Splango Documentation

Release 0.1.0

Shimon Rura

Sep 27, 2017

Contents

Splango is a project intended to help developers do split-testing in Django websites.

Splango is designed to help you take the first steps with split (A/B) testing with minimal friction. It allows you to instantly declare and run a split test experiment in your templates or in Python code, and provides an admin UI for viewing simple funnel reports on the results.

Contents:

CHAPTER 1

Introduction

Splango is a project intended to help developers do split-testing in Django websites.

Splango is designed to help you take the first steps with split (A/B) testing with minimal friction. It allows you to instantly declare and run a split test experiment in your templates or in Python code, and provides an admin UI for viewing simple funnel reports on the results.

Features

This application provides... Some features are:

- X:
  - **‘A‘_**
  - **‘B‘_**
  - **‘C‘_**
- Y

CHAPTER 2

Demo

There will be a demo at X.

CHAPTER 3

Installation

Dependencies

Dependencies that **must** be met to use the application:

- Django, obviously

Get a copy

From pypi:

```
$ pip install Splango
```

or:

```
$ easy_install Splango
```

or clone from github:

```
$ git clone git://github.com/lookup/Splango.git
```

and add it to the Python path.

or:

```
$ cd Splango
$ sudo python setup.py install
```

CHAPTER 4

Miscellaneous

Mailing list

Coming soon.

CHAPTER 5

Bugs

Please report issues in github

CHAPTER 6

ChangeLog

0.1.1 / 2013-03-13

- Blah

0.1.0 / 2013-03-12

- Buh
- Burrito

CHAPTER 7

Copyrights and License

`Splango` is protected by MIT license.

Some bits were derived from others work and copyrighted by:

- django-social-auth:

  ```
  Original Copyright goes to Matías Aguirre (omab)
  Documentation source borrowed from https://github.com/omab/django-social-auth
  ```

CHAPTER 8

splango

splango Package

`splango` Package

class `splango.__init__.`**`RequestExperimentManager`**(*request*)

> **`declare_and_enroll`**(*exp_name*, *variants*)
>
> **`enqueue`**(*action*, *params*)
>
> **`finish`**(*response*)
> Decide what to do if subject is human or not.
>
> **`get_subject`**()
>
> **`log_goal`**(*goal_name*, *extra=None*)
>
> **`process_from_queue`**(*action*, *params*)

`middleware` Module

class `splango.middleware.`**`ExperimentsMiddleware`**

> **`process_request`**(*request*)
> Assign the Experiment Manager to the request.
>
> **`process_response`**(*request*, *response*)
> Retrieve the Experiment Manager from the request and assign it to the response.

`models` Module

class `splango.models.`**`Enrollment`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

Identifies which variant a subject is assigned to in a given experiment.

`experiment`

`get_next_by_created``(`**moreargs*, ***morekwargs*`)`

`get_previous_by_created``(`**moreargs*, ***morekwargs*`)`

`subject`

`variant`

class `splango.models.`**`Experiment`**`(`**args*, ***kwargs*`)`
Bases: `django.db.models.base.Model`

A named experiment.

An experiment has a lot of variants, and a variant belongs to only one experiment.

classmethod `declare``(`*name*, *variants_names*`)`
create or update an experiment and its variants (variant names given).

`enrollment_set`

`experimentreport_set`

`get_next_by_created``(`**moreargs*, ***morekwargs*`)`

`get_or_create_enrollment``(`*subject*, *variant=None*`)`
Get or create an *`Enrollment`* object for `subject`.

Only if the object is to be created will `variant` be used. If `variant` is None, a random variant will be assigned.

Parameters

- **`subject`** (*`Subject`*) – the subject of the enrollment
- **`variant`** (*`str or None`*) – when creating the object, it is the variant to use; if None, a random variant will be used created, this will be the value for *`Enrollment.variant`*

Returns the enrollment for `subject`

Return type *`Enrollment`*

`get_previous_by_created``(`**moreargs*, ***morekwargs*`)`

`get_random_variant``()`
Return one of the object's variants chosen in a random way.

Warning: There is a reason why a `random.Random` generator is created in every call: using `random.choice()` used use the same generator every time because of it is not really a function but a method of a hidden instance of `random.Random`, defined in `random`. Debugging we could see that the instance's internal state was always the same, thus the output will not be random!

Also, `random.Random.jumpahead()` seemed to be the solution but it is not recommended and was removed in Python 3.

Returns variant

Return type basestring

get_variants()

variants

variants_commasep()

class splango.models.**ExperimentReport**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

A report on the results of an experiment.

experiment

generate()
Generate the report for experiment.

Generate the report of a experiment goals and variants.

Associate each variant with a goal, and associate the variant count too.

Returns A dict with goals, variants and variants counts associated to each goal

get_funnel_goals()

class splango.models.**Goal**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

An experiment goal, that is what we are waiting to happen.

get_next_by_created(**moreargs*, ***morekwargs*)

get_previous_by_created(**moreargs*, ***morekwargs*)

get_records_count_per_variant(*experiment*)
Get the goal records count and the respective percentage per variant.

>> goal.get_records_count_per_variant(experiment) {8: (1, 25.0), 1: (2, 50.0), 2: (0, 0.0), 6: (1, 25.0), 9: (0, 0.0)}

Parameters **experiment** (*Experiment*) –

Returns count of *GoalRecord* objects and percentage for each variant of experiment

Return type dict

get_records_total(*experiment*)
Get the goal records total for an experiment, including all its variants

goalrecord_set

subject_set

class splango.models.**GoalRecord**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

Associate the goal reached by a subject with that subject.

static extract_request_info(*request*)

get_next_by_created(**moreargs*, ***morekwargs*)

get_previous_by_created(**moreargs*, ***morekwargs*)

goal

classmethod **`record`**(*subject*, *goal_name*, *request_info*, *extra=None*)

classmethod **`record_user_goal`**(*user*, *goal_name*)

`subject`

class `splango.models.`**`Subject`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

An experimental subject, possibly also a registered user (at creation or later on.

`enrollment_set`

`get_next_by_created`(**moreargs*, ***morekwargs*)

`get_previous_by_created`(**moreargs*, ***morekwargs*)

`get_variants`()

Return all the variants shown to this subject.

The relationship is established through *`Enrollment`*, which has foreign keys to both *`Variant`* and *`Subject`*.

See also:

See analogous method *`Variant.get_subjects()`*.

Returns the variants

Return type queryset of *`Variant`*

`goalrecord_set`

`goals`

`is_registered_user`()

Is this subject associated to a registered user?

Returns True if subject is a registered user i.e. associated to a `django.contrib.auth.models.User`

Return type bool

`merge_into`(*other_subject*)

Move the enrollments and goal records associated with this subject into `other_subject`, preserving `other_subject`'s enrollments in case of conflict.

`registered_as`

class `splango.models.`**`Variant`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

An Experiment Variant, with optional weight

(The weight is not considered at the moment)

`enrollment_set`

`experiment`

`get_goal_records`(*goal*)

Return all the records of `goal` for this variant.

Parameters **`goal`** (*`Goal`*) –

Returns the goal records

Return type queryset of `GoalRecord`

`get_subjects``()`

Return all the subjects to whom this variant was shown.

The relationship is established through `Enrollment`, which has foreign keys to both `Variant` and `Subject`.

See also:

See analogous method `Subject.get_variants()`.

Returns the subjects

Return type queryset of `Subject`

`utils` Module

Utilities for project Splango.

`splango.utils.`**`is_first_visit`**`(`*request*`)`

Tell whether it is the first visit by `request`'s visitor.

Current algorithm is very basic. It performs the following nested checks:

- if `user` in `request` is authenticated
- if there is a HTTP_REFERER
- if `request`'s host matches the referer

Parameters **`request`** (`django.http.HttpRequest`) – HTTP request

Returns True if this `request` is the first one by `request`'s visitor

Return type bool

`splango.utils.`**`replace_insensitive`**`(`*string*, *target*, *replacement*`)`

Similar to string.replace() but is case insensitive

Code borrowed from `debug_toolbar` and http://forums.devshed.com/python-programming-11/case-insensitive-string-replace-490921.html

`views` Module

`splango.views.`**`experiment_detail`**`(`*request*, **args*, ***kwargs*`)`

Show the experiment and its reports.

`splango.views.`**`experiment_goal_report`**`(`*request*, **args*, ***kwargs*`)`

Goal results for all the variants in a experiment.

Parameters

- **`request`** –
- **`exp_name`** – The name of the experiment
- **`goal_name`** – The name of the goal

Returns

`splango.views.`**`experiment_log`**(*request*, **args*, ***kwargs*)

Show an enrollment, that dentifies which variant a subject is assigned to in a given experiment.

In the response, it returns the experiment itself; the activities, that shows what goals reached by the subject, with the given variant, and the title, that shows the activity in string format.

Returns The experiment log response

`splango.views.`**`experiment_report`**(*request*, **args*, ***kwargs*)

Show the experiment report.

`splango.views.`**`experiments_overview`**(*request*, **args*, ***kwargs*)

Show experiments list.

`splango.views.`**`goal_report`**(*request*, **args*, ***kwargs*)

Goal results for all the variants.

Parameters **`goal_name`** – The name of the goal

Returns

Subpackages

templatetags Package

`splangotags` Module

class `splango.templatetags.splangotags.`**`ExperimentNode`**(*exp_name*, *variants_str*)

Bases: `django.template.base.Node`

Template node for the {% experiment ... %} template tag.

This

`render()` returns an empty string (thus `{% experiment "..." variants "...,...,..." %}` renders nothing) but it must be called so that the experiment is recorded appropriately.

`render`(*context*)

Declare the experiment and enroll a variant. Render nothing.

Parameters **`context`** (`django.template.context.Context`) – template context

Returns empty string

Return type basestring

Raises `django.template.TemplateSyntaxError` if `'request'` is not in `context`, or if the former does not have an experiments manager.

class `splango.templatetags.splangotags.`**`HypNode`**(*exp_name*, *exp_variant*, *node_list*)

Bases: `django.template.base.Node`

Template node for a `{% hyp %}` template tag.

This *`render()`* method of this class either returns an empty string or the inner nodes rendered.

`render`(*context*)

Render the node list if `exp_variant` is the enrolled variant.

Parameters **`context`** (`django.template.context.Context`) – template context

Returns `node_list` rendered or an empty string

Return type basestring

Raises `django.template.TemplateSyntaxError` if the experiment named `exp_name` has not been declared yet

`splango.templatetags.splangotags.`**`experiment`**(*parser*, *token*)

Return a *`ExperimentNode`* according to the contents of `token`.

Example:: {% experiment "signup_button" variants "red,blue" %}

Parameters

- **`parser`** – template parser object, not used
- **`token`** (`django.template.base.Token`) – tag contents i.e. between `{% `` and `` %}`

Returns experiment node

Return type *`ExperimentNode`*

Raises `django.template.TemplateSyntaxError` if tag arguments in `token` are different than three

`splango.templatetags.splangotags.`**`hyp`**(*parser*, *token*)

Return a *`HypNode`* according to the contents of `token`.

Example:: {% hyp "signup_button" "blue" %}

Parameters

- **`parser`** (`django.template.base.Parser`) – template parser object
- **`token`** (`django.template.base.Token`) – tag contents i.e. between `{% `` and `` %}`

Returns experiment node

Return type *`ExperimentNode`*

Raises `django.template.TemplateSyntaxError` if tag arguments in `token` are different than two

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index