flask-bitmapist Documentation

Release 0.1.2

Cuttlesoft

September 14, 2016

Contents

Flask-Bitmapist is a Flask extension that creates a simple interface to the Bitmapist analytics library.

Events are registered with the name of the event (e.g., "user:logged_in") and the object id (e.g., the logged in user's id).

There are four different ways to register events from your Flask app:

- Call a function decorated with the `@mark()` decorator
- Use the `Bitmapistable` mixin (note: current ORM support is limited to SQLAlchemy)
- With the Flask-Login extension, user login/logout will register corresponding events automatically
- Call the `mark_event()` function directly

To use the `@mark()` decorator:

```
@mark('user:reset_password', user.id)
def reset_password():
    pass
```

To use the `Bitmapistable` mixin:

```
from flask_bitmapist import Bitmapistable

class User(db.Model, Bitmapistable):
    pass
```

If you are using Flask-Login, "user:logged_in" and "user:logged_out" events will be registered automatically on user login and user logout, respectively:

```
>>> flask_login.login_user(user)
>>> flask_login.logout_user()
```

You can also call the `mark_event()` function directly:

```
>>> mark_event('user:action_taken', user.id)
```

CHAPTER 1

Installation

Install the extension using pip:

```
$ pip install flask-bitmapist
```

CHAPTER 2

Quickstart

2.1 Initialization

Marking a user-based event is very simple with Flask-Bitmapist.

Begin by importing FlaskBitmapist and initializing the FlaskBitmapist application (this will need to be a Flask app):

```
from flask import Flask
from flask_bitmapist import FlaskBitmapist

# create Flask app object
app = Flask(__name__)

# initialize flask_bitmapist with the app object
flaskbitmapist = FlaskBitmapist()
flaskbitmapist.init_app(app)
```

Ensure that Redis is running; you can specify a port (default: 6379) with the `--port` flag:

```
$ redis-server
```

You are then free to use whichever method(s) you find best suited to your application for marking and registering events.

2.2 Configuration Options

Configuration Options	Description	Default
BITMAPIST_REDIS_URL	Location where Redis server is running	"redis://localhost:6379"
BITMAPIST_REDIS_SYSTEM	Name of Redis system to use for Bitmapist	"default"
BITMAPIST_TRACK_HOURLY	Whether to track events down to the hour	False
BITMAPIST_DISABLE_BLUEPRINT	Whether to disable registration of the default blueprint	False

2.3 Usage

2.3.1 Decorator

Usage of the `@mark()` decorator can be useful when you want to track interactions that do not deal directly with the database model.

To use, import the decorator and attach it to the function, providing the event name and user id:

```
from flask_bitmapist import mark

@mark('index:visited', current_user.id)
def index():
  return render_template('index.html')
```

2.3.2 Mixin

The mixin can be used to track when a user object is created, updated, or deleted. It interacts directly with the ORM to register events on insert, update, or delete.

To use, import the mixin and extend the desired class with it:

```
from flask_bitmapist import Bitmapistable

class User(db.Model, Bitmapistable):
  id = db.Column(db.Integer, primary_key=True)
```

The event "user:created" will then be registered when a new user is instantiated and committed to the database:

```
user = User()
db.session.add(user)
db.session.commit()
```

Similarly, "user:updated" and "user:deleted" will be registered for a given user on updating and deleting, respectively.

2.3.3 Flask-Login

The Flask-Login extension is a common means of user management for many Flask applications. Flask-Bitmapist integrates with this extension to track user login and logout events automatically via Flask-Login's LoginManager and UserMixin:

```
from flask_login import LoginManager, UserMixin

class User(UserMixin):
    id = None

login_manager = LoginManager()
login_manager.init_app(app)
```

Create and log in the user, and the event "user:logged_in" will be registered automatically; the same works for logging out a user and the "user:logged_out" event:

```
from flask_login import login_user, logout_user

user = User(id=user_id)
```

```
# login user
login_user(user)

# logout user
logout_user()
```

2.3.4 Function Call

The most raw way to use Flask-Bitmapist is to directly call `mark_event()`:

```
from flask_bitmapist import mark_event

mark_event('event:completed', current_user.id)
```

2.4 Small Example App

```
from flask import Flask
from flask_bitmapist import FlaskBitmapist, mark

app = Flask(__name__)

flaskbitmapist = FlaskBitmapist()
flaskbitmapist.init_app(app)

@app.route('/')
@mark('index:visited', 1)  # current_user.id
def index():
  """using the mark decorator, the first argument is the event
     and the second is the id of the current_user
  """
  return 'Hello, world!'

if __name__ == '__main__':
  app.run()
```

CHAPTER 3

Testing

To run the tests, ensure that you have Redis running on port 6399:

```
$ redis-server --port 6399
```

Then you can simply run:

```
$ python setup.py test
```

To seed fake data for testing, run:

```
$ python scripts/seed.py
```

CHAPTER 4

API

flask_bitmapist.utils.**get_event_data**(*event_name*, *time_group='days'*, *now=None*, *system='default'*)

Get the data for a single event at a single event in time.

Parameters

- **event_name** (*str*) – Name of event for retrieval
- **time_group** (*str*) – Time scale by which to group results; can be *days*, *weeks*, *months*, *years*
- **now** (*datetime*) – Time point at which to get event data (defaults to current time if None)
- **system** (*str*) – Which bitmapist should be used

Returns Bitmapist events collection

flask_bitmapist.utils.**get_cohort**(*primary_event_name*, *secondary_event_name*, *additional_events=[]*, *time_group='days'*, *num_rows=10*, *num_cols=10*, *system='default'*, *with_replacement=False*)

Get the cohort data for multiple chained events at multiple points in time.

Parameters

- **primary_event_name** (*str*) – Name of primary event for defining cohort
- **secondary_event_name** (*str*) – Name of secondary event for defining cohort
- **additional_events** (*list*) – List of additional events by which to filter cohort (e.g., `[{'name': 'user:logged_in', 'op': 'and'}]`)
- **time_group** (*str*) – Time scale by which to group results; can be *days*, *weeks*, *months*, *years*
- **num_rows** (*int*) – How many results rows to get; corresponds to how far back to get results from current time
- **num_cols** (*int*) – How many results cols to get; corresponds to how far forward to get results from each time point
- **system** (*str*) – Which bitmapist should be used
- **with_replacement** (*bool*) – Whether more than one occurence of an event should be counted for a given user; e.g., if a user logged in multiple times, whether to include subsequent logins for the cohort

Returns Tuple of (list of lists of cohort results, list of dates for cohort, primary event total for each date)

flask_bitmapist.utils.**chain_events**(*base_event_name*, *events_to_chain*, *now*, *time_group*, *system='default'*)

Chain additional events with a base set of events.

Note: `OR` operators will apply only to their direct predecessors (i.e., `A && B && C || D` will be handled as `A && B && (C || D)`, and `A && B || C && D` will be handled as `A && (B || C) && D`).

Parameters

- **base_event_name** (*str*) – Name of event to chain additional events to/with
- **events_to_chain** (*list*) – List of additional event names to chain (e.g., `[{'name': 'user:logged_in', 'op': 'and'}]`)
- **now** (*datetime*) – Time point at which to get event data
- **time_group** (*str*) – Time scale by which to group results; can be *days*, *weeks*, *months*, *years*
- **system** (*str*) – Which bitmapist should be used

Returns Bitmapist events collection

4.1 Indices and tables

- genindex
- modindex
- search

C

G