define domain name and optional URI path to your Matomo server, as well as the Matomo ID of the website you're tracking with your Matomo server, in your project settings.

Setting the domain

Your Django project needs to know where your Matomo server is located. Typically, you'll have Matomo installed on a subdomain of its own (e.g. `matomo.example.com`), otherwise it runs in a subdirectory of a website of yours (e.g. `www.example.com/matomo`). Set `MATOMO_DOMAIN_PATH` in the project `settings.py` file accordingly:

```
MATOMO_DOMAIN_PATH = 'matomo.example.com'
```

If you do not set a domain the tracking code will not be rendered.

Setting the site ID

Your Matomo server can track several websites. Each website has its site ID (this is the `idSite` parameter in the query string of your browser's address bar when you visit the Matomo Dashboard). Set `MATOMO_SITE_ID` in the project `settings.py` file to the value corresponding to the website you're tracking:

```
MATOMO_SITE_ID = '4'
```

If you do not set the site ID the tracking code will not be rendered.

User variables

Matomo supports sending custom variables along with default statistics. If you want to set a custom variable, use the context variable `matomo_vars` when you render your template. It should be an iterable of custom variables represented by tuples like: `(index, name, value[, scope])`, where scope may be `'page'` (default) or `'visit'`. `index` should be an integer and the other parameters should be strings.

```
context = Context({
    'matomo_vars': [(1, 'foo', 'Sir Lancelot of Camelot'),
                    (2, 'bar', 'To seek the Holy Grail', 'page'),
                    (3, 'spam', 'Blue', 'visit')]
})
return some_template.render(context)
```

Matomo default settings allow up to 5 custom variables for both scope. Variable mapping between index and name must stay constant, or the latest name override the previous one.

If you use the same user variables in different views and its value can be computed from the HTTP request, you can also set them in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`.

User tracking

If you use the standard Django authentication system, you can allow Matomo to track individual users by setting the *ANALYTICAL_AUTO_IDENTIFY* setting to `True`. This is enabled by default. Matomo will identify users based on their `username`.

If you disable this settings, or want to customize what user id to use, you can set the context variable `analytical_identity` (for global configuration) or `matomo_identity` (for Matomo specific configuration). Setting one to `None` will disable the user tracking feature:

```
# Matomo will identify this user as 'BDFL' if ANALYTICAL_AUTO_IDENTIFY is True or␣
↪unset
request.user = User(username='BDFL', first_name='Guido', last_name='van Rossum')

# Matomo will identify this user as 'Guido van Rossum'
request.user = User(username='BDFL', first_name='Guido', last_name='van Rossum')
context = Context({
    'matomo_identity': request.user.get_full_name()
})

# Matomo will not identify this user (but will still collect statistics)
request.user = User(username='BDFL', first_name='Guido', last_name='van Rossum')
context = Context({
    'matomo_identity': None
})
```

Disabling cookies

If you want to disable cookies, set `MATOMO_DISABLE_COOKIES` to `True`. This is disabled by default.

Internal IP addresses

Usually, you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the *ANALYTICAL_INTERNAL_IPS* (which takes the value of `INTERNAL_IPS` by default) the tracking code is commented out. See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.19 Mixpanel – event tracking

Mixpanel tracks events and actions to see what features users are using the most and how they are trending. You could use it for real-time analysis of visitor retention or funnels.

Installation

To start using the Mixpanel integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Mixpanel template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Mixpanel Javascript code is inserted into templates using a template tag. Load the `mixpanel` template tag library and insert the `mixpanel` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load mixpanel %}
...
{% mixpanel %}
</head>
<body>
...
```

Configuration

Before you can use the Mixpanel integration, you must first set your token.

Setting the token

Every website you track events for with Mixpanel gets its own token, and the `mixpanel` tag will include it in the rendered Javascript code. You can find the project token on the Mixpanel *projects* page. Set `MIXPANEL_API_TOKEN` in the project `settings.py` file:

```
MIXPANEL_API_TOKEN = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```

If you do not set a token, the tracking code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `MIXPANEL_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Identifying users

If your websites identifies visitors, you can pass this information on to Mixpanel so that you can tie events to users. By default, the username of an authenticated user is passed to Mixpanel automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `mixpanel_identity` or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'mixpanel_identity': identity})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'mixpanel_identity': request.user.email}
```

(continues on next page)

(continued from previous page)

```
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Mixpanel can also receive properties for your identified user, using mixpanel.people.set. If want to send extra properties, just set a dictionary instead of a string in the `mixpanel_identity` context variable. The key `id` or `username` will be used as the user unique id, and any other key-value pair will be passed as *people properties*. For example:

```
def identify(request):
    try:
        return {
            'mixpanel_identity': {
                'id': request.user.id,
                'last_login': str(request.user.last_login),
                'date_joined': str(request.user.date_joined),
            }
        }
    except AttributeError:
        return {}
```

Tracking events

The django-analytical app integrates the Mixpanel Javascript API in templates. To tracking events in views or other parts of Django, you can use Wes Winham's mixpanel-celery package.

If you want to track an event in Javascript, use the asynchronous notation, as described in the section titled "Asynchronous Tracking with Javascript" in the Mixpanel documentation. For example:

```
mixpanel.track("play-game", {"level": "12", "weapon": "sword", "character": "knight"}
↪);
```

2.4.20 Olark – visitor chat

Olark is a lightweight tool to chat with visitors to your website using your existing instant messaging client. Chat with your website visitors while they browse, using your mobile device or instant messenger. Olark is fully customizable, supports multiple operators and keeps chat transcripts.

Installation

To start using the Olark integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Olark template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Olark Javascript code is inserted into templates using a template tag. Load the `olark` template tag library and insert the `olark` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load olark %}
...
{% olark %}
</body>
</html>
```

Configuration

Before you can use the Olark integration, you must first set your site ID. You can customize the visitor nickname and add information to their status in the operator buddy list, and customize the text used in the chat window.

Setting the site ID

In order to install the chat code, you need to set your Olark site ID. The `olark` tag will include it in the rendered Javascript code. You can find the site ID on installation page of you Olark account. Set `OLARK_SITE_ID` in the project `settings.py` file:

```
OLARK_SITE_ID = 'XXXX-XXX-XX-XXXX'
```

If you do not set the site ID, the chat window will not be rendered.

Setting the visitor nickname

If your website identifies visitors, you can use that to set their nickname in the operator buddy list. By default, the name and username of an authenticated user are automatically used to set the nickname. See *Identifying authenticated users*.

You can also set the visitor nickname yourself by adding either the `olark_nickname` (alias: `olark_identity`) or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'olark_nickname': nick})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def set_olark_nickname(request):
    try:
        return {'olark_nickname': request.user.email}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

See also api.chat.updateVisitorNickname in the Olark Javascript API documentation.

Adding status information

If you want to send more information about the visitor to the operators, you can add text snippets to the status field in the buddy list. Set the `olark_status` context variable to a string or a list of strings and the `olark` tag will pass

them to Olark as status messages:

```
context = RequestContext({'olark_status': [
    'has %d items in cart' % cart.item_count,
    'value of items is $%0.2f' % cart.total_value,
]})
return some_template.render(context)
```

See also api.chat.updateVisitorStatus in the Olark Javascript API documentation.

Customizing the chat window messages

Olark lets you customize the appearance of the Chat window by changing location, colors and messages text. While you can configure these on the Olark website, sometimes one set of messages is not enough. For example, if you want to localize your website, you want to address every visitor in their own language. Olark allows you to set the messages on a per-page basis, and the `olark` tag supports this feature by way of the following context variables:

Context variable	Example message
`olark_welcome_title`	Click to Chat
`olark_chatting_title`	Live Help: Now Chatting
`olark_unavailable_title`	Live Help: Offline
`olark_busy_title`	Live Help: Busy
`olark_away_message`	Our live support feature is currently offline, Please try again later.
`olark_loading_title`	Loading Olark...
`olark_welcome_message`	Welcome to my website. You can use this chat window to chat with me.
`olark_busy_message`	All of our representatives are with other customers at this time. We will be with you shortly.
`olark_chat_input_text`	Type here and hit to chat
`olark_name_input_text`	and type your Name
`olark_email_input_text`	and type your Email
`olark_offline_note_message`	We are offline, send us a message
`olark_send_button_text`	Send
`olark_offline_note_thankyou_text`	Thank you for your message. We will get back to you as soon as we can.
`olark_offline_note_error_text`	You must complete all fields and specify a valid email address
`olark_offline_note_sending_text`	Sending...
`olark_operator_is_typing_text`	is typing...
`olark_operator_has_stopped_typing_text`	has stopped typing
`olark_introduction_error_text`	Please leave a name and email address so we can contact you in case we get disconnected
`olark_introduction_messages`	Welcome, just fill out some brief information and click 'Start chat' to talk to us
`olark_introduction_submit_button_text`	Start chat

As an example, you could set the texts site-wide base on the current language using a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
OLARK_TEXTS = {
    'en': {
        'welcome title':  "Click for Live Help",
        'chatting_title': "Live Help: Now chatting",
```

(continues on next page)

(continued from previous page)

```
        ...
    },
    'nl': {
        'welcome_title':  "Klik voor online hulp",
        'chatting_title': "Online hulp: in gesprek",
        ...
    },
    ...
}

def set_olark_texts(request):
    lang = request.LANGUAGE_CODE.split('-', 1)[0]
    texts = OLARK_TEXTS.get(lang)
    if texts is None:
        texts = OLARK_TEXTS.get('en')
    return dict(('olark_%s' % k, v) for k, v in texts.items())
```

See also the Olark blog post on supporting multiple languages.

Thanks go to Olark for their support with the development of this application.

2.4.21 Optimizely – A/B testing

Optimizely is an easy way to implement A/B testing. Try different decisions, images, layouts, and copy without touching your website code and see exactly how your experiments are affecting pageviews, retention and sales.

Installation

To start using the Optimizely integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Optimizely template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Optimizely Javascript code is inserted into templates using a template tag. Load the `optimizely` template tag library and insert the `optimizely` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the top of the HTML head:

```
{% load optimizely %}
<html>
<head>
{% optimizely %}
...
```

Configuration

Before you can use the Optimizely integration, you must first set your account number.

Setting the account number

Optimizely gives you a unique account number, and the `optimizely` tag will include it in the rendered Javascript code. You can find your account number by clicking the *Implementation* link in the top right-hand corner of the Optimizely website. A pop-up window will appear containing HTML code looking like this:

```
<script src="//cdn.optimizely.com/js/XXXXXXX.js"></script>
```

The number `XXXXXXX` is your account number. Set `OPTIMIZELY_ACCOUNT_NUMBER` in the project `settings.py` file:

```
OPTIMIZELY_ACCOUNT_NUMBER = 'XXXXXXX'
```

If you do not set an account number, the Javascript code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `OPTIMIZELY_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.22 Performable – web analytics and landing pages

Performable provides a platform for inbound marketing, landing pages and web analytics. Its analytics module tracks individual customer interaction, funnel and e-commerce analysis. Landing pages can be created and designed on-line, and integrated with you existing website.

Installation

To start using the Performable integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Performable template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Performable Javascript code is inserted into templates using a template tag. Load the `performable` template tag library and insert the `performable` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load performable %}
...
{% performable %}
</body>
</html>
```

Configuration

Before you can use the Performable integration, you must first set your API key.

Setting the API key

You Performable account has its own API key, which `performable` tag will include it in the rendered Javascript code. You can find your API key on the *Account Settings* page (click 'Account Settings' in the top right-hand corner of your Performable dashboard). Set `PERFORMABLE_API_KEY` in the project `settings.py` file:

```
PERFORMABLE_API_KEY = 'XXXXXX'
```

If you do not set an API key, the Javascript code will not be rendered.

Identifying authenticated users

If your websites identifies visitors, you can pass this information on to Performable so that you can track individual users. By default, the username of an authenticated user is passed to Performable automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `performable_identity` or the `analytical_identity` variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'performable_identity': identity})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'performable_identity': request.user.email}
    except AttributeError:
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `PERFORMABLE_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Embedding a landing page

You can embed a Performable landing page in your Django website. The `performable_embed` template tag adds the Javascript code to embed the page. It takes two arguments: the hostname and the page ID:

```
{% performable_embed HOSTNAME PAGE_ID %}
```

To find the hostname and page ID, select *Manage* → *Manage Landing Pages* on your Performable dashboard. Select the landing page you want to embed. Look at the URL in your browser address bar; it will look like this:

```
http://my.performable.com/s/HOSTNAME/page/PAGE_ID/
```

(If you are placing the hostname and page id values in the template, do not forget to enclose them in quotes or they will be considered context variable names.)

Thanks go to Performable for their support with the development of this application.

2.4.23 Rating@Mail.ru – traffic analysis

Rating@Mail.ru is an analytics tool like as google analytics.

Installation

To start using the Rating@Mail.ru integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Rating@Mail.ru template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Rating@Mail.ru counter code is inserted into templates using a template tag. Load the `rating_mailru` template tag library and insert the `rating_mailru` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load rating_mailru %}
<html>
<head>
...
{% rating_mailru %}
</head>
...
```

Configuration

Before you can use the Rating@Mail.ru integration, you must first set your website counter ID.

Setting the counter ID

Every website you track with Rating@Mail.ru gets its own counter ID, and the `rating_mailru` tag will include it in the rendered Javascript code. You can find the web counter ID on the overview page of your account. Set `RATING_MAILRU_COUNTER_ID` in the project `settings.py` file:

```
RATING_MAILRU_COUNTER_ID = '1234567'
```

If you do not set a counter ID, the counter code will not be rendered.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `RATING_MAILRU_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.4.24 SnapEngage – live chat

SnapEngage is a live chat widget for your site which integrates with your existing chat client. It integrates with many online applications and even allows you to make a remote screenshot of the webpage. SnapEngage can be customized to fit your website look and feel, offers reports and statistics and is available in many languages.

Installation

To start using the SnapEngage integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the SnapEngage template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The SnapEngage Javascript code is inserted into templates using a template tag. Load the `SnapEngage` template tag library and insert the `SnapEngage` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load snapengage %}
...
{% snapengage %}
</body>
</html>
```

Configuration

Before you can use the SnapEngage integration, you must first set the widget ID. You can customize the visitor nickname and add information to their status in the operator buddy list, and customize the text used in the chat window.

Setting the widget ID

In order to install the chat code, you need to set the ID of the SnapEngage widget. You can find the site ID on the Your Widget ID page of your SnapEngage account. Set `SNAPENGAGE_WIDGET_ID` in the project `settings.py` file:

```
SNAPENGAGE_WIDGET_ID = 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX'
```

If you do not set the widget ID, the chat window will not be rendered.

Customizing the widget

The SnapEngage widget can be customized in various ways using either context variables or settings. More information about controlling the widget can be found on the customization FAQ section of the SnapEngage website.

Setting	Context variable	Description
SNAPENGAGE_DOMAIN	snapengage_domain	Manually set the domain name to follow users across subdomains.
SNAPENGAGE_SECURE_CONNECTION	snapengage_secure_connection	Force the use of SSL for the chat connection, even on unencrypted pages. (Default: False)
SNAPENGAGE_BUTTON_EFFECT	snapengage_button_effect	An effect applied when the mouse hovers over the button. (Example: "-4px")
SNAPENGAGE_BUTTON_STYLE	snapengage_button_style	What the chat button should look like. Use any of the BUTTON_STYLE_* constants, or a URL to a custom button image.
SNAPENGAGE_BUTTON_LOCATION	snapengage_button_location	The location of the chat button. Use any of the BUTTON_LOCATION_* constants.
SNAPENGAGE_BUTTON_LOCATION_OFFSET	snapengage_button_location_offset	The offset of the button from the top or left side of the page. (Default: "55%")
SNAPENGAGE_FORM_POSITION	snapengage_form_position	Configure the location of the chat window. Use any of the FORM_POSITION_* constants.
SNAPENGAGE_FORM_TOP_POSITION	snapengage_form_top_position	The chat window offset in pixels from the top of the page.
SNAPENGAGE_READONLY_EMAIL	snapengage_readonly_email	Whether a preset e-mail address can be changed by the visitor. (Default: False)
SNAPENGAGE_SHOW_OFFLINE	snapengage_show_offline	Whether to show the chat button when all operators are offline. (Default: True)
SNAPENGAGE_SCREENSHOTS	snapengage_screenshots	Whether to allow the user to take a screenshot. (Default: True)
SNAPENGAGE_OFFLINE_SCREENSHOTS	snapengage_offline_screenshots	Whether to allow the user to take a screenshot when all operators are offline. (Default: True)
SNAPENGAGE_SOUNDS	snapengage_sounds	Whether to play chat sound notifications. (Default: True)

There are also two customizations that can only be used with context variables.

Context variable	Description
snapengage_proactive_chat	Set to False to disable proactive chat, for example for users who are already converted.
snapengage_email	Set the e-mail address of the website visitor. (See *Setting the visitor e-mail address*)

Setting the visitor e-mail address

If your website identifies visitors, you can use that to pass their e-mail address to the support agent. By default, the e-mail address of an authenticated user is automatically used. See *Identifying authenticated users*.

You can also set the visitor e-mail address yourself by adding either the snapengage_email (alias: snapengage_identity) or the analytical_identity variable to the template context. If both variables are set, the former takes precedence. For example:

```
context = RequestContext({'snapengage_email': email})
return some_template.render(context)
```

If you can derive the e-mail address from the HTTP request, you can also use a context processor that you add to the TEMPLATE_CONTEXT_PROCESSORS list in settings.py:

```
from django.core.exceptions import ObjectDoesNotExist

def set_snapengage_email(request):
    try:
        profile = request.user.get_profile()
        return {'snapengage_email': profile.business_email}
    except (AttributeError, ObjectDoesNotExist):
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

If the user should not be able to edit the pre-set e-mail address, you can set either the `snapengage_readonly_email` context variable or the `SNAPENGAGE_READONLY_EMAIL` setting to `True`.

Thanks go to SnapEngage for their support with the development of this application.

2.4.25 Spring Metrics – conversion tracking

Spring Metrics is a conversions analysis tool. It shows you the top converting sources, search keywords and landing pages. The real-time dashboard shows you how customers interact with your website and how to increase conversion.

Installation

To start using the Spring Metrics integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Spring Metrics template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Spring Metrics tracking code is inserted into templates using a template tag. Load the `spring_metrics` template tag library and insert the `spring_metrics` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load spring_metrics %}
<html>
<head>
...
{% spring_metrics %}
</head>
...
```

Configuration

Before you can use the Spring Metrics integration, you must first set your website Tracking ID and tag a page for conversion. You can also customize the data that Spring Metrics tracks.

Setting the Tracking ID

Every website you track with Spring Metrics gets its own Tracking ID, and the `spring_metrics` tag will include it in the rendered Javascript code. You can find the Tracking ID in the Site Settings of your Spring Metrics account. Set `SPRING_METRICS_TRACKING_ID` in the project `settings.py` file:

```
SPRING_METRICS_TRACKING_ID = 'XXXXXXXXXX'
```

If you do not set a Tracking ID, the tracking code will not be rendered.

Tagging conversion

In order to make use of Spring Metrics, you must tell it when visitors become customers. This is called conversion. Usually, it marked by the client requesting a specific page, such as the "thank you" page of a webshop checkout. You tag these pages in the Site Settings of your Spring Metrics account.

Alternatively, you can mark conversion pages using the `spring_metrics_convert` template context variable:

```
context = RequestContext({'spring_metrics_convert': 'mailinglist signup'})
return some_template.render(context)
```

Tracking revenue

Spring Metrics allows you to track the value of conversions. Using the `spring_metrics_revenue` template context variable, you can let the `spring_metrics` tag pass earned revenue to Spring Metrics. You can set the context variable in your view when you render a template containing the tracking code:

```
context = RequestContext({
    'spring_metrics_convert': 'sale',
    'spring_metrics_revenue': '30.53',
})
return some_template.render(context)
```

(You would not need to use the `spring_metrics_convert` variable if you already tagged the page in Spring Metrics.)

Custom data

Spring Metrics can also track other data. Interesting examples could be transaction IDs or the e-mail addresses from logged in users. By setting any `spring_metrics_X` template context variable, Spring Metrics will track a variable named `X`. For example:

```
context = RequestContext({
    'spring_metrics_revenue': '30.53',
    'spring_metrics_order_id': '15445',
})
return some_template.render(context)
```

Some variables should be passed on every page and can be computed from the request object. In such cases you will want to set custom variables in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def spring_metrics_global_variables(request):
    try:
        profile = request.user.get_profile()
        return {'spring_metrics_city': profile.address.city}
    except (AttributeError, ObjectDoesNotExist):
        return {}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Identifying authenticated users

If you have not set the `spring_metrics_email` property explicitly, the e-mail address of an authenticated user is passed to Spring Metrics automatically. See *Identifying authenticated users*.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `SPRING_METRICS_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Thanks go to Spring Metrics for their support with the development of this application.

2.4.26 UserVoice – user feedback and helpdesk

UserVoice makes it simple for your customers to give, discuss, and vote for feedback. An unobtrusive feedback tab allows visitors to easily submit and discuss ideas without having to sign up for a new account. The best ideas are delivered to you based on customer votes.

Installation

To start using the UserVoice integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the UserVoice template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The UserVoice Javascript code is inserted into templates using a template tag. Load the `uservoice` template tag library and insert the `uservoice` tag. Because every page that you want to have the feedback tab to appear on must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML body:

```
{% load uservoice %}
...
{% uservoice %}
</body>
</html>
```

Configuration

Before you can use the UserVoice integration, you must first set the widget key.

Setting the widget key

In order to use the feedback widget, you need to configure which widget you want to show. You can find the widget keys in the *Channels* tab on your UserVoice *Settings* page. Under the *Javascript Widget* heading, find the Javascript embed code of the widget. The widget key is the alphanumerical string contained in the URL of the script imported by the embed code:

```
<script type="text/javascript">

  UserVoice=window.UserVoice||[];(function(){
        var uv=document.createElement('script');uv.type='text/javascript';
        uv.async=true;uv.src='//widget.uservoice.com/XXXXXXXXXXXXXXXXXXXX.js';
        var s=document.getElementsByTagName('script')[0];
        s.parentNode.insertBefore(uv,s)})();
</script>
```

(The widget key is shown as `XXXXXXXXXXXXXXXXXXXX`.)

The default widget

Often you will use the same widget throughout your website. The default widget key is configured by setting `USERVOICE_WIDGET_KEY` in the project `settings.py` file:

```
USERVOICE_WIDGET_KEY = 'XXXXXXXXXXXXXXXXXXXX'
```

If the setting is present but empty, no widget is shown by default. This is useful if you want to set a widget using a template context variable, as the setting must be present for the generic `analytical.*` tags to work.

Widget options

You can set `USERVOICE_WIDGET_OPTIONS` to customize your widget with UserVoice's options.

Tip: See the JS SDK Overview and the reference for the details of available options.

For example, to override the default icon style with a tab and on the left, you could define:

```
USERVOICE_WIDGET_OPTIONS = {"trigger_position": "left",
                            "trigger_style": "tab"}
```

Per-view widget

The widget configuration can be overriden in a view using `uservoice_widget_options` template context variable. For example:

```
context = RequestContext({'uservoice_widget_options': 'mode': 'satisfaction'})
return some_template.render(context)
```

It's also possible to set a different widget key for a particular view with `uservoice_widget_key`:

```
context = RequestContext({'uservoice_widget_key': 'XXXXXXXXXXXXXXXXXXX'})
return some_template.render(context)
```

These variable passed in the context overrides the default widget configuration.

Using a custom link

Instead of showing the default feedback icon or tab, you can make the UserVoice widget launch when a visitor clicks a link or when some other event occurs. As the documentation describe, simply add the `data-uv-trigger` HTML attribute to the element. For example:

```
<a href="mailto:questions@yoursite.com" data-uv-trigger>Contact us</a>
```

In order to hidden the default trigger, you should disable it putting `uservoice_add_trigger` to `False`:

```
context = RequestContext({'uservoice_add_trigger': False})
return your_template_with_custom_uservoice_link.render(context)
```

If you want to disable the automatic trigger globally, set in `settings.py`:

```
USERVOICE_ADD_TRIGGER = False
```

Setting the widget key in a context processor

You can also set the widget keys in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`. For example, to show a specific widget to logged in users:

```
def uservoice_widget_key(request):
    try:
        if request.user.is_authenticated():
            return {'uservoice_widget_key': 'XXXXXXXXXXXXXXXXXXX'}
    except AttributeError:
        pass
    return {}
```

The widget key passed in the context variable overrides both the default and the per-view widget key.

Identifying users

If your websites identifies visitors, you can pass this information on to Uservoice. By default, the name and email of an authenticated user is passed to Uservoice automatically. See *Identifying authenticated users*.

You can also send the visitor identity yourself by adding either the `uservoice_identity` or the `analytical_identity` variable to the template context. (If both are set, the former takes precedence.) This should be a dictionary with the desired user traits as its keys. Check the documentation on identifying users to see valid traits. For example:

```
context = RequestContext({'uservoice_identity': {'email': user_email,
                                                 'name': username }})
return some_template.render(context)
```

If you can derive the identity from the HTTP request, you can also use a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
def identify(request):
    try:
        return {'uservoice_identity': {
          email: request.user.username,
          name: request.user.get_full_name(),
          id: request.user.id,
          type: 'vip',
          account: {
            name: 'Acme, Co.',
            monthly_rate: 9.99,
            ltv: 1495.00,
            plan: 'Enhanced'
          }
         }
        }
    except AttributeError:
        return {}
```

Thanks go to UserVoice for their support with the development of this application.

2.4.27 Woopra – website analytics

Woopra generates live detailed statistics about the visitors to your website. You can watch your visitors navigate live and interact with them via chat. The service features notifications, campaigns, funnels and more.

Installation

To start using the Woopra integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Woopra template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Woopra tracking code is inserted into templates using a template tag. Load the `woopra` template tag library and insert the `woopra` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load woopra %}
<html>
<head>
...
{% woopra %}
</head>
...
```

Because Javascript code is asynchronous, putting the tag in the head section increases the chances that a page view is going to be tracked before the visitor leaves the page. See for details the Asynchronous JavaScript Developer's Guide on the Woopra website.

Configuration

Before you can use the Woopra integration, you must first set the website domain. You can also customize the data that Woopra tracks and identify authenticated users.

Setting the domain

A Woopra account is tied to a website domain. Set `WOOPRA_DOMAIN` in the project `settings.py` file:

```
WOOPRA_DOMAIN = 'XXXXXXXX.XXX'
```

If you do not set a domain, the tracking code will not be rendered.

(In theory, the django-analytical application could get the website domain from the current `Site` or the `request` object, but this setting also works as a sign that the Woopra integration should be enabled for the `analytical.*` template tags.)

Visitor timeout

The default Woopra visitor timeout – the time after which Woopra ignores inactive visitors on a website – is set at 4 minutes. This means that if a user opens your Web page and then leaves it open in another browser window, Woopra will report that the visitor has gone away after 4 minutes of inactivity on that page (no page scrolling, clicking or other action).

If you would like to increase or decrease the idle timeout setting you can set `WOOPRA_IDLE_TIMEOUT` to a time in milliseconds. For example, to set the default timout to 10 minutes:

```
WOOPRA_IDLE_TIMEOUT = 10 * 60 * 1000
```

Keep in mind that increasing this number will not only show you more visitors on your site at a time, but will also skew your average time on a page reporting. So it's important to keep the number reasonable in order to accurately make predictions.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `WOOPRA_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

Custom data

As described in the Woopra documentation on custom visitor data, the data that is tracked by Woopra can be customized. Using template context variables, you can let the `woopra` tag pass custom data to Woopra automatically. You can set the context variables in your view when your render a template containing the tracking code:

```
context = RequestContext({'woopra_cart_value': cart.total_price})
return some_template.render(context)
```

For some data, it is annoying to do this for every view, so you may want to set variables in a context processor that you add to the `TEMPLATE_CONTEXT_PROCESSORS` list in `settings.py`:

```
from django.utils.hashcompat import md5_constructor as md5

GRAVATAR_URL = 'http://www.gravatar.com/avatar/'

def woopra_custom_data(request):
    try:
        email = request.user.email
    except AttributeError:
        return {}
    email_hash = md5(email).hexdigest()
    avatar_url = "%s%s" % (GRAVATAR_URL, email_hash)
    return {'woopra_avatar': avatar_url}
```

Just remember that if you set the same context variable in the `RequestContext` constructor and in a context processor, the latter clobbers the former.

Standard variables that will be displayed in the Woopra live visitor data are listed in the table below, but you can define any `woopra_*` variable you like and have that detail passed from within the visitor live stream data when viewing Woopra.

Context variable	Description
`woopra_name`	The visitor's full name.
`woopra_email`	The visitor's email address.
`woopra_avatar`	A URL link to a visitor avatar.

Identifying authenticated users

If you have not set the `woopra_name` or `woopra_email` variables explicitly, the username and email address of an authenticated user are passed to Woopra automatically. See *Identifying authenticated users*.

Thanks go to Woopra for their support with the development of this application.

2.4.28 Yandex.Metrica – traffic analysis

Yandex.Metrica is an analytics tool like as google analytics.

Installation

To start using the Yandex.Metrica integration, you must have installed the django-analytical package and have added the `analytical` application to `INSTALLED_APPS` in your project `settings.py` file. See *Installation and configuration* for details.

Next you need to add the Yandex.Metrica template tag to your templates. This step is only needed if you are not using the generic `analytical.*` tags. If you are, skip to *Configuration*.

The Yandex.Metrica counter code is inserted into templates using a template tag. Load the `yandex_metrica` template tag library and insert the `yandex_metrica` tag. Because every page that you want to track must have the tag, it is useful to add it to your base template. Insert the tag at the bottom of the HTML head:

```
{% load yandex_metrica %}
<html>
<head>
...
{% yandex_metrica %}
</head>
...
```

Configuration

Before you can use the Yandex.Metrica integration, you must first set your website counter ID.

Setting the counter ID

Every website you track with Yandex.Metrica gets its own counter ID, and the `yandex_metrica` tag will include it in the rendered Javascript code. You can find the web counter ID on the overview page of your account. Set `YANDEX_METRICA_COUNTER_ID` in the project `settings.py` file:

```
YANDEX_METRICA_COUNTER_ID = '12345678'
```

If you do not set a counter ID, the counter code will not be rendered.

You can set additional options to tune your counter:

Constant	Default Value	Description
`YANDEX_METRICA_WEBVISOR`	False	Webvisor, scroll map, form analysis.
`YANDEX_METRICA_TRACKHASH`	False	Hash tracking in the browser address bar.
`YANDEX_METRICA_NOINDEX`	False	Stop automatic page indexing.
`YANDEX_METRICA_ECOMMERCE`	False	Dispatch ecommerce data to Metrica.

Internal IP addresses

Usually you do not want to track clicks from your development or internal IP addresses. By default, if the tags detect that the client comes from any address in the `YANDEX_METRICA_INTERNAL_IPS` setting, the tracking code is commented out. It takes the value of *`ANALYTICAL_INTERNAL_IPS`* by default (which in turn is `INTERNAL_IPS` by default). See *Identifying authenticated users* for important information about detecting the visitor IP address.

2.5 Settings

Here's a full list of all available settings, in alphabetical order, and their default values.

`ANALYTICAL_AUTO_IDENTIFY`
Default: `True`

Automatically identify logged in users by their username. See *Identifying authenticated users*.

`ANALYTICAL_INTERNAL_IPS`
Default: `INTERNAL_IPS`

A list or tuple of internal IP addresses. Tracking code will be commented out for visitors from any of these addresses. You can configure this setting for each service individually by substituting `ANALYTICAL` for the

upper-case service name. For example, set `GOOGLE_ANALYTICS_INTERNAL_IPS` to configure for Google Analytics.

See *Internal IP addresses*.

2.6 History and credits

2.6.1 Changelog

The project follows the Semantic Versioning specification for its version numbers. Patch-level increments indicate bug fixes, minor version increments indicate new functionality and major version increments indicate backwards incompatible changes.

Version 1.0.0 is the last to support Django < 1.7. Users of older Django versions should stick to 1.0.0, and are encouraged to upgrade their setups. Starting with 2.0.0, dropping support for obsolete Django versions is not considered to be a backward-incompatible change.

Unreleased

- Remove deprecated Piwik integration. Use Matomo instead! (Peter Bittner)

Version 3.1.0

- Rename default branch from master to main (Peter Bittner, Jannis Leidel)
- Modernize packaging setup, add pyproject.toml (Peter Bittner)
- Integrate isort, reorganize imports (David Smith)
- Refactor test suite from Python unit tests to Pytest (David Smith)
- Add Heap integration (Garrett Coakley)
- Drop Django 3.1, cover Django 4.0 and Python 3.10 in test suite (David Smith)

Version 3.0.0

- Add support for Lucky Orange (Peter Bittner)
- Add missing instructions in Installation chapter of the docs (Peter Bittner)
- Migrate test setup to Pytest (David Smith, Peter Bittner, Pi Delport)
- Support Django 3.1 and Python 3.9, drop Django 1.11 and Python 2.7/3.5 (David Smith)
- Migrate from Travis CI to GitHub Actions (Jannis Leidel)
- Update accepted patterns (regex) for Google Analytics GTag (Taha Rushain)
- Scope Piwik warning to use of Piwik (Hugo Barrera)
- Add `user_id` to Google Analytics GTag (Sean Wallace)

Version 2.6.0

- Support Django 3.0 and Python 3.8, drop Django 2.1
- Add support for Google Analytics Tag Manager (Marc Bourqui)
- Add Matomo, the renamed version of Piwik (Scott Karlin)
- Move Joost's project over to the Jazzband

Version 2.5.0

- Add support for Google analytics.js (Marc Bourqui)
- Add support for Intercom HMAC identity verification (Pi Delport)
- Add support for Hotjar (Pi Delport)
- Make sure _trackPageview happens before other settings in Google Analytics (Diederik van der Boor)

Version 2.4.0

- Support Django 2.0 (Matthäus G. Chajdas)

Version 2.3.0

- Add Facebook Pixel support (Pi Delport)
- Add Python 3.6 and Django 1.10 & 1.11 tests (Pi Delport)
- Drop Python 3.2 support

Version 2.2.2

- Allow port in Piwik domain path. (Alex Ramsay)

Version 2.2.1

- Fix a bug with the extra Google Analytics variables also pushing the *_gat.* flag onto the configuration array.

Version 2.2.0

- Update Woopra JavaScript snippet (Aleck Landgraf)

Version 2.1.0

- Support Rating@mail.ru (Nikolay Korotkiy)
- Support Yandex.Metrica (Nikolay Korotkiy)
- Add support for extra Google Analytics variables (Steve Schwarz)
- Remove support for Reinvigorate (service shut down)

Version 2.0.0

- Support Django 1.9, drop support for Django < 1.7 (Hugo Osvaldo Barrera)
- Support custom user models with an alternative username field (Brad Pitcher)

Version 1.0.0

- Add Piwik user variables support (Alexandre Pocquet)

Version 0.22.0

- Mark package as Python 3 compatible (Martín Gaitán)
- Fix Clickmap tracker id regular expression
- Test with Django 1.8

Version 0.21.0

- Added compatibility with Python 3 (Eric Amador)

Version 0.20.0

- Support Django 1.7 (Craig Bruce)
- Update Mixpanel identity code (Martín Gaitán)
- Identify authenticated users in Uservoice (Martín Gaitán)
- Add full name and email to Olark (Scott Adams)

Version 0.19.0

- Add Piwik integration (Peter Bittner)

Version 0.18.0

- Update HubSpot code (Craig Bruce)

Version 0.17.1

- Fix typo in Intercom.io support (Steven Skoczen)

Version 0.17.0

- Update UserVoice support (Martín Gaitán)
- Add support for Intercom.io (Steven Skoczen)

Version 0.16.0

- Add support for GA Display Advertising features (Max Arnold)

Version 0.15.0

- Add IP anonymization setting to GA tracking pixel (Tinnet Coronam)
- Include Django 1.5 in tox.ini (Tinnet Coronam)
- Add Clickmap integration (Philippe O. Wagner)

Version 0.14.0

- Update mixpanel integration to latest code (Simon Ye)

Version 0.13.0

- Add support for the KISSmetrics alias feature (Sandra Mau)
- Update testing code for Django 1.4 (Pi Delport)

Version 0.12.0

- Add support for the UserVoice service.

Version 0.11.3

- Added support for Gaug.es (Steven Skoczen)

Version 0.11.2

- Fix Spring Metrics custom variables.
- Update Spring Metrics documentation.

Version 0.11.1

- Fix Woopra for anonymous users (Steven Skoczen).

Version 0.11.0

- Added support for the Spring Metrics service.
- Allow sending events and properties to KISSmetrics (Paul Oswald).
- Add support for the Site Speed report in Google Analytics (Uros Trebec).

Version 0.10.0

- Added multiple domains support for Google Analytics.
- Fixed bug in deleted settings testing code (Eric Davis).

Version 0.9.2

- Added support for the SnapEngage service.
- Updated Mixpanel code (Julien Grenier).

Version 0.9.1

- Fixed compatibility with Python 2.5 (Iván Raskovsky).

Version 0.9.0

- Updated Clicky tracking code to support multiple site ids.
- Fixed Chartbeat auto-domain bug when the Sites framework is not used (Eric Davis).
- Improved testing code (Eric Davis).

Version 0.8.1

- Fixed MANIFEST bug that caused GoSquared support to be missing from the source distribution.

Version 0.8.0

- Added support for the GoSquared service.
- Updated Clicky tracking code to use relative URLs.

Version 0.7.0

- Added support for the Woopra service.
- Added chat window text customization to Olark.
- Renamed `MIXPANEL_TOKEN` setting to `MIXPANEL_API_TOKEN` for compatibility with Wes Winham's mixpanel-celery package.
- Fixed the `<script>` tag for Crazy Egg.

Version 0.6.0

- Added support for the Reinvigorate service.
- Added support for the Olark service.

Version 0.5.0

- Split off Geckoboard support into django-geckoboard.

Version 0.4.0

- Added support for the Geckoboard service.

Version 0.3.0

- Added support for the Performable service.

Version 0.2.0

- Added support for the HubSpot service.
- Added template tags for individual services.

Version 0.1.0

- First project release.

2.6.2 Credits

The django-analytical package was originally written by Joost Cassee and is now maintained by the Jazzband community, with contributions from Eric Davis, Paul Oswald, Uros Trebec, Steven Skoczen, Pi Delport, Sandra Mau, Simon Ye, Tinnet Coronam, Philippe O. Wagner, Max Arnold , Martín Gaitán, Craig Bruce, Peter Bittner, Scott Adams, Eric Amador, Alexandre Pocquet, Brad Pitcher, Hugo Osvaldo Barrera, Nikolay Korotkiy, Steve Schwarz, Aleck Landgraf, Marc Bourqui, Diederik van der Boor, Matthäus G. Chajdas, Scott Karlin and others.

Included Javascript code snippets for integration of the analytics services were written by the respective service providers.

The application was inspired by and uses ideas from Analytical, Joshua Krall's all-purpose analytics front-end for Rails.

The work on Crazy Egg was made possible by Bateau Knowledge. The work on Intercom was made possible by GreenKahuna.

2.6.3 Helping out

If you want to help out with the development of django-analytical, by posting detailed bug reports, proposing new features or other analytics services to support, or suggesting documentation improvements, use the issue tracker. If you want to get your hands dirty, great! Clone the repository, make changes and place a pull request. Creating an issue to discuss your plans is useful.

This is a Jazzband project. By contributing you agree to abide by the Contributor Code of Conduct and follow the guidelines.

2.7 License

The django-analytical package is distributed under the MIT License. The complete license term are included below. The copyright of the integration code snippets of individual services rest solely with the respective service providers.

2.7.1 License terms

Index

A