Swingtime Documentation

Release 0.3.2

David Krauth

September 05, 2016

Contents

CHAPTER 1

Table of Contents

1.1 Calendar application for Mezzanine using the fullcalendar.io widget

Mezzanine-fullcalendar is a calendar application for Mezzanine, using the awesome fullcalendar.io javascript widget, which provides a modern looking calendar.

This project originally started out as a fork of django-swingtime, but deviated so much from the original project, it is now a separate project.

1.1.1 Current features

- Admin interface for managing events and their occurrences.
- Event categories
- Class based views for the calendar, JSON data, and an agenda view with upcoming events.
- Support for multiple sites

1.1.2 Planned features

- Make use of the editing features of the fullcalendar javascript widget.
- Create forms for easily adding new events through the calendar.
- ICal export
- More tests and docs

1.1.3 Requirements

- Python 2.7+, 3.4+
- Django 1.6+
- python-dateutil

1.2 Installation

1.2.1 Get `Swingtime`

Options:

- Source: https://github.com/jonge-democraten/mezzanine-swingtime
- `curl -o swingtime.zip -L https://github.com/jonge-democraten/mezzanine-swingtime/archi`

1.3 Demo

1.3.1 Intro

Swingtime comes with its own demo project and application. The demo is themed as a Karate studio's website and allows you see and interact with the Swingtime application.

Within the Swingtime demo is an app named `karate`, which defines the custom management command `loaddemo`. This command will pre-populate your initial database with some events and occurrences based upon the current date and time.

Currently, Swingtime does not include any templates of its own. The demo project provides some sample templates to use as a guide or starting point.

1.3.2 Running the demo

Get the code:

```
$ git clone https://github.com/jonge-democraten/mezzanine-swingtime.git mezzanine-swingtime-master
```

– or –

```
$ curl -o swingtime.zip -L https://github.com/jonge-democraten/mezzanine-swingtime/archive/master.zip
$ unzip swingtime.zip
```

You can set up your environment to run the demo in a `virtualenv` by doing the following (please note that for the following commands you have already installed `virtualenv` and `virtualenvwrapper`):

```
$ cd mezzanine-swingtime-master/demo
$ mkvirtualenv swingtime_demo
$ pip install -r requirements.txt
$ pip install pytz django-extensions Sphinx # <-- optional
```

And, finally:

```
$ python manage.py loaddemo
$ python manage.py runserver
```

`loaddemo` is just a simple wrapper around `syncdb` and a short script to load some data into your new database (by default, a sqlite3 database named `karate.db`) in the root directory of the demo.

Now, you are ready to browse to http://127.0.0.1:8000/

1.3.3 Optional

- Build the documentation files as HTML:

```
$ cd docs && make html  # assumes you have Sphinx installed
```

- Run development server to check for deprecation warning:

```
$ python -Wd manage.py runserver
```

1.4 `models` — Swingtime Object Model Definitions

1.4.1 Functions

`create_event`

`models.`**`create_event`**(*title*, *event_type*[, *description*, *start_time*, *end_time*, *note*, ***rrule_params*])

Convenience function to create an *`Event`*, optionally create an *`EventType`*, and associated *`Occurrence`* instances. *`Occurrence`* creation rules match those for *`Event.add_occurrences()`*.

Returns the newly created *`Event`* instance.

Parameters

`event_type` can be either an *`EventType`* object or 2-tuple of `(abbreviation,label)`, from which an *`EventType`* is either created or retrieved.

`description` sets the event's description if not `None`

`start_time` will default to the current hour if `None`

`end_time` will default to `start_time` plus *`swingtime_settings.DEFAULT_OCCURRENCE_DURATION`* hour if `None`

`note` if not `None`, add a `Note` instance to the new event

`rrule_params` follows the `dateutils` API (see http://labix.org/python-dateutil)

Example:

```
from datetime import datetime, time
from swingtime import models as swingtime
from dateutil import rrule

event = swingtime.create_event(
    'Beginner Class',
    ('bgn', 'Beginner Classes'),
    description='Open to all beginners',
    start_time=datetime.combine(datetime.now().date(), time(19)),
    count=6,
    byweekday=(rrule.MO, rrule.WE, rrule.FR)
)
```

1.4.2 Classes

Note

class models.**Note**(*django.db.models.Model*)

A generic model for adding simple, arbitrary notes to other models such as Event or Occurrence.

note
models.TextField

created
models.DateTimeField

EventType

class models.**EventType**(*django.db.models.Model*)

Simple Event classifcation.

abbr
models.CharField

label
models.CharField

Event

class models.**Event**(*django.db.models.Model*)

Container model for general metadata and associated Occurrence entries.

title
models.CharField

description
models.CharField

event_type
models.ForeignKey for EventType

notes
generic.GenericRelation for Note

get_absolute_url()
return ('swingtime-event', [str(self.id)])

add_occurrences(*start_time*, *end_time*[, ***rrule_params*])
Add one or more occurences to the event using a comparable API to dateutil.rrule.

If rrule_params does not contain a freq, one will be defaulted to rrule.DAILY.

Because rrule.rrule returns an iterator that can essentially be unbounded, we need to slightly alter the expected behavior here in order to enforce a finite number of occurrence creation.

If both count and until entries are missing from rrule_params, only a single Occurrence instance will be created using the exact start_time and end_time values.

upcoming_occurrences()
Return all occurrences that are set to start on or after the current time.

next_occurrence()
Return the single occurrence set to start on or after the current time if available, otherwise None.

daily_occurrences([*dt*])
Convenience method wrapping `Occurrence.objects.daily_occurrences`.

OccurrenceManager

class `models.`**OccurrenceManager**(*models.Manager*)

daily_occurrences([*dt*, *event*])
Returns a queryset of for instances that have any overlap with a particular day.

Parameters

dt may be either a datetime.datetime, datetime.date object, or `None`. If `None`, default to the current day

event can be an `Event` instance for further filtering

Occurrence

class `models.`**Occurrence**(*django.db.models.Model*)
Represents the start end time for a specific occurrence of a master *Event* object.

start_time
models.DateTimeField

end_time
models.DateTimeField

event
models.ForeignKey - a non-editable Event object

notes
generic.GenericRelation `Note`

get_absolute_url()
'swingtime-occurrence', [str(self.event.id), str(self.id)])

__cmp__()
Compare two `Occurrence` start times

title
Shortcut for the occurrence's `Event.title`

event_type
Shortcut for the occurrence's `EventType`

1.5 views — Swingtime Views

1.5.1 Functions

event_listing

`views.`**event_listing**(*request*[, *template='swingtime/event_list.html'*, *events=None*, ***extra_context*])
View all `events`.

If `events` is a queryset, clone it. If `None` default to all `Event` objects.

Context parameters:

events an iterable of `Event` objects

extra_context extra variables passed to the template context

event_view

views.**event_view**(*request*, *pk*[, *template='swingtime/event_detail.html'*, *event_form_class=forms.EventForm*, *recurrence_form_class=forms.MultipleOccurrenceForm*])

View an `Event` instance and optionally update either the event or its occurrences.

Context parameters:

event the event keyed by `pk`

event_form a form object for updating the event

recurrence_form a form object for adding occurrences

occurrence_view

views.**occurrence_view**(*request*, *event_pk*, *pk*[, *template='swingtime/occurrence_detail.html'*, *form_class=forms.SingleOccurrenceForm*])

View a specific occurrence and optionally handle any updates.

Context parameters:

occurrence the occurrence object keyed by `pk`

form a form object for updating the occurrence

add_event

views.**add_event**(*request*[, *template='swingtime/add_event.html'*, *event_form_class=forms.EventForm*, *recurrence_form_class=forms.MultipleOccurrenceForm*])

Add a new `Event` instance and 1 or more associated `Occurrence` instancess.

Context parameters:

dtstart a datetime.datetime object representing the GET request value if present, otherwise None

event_form a form object for updating the event

recurrence_form a form object for adding occurrences

_datetime_view

views.**_datetime_view**(*request template*, *dt*[, *timeslot_factory=None*, *items=None*, *params=None*])

Build a time slot grid representation for the given datetime `dt`. See utils.create_timeslot_table documentation for items and params.

Context parameters:

day the specified datetime value (dt)

next_day day + 1 day

prev_day day - 1 day

timeslots time slot grid of (time, cells) rows

day_view

views.**day_view**(*request*, *year*, *month*, *day*[, *template='swingtime/daily_view.html'*, ***params*])
See documentation for function''_datetime_view''.

today_view

views.**today_view**(*request*[, *template='swingtime/daily_view.html'*, ***params*])
See documentation for function''_datetime_view''.

year_view

views.**year_view**(*request*, *year*[, *template='swingtime/yearly_view.html'*, *queryset=None*])
Context parameters:

year an integer value for the year in questin

next_year year + 1

last_year year - 1

by_month a sorted list of (month, occurrences) tuples where month is a datetime.datetime object for the first day of a month and occurrences is a (potentially empty) list of values for that month. Only months which have at least 1 occurrence is represented in the list

month_view

views.**month_view**(*request*, *year*, *month*[, *template='swingtime/monthly_view.html'*, *queryset=None*])
Render a tradional calendar grid view with temporal navigation variables.

Context parameters:

today the current datetime.datetime value

calendar a list of rows containing (day, items) cells, where day is the day of the month integer and items is a (potentially empty) list of occurrence for the day

this_month a datetime.datetime representing the first day of the month

next_month this_month + 1 month

last_month this_month - 1 month

1.6 forms — Swingtime Forms

Convenience forms for adding and updating Event and Occurrence objects.

1.6.1 Functions

timeslot_options

forms.**timeslot_options**([*interval=swingtime_settings.TIMESLOT_INTERVAL*, *start_time=swingtime_settings.TIMESLOT_START_TIME*, *end_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION*, *fmt=swingtime_settings.TIMESLOT_TIME_FORMAT*])

Create a list of time slot options for use in swingtime forms.

The list is comprised of 2-tuples containing a 24-hour time value and a 12-hour temporal representation of that offset.

timeslot_offset_options

forms.**timeslot_offset_options**([*interval=swingtime_settings.TIMESLOT_INTERVAL*, *start_time=swingtime_settings.TIMESLOT_START_TIME*, *end_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION*, *fmt=swingtime_settings.TIMESLOT_TIME_FORMAT*])

Create a list of time slot options for use in swingtime forms.

The list is comprised of 2-tuples containing the number of seconds since the start of the day and a 12-hour temporal representation of that offset.

1.6.2 Data

default_timeslot_options

forms.**default_timeslot_options**

defaults to `timeslot_options()`

default_timeslot_offset_options

forms.**default_timeslot_offset_options**

defaults to `timeslot_offset_options()`

1.6.3 Classes

MultipleIntegerField

class forms.**MultipleIntegerField**(*django.forms.MultipleChoiceField*)

A form field for handling multiple integers.

def __init__(self, choices, size=None, label=None, widget=None):

if widget is None: widget = forms.SelectMultiple(attrs={'size' : size or len(choices)})

SplitDateTimeWidget

class forms.**SplitDateTimeWidget**(*django.forms.MultiWidget*)

A Widget that splits datetime input into a SelectDateWidget for dates and Select widget for times.

__init__(*attrs=None*)
uses widgets `SelectDateWidget` and `forms.Select(choices=default_timeslot_options`

MultipleOccurrenceForm

class `forms.`**MultipleOccurrenceForm**(*django.forms.Form*)

day
forms.DateField

start_time_delta
forms.IntegerField

end_time_delta
forms.IntegerField

repeats
forms.ChoiceField

count
forms.IntegerField

until
forms.DateField

freq
forms.IntegerField

interval
forms.IntegerField

week_days
MultipleIntegerField

month_ordinal
forms.IntegerField

month_ordinal_day
forms.IntegerField

each_month_day = MultipleIntegerField(

year_months
MultipleIntegerField

is_year_month_ordinal
forms.BooleanField(required=False)

year_month_ordinal
forms.IntegerField(widget=forms.Select(choices=ORDINAL))

year_month_ordinal_day
forms.IntegerField(widget=forms.Select(choices=WEEKDAY_LONG))

__init__(*[*args, **kws*]*)
if `initial` contains `dtstart` - a `datetime.datetime` instance - the appropriate unspecified `initial` will be defaulted for the form.

clean()
populates `cleaned_data` with `start_time` and `end_time` values

save(event):
Returns an `Event` object

EventForm

class `forms.`**EventForm**(*django.forms.ModelForm*)
A simple form for adding and updating Event attributes

SingleOccurrenceForm

class `forms.`**SingleOccurrenceForm**(*django.forms.ModelForm*)
A simple form for adding and updating single Occurrence attributes

1.7 `utils` — Swingtime Utilities

Common features and functions for swingtime

1.7.1 Functions

html_mark_safe

`utils.`**html_mark_safe**(*func*)
Decorator for functions return strings that should be treated as template safe.

time_delta_total_seconds

`utils.`**time_delta_total_seconds**(*time_delta*)
Calculate the total number of seconds represented by a `datetime.timedelta` object

month_boundaries

`utils.`**month_boundaries**(*[dt=None]*)
Return a 2-tuple containing the datetime instances for the first and last dates of the current month or using `dt` as a reference.

css_class_cycler

`utils.`**css_class_cycler**()
Return a dictionary keyed by *EventType* abbreviations, whose values are an iterable or cycle of CSS class names.

create_timeslot_table

utils.**create_timeslot_table**([*dt=None*, *items=None*, *start_time=swingtime_settings.TIMESLOT_START_TIME*, *end_time_delta=swingtime_settings.TIMESLOT_END_TIME_DURATION*, *time_delta=swingtime_settings.TIMESLOT_INTERVAL*, *min_columns=swingtime_settings.TIMESLOT_MIN_COLUMNS*, *css_class_cycles=css_class_cycler*, *proxy_class=DefaultOccurrenceProxy*])

Create a grid-like object representing a sequence of times (rows) and columns where cells are either empty or reference a wrapper object for event occasions that overlap a specific time slot.

Currently, there is an assumption that if an occurrence has a `start_time` that falls with the temporal scope of the grid, then that `start_time` will also match an interval in the sequence of the computed row entries.

dt a `datetime.datetime` instance or `None` to default to now

items a queryset or sequence of *Occurrence* instances. If `None`, default to the daily occurrences for `dt`

start_time a `datetime.time` instance, defaulting to *swingtime_settings.TIMESLOT_START_TIME*

end_time_delta a `datetime.timedelta` instance, defaulting to *swingtime_settings.TIMESLOT_END_TIME_DURATION*

time_delta a `datetime.timedelta` instance, defaulting to *swingtime_settings.TIMESLOT_INTERVAL*

min_column the minimum number of columns to show in the table, defaulting to *swingtime_settings.TIMESLOT_MIN_COLUMNS*

css_class_cycles if not `None`, a callable returning a dictionary keyed by desired `EventType` abbreviations with values that iterate over progressive CSS class names for the particular abbreviation; defaults to *css_class_cycler()*

proxy_class a wrapper class for accessing an `Occurrence` object, which should also expose `event_type` and `event_class` attrs, and handle the custom output via its __unicode__ method; defaults to *DefaultOccurrenceProxy*

1.7.2 Classes

BaseOccurrenceProxy

class utils.**BaseOccurrenceProxy**(*object*)

A simple wrapper class for handling the representational aspects of an *Occurrence* instance.

DefaultOccurrenceProxy

class utils.**DefaultOccurrenceProxy**(*BaseOccurrenceProxy*)

Through the `__unicode__` method, outputs a **safe** string anchor tag for the *Occurrence* instance, followed by simple token placeholders to represent additional slot fillings.

1.8 swingtime_settings — Configuration Settings

1.8.1 Settings

Swingtime has it's settings module (`conf/swingtime_settings.py`) that simulates how each Django project's `setting.py` file functions. You can overwrite any or all of the configuration parameters described in

swingtime_settings by creating a file in your own project and referencing that file in your project settings using the name SWINGTIME_SETTINGS_MODULE.

For example, from the demo's configuration:

```
SWINGTIME_SETTINGS_MODULE = 'demo.swingtime_settings'
```

swingtime_settings.**TIMESLOT_TIME_FORMAT**
A "strftime" string for formatting start and end time selectors in forms. The default format is:

```
    '%I:%M %p'
```

swingtime_settings.**TIMESLOT_INTERVAL**
Used for creating start and end time form selectors as well as time slot grids. Value should be datetime.timedelta value representing the incremental differences between temporal options. The default is:

```
    ``datetime.timedelta(minutes=15)``
```

swingtime_settings.**TIMESLOT_START_TIME**
A datetime.time value indicting the starting time for time slot grids and form selectors. The default is:

```
    ``datetime.time(9)``
```

swingtime_settings.**TIMESLOT_END_TIME_DURATION**
A datetime.timedelta value indicating the offset value from TIMESLOT_START_TIME for creating time slot grids and form selectors. The purpose for using a time delta is that it possible to span dates. For instance, one could have a starting time of 3pm (15:00) and wish to indicate a ending value 1:30am (01:30), in which case a value of datetime.timedelta(hours=10.5) could be specified to indicate that the 1:30 represents the following date's time and not the current date. Default:

```
    ``datetime.timedelta(hours=+8)``
```

swingtime_settings.**TIMESLOT_MIN_COLUMNS**
Indicates a minimum value (default: 4) for the number grid columns to be shown in the time slot table.

swingtime_settings.**DEFAULT_OCCURRENCE_DURATION**
Indicate the default length in time for a new occurrence, specifed by using a datetime.timedelta object, defaulting to:

```
    ``datetime.timedelta(hours=+1)``
```

swingtime_settings.**CALENDAR_FIRST_WEEKDAY**
If not None, passed to calendar.setfirstweekday function. Default: 6

1.9 Changes in Swingtime

1.9.1 Release 0.4 (September 18, 2014)

- Added support for Python 3.4

1.9.2 Release 0.3.3 (September 17, 2014)

- Added support for Django 1.5, 1.6, 1.7

1.9.3 Release 0.2.2 (March 16, 2013)

- Registered in PyPI
- Installs with *pip*

1.9.4 Release 0.2 (Decemeber 18, 2008)

- First public release.

CHAPTER 2

Index

- genindex
- modindex
- search

f

m

s

u

v

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

R

S

T

U

V

W

Y