django-baton Documentation

Release 2.8.0

abidibo

Aug 11, 2023

Contents

A cool, modern and responsive django admin application based on bootstrap 5

Baton was developed with one concept in mind: **overwrite as few django templates as possible**. Everything is done with css (sass and bootstrap mixins), and when the markup needs some edit, then DOM manipulation through js is used.

CHAPTER 1

Features

- Supports django >= 2.1
- Based on bootstrap 5 and FontAwesome 6
- Fully responsive
- Custom and flexible sidebar menu
- Text input filters facility
- Configurable form tabs
- Easy way to include templates in the change form page
- Collapsable stacke inline entries
- Lazy load of current uploaded images
- Optional index page filled with google analytics widgets
- Full customization available recompiling the provided js app
- it translations

CHAPTER 2

Getting started

2.1 Installation

2.1.1 Using pip

1. Install the last available version:

```
pip install django-baton
```

Note: In order to use the Google Analytics index, install baton along the optional dependencies with `pip install django-baton[analytics]`

2. Add `baton` and `baton.autodiscover` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    'baton',
    'django.contrib.admin',
    # ... (place baton.autodiscover at the very end)
    'baton.autodiscover',
)
```

Important: `baton` must be placed before `django.contrib.admin` and `baton.autodiscover` as the last app.

3. Replace django.contrib.admin in your project urls, and add baton urls:

```
from baton.autodiscover import admin
from django.urls import path, include
```

(continues on next page)

(continued from previous page)

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('baton/', include('baton.urls')),

]
```

Important: If you get a "__No crypto library **available**__" when using the google analytics index, then install this package:

$ pip install PyOpenSSL

Why two installed apps?

The first baton has to be placed before the `django.contrib.admin` app, because it overrides 3 templates and resets all css. The `baton.autodiscover` entry is needed as the last installed app in order to register all applications for the admin. I decided to create a custom `AdminSite` class, in order to allow the customization of some variables the django way (`site_header`, `index_title`, ...). I think this is a good approach, better than customizing this vars overwriting the orignal templates. The problem is that when creating a custom `AdminSite`, you should register manually all the apps. I didn't like this, so I wrote this autodiscover module, which automatically registers all the apps already registered with the django default `AdminSite`. In order to do this, all the apps must be already registered, so it comes as the last installed app.

2.2 Configuration

You can configure your baton installation defining a config dictionary in your `settings.py`

2.2.1 Example

This is an example of configuration:

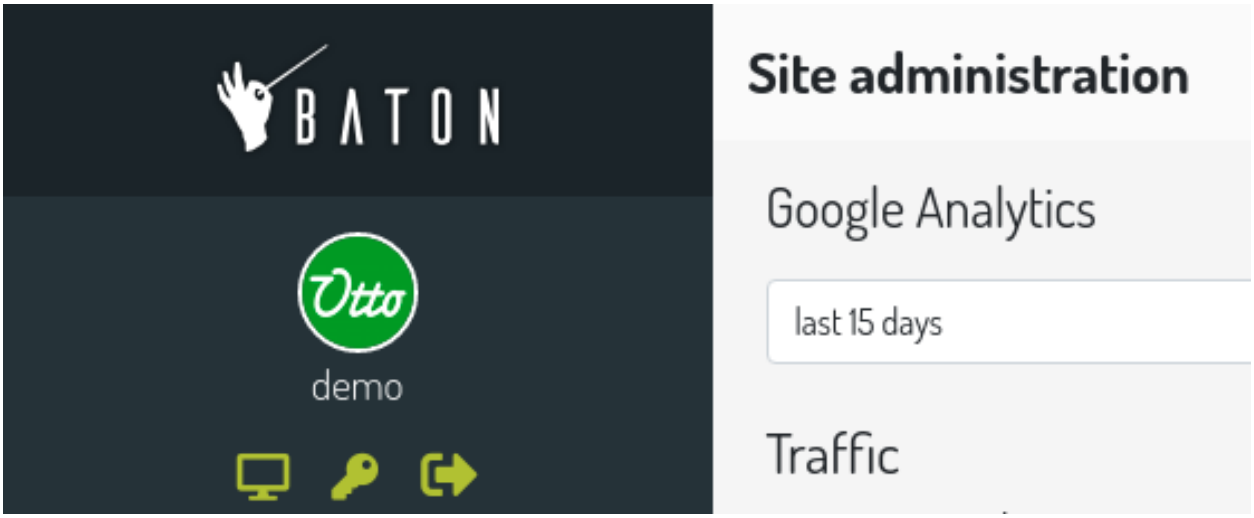
```
BATON = {
    'SITE_HEADER': 'Baton',
    'SITE_TITLE': 'Baton',
    'INDEX_TITLE': 'Site administration',
    'SUPPORT_HREF': 'https://github.com/otto-torino/django-baton/issues',
    'COPYRIGHT': 'copyright © 2017 <a href="https://www.otto.to.it">Otto srl</a>', #␣
↪noqa
    'POWERED_BY': '<a href="https://www.otto.to.it">Otto srl</a>',
    'CONFIRM_UNSAVED_CHANGES': True,
    'SHOW_MULTIPART_UPLOADING': True,
    'ENABLE_IMAGES_PREVIEW': True,
    'CHANGELIST_FILTERS_IN_MODAL': True,
    'CHANGELIST_FILTERS_ALWAYS_OPEN': False,
    'CHANGELIST_FILTERS_FORM': True,
    'COLLAPSABLE_USER_AREA': False,
    'MENU_ALWAYS_COLLAPSED': False,
    'MENU_TITLE': 'Menu',
    'MESSAGES_TOASTS': False,
    'GRAVATAR_DEFAULT_IMG': 'retro',
```

(continues on next page)

(continued from previous page)

```
    'GRAVATAR_ENABLED': True,
    'FORCE_THEME': None
    'LOGIN_SPLASH': '/static/core/img/login-splash.png',
    'SEARCH_FIELD': {
        'label': 'Search contents...',
        'url': '/search/',
    },
    'MENU': (
        { 'type': 'title', 'label': 'main', 'apps': ('auth', ) },
        {
            'type': 'app',
            'name': 'auth',
            'label': 'Authentication',
            'icon': 'fa fa-lock',
            'default_open': True,
            'models': (
                {
                    'name': 'user',
                    'label': 'Users'
                },
                {
                    'name': 'group',
                    'label': 'Groups'
                },
            )
        },
        { 'type': 'title', 'label': 'Contents', 'apps': ('flatpages', ) },
        { 'type': 'model', 'label': 'Pages', 'name': 'flatpage', 'app': 'flatpages' },
        { 'type': 'free', 'label': 'Custom Link', 'url': 'http://www.google.it',
↪'perms': ('flatpages.add_flatpage', 'auth.change_user') },
        { 'type': 'free', 'label': 'My parent voice', 'children': [
            { 'type': 'model', 'label': 'A Model', 'name': 'mymodelname', 'app':
↪'myapp', 'icon': 'fa fa-gavel' },
            { 'type': 'free', 'label': 'Another custom link', 'url': 'http://www.
↪google.it' },
        ] },
    ),
    'ANALYTICS': {
        'CREDENTIALS': os.path.join(BASE_DIR, 'credentials.json'),
        'VIEW_ID': '12345678',
    }
}
```

2.2.2 Site header



Default: baton logo

Important: `SITE_HEADER` is marked as safe, so you can include img tags or links

2.2.3 Site title

Default: 'Baton'

2.2.4 Index title

Default: 'Site administration'

2.2.5 Support href

This is the content of the href attribute of the support link rendered in the footer.

Default: 'https://github.com/otto-torino/django-baton/issues'

Example: 'mailto:support@company.org'

2.2.6 Copyright

A copyright string inserted centered in the footer

Default: 'copyright © 2017 Otto srl'

Important: `COPYRIGHT` is marked as safe, so you can include img tags or links

2.2.7 Powered by

A powered by information included in the right part of the footer, under the `SITE_TITLE` string

Default: 'Otto srl'

Important: `POWERED_BY` is marked as safe, so you can include img tags or links

---

2.2.8 Confirm unsaved changes

Alert the user when he's leaving a change or add form page without saving changes

Default: True

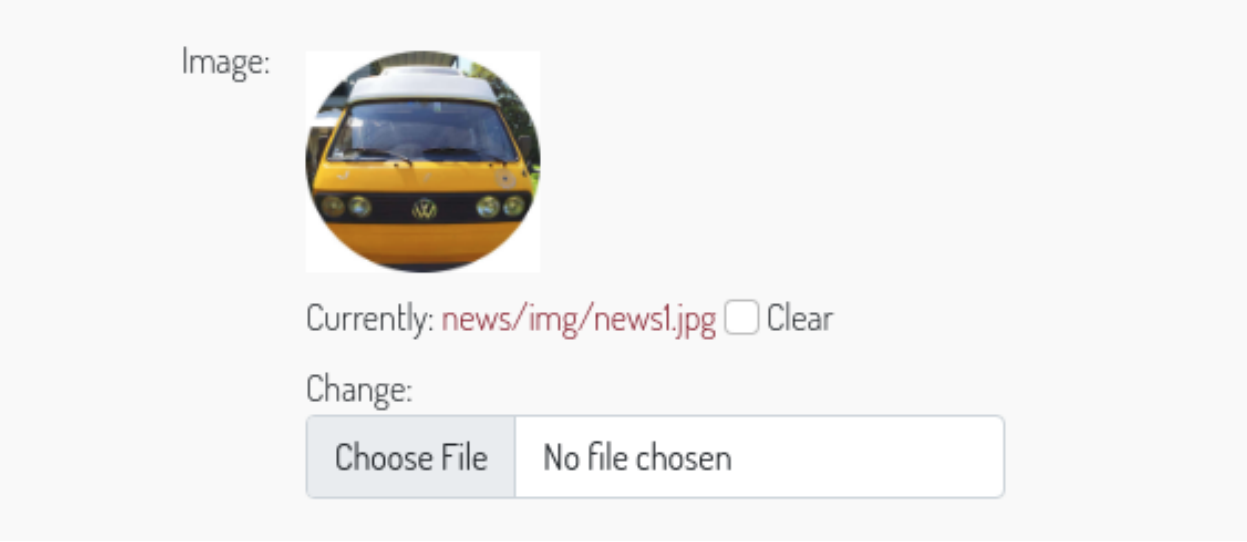
Important: The check for a dirty form relies on the jQuery serialize method, so it's not 100% safe. Disabled inputs, particular widgets (ckeditor) can not be detected.

2.2.9 Show multipart uploading

Show an overlay with a spinner when a `multipart/form-data` form is submitted

Default: True

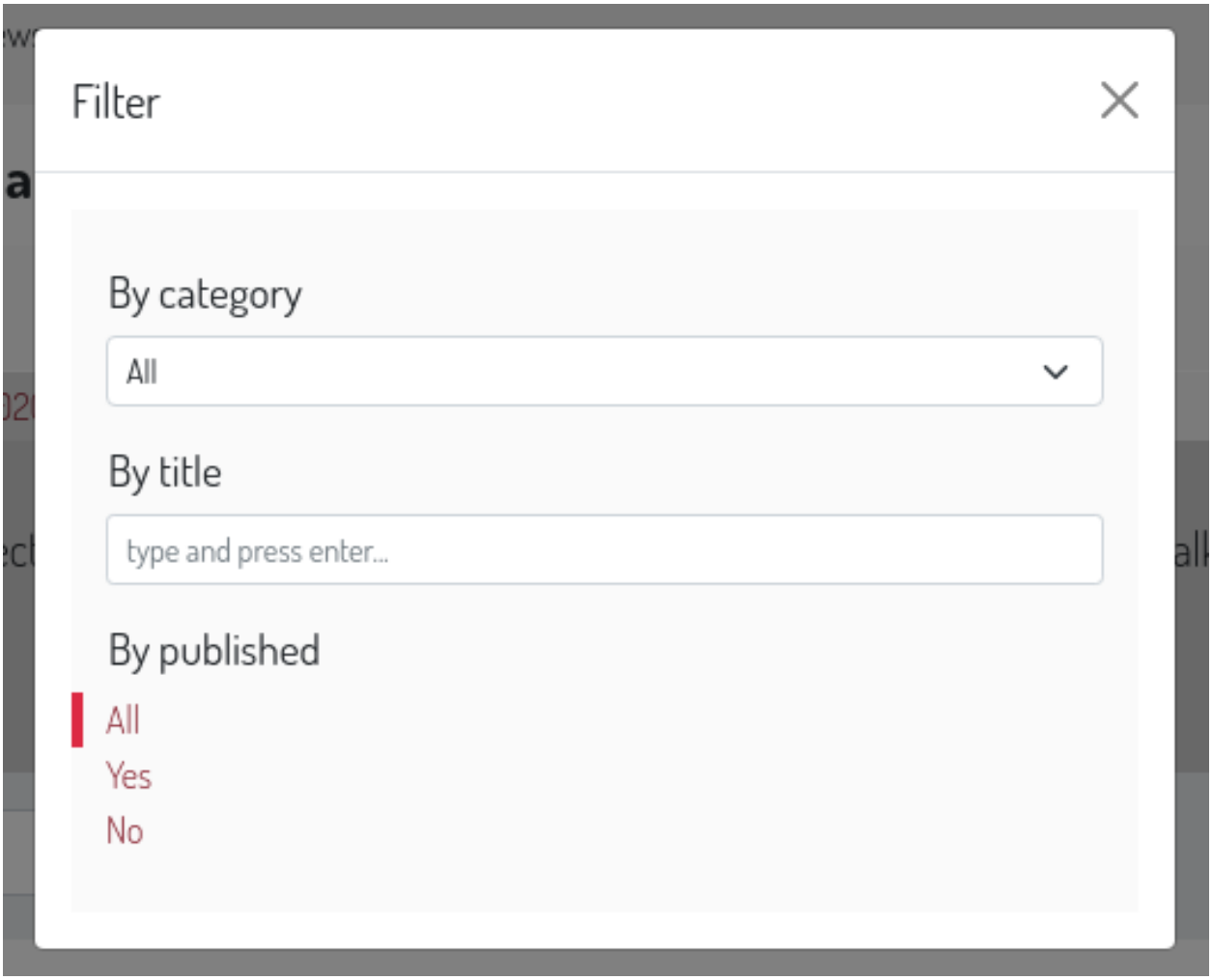
2.2.10 Enable images preview



Displays a preview above all input file fields which contain images. You can control how the preview is displayed overriding the class `.baton-image-preview`. By default previews are 100px height and with a box shadow on over event

Default: True

2.2.11 Changelist filters in modal



If set to `True` the changelist filters are opened in a centered modal above the document, useful when you set many filters. By default, its value is `False` and the changelist filters appears from the right side of the changelist table.

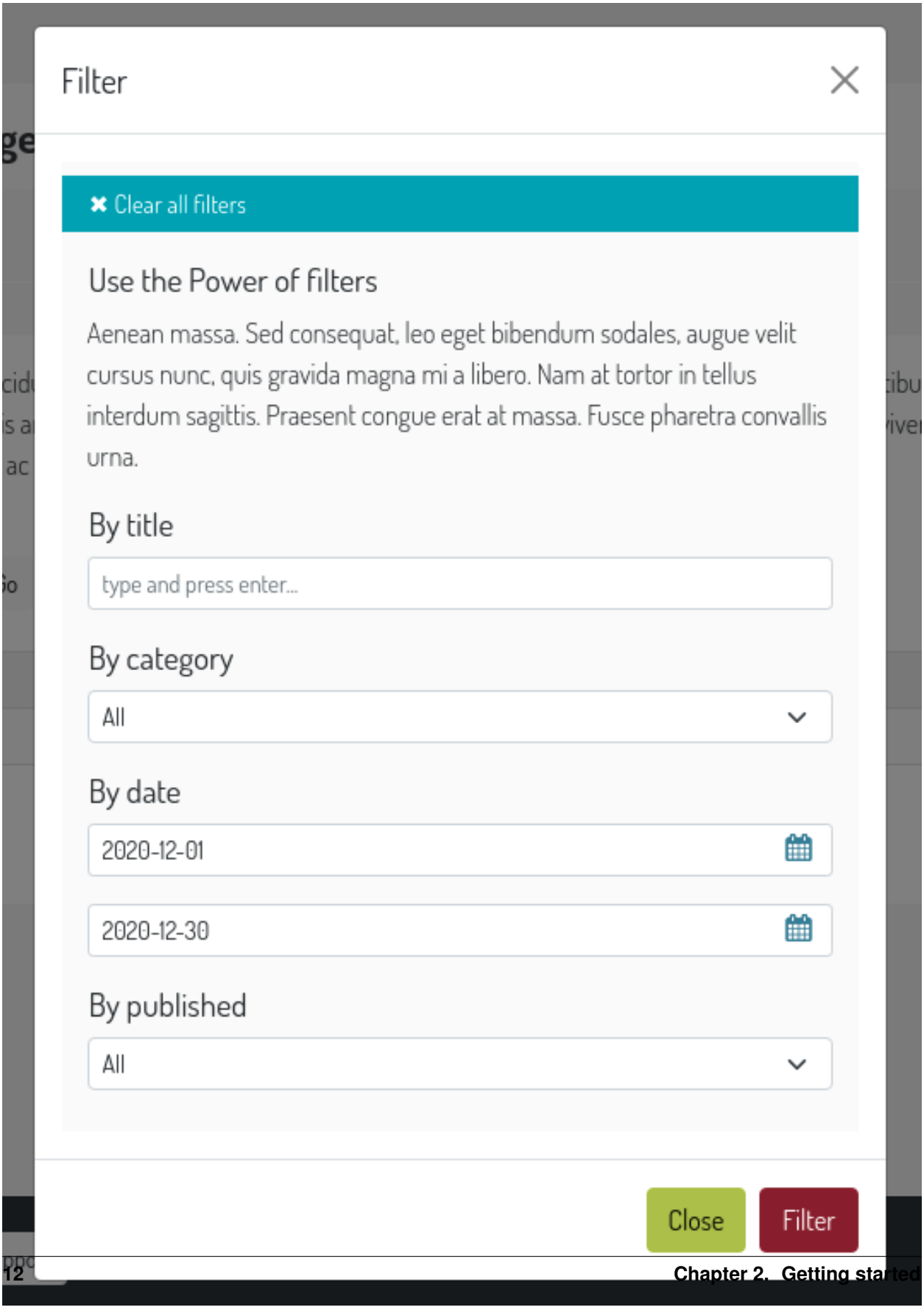
Default: False

2.2.12 Changelist filters always open

If set to `True` the changelist filters are opened by default. By default, its value is `False` and the changelist filters can be expanded clicking a toggler button. This option is considered only if `CHANGELIST_FILTERS_IN_MODAL` is `False`

Default: False

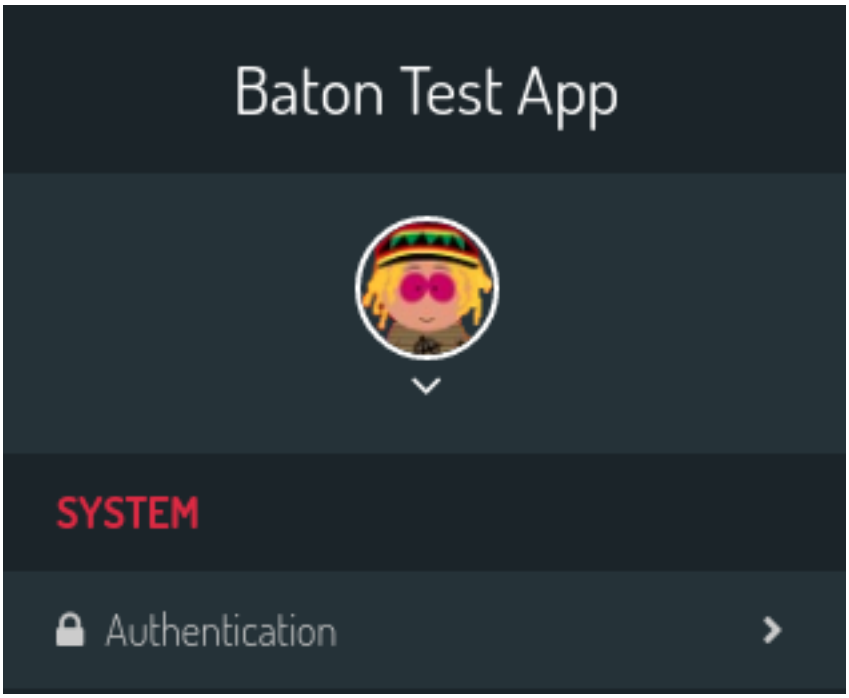
2.2.13 Changelist filters form



If set to `True` the changelist filters are treated as in a form, you can set many of them at once and then press a filter button in order to actually perform the filtering. With such option all standard filters are displayed as dropdowns.

Default: False

2.2.14 Collapsable user area



If set to `True` the sidebar user area is collapsed and can be expanded to show links.

Default: False

2.2.15 Menu always collapsed

If set to `True` the menu is hidden at page load, and the navbar toggler is always visible, just click it to show the sidebar menu.

Default: False

2.2.16 Menu title

The menu title shown in the sidebar. If an empty string, the menu title is hidden and takes no space on larger screens, the default menu voice will still be visible in the mobile menu.

2.2.17 Messages toasts

You can decide to show all or specific level admin messages in toasts. Set it to `True` to show all message in toasts. set it to `['warning', 'error']` to show only warning and error messages in toasts.

Default: False

2.2.18 Gravatar default image

The default gravatar image displayed if the user email is not associated to any gravatar image. Possible values: 404, mp, identicon, monsterid, wavatar, retro, robohash, blank (see *gravatar docs [http://en.gravatar.com/site/implement/images/]*).

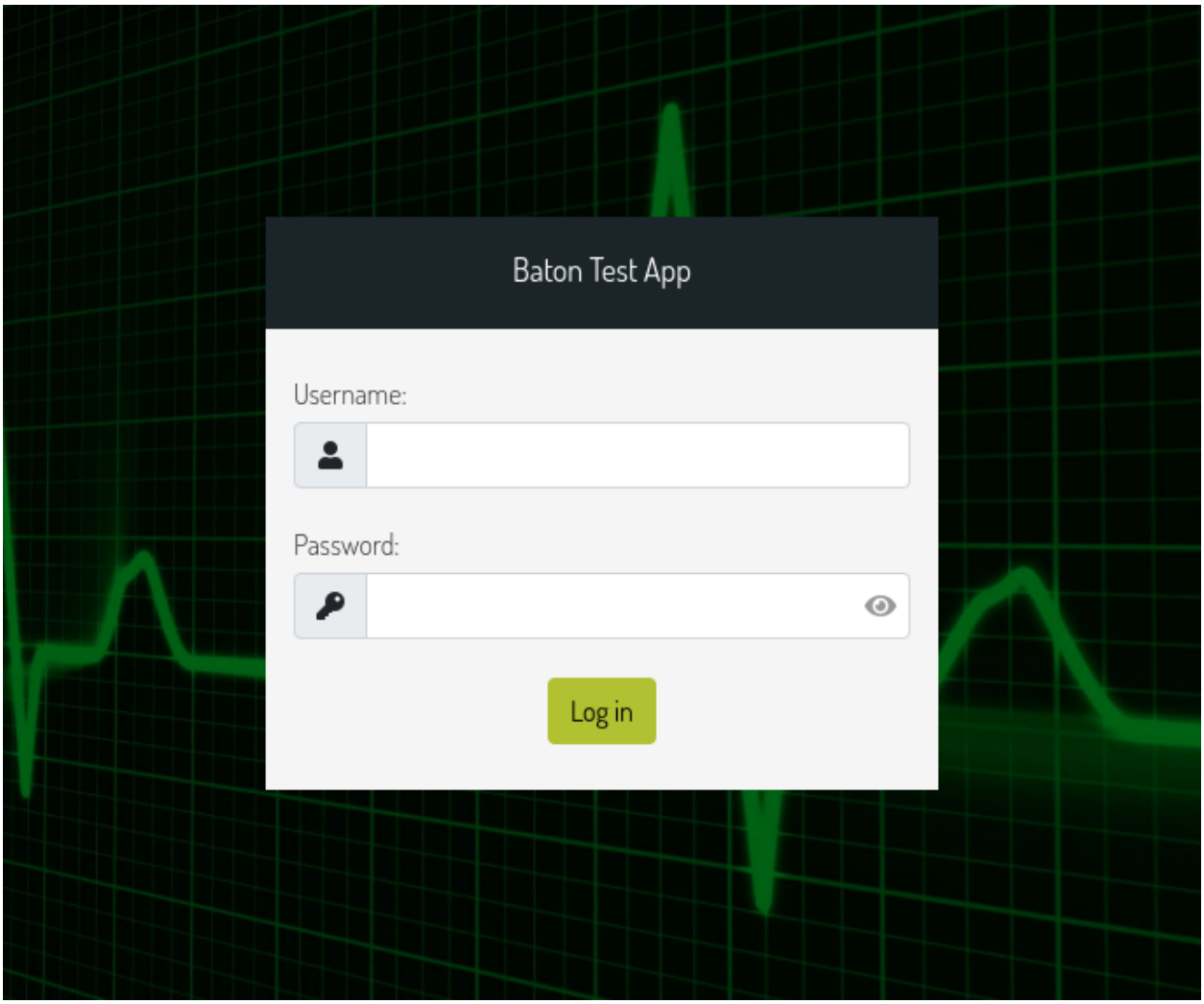
Default: 'retro'

2.2.19 Gravatar enabled

Should a gravatar image be shown for the user in the menu?

Default: True

2.2.20 Login splash image



An image used as body background in the login page. The image is centered and covers the whole viewport.

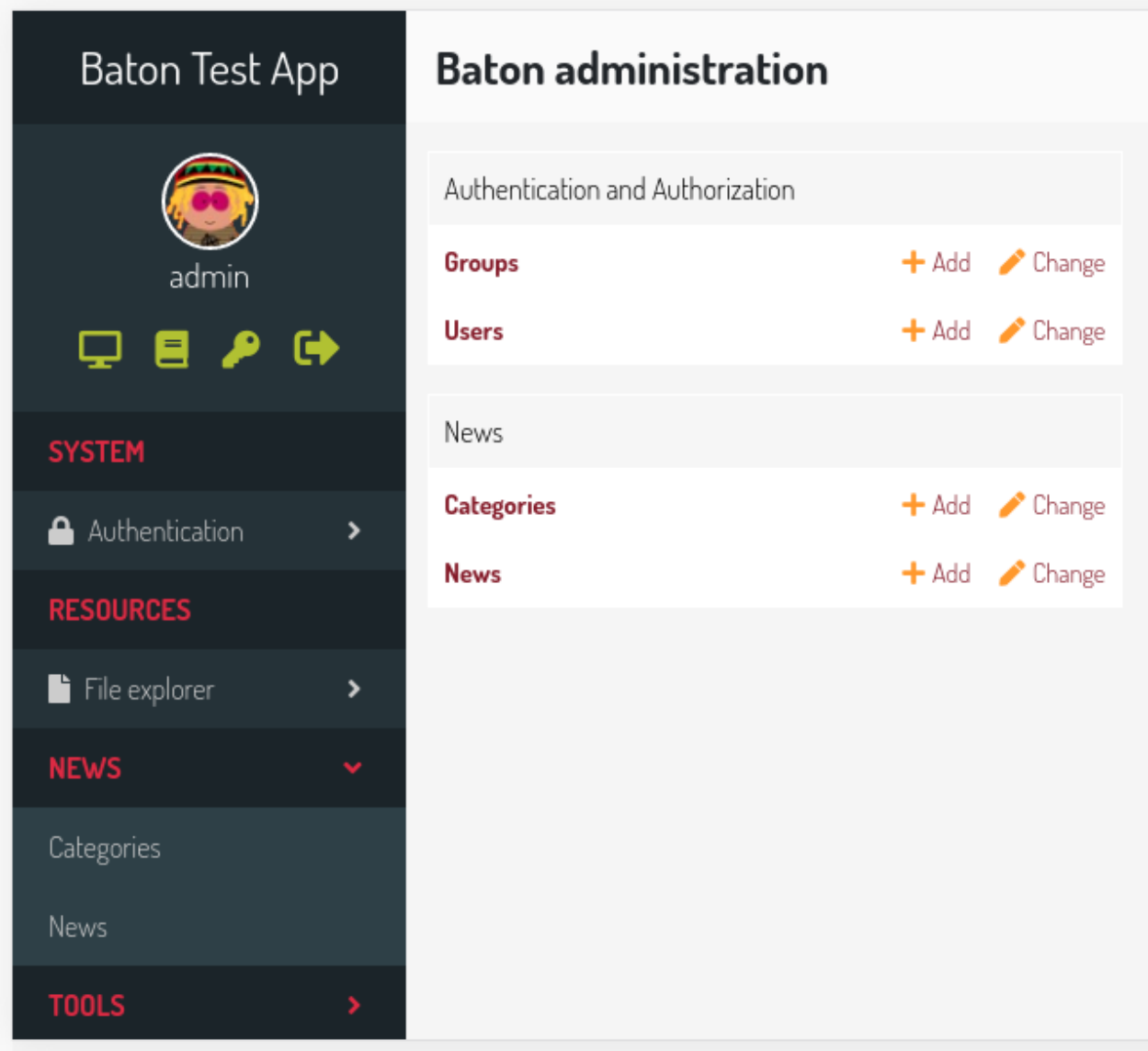
Default: None

2.2.21 Force theme

You can force the light or dark theme, and the theme toggle disappears from the user area.

Default: None

2.2.22 Menu



The sidebar menu is rendered through javascript.

If you don't define a custom menu, the default menu is rendered, which includes all the apps and models registered in the admin that the user can view.

When defining a custom menu you can use 4 different kinds of voices:

- title
- app

- model
- free

Title and free voices can have children. Children follow these rules:

- children children are ignored (do not place an app voice as child)

Voices with children can specify a `default_open` option, used to expand the submenu by default.

Title

Like the voices *MAIN* and *CONTENTS* in the above image, it represents a menu section. You should set a `label` and optionally an `apps` or `perms` key, used for visualization purposes.

If the title voice should act as a section title for a group of apps, you'd want to specify these apps, because if the user can't operate over them, then the voice is not shown. At the same time you can define some perms (OR condition), something like:

```
{ 'type': 'title', 'label': 'main', 'perms': ('auth.add_user', ) },
```

or

```
{ 'type': 'title', 'label': 'main', 'apps': ('auth', ) },
```

It accepts children voices, though you can specify the `default_open` key.

App

In order to add an application with all its models to the menu, you need an *app* menu voice.

You must specify the `type` and `name` keys, optionally an `icon` key (you can use FontAwesome classes which are included by default), a `default_open` key and a `models` key.

Important: If you don't define the models key then the default app models are listed under your app, otherwise only the specified models are listed (in the order you provide).

The `models` key must be a tuple, where every item represents a model in the form of a dictionary with keys `label` and `name`

```
{
    'type': 'app',
    'name': 'auth',
    'label': 'Authentication',
    'icon': 'fa fa-lock',
    'models': (
        {
            'name': 'user',
            'label': 'Users'
        },
        {
            'name': 'group',
            'label': 'Groups'
        },
    )
},
```

Important: App name should be lowercase.

Model

If you want to add only a link to the admin page of a single model, you can use this voice. For example, the *flatpages* app has only one model *Flatpage*, so I think it may be better to avoid a double selection.

In this case you must specify the `type`, `name` and `app` keys, optionally an `icon` key (you can use FontAwesome classes which are included by default). An example:

```
{ 'type': 'model', 'label': 'Pages', 'name': 'flatpage', 'app': 'flatpages', 'icon':
↪'fa fa-file-text-o' },
```

Important: Model name should be lowercase.

Free

If you want to link an external site, a documentation page, an add element page and in general every custom resource, you may use this voice.

In such case you must define an `url` and if you want some visibility permissions (OR clause)

```
{ 'type': 'free', 'label': 'Docs', 'url': 'http://www.mydocssite.com' },
```

or

```
{ 'type': 'free', 'label': 'Add page', 'url': '/admin/flatpages/flatpage/add/', 'perms
↪': ('flatpages.add_flatpage', ) },
```

It accepts children voices

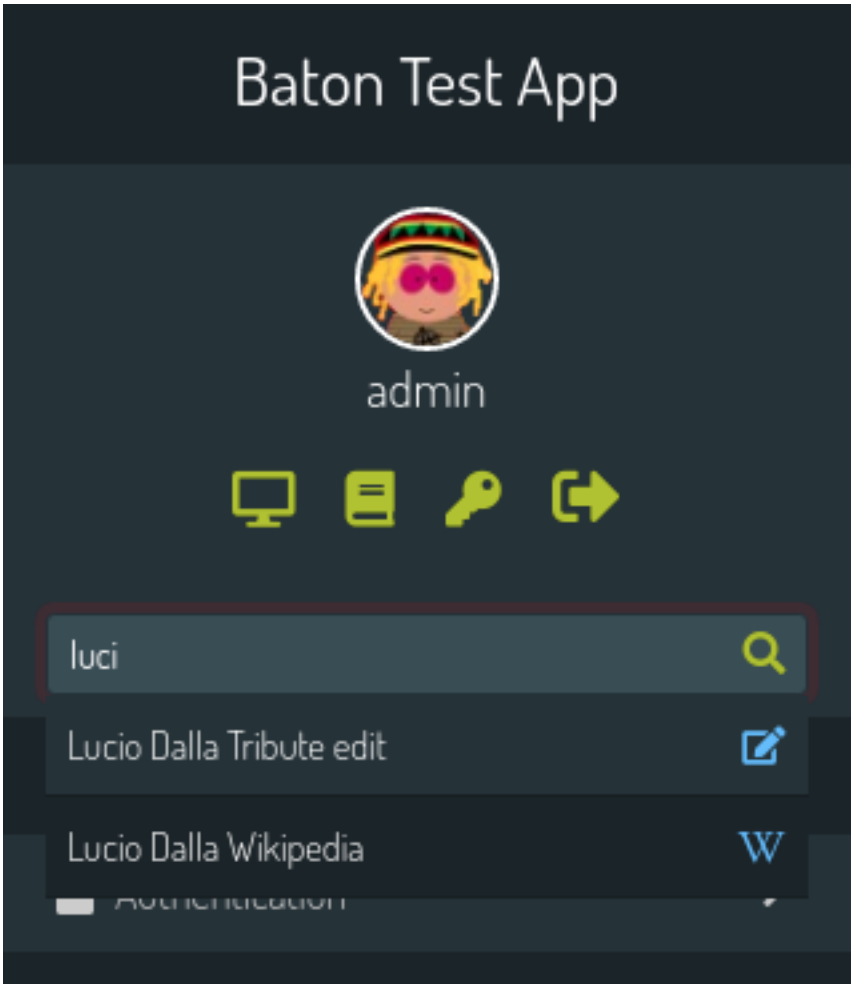
```
{ 'type': 'free', 'label': 'My parent voice', 'children': [
    { 'type': 'free', 'label': 'Docs', 'url': 'http://www.mydocssite.com' },
    { 'type': 'free', 'label': 'Photos', 'url': 'http://www.myphotossite.com' },
] },
```

Since free voices can have children you can specify the `default_open` key.

Free voices also accept a _re_ property, which specifies a regular expression used to decide whether to highlight the voice or not (the regular expression is evaluated against the document location pathname):

```
{
        'type': 'free',
    'label': 'Categories',
    'url': '/admin/news/category/',
    're': '^/admin/news/category/(\d*)?'
}
```

2.2.23 Search Field



With this functionality, you can configure a sidebar input search field with autocomplete functionality that can let you surf easily and quickly to any page you desire.

```
'SEARCH_FIELD': {
    'label': 'Label shown as placeholder',
    'url': '/api/path/',
},
```

The autocomplete field will call a custom api at every keyup event. Such api receives the `text` param in the querystring and should return a json response including the search results in the form:

```
{
    length: 2,
    data: [
        { label: 'My result #1', icon: 'fa fa-edit', url: '/admin/myapp/mymodel/1/
↪change' },
        // ...
    ]
}
```

You should provide the results length and the data as an array of objects which must contain the `label` and `url` keys. The `icon` key is optional and is treated as css class given to an `i` element.

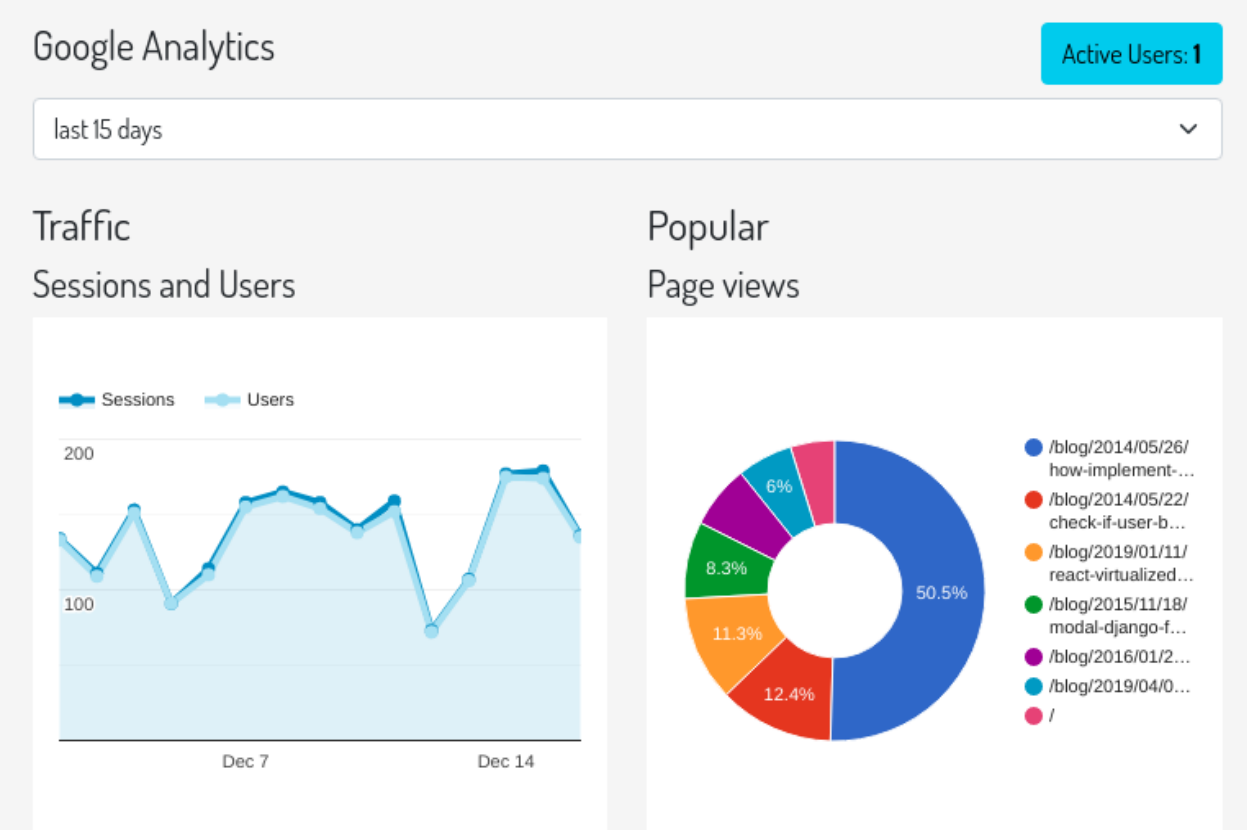
Let's see an example:

```
@staff_member_required
def admin_search(request):
    text = request.GET.get('text', None)
    res = []
    news = News.objects.all()
    if text:
        news = news.filter(title__icontains=text)
    for n in news:
        res.append({
            'label': str(n) + ' edit',
            'url': '/admin/news/news/%d/change' % n.id,
            'icon': 'fa fa-edit',
        })
    if text.lower() in 'Lucio Dalla Wikipedia'.lower():
        res.append({
            'label': 'Lucio Dalla Wikipedia',
            'url': 'https://www.google.com',
            'icon': 'fab fa-wikipedia-w'
        })
    return JsonResponse({
        'length': len(res),
        'data': res
    })
```

You can move between the results using the keyboard up and down arrows, and you can browse to the voice url pressing Enter.

2.2.24 Analytics

Note: In order to use the Google Analytics index, install baton along the optional dependencies with `pip install django-baton[analytics]`



Baton provides an index view which displays google analytics statistics charts for the last 15 days, 1 month, 3 month and 1 year.

In order to activate it you need to create a service account and link it to your google analytics view, then you must define the keys:

- `CREDENTIALS`: path to the credentials json file
- `VIEW_ID`: id of the analytics view which serves the data

You can add contents before and after the analytics dashboard by extending the `baton/analytics.html` template and filling the `baton_before_analytics` and `baton_after_analytics` blocks.

How to generate a credentials json file

Follow the steps in the Google Identity Platform documentation to create a service account from the Google Developer Console.

Once the service account is created, you can click the Generate New JSON Key button to create and download the key and add it to your project.

Add the service account as a user in Google Analytics. The service account you created in the previous step has an email address that you can add to any of the Google Analytics views you'd like to request data from. It's generally best to only grant the service account read-only access.

2.3 Page Detection

Baton triggers some of its functionalities basing upon the current page. For example, it will trigger the tab functionality only when the current page is an add form or change form page.

Baton understands which page is currently displayed performing some basic regular expressions against the location pathname. There may be cases in which you'd like to serve such contents at different and custom urls, in such cases you need a way to tell Baton which kind of page is tied to that url.

For this reason you can inject your custom hook, a javascript function which should return the page type and that receives as first argument the Baton's default function to use as fallback, i.e.

```
<!-- admin/base_site.html -->
<script>
    (function () {
        Baton.detectPageHook = fn => /newschange/.test(location.pathname) ? 'change_
↪form' : fn()
    })()
</script>
<script src="{% static 'baton/js_snippets/init_baton.js' %}"></script>
```

In this case we tell Baton that when the location pathname includes the string `newschange`, then the page should be considered a `change_form`, otherwise we let Baton guess the page type.

So, in order to hook into the Baton page detection system, just define a `Baton.detectPageHook` function which receives the default function as first argument and should return the page type.

The available page types are the following: `dashboard`, `admindocs`, `login`, `logout`, `passowrd_change`, `password_change_success`, `add_form`, `change_form`, `changelist`, `filer`, `default`.

2.4 Signals

Baton provides a dispatcher that can be used to register function that will be called when some events occurr. At this moment Baton emits four types of events:

- `onNavbarReady`: dispatched when the navbar is fully rendered
- `onMenuReady`: dispatched when the menu is fully rendered (probably the last event fired, since the menu contents are retrieves async)
- `onTabsReady`: dispatched when the changeform tabs are fully
- `onMenuError`: dispatched if the request sent to retrieve menu contents fails
- `onReady`: dispatched when Baton js has finished its sync job

In order to use them just override the baton *admin/base_site.html* template and register your listeners **before** calling *Baton.init*, i.e.

```
<!-- ... -->
<script>
    (function ($, undefined) {
        // init listeners
        Baton.Dispatcher.register('onReady', function () { console.log('BATON IS READY
↪') })
        Baton.Dispatcher.register('onMenuReady', function () { console.log('BATON␣
↪MENU IS READY') })
        Baton.Dispatcher.register('onNavbarReady', function () { console.log('BATON␣
↪NAVBAR IS READY') })
```

(continues on next page)

(continued from previous page)

```
        // end listeners
    })(jQuery, undefined)
</script>
<script src="{% static 'baton/js_snippets/init_baton.js' %}"></script>
<!-- ... -->
```

2.5 Js Utilities

Baton comes with a number of exported js modules you can use to enhance your admin application.

2.5.1 Dispatcher

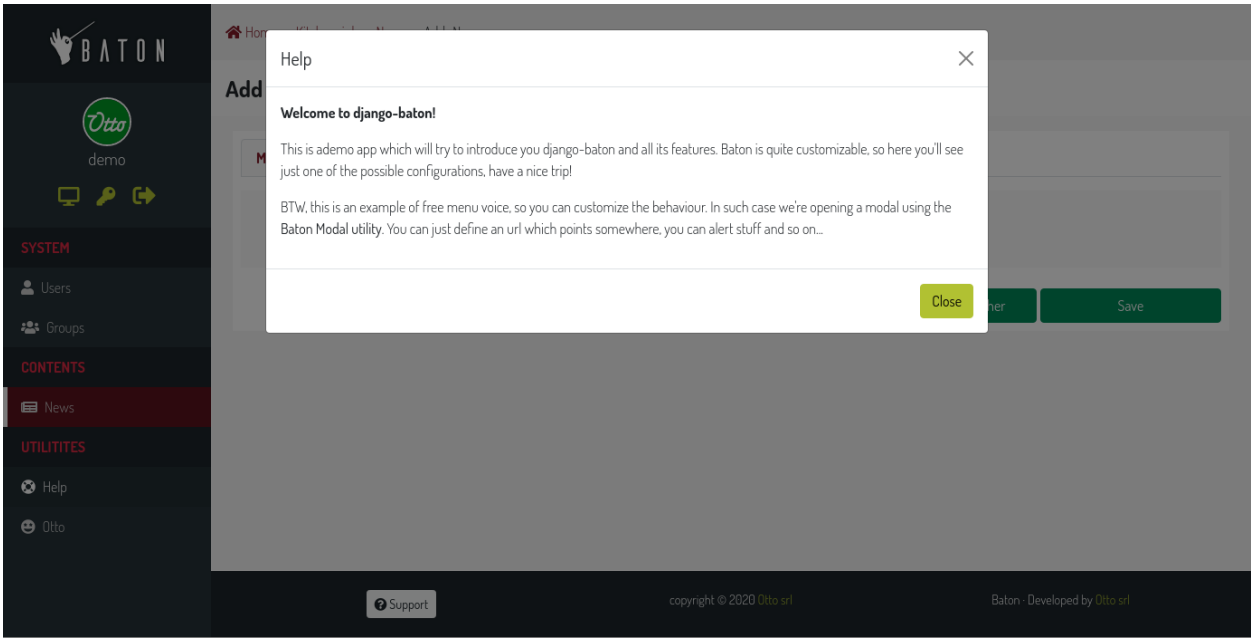
Baton Dispatcher singleton module lets you subscribe to event and dispatch them, making use of the Mediator pattern.

Example:

```
// register a callback tied to the event
Baton.Dispatcher.register('myAppLoaded', function (evtName, s) { console.log('COOL '␣
↪+ s) })

// emit the event
Baton.Dispatcher.emit('myAppLoaded', 'STUFF!')
```

2.5.2 Modal



Baton Modal class lets you insert some content on a bootstrap modal without dealing with all the markup.

Usage:

```
// modal configuration:
//
// let config = {
//     title: 'My modal title',
//     subtitle: 'My subtitle', // optional
//     content: '<p>my html content</p>', // alternative to url
//     url: '/my/url', // url used to perform an ajax request, the response is put␣
↪inside the modal body. Alternative to content.
//     hideFooter: false, // optional
//     showBackBtn: false, // show a back button near the close icon, optional
//     backBtnCb: function () {}, // back button click callback (useful to have a␣
↪multi step modal), optional
//     actionBtnLabel: 'save', // action button label, default 'save', optional
//     actionBtnCb: null, // action button callback, optional
//     onUrlLoaded: function () {}, // callback called when the ajax request has␣
↪completed, optional
//     size: 'lg', // modal size: sm, md, lg, xl, optional
//     onClose: function () {} // callback called when the modal is closed, optional
// }
//
// constructs a new modal instance
// let myModal = new Baton.Modal(config)

let myModal = new Baton.Modal({
    title: 'My modal title',
    content: '<p>my html content</p>',
    size: 'lg'
})

myModal.open();
myModal.close();

myModal.update({
    title: 'Step 2',
    content: '<p>cool</p>'
})
myModal.toggle();
```

2.6 Js Translations

There are some circustamces in which Baton will print to screen some js message. Baton detects the user locale and will localize such messages, but it comes with just `en` and `it` translations provided.

Important: Baton retrieves the current user locale from the `lang` attribute of the `html` tag.

However you can provide or add your own translations by attaching an object to the *Baton* namespace:

```
// these are the default translations, you can just edit the one you need, or add␣
↪some locales. Baton engione will always
// pick up your custom translation first, if it find them.
// you can define thi object before Baton.init in the base_site template
Baton.translations = {
```

(continues on next page)

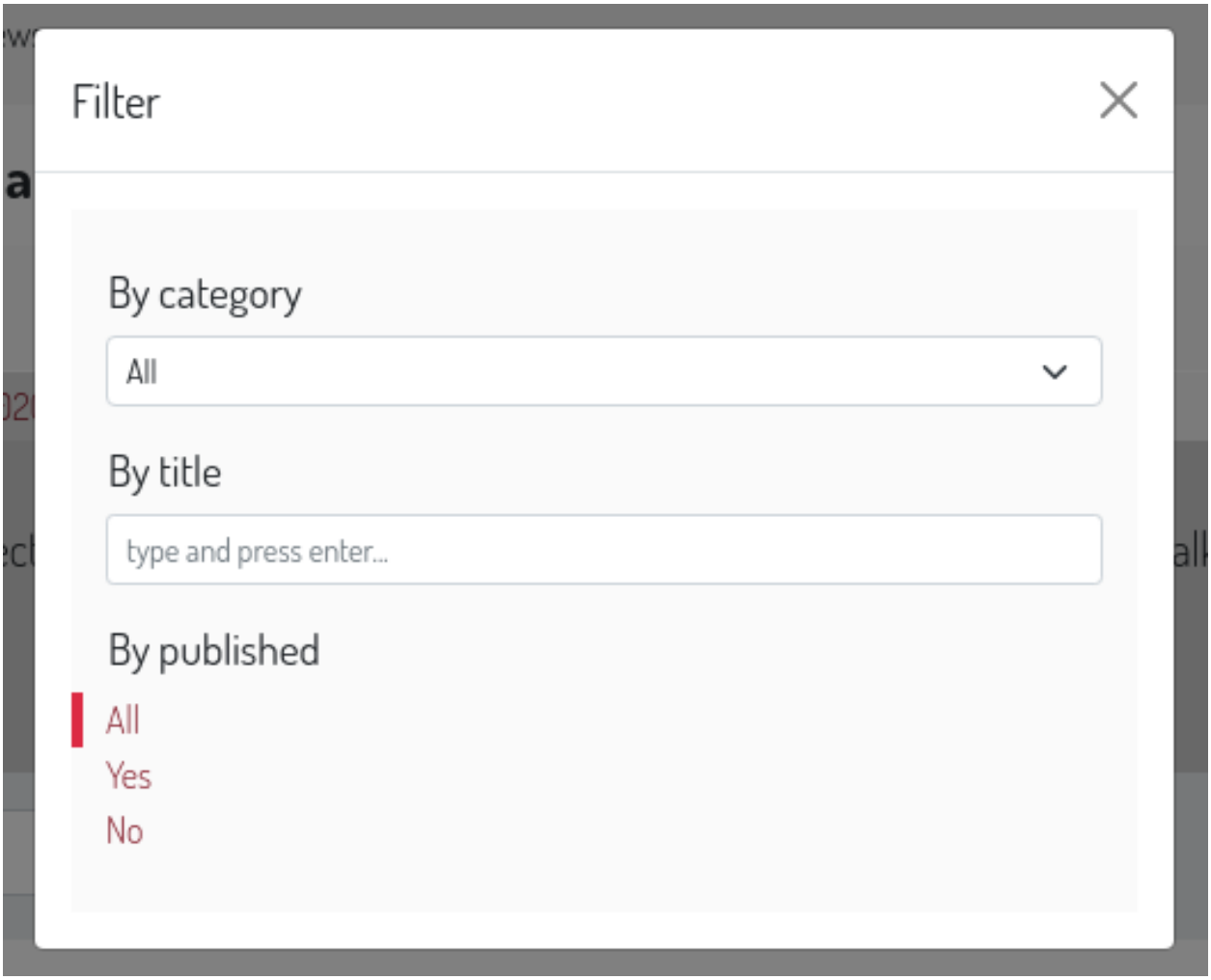
(continued from previous page)

```
  unsavedChangesAlert: 'You have some unsaved changes.',
  uploading: 'Uploading...',
  filter: 'Filter',
  close: 'Close',
  save: 'Save',
  search: 'Search',
  cannotCopyToClipboardMessage: 'Cannot copy to clipboard, please do it manually:␣
↪Ctrl+C, Enter',
  retrieveDataError: 'There was an error retrieving the data',
  lightTheme: 'Light theme',
  darkTheme: 'Dark theme',
}
```

Important: Just use the `trans` templatetag to deal with multilanguage web applications

If Baton can't find the translations for the user locale, it will default to `en`. Keep in mind that Baton will use `en` translations for all `en-xx` locales, but of course you can specify your custom translations!

2.7 List Filters



2.7.1 Input Text Filters

Idea taken from this medium article.

Baton defines a custom InputFilter class that you can use to create text input filters and use them as any other `list_filters`, for example

```
# your app admin

from baton.admin import InputFilter

class IdFilter(InputFilter):
    parameter_name = 'id'
    title = 'id'

    def queryset(self, request, queryset):
        if self.value() is not None:
            search_term = self.value()
```

(continues on next page)

(continued from previous page)

```
            return queryset.filter(
                id=search_term
            )


class MyModelAdmin(admin.ModelAdmin):
    list_filters = (
        'my_field',
        IdFilter,
        'my_other_field',
    )
```

Just define in the `queryset` method the logic used to retrieve the results.

2.7.2 Dropdown Filters

Taken from the github app django-admin-list-filter-dropdown.

Baton provides a dropdown form of the following list filters:

Django admin filter name	Baton name
SimpleListFilter	SimpleDropdownFilter
AllValuesFieldListFilter	DropdownFilter
ChoicesFieldListFilter	ChoicesDropdownFilter
RelatedFieldListFilter	RelatedDropdownFilter
RelatedOnlyFieldListFilter	RelatedOnlyDropdownFilter

The dropdown is visible only if the filter contains at least three options, otherwise the default template is used.

Usage:

```
from baton.admin import DropdownFilter, RelatedDropdownFilter, ChoicesDropdownFilter

class MyModelAdmin(admin.ModelAdmin):

    list_filter = (
        # for ordinary fields
        ('a_charfield', DropdownFilter),
        # for choice fields
        ('a_choicefield', ChoiceDropdownFilter),
        # for related fields
        ('a_foreignkey_field', RelatedDropdownFilter),
    )
```

2.7.3 Multiple Choice Filters

Baton defines a custom MultipleChoiceListFilter class that you can use to filter on multiple options, for example:

```
# your app admin

from baton.admin import MultipleChoiceListFilter

class StatusListFilter(MultipleChoiceListFilter):
```

(continues on next page)

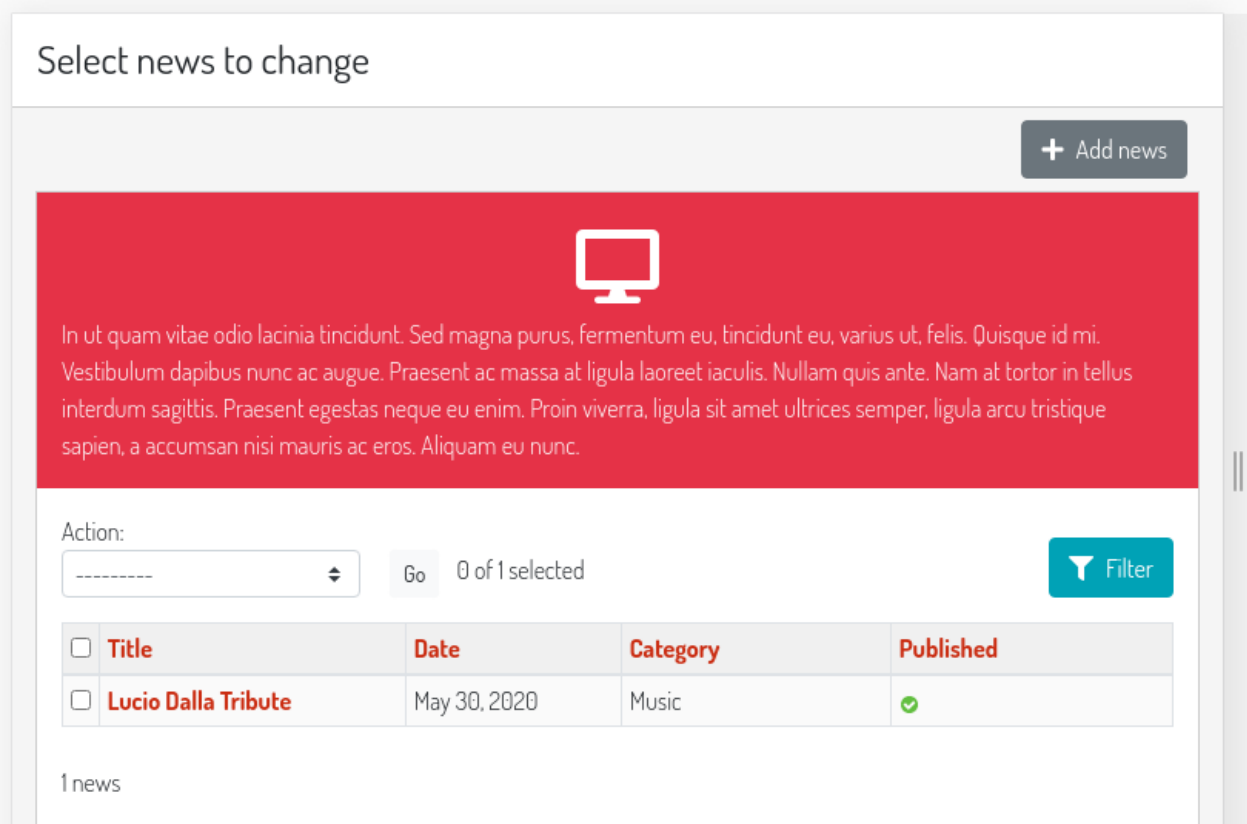
(continued from previous page)

```
    title = 'Status'
    parameter_name = 'status__in'

    def lookups(self, request, model_admin):
        return News.Status.choices


class MyModelAdmin(admin.ModelAdmin):
    list_filters = (
        'my_field',
        StatusListFilter,
        'my_other_field',
    )
```

2.8 Changelist includes



Important: In order for this feature to work, the user browser must support html template tags.

Baton lets you include templates directly inside the change list page, in any position you desire. It's as simple as specifying the template path and the position of the template:

```
@admin.register(News)
class NewsAdmin(admin.ModelAdmin):
```

(continues on next page)

(continued from previous page)

```
    #...
    baton_cl_includes = [
        ('news/admin_include_top.html', 'top', ),
        ('news/admin_include_below.html', 'below', )
    ]
```

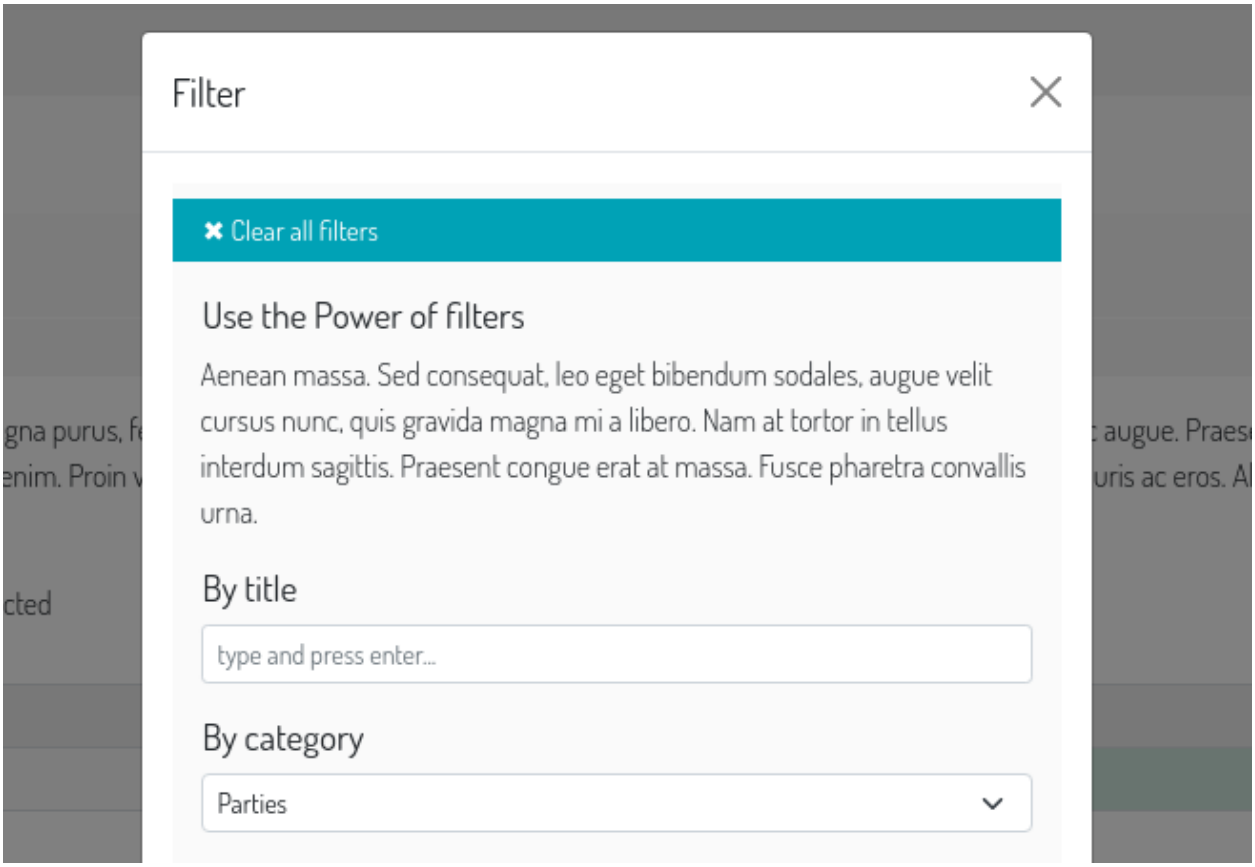
In this case, Baton will place the content of the `admin_include_top.html` template at the top of the changelist section (above the search field), and the content of the `admin_include_below.html` below the changelist form.

You can specify the following positions:

Position	Description
top	the template is placed inside the changelist form, at the top
bottom	the template is placed inside the changelist form, at the bottom
above	the template is placed above the changelist form
below	the template is placed below the changelist form

And, of course, you can access the all the changelist view context variables inside your template.

2.9 Changelist filters includes



Important: In order for this feature to work, the user browser must support html template tags.

Baton lets you include templates directly inside the change list filter container, at the top or the bottom. It's as simple as specifying the template path and the position of the template:

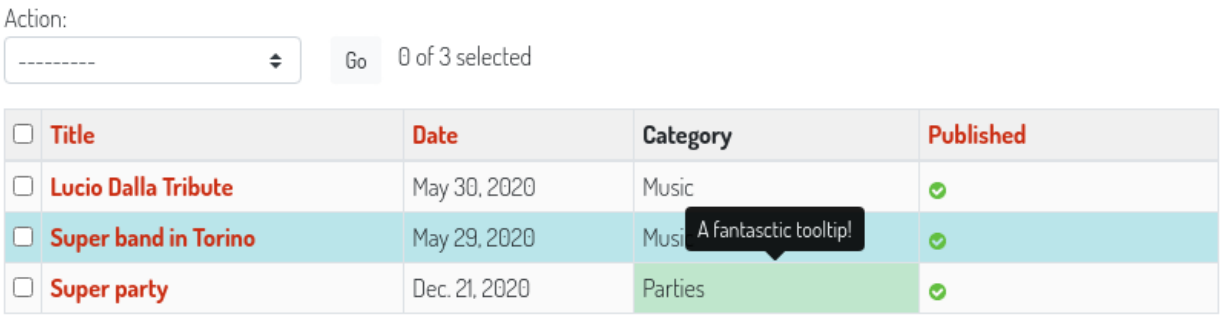
```
@admin.register(News)
class NewsAdmin(admin.ModelAdmin):
    #...
    baton_cl_filters_includes = [
        ('news/admin_filters_include_top.html', 'top', ),
        ('news/admin_filters_include_below.html', 'bottom', )
    ]
```

You can specify the following positions:

Position	Description
top	the template is placed inside the changelist filters container, at the top
bottom	the template is placed inside the changelist filters container, at the bottom

And, of course, you can access the all the changelist view context variables inside your template.

2.10 Changelist Row Attributes



Important: In order for this feature to work, the user browser must support html template tags.

With Baton you can add every kind of html attribute (including css classes) to any element in the changelist table (cell, rows, ...)

It's a bit tricky, let's see how:

1. Add a `baton_cl_rows_attributes` function to your `ModelAdmin` class, which takes `request` and `cl` (changelist view) as parameters.
2. Return a json dictionary where the keys are used to match an element and the values specifies the attributes and other rules to select the element.

Better to see an example:

```
class NewsModelAdmin(admin.ModelAdmin):
    # ...

    def get_category(self, instance):
        return mark_safe('<span class="span-category-id-%d">%s</span>' % (instance.id,
↪ str(instance.category)))
```

(continues on next page)

(continued from previous page)

```
    get_category.short_description = 'category'

    def baton_cl_rows_attributes(self, request, cl):
        data = {}
        for news in cl.queryset.filter(category__id=2):
            data[news.id] = {
                'class': 'table-info',
            }
        data[news.id] = {
            'class': 'table-success',
            'data-lol': 'lol',
            'title': 'A fantasctic tooltip!',
            'selector': '.span-category-id-%d' % 1,
            'getParent': 'td',
        }
        return json.dumps(data)
```

In such case we're returning a dictionary with possibly many keys (each key is an id of a news instance).

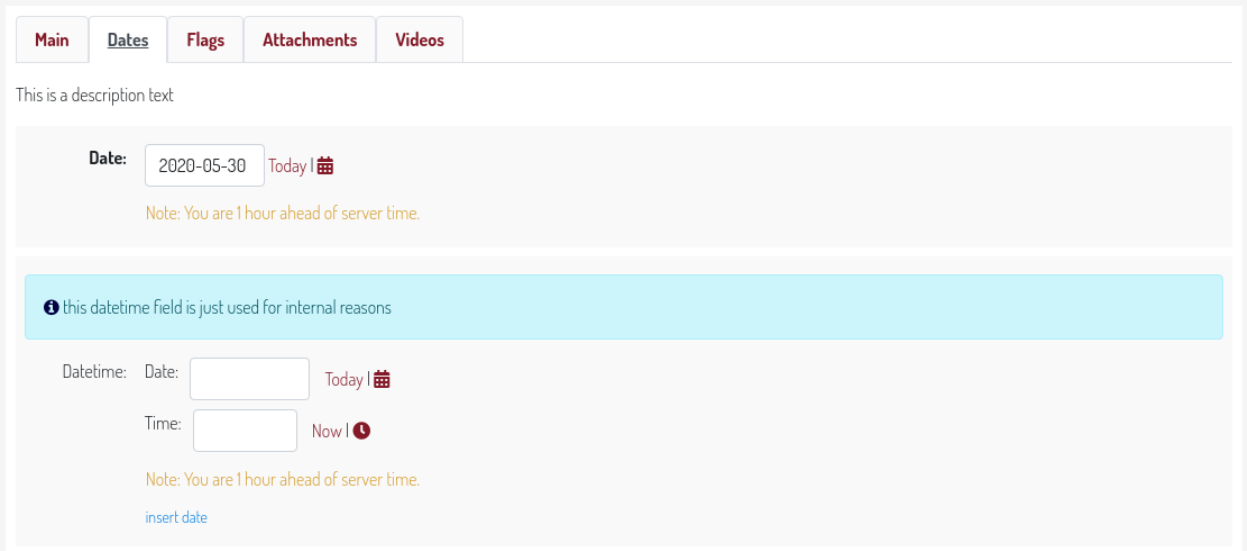
The first kind of dictionary elements will add a `table-info` class to the `tr` (rows) containing the news respecting the rule `category__id=2`

The second kind of element instead uses some more options to customize the element selection: you can specify a css selector, and you can specify if Baton should then take one of its parents, and in such case you can give a parent selector also. In the example provided Baton will add the class `table-success`, `data-attribute` and the `title` attribute to the cell which contains the element `.span-category-id-1`.

So these are the rules:

- the default `selector` is `#result_list tr input[name=_selected_action][value=' + key + ']`, meaning that it can work only if the model is editable (you have the checkox inputs for selecting a row), and selects the row of the instance identified by `key`. If you use a custom selector the dictionary `key` is unuseful.
- the default `getParent` is `tr`. You can change it at you will, or set it to *False*, in such case the element to which apply the given attributes will be the one specified by `selector`.
- Every other key different from `selector` and `getParent` will be considered an attribute and added to the element.

2.11 Form tabs



Baton provides an easy way to define form tabs in yor change form templates. Everything is done through javascript and you only need to add some classes you your `ModelAdmin` fieldsets

```
from django.contrib import admin
from .models import Item, Attribute, Feature

class AttributeInline(admin.StackedInline):
    model = Attribute
    extra = 1

class FeatureInline(admin.StackedInline):
    model = Feature
    extra = 1

class ItemAdmin(admin.ModelAdmin):
    list_display = ('label', 'description', 'main_feature', )
    inlines = [AttributeInline, FeatureInline, ]

    fieldsets = (
        ('Main', {
            'fields': ('label', ),
            'classes': ('baton-tabs-init', 'baton-tab-inline-attribute', 'baton-tab-
↪fs-content', 'baton-tab-group-fs-tech--inline-feature', ),
            'description': 'This is a description text'

        }),
        ('Content', {
            'fields': ('text', ),
            'classes': ('tab-fs-content', ),
            'description': 'This is another description text'

        }),
        ('Tech', {
            'fields': ('main_feature', ),
            'classes': ('tab-fs-tech', ),
```

(continues on next page)

(continued from previous page)

```
            'description': 'This is another description text'

        }),
    )
```

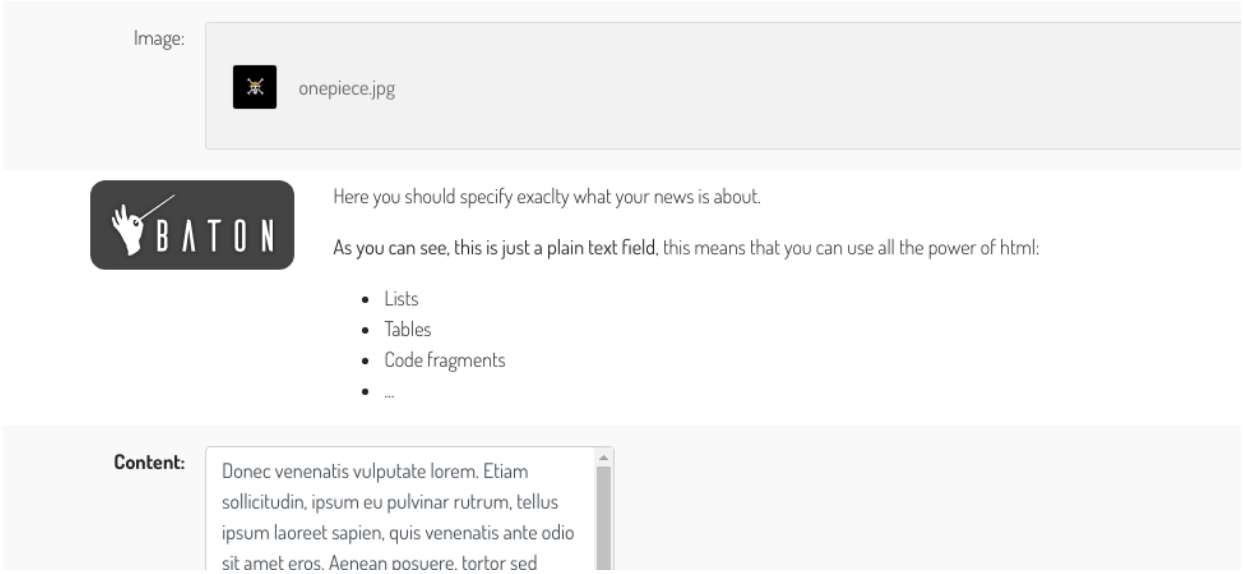
2.11.1 Rules

- Inline classes remain the same, no action needed
- In the first fieldset define a `baton-tabs-init` class which enables tabs
- On the first fieldset, you can add an `order-[NUMBER]` class, which will be used to determined in which position to place the first fieldset. The order starts from 0, and if omitted, the first fieldset has order 0. If you assign for example the class `order-2` to the first fieldset, then the first fieldset will be the third tab, while all other tabs will respect the order of declaration.
- For every `InLine` you want to put in a separate tab, add a class `baton-tab-inline-MODELNAME` or `baton-tab-inline-RELATEDNAME` if you've specified a related_name to the foreign key
- For every fieldset you want to put in a separate tab, add a class `baton-tab-fs-CUSTOMNAME`, and add a class `tab-fs-CUSTOMNAME` on the fieldset
- For every group you want to put in a separate tab, add a class `baton-tab-group-ITEMS`, where items can be inlines (`inline-RELATEDNAME`) and/or fieldsets (`fs-CUSTOMNAME`) separated by a double hypen `--`. Also add a class `tab-fs-CUSTOMNAME` on the fieldset items.
- Tabs order respects the defined classes order
- Fieldsets without a specified tab will be added to the main tab. If you want the fieldset to instead display outside of any tabs, add a class `tab-fs-none` to the fieldset. The fieldset will then always be visible regardless of the current tab.

Other features:

- when some field has an error, the first tab containing errors is opened automatically
- you can open a tab on page load just by adding an hash to the url, i.e. #inline-feature, #fs-content, #group-fs-tech–inline-feature

2.12 Form includes



Important: In order for this feature to work, the user browser must support html template tags.

Baton lets you include templates directly inside the change form page, in any position you desire. It's as simple as specifying the template path, the field name used as anchor and the position of the template:

```
@admin.register(News)
class NewsAdmin(admin.ModelAdmin):
    #...
    baton_form_includes = [
        ('news/admin_datetime_include.html', 'datetime', 'top', ),
        ('news/admin_content_include.html', 'content', 'above', )
    ]
```

In this case, Baton will place the content of the `admin_datetime_include.html` template at the top of the datetime field row, and the content of the `admin_content_include.html` above the content field row.

You can specify the following positions:

Position	Description
top	the template is placed inside the form row, at the top
bottom	the template is placed inside the form row, at the bottom
above	the template is placed above the form row
below	the template is placed below the form row
right	the template is placed inline at the field right side

And, of course, you can access the *{{ original }}* object variable inside your template.

It works seamlessly with the tab facility, if you include content related to a field inside one tab, then the content will be placed in the same tab.

2.13 Collapsable StackedInline entries

Baton lets you collapse single stacked inline entries, just add a *collapse-entry* class to the inline, with or without the entire collapse class

```
class VideosInline(admin.StackedInline):
    model = Video
    extra = 1
    classes = ('collapse-entry', )  # or ('collapse', 'collapse-entry', )
```

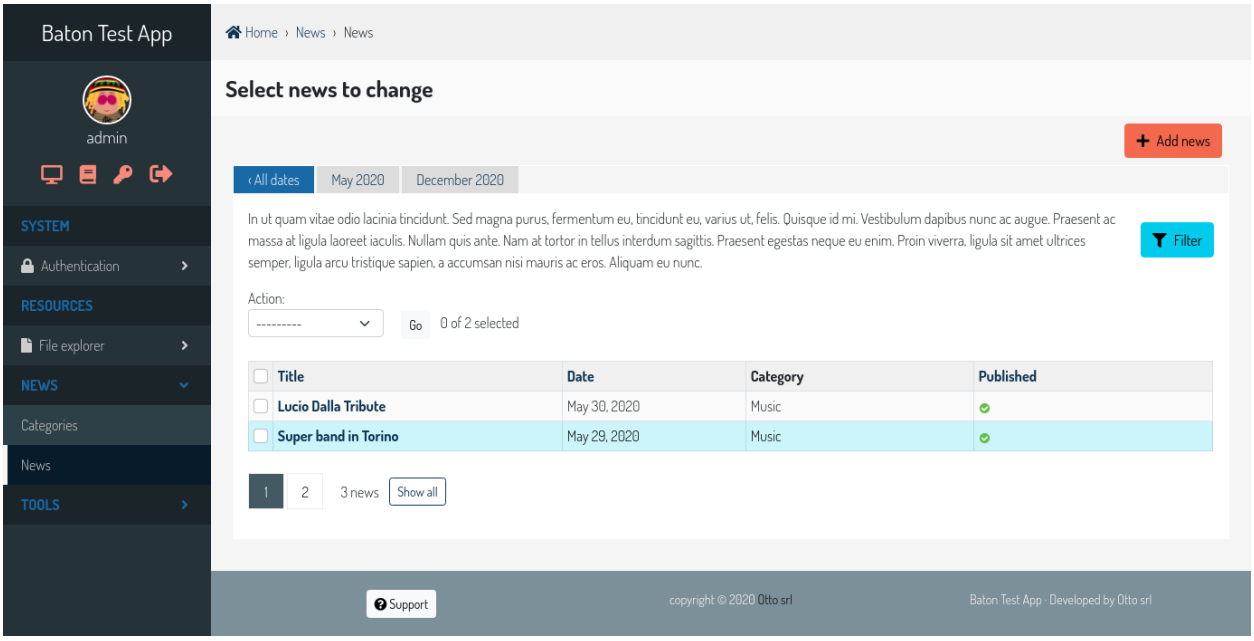
And if you want the first entry to be initially expanded, add also the *expand-first* class

```
class VideosInline(admin.StackedInline):
    model = Video
    extra = 1
    classes = ('collapse-entry', 'expand-first', )
```

CHAPTER 3

Advanced customization

3.1 Customization



It's easy to heavily customize the appeareance of baton. All the stuff is compiled from a modern js app which resides in *baton/static/baton/app*.

3.1.1 The Baton js app

The js app which **baton** provides is a modern js app, written using es2015 and stage-0 code features, which are then transpiled to a code browsers can understand using babel and webpack.

All css are written using sass on top of bootstrap 4.5.0, and transpiled with babel so that the final output is a single js file ready to be included in the html template.

The app entry point is index.js, where the only variable attached to the window object `Baton` is defined.

All the js modules used are inside the core directory.

3.1.2 Change the baton appearance

It's quite easy to change completely the appearance of baton, just overwrite the sass variables as you like and recompile the app. Then make sure to serve your recompiled app in place of the baton one.

Here comes what you have to do:

- place one of your django apps before *baton* in the *INSTALLED_APPS* settings, I'll call this app ROOTAPP
- clone the repository (or copy the *static/baton/app* dir from your virtualenv)

```
$ git clone https://github.com/otto-torino/django-baton.git
```

- install the app requirements

```
$ cd django-baton/baton/static/baton/app/
$ npm install
```

- edit the `src/styles/_variables.scss` file as you like
- recompile the app

```
$ npm run compile
```

- copy the generated bundle `dist/baton.min.js` in `ROOTAPP/static/baton/app/dist/`

You can also perform live development, in this case:

- place one of your django apps before *baton* in the *INSTALLED_APPS* settings, I'll call this app ROOTAPP
- create an admin base_site template `ROOTAPP/templates/admin/base_site.html` with the following content:

```
{% baton_config as conf %}
{{ conf | json_script:"baton-config" }}
<script charset="utf-8">
    (function () {
        // immediately set the theme mode to avoid flashes
        var systemTheme = window.matchMedia("(prefers-color-scheme: dark)");
        var theme = JSON.parse(document.getElementById('baton-config').
↪textContent).forceTheme || localStorage.getItem('baton-theme') || (systemTheme.
↪matches ? 'dark' : 'light');
        document.getElementsByTagName('html')[0].setAttribute('data-bs-theme',␣
↪theme);
    })()
</script>
<meta content="width=device-width, initial-scale=1.0" name="viewport" />
<link rel="stylesheet" href="{% static 'baton/css/theme.css' %}" />
<script src="{% static 'baton/app/dist/baton.min.js' %}"></script>
<!-- <script src="http://localhost:8080/static/baton/app/dist/baton.min.js"></
↪script> -->
<script src="{% static 'baton/js_snippets/init_baton.js' %}"></script>
```

- or you can edit directly the baton template and switch the comment of the two lines:

```
<!-- <script src="{% static 'baton/app/dist/baton.min.js' %}"></script> comment␣
↪the compiled src and uncomment the webpack served src -->
<script src="http://localhost:8080/static/baton/app/dist/baton.min.js"></script>
```

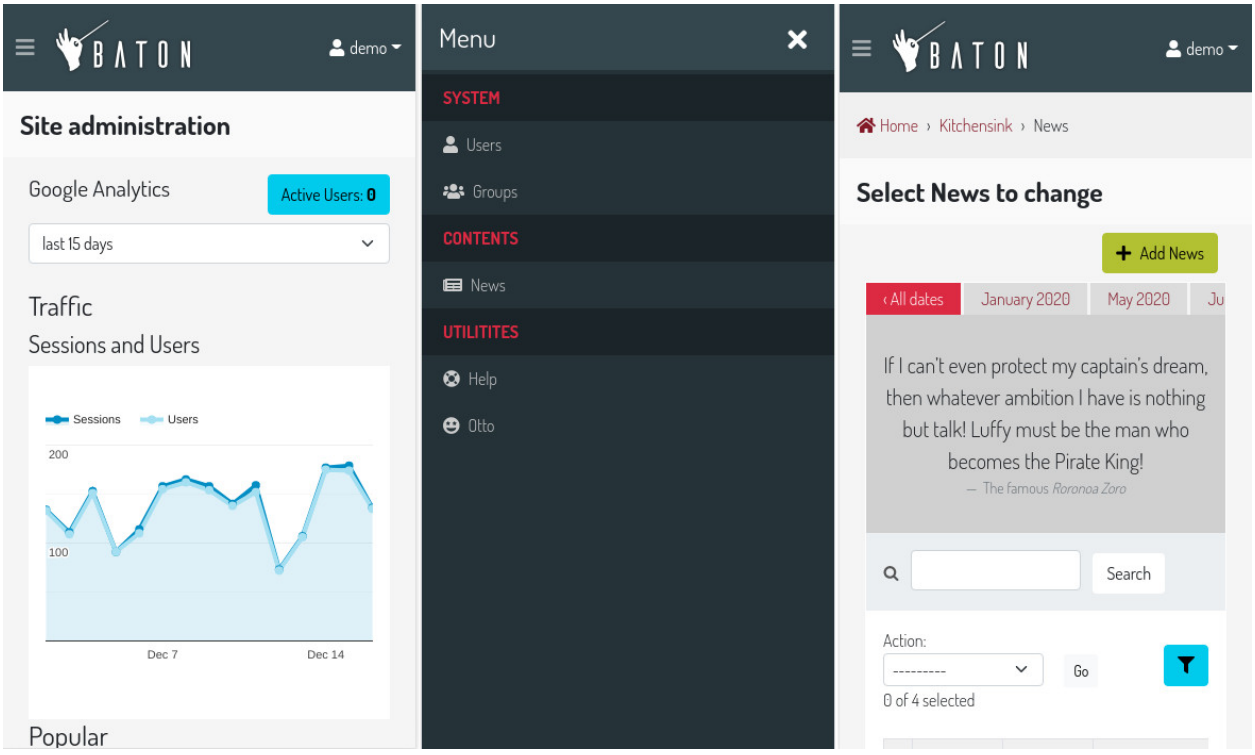
- start the webpack development server

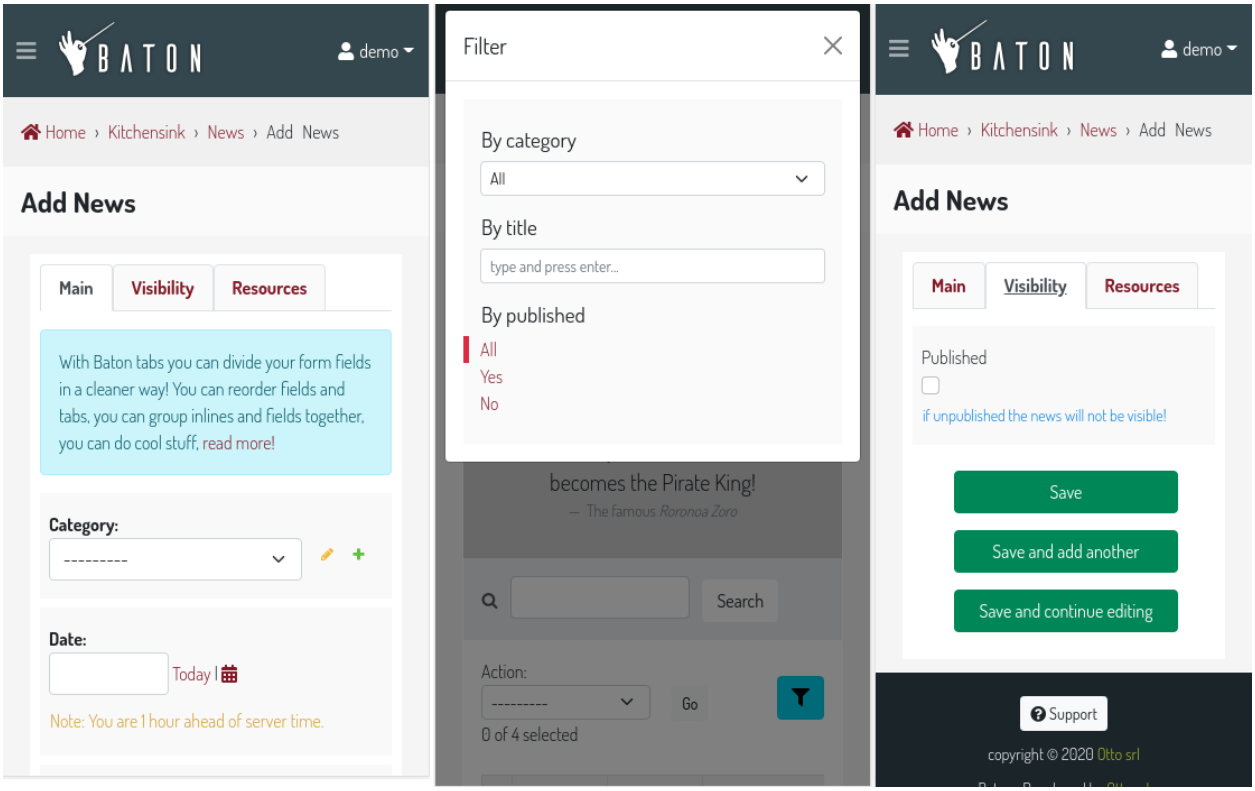
```
$ npm run dev
```

Now while you make your changes to the js app (css included), webpack will update the bundle automatically, so just refresh the page and you'll see your changes.

CHAPTER 4

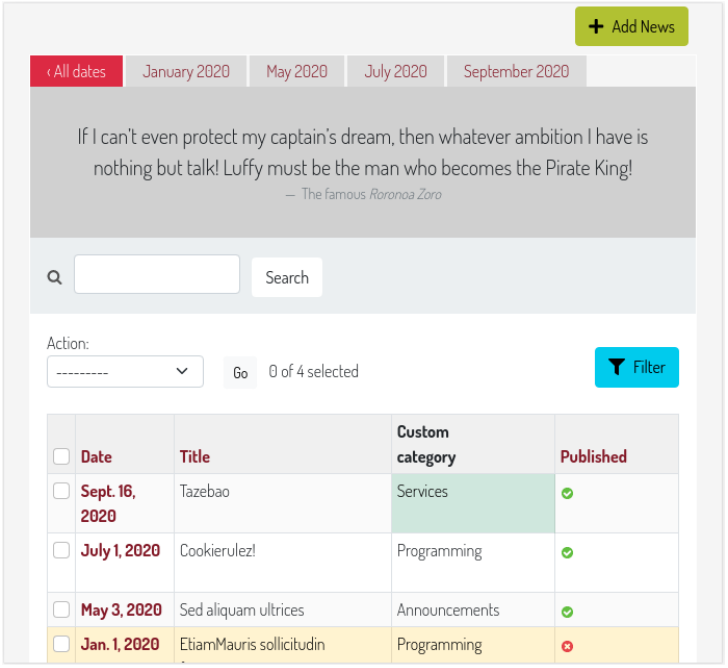
Screenshots



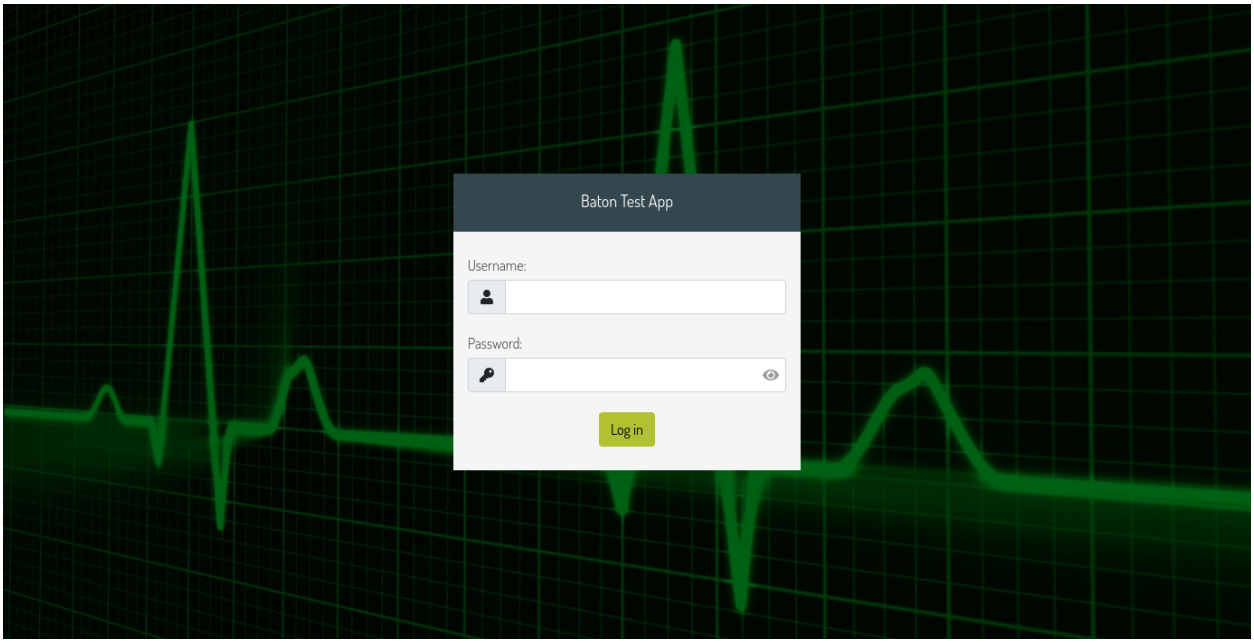

BATON
demo
Home › Kitchensink › News › Add News
Add News
Main
Visibility
Resources
With Baton tabs you can divide your form fields in a cleaner way! You can reorder fields and tabs, you can group inlines and fields together, you can do cool stuff, read more!
Category:

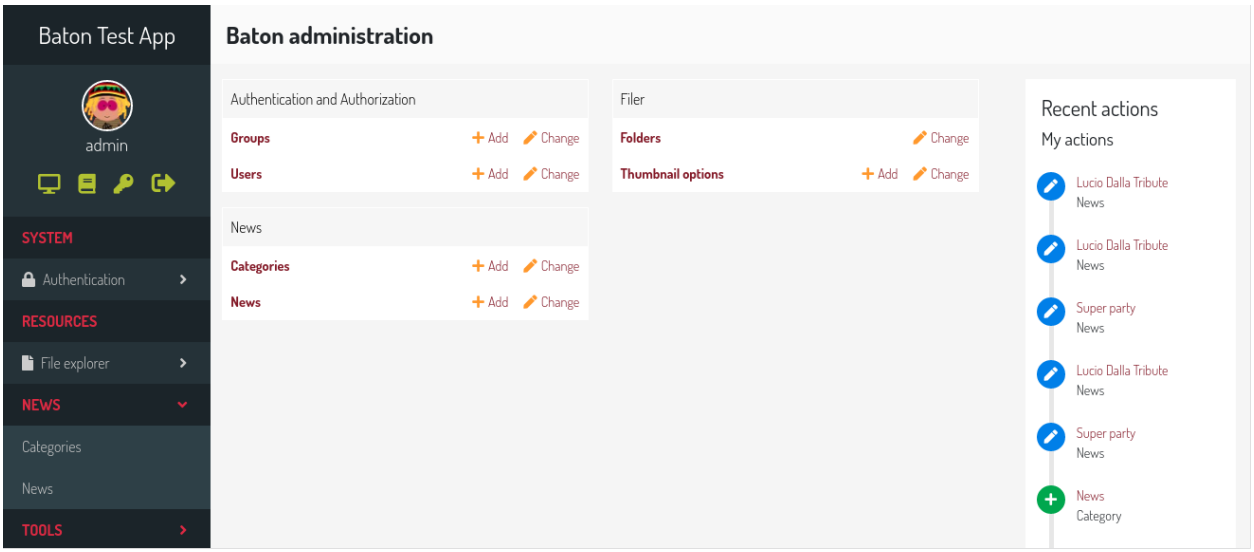
Date:
Today |
Note: You are 1 hour ahead of server time.
Filter
By category
All
By title
type and press enter...
By published
All
Yes
No
becomes the Pirate King!
— The famous Roronoa Zoro
Search
Action:

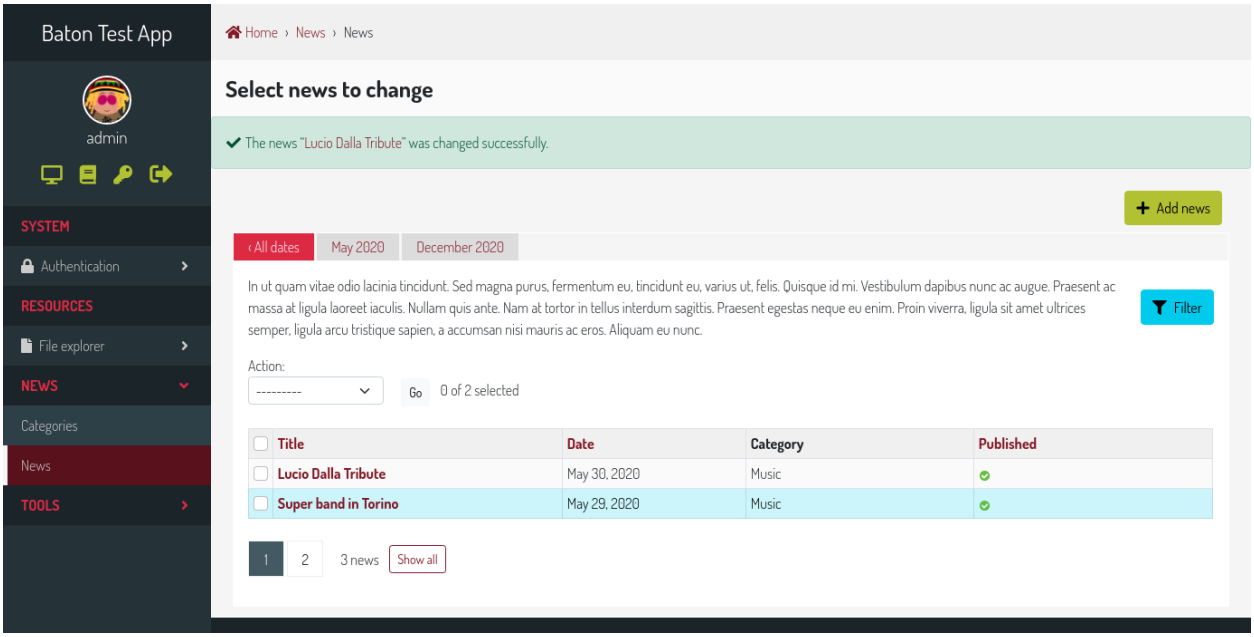
Go
0 of 4 selected
BATON
demo
Home › Kitchensink › News › Add News
Add News
Main
Visibility
Resources
Published
if unpublished the news will not be visible!
Save
Save and add another
Save and continue editing
Support
copyright © 2020 Otto srl


Add News
‹ All dates
January 2020
May 2020
July 2020
September 2020
If I can't even protect my captain's dream, then whatever ambition I have is nothing but talk! Luffy must be the man who becomes the Pirate King!
— The famous Roronoa Zoro
Search
Action:

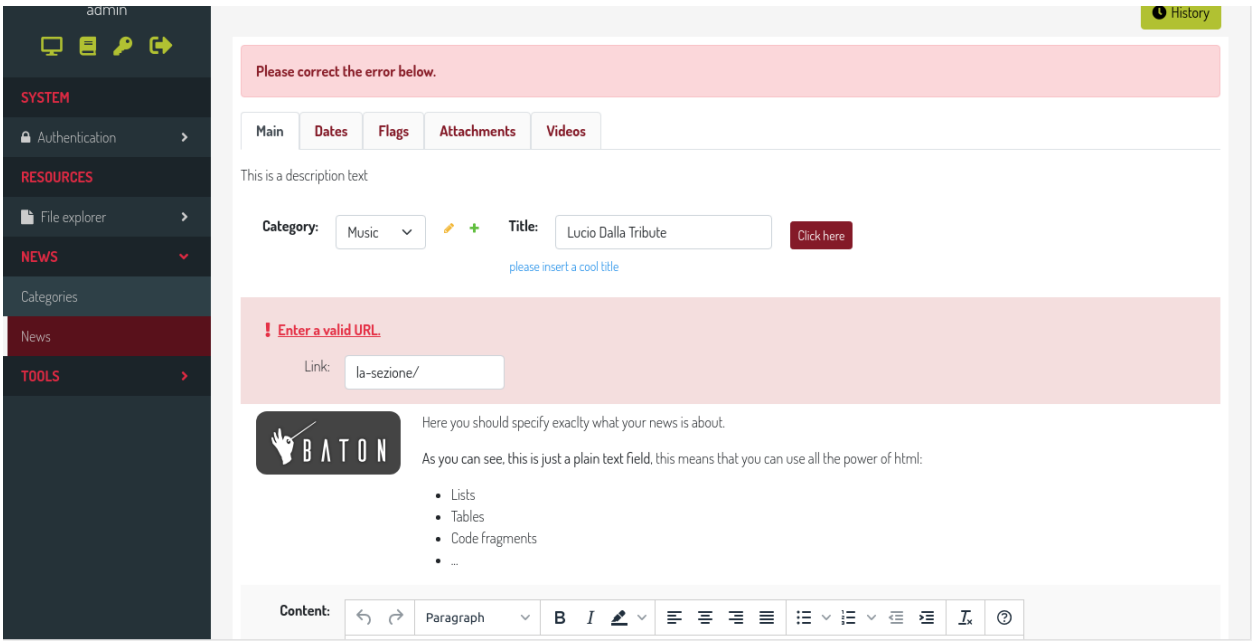
Go
0 of 4 selected
Filter
Date
Title
Custom category
Published
Sept. 16, 2020
Tazebao
Services
July 1, 2020
Cookierulez!
Programming
May 3, 2020
Sed aliquam ultrices
Announcements
Jan. 1, 2020
EtiamMauris sollicitudin
Programming

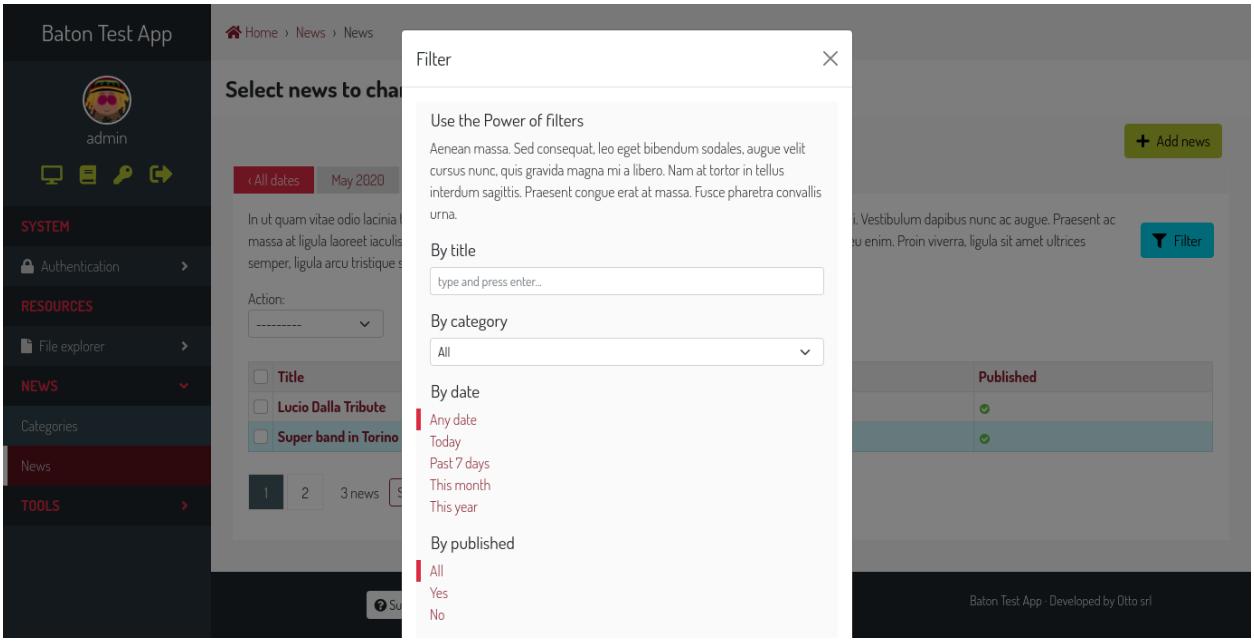

Baton Test App
Username:
Password:
Log in

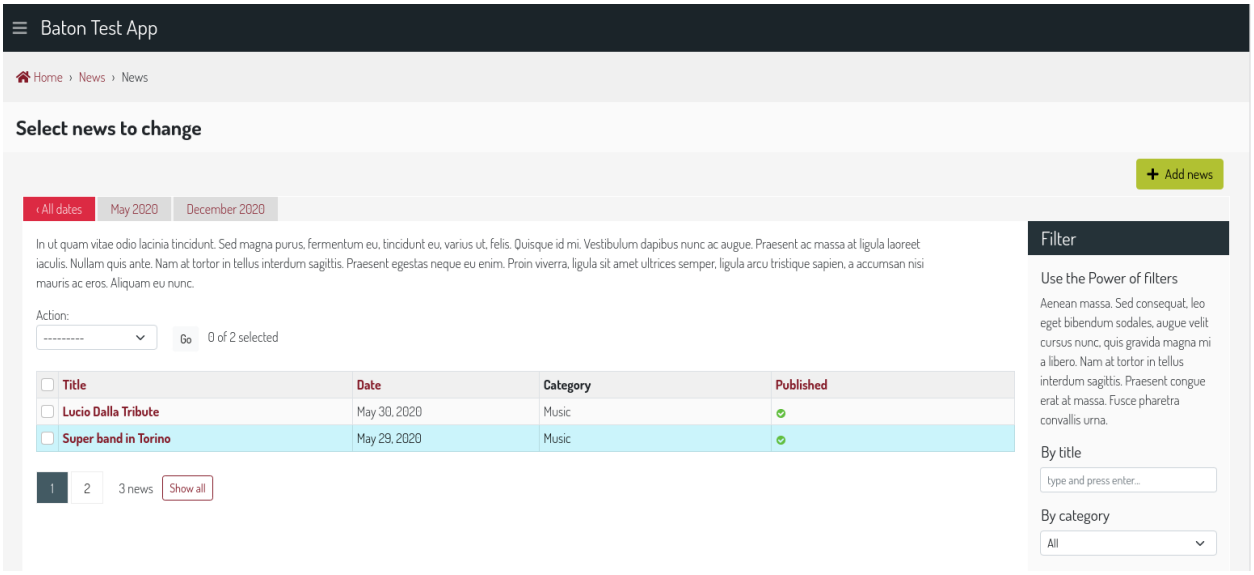

Baton Test App
Baton administration
admin
SYSTEM
Authentication
RESOURCES
File explorer
NEWS
Categories
News
TOOLS
Authentication and Authorization
Groups + Add Change
Users + Add Change
News
Categories + Add Change
News + Add Change
Filer
Folders Change
Thumbnail options + Add Change
Recent actions
My actions
Lucio Dalla Tribute
News
Lucio Dalla Tribute
News
Super party
News
Lucio Dalla Tribute
News
Super party
News
News
Category


Baton Test App
admin
SYSTEM
Authentication
RESOURCES
File explorer
NEWS
Categories
News
TOOLS
Home › News › News
Select news to change
The news "Lucio Dalla Tribute" was changed successfully.
Add news
‹ All dates
May 2020
December 2020
In ut quam vitae odio lacinia tincidunt. Sed magna purus, fermentum eu, tincidunt eu, varius ut, felis. Quisque id mi. Vestibulum dapibus nunc ac augue. Praesent ac massa at ligula laoreet iaculis. Nullam quis ante. Nam at tortor in tellus interdum sagittis. Praesent egestas neque eu enim. Proin viverra, ligula sit amet ultrices semper, ligula arcu tristique sapien, a accumsan nisi mauris ac eros. Aliquam eu nunc.
Filter
Action:

Go
0 of 2 selected
Title
Date
Category
Published
Lucio Dalla Tribute
May 30, 2020
Music
Super band in Torino
May 29, 2020
Music
1
2
3 news
Show all


admin
SYSTEM
Authentication
RESOURCES
File explorer
NEWS
Categories
News
TOOLS
History
Please correct the error below.
Main
Dates
Flags
Attachments
Videos
This is a description text
Category:
Music
Title:
Lucio Dalla Tribute
Click here
please insert a cool title
Enter a valid URL.
Link:
la-sezione/
BATON
Here you should specify exaclty what your news is about.
As you can see, this is just a plain text field, this means that you can use all the power of html:
Lists
Tables
Code fragments
...
Content:
Paragraph


Baton Test App
admin
SYSTEM
Authentication
RESOURCES
File explorer
NEWS
Categories
News
TOOLS
Home › News › News
Select news to cha
Add news
All dates
May 2020
Filter
Action:
Title
Lucio Dalla Tribute
Super band in Torino
Published
3 news
Filter
Use the Power of filters
Aenean massa. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc, quis gravida magna mi a libero. Nam at tortor in tellus interdum sagittis. Praesent congue erat at massa. Fusce pharetra convallis urna.
By title
type and press enter...
By category
All
By date
Any date
Today
Past 7 days
This month
This year
By published
All
Yes
No
Baton Test App · Developed by Otto srl


Baton Test App
Home › News › News
Select news to change
Add news
All dates
May 2020
December 2020
In ut quam vitae odio lacinia tincidunt. Sed magna purus, fermentum eu, tincidunt eu, varius ut, felis. Quisque id mi. Vestibulum dapibus nunc ac augue. Praesent ac massa at ligula laoreet iaculis. Nullam quis ante. Nam at tortor in tellus interdum sagittis. Praesent egestas neque eu enim. Proin viverra, ligula sit amet ultrices semper, ligula arcu tristique sapien, a accumsan nisi mauris ac eros. Aliquam eu nunc.
Action:
Go
0 of 2 selected
Title
Date
Category
Published
Lucio Dalla Tribute
May 30, 2020
Music
Super band in Torino
May 29, 2020
Music
1
2
3 news
Show all
Filter
Use the Power of filters
Aenean massa. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc, quis gravida magna mi a libero. Nam at tortor in tellus interdum sagittis. Praesent congue erat at massa. Fusce pharetra convallis urna.
By title
type and press enter...
By category
All