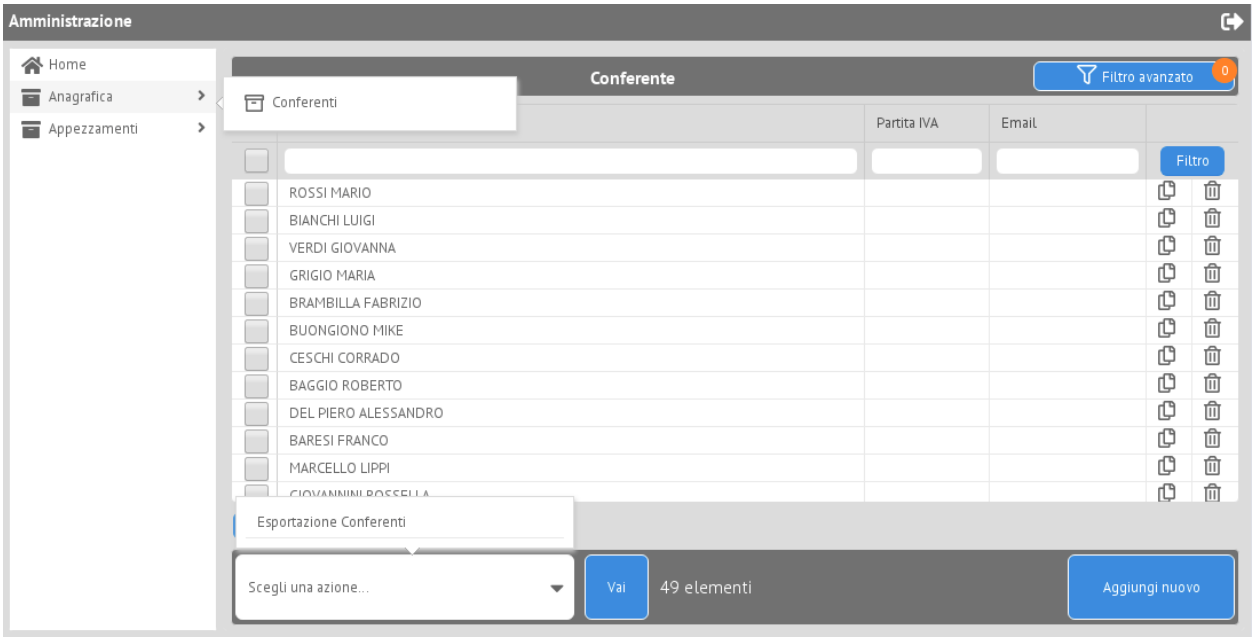
Django Webix Documentation

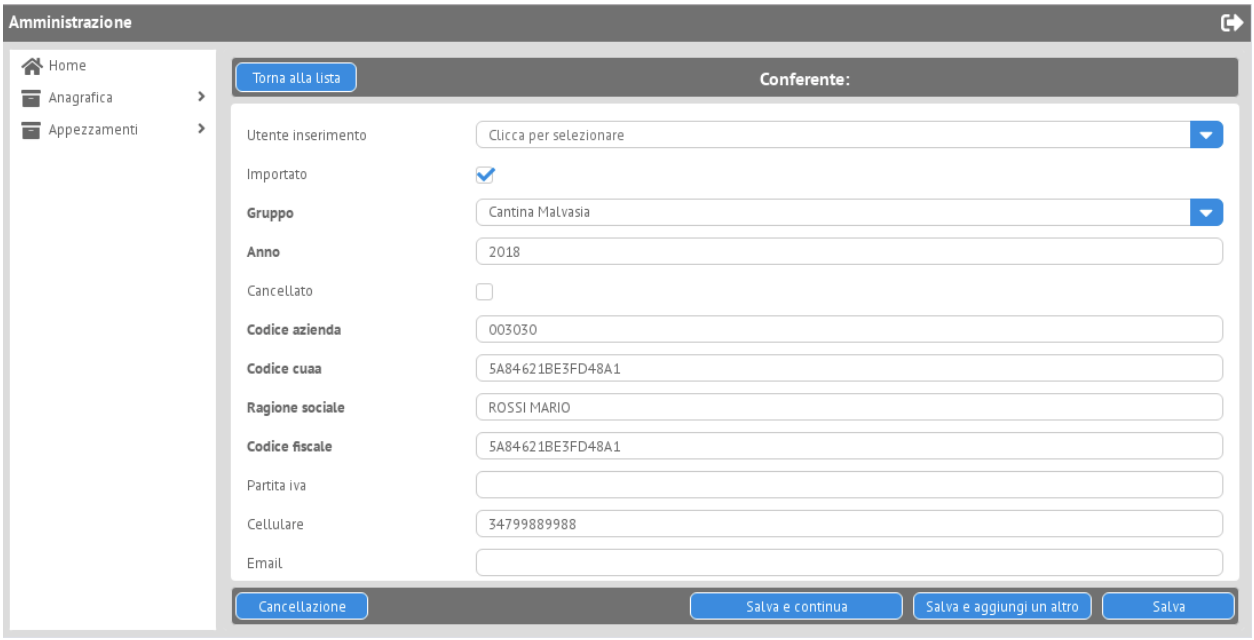
Release 1.4.0

MPA Solutions

May 17, 2022

Contents


Amministrazione
Home
Anagrafica
Appezzamenti
Conferenti
Conferente
Filtro avanzato
0
Partita IVA
Email
Filtro
ROSSI MARIO
BIANCHI LUIGI
VERDI GIOVANNA
GRIGIO MARIA
BRAMBILLA FABRIZIO
BUONGIONO MIKE
CESCHI CORRADO
BAGGIO ROBERTO
DEL PIERO ALESSANDRO
BARESI FRANCO
MARCELLO LIPPI
Esportazione Conferenti
Scegli una azione...
Vai
49 elementi
Aggiungi nuovo


Amministrazione
Home
Anagrafica
Appezzamenti
Torna alla lista
Conferente:
Utente inserimento
Clicca per selezionare
Importato
Gruppo
Cantina Malvasia
Anno
2018
Cancellato
Codice azienda
003030
Codice cuaa
5A84621BE3FD48A1
Ragione sociale
ROSSI MARIO
Codice fiscale
5A84621BE3FD48A1
Partita iva
Cellulare
34799889988
Email
Cancellazione
Salva e continua
Salva e aggiungi un altro
Salva

CHAPTER 1

Contents

1.1 Django Webix

Use the Webix JavaScript UI library with Django

1.1.1 Documentation

The full documentation is at https://django-webix.readthedocs.io.

1.1.2 Quickstart

Install Django Webix:

```
$ pip install django-webix
```

Add `django-webix` to your `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'django_webix',
```

(continues on next page)

(continued from previous page)

```
    # ...
]
```

Add `django-webix` URLconf to your project `urls.py` file

```
from django.conf.urls import url, include

urlpatterns = [
    # ...
    url(r'^django-webix/', include('django_webix.urls')),
    # ...
]
```

Add internationalization to *TEMPLATES*

```
TEMPLATES = [
    {
        # ...
        'OPTIONS': {
            'context_processors': [
                # ...
                'django.template.context_processors.i18n',
            ],
        },
    },
]
```

Include `webix static files` folder in your django staticfiles folder as `webix` and add static configuration

```
STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
)
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'staticfiles'),
)
STATIC_URL = '/static/'
```

1.1.3 Running Tests

Does the code actually work?

```
$ source <YOURVIRTUALENV>/bin/activate
$ (myenv) $ pip install tox
$ (myenv) $ tox
```

1.1.4 Contributors

Here is a list of Django-Webix's contributors.

1.2 Configurations

1.2.1 Install

`django-webix` is available on `https://pypi.python.org/pypi/django-webix/` install it simply with:

```
$ pip install django-webix
```

1.2.2 Configure

Settings

Add `django_webix` to your `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'django_webix',
    # ...
]
```

Add `django-webix` URLconf to your project `urls.py` file

```
from django.conf.urls import url, include

urlpatterns = [
    # ...
    url(r'^django-webix/', include('django_webix.urls')),
    # ...
]
```

Add internationalization to *TEMPLATES*

```
TEMPLATES = [
    {
        # ...
        'OPTIONS': {
            'context_processors': [
                # ...
                'django.template.context_processors.i18n',
            ],
        },
    },
]
```

Include `webix static files` folder in your django staticfiles folder as `webix` and add static configuration

```
STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
)
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'staticfiles'),
)
STATIC_URL = '/static/'
```

Configuration variables

```
WEBIX_LICENSE = 'PRO'  # FREE
WEBIX_VERSION = '7.0.3'
WEBIX_CONTAINER_ID = 'content_right'
WEBIX_FONTAWESOME_CSS_URL = 'fontawesome/css/all.min.css'
WEBIX_FONTAWESOME_VERSION = '5.12.0'
WEBIX_HISTORY_ENABLE = True # optional
```

1.3 Views

1.3.1 Usage

Forms

Create the forms (e.g. <app_name>/forms.py)

```
from django_webix.forms import WebixModelForm

from <app_name>.models import MyModel

class MyModelForm(WebixModelForm):
    class Meta:
        model = MyModel
        fields = '__all__'
```

Views

Create the views (e.g. <app_name>/views.py)

```
import json

from django.views.generic import TemplateView

from django_webix.formsets import WebixTabularInlineFormSet, WebixStackedInlineFormSet
from django_webix.views import WebixListView, WebixCreateView, WebixUpdateView,␣
↪WebixDeleteView

from <app_name>.forms import MyModelForm
from <app_name>.models import MyModel, InlineModel


class HomeView(TemplateView):
    template_name = 'base.html'


class InlineModelInline(WebixStackedInlineFormSet):
    model = InlineModel
    fields = '__all__'


class MyModelListView(WebixListView):
    model = MyModel
```

(continues on next page)

(continued from previous page)

```
    footer = True

    # paging
    enable_json_loading = True
    paginate_count_default = 100
    paginate_start_default = 0
    paginate_count_key = 'count'
    paginate_start_key = 'start'

    def get_queryset(self, initial_queryset=None):
        # custom queryset with annotate etc? is possibile :-)

        initial_queryset = MyModel.objects.all()
        return super().get_queryset(initial_queryset=initial_queryset)

    fields = [
        { # char example
            'field_name': 'XXX',
            'datalist_column': '''{id: "XXX", serverFilterType:"icontains", header: ["
↪{{_("TEXT1")|escapejs}}", {content: "serverFilter"}], fillspace: true, sort: "server
↪"}'''
        },
        { # FK example
            'field_name': 'YYYY',
            'datalist_column': ''' {id: "YYYY", serverFilterType:"exact", header: ["{
↪{_("TEXT2")|escapejs}}", {content: "serverSelectFilter", options:YYYY_options}],␣
↪adjust: "all", sort: "server"}'''
        },
        { # number example (in this case by interface is possibile to write for␣
↪example "<=5" )
            'click_action': '''custom_js_function_to_add_into_js(el['id']);''',
            'field_name': 'ZZZZ',
            'footer': Sum('ZZZZ'),
            'datalist_column': '''{id: "ZZZZ", serverFilterType:"numbercompare",␣
↪header: ["{{_("TEXT3")|escapejs}}", {content: "numberFilter"}], css: {'text-align':
↪'right'}, adjust: "all", sort: "server"}'''
        },
    ]


class MyModelCreateView(WebixCreateView):
    model = MyModel
    inlines = [InlineModelInline]
    form_class = MyModelForm


class MyModelUpdateView(WebixUpdateView):
    model = MyModel
    inlines = [InlineModelInline]
    form_class = MyModelForm


class MyModelDeleteView(WebixDeleteView):
    model = MyModel
```

CreateView and UpdateView Signals

When createview and updateview work some signals are sended.

```
```

django_webix_view_pre_save.send(sender=self, instance=None, created=True, form=form, inlines=inlines) django_webix_view_pre_inline_save.send(sender=self, instance=self.object, created=True, form=form, inlines=inlines) django_webix_view_post_save.send(sender=self, instance=self.object, created=True, form=form, inlines=inlines)

DeleteView Signals

When deleteview works some signals are sended.

```
```

django_webix_view_pre_delete.send(sender=self, instance=self.object) django_webix_view_post_delete.send(sender=self, instance=self.copied_object)

ListView Actions

Create the actions (e.g. <app_name>/actions.py)

```
from django.http import JsonResponse

from django_webix.views.generic.decorators import action_config

# list checkboxes actions
@action_config(action_key='CUSTOMKEY',
               response_type='json',
               short_description='TEXT4')
def my_action(self, request, qs):
    qs.update(status='p')
    return JsonResponse({
        "status": True,
        "message": 'Updated {} items'.format(qs.count()),
        "redirect_url": self.get_url_list(),
    }, safe=False)
```

Urls

Register the views url (e.g. <project_name>/urls.py)

```
from django.urls import path

from <app_name>.views import HomeView, MyModelListView, MyModelCreateView,␣
↪MyModelUpdateView, MyModelDeleteView

urlpatterns = [
    # ...
```

(continues on next page)

(continued from previous page)

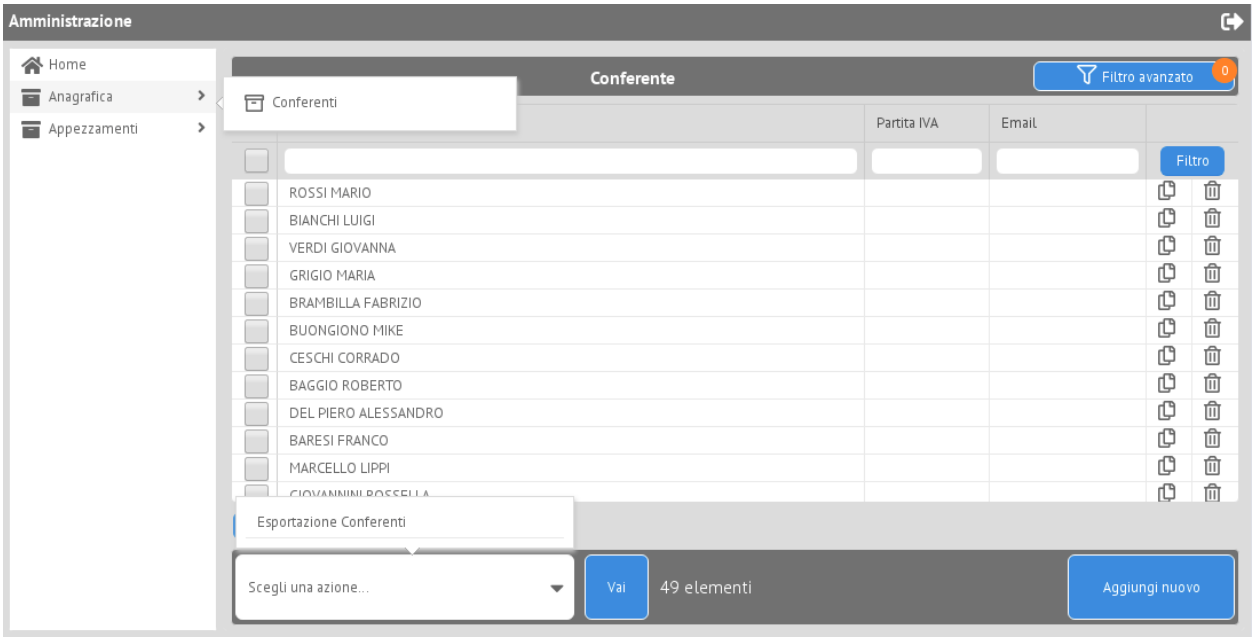
```
    path('', HomeView.as_view(), name='home'),

    path('mymodel/list', MyModelListView.as_view(), name='myapplication.mymodel.list
↪'),
    path('mymodel/create', MyModelCreateView.as_view(), name='myapplication.mymodel.
↪create'),
    path('mymodel/update/<int:pk>', MyModelUpdateView.as_view(), name='myapplication.
↪mymodel.update'),
    path('mymodel/delete/<int:pk>', MyModelDeleteView.as_view(), name='myapplication.
↪mymodel.delete'),
    # ...
]
```

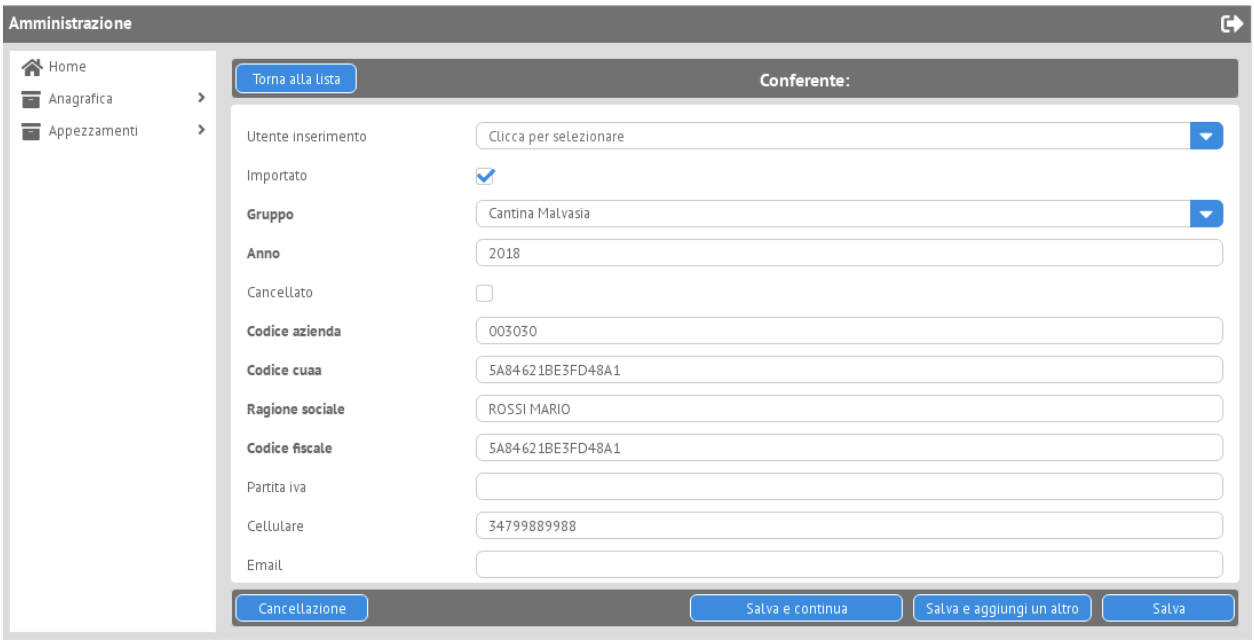
Base Template

Create a base html template (e.g. <app_name>/templates/base.html)

```
{% load i18n %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    {% include "django_webix/static_meta.html" %}
</head>
<body>
</body>

<script type="text/javascript" charset="utf-8">
    webix.ready(function () {
        webix.ui({
            id: 'content_right',
            rows: []
        });

        webix.extend($$('content_right'), webix.OverlayBox);

        load_js('{% url 'myapplication.mymodel.list' %}');
    });
</script>
</html>
```

1.4 Admin





1.4.1 Configure

Settings

Description

```
INSTALLED_APPS = [
    # ...
```

(continues on next page)

(continued from previous page)

```
    'django_webix.admin_webix',
    # ...
]
```

Urls

Register the views url (e.g. <project_name>/urls.py)

```
from django.conf.urls import url

from <somewhere> import admin_webix

urlpatterns = [
  path('admin_webix/', admin_webix.site.urls), # or another paths :-)
]
```

1.4.2 Basic Installation

Admin Webix Site

For a basic usage it's enough register admin models on predefined site

```
from django_webix import admin_webix as admin

@admin.register(XXXModelName)
class XXXAdmin(admin.ModelWebixAdmin):
    pass
```

1.4.3 Advanced Installation

Admin Webix Site

If an advanced use is needed it's enough extend default Site class. Here an example:

```
import datetime

from django.apps import apps
from django.conf import settings
from django.utils.functional import LazyObject
from django.utils.module_loading import import_string


class CustomSiteAdminWebixSite(LazyObject):
    def _setup(self):
        AdminWebixSiteClass = import_string(apps.get_app_config('admin_webix').
↪default_site)

        def extra_index_context(self, request):
            if request.session.get('year',None) == None:
                request.session['year'] = datetime.datetime.today().year
            return {
```

(continues on next page)

(continued from previous page)

```
                'DEBUG': settings.DEBUG,
                'years': list(range(2020, datetime.datetime.today().year + 1)),
            }

        AdminWebixSiteClass.extra_index_context = extra_index_context

        self._wrapped = AdminWebixSiteClass()

custom_site = CustomSiteAdminWebixSite()

# others customization parameters
custom_site.site_title = gettext_lazy('Django webix site admin')
custom_site.site_header = gettext_lazy('Django webix administration')
custom_site.index_title = gettext_lazy('Site administration')
custom_site.site_url = '/'
custom_site.login_form = None
custom_site.webix_container_id = 'content_right'
custom_site.index_template = None
custom_site.login_template = None
custom_site.logout_template = None
custom_site.password_change_template = None
custom_site.password_change_done_template = None
```

1.4.4 Basic Usage

Admin Webix

Create the files (e.g. <app_name>/admin_webix.py) and there is a simple example:

```
from anagrafica.models import Conferente
from appezzamenti.models import Appezzamento, UnitaVitata
from django_webix import admin_webix as admin


@admin.register(Conferente)
class ConferenteAdmin(admin.ModelWebixAdmin):
    list_display = ['ragione_sociale', 'partita_iva', 'email']
    enable_json_loading = True


@admin.register(Appezzamento)
class AppezzamentoAdmin(admin.ModelWebixAdmin):
    list_display = ['conferente__ragione_sociale', 'codice', 'denominazione']
    enable_json_loading = True
```

1.4.5 Advanced Usage

Parameters

Create the files (e.g. <app_name>/admin_webix.py) and here there is an example of full list of parameters and funcions that can be override.

```
from anagrafica.models import Conferente
from appezzamenti.models import Appezzamento, UnitaVitata
from django_webix import admin_webix as admin

# INLINE
class IndiceStambeccoInline(IndiceInlineMixin, WebixStackedInlineFormSet):
    model = IndiceStambecco
    form_class = IndiceStambeccoForm

# ACTION
@action_config(action_key='validazione',
               response_type='json',
               short_description='Validazione, investimenti e rinvenimenti',
               allowed_permissions=['delete'])
def validazione_trimestrale(self, request, qs):
    user = request.user
    stazioni = []
    if user.get_profilo().direttore_distretto:
        stazioni = user.get_profilo().stazioni_forestali.values_list('ogc_fid',␣
↪flat=True)
    qs = qs.filter(Q(tipo_dato=TipoDato.objects.get(codice='rinvenimento')) |
                   Q(tipo_dato=TipoDato.objects.get(codice='investimento'))
                   )

    if not user.is_admin():
        qs = qs.filter(stazione_forestale__in=stazioni)
    count = int(qs.count())
    qs.update(validata=True)
    return JsonResponse({
        "status": True,
        "message": '{count_delete_instances} schede sono state validate e non sono␣
↪piu modificabili'.format(
            count_delete_instances=count),
        "redirect_url": self.get_url_list(),
    }, safe=False)

@admin.register(Conferente)
class ConferenteAdmin(admin.ModelWebixAdmin):
    # WEBIX VIEWS (for fully override)
    create_view = None
    update_view = None
    delete_view = None
    list_view = None

    # JS TEMPLATES
    add_form_template = None
    change_form_template = None
    change_list_template = None
    delete_template = None
    dashboard_template = 'admin_webix/dashboard.js'

    # CREATE AND UPDATE SETTINGS
    enable_button_save_continue = True
    enable_button_save_addanother = True
    enable_button_save_gotolist = True

    # DJANGO WEBIX FORM: OPTION 1
```

(continues on next page)

(continued from previous page)

```
    autocomplete_fields = []
    readonly_fields = []
    fields = None
    exclude = None
    # DJANGO WEBIX FORM: OPTION 2
    form = None

    inlines = [IndiceStambeccoInline]

    # LIST SETTINGS
    ordering = None
    actions = [multiple_delete_action, validazione_trimestrale]
    list_display = [
        {
            'field_name': 'codice',
            'datalist_column': '''{id: "codice", serverFilterType:"icontains",␣
↪header: ["{{_("Codice appezzamento")|escapejs}}", {content: "serverFilter"}],␣
↪adjust: "all", sort: "server"}'''
        },
        {
            'field_name': 'numero_piante_sintomatiche',
            'click_action': '''set_webgis_item('Appezzamento', [el['id']], el['bbox
↪']);''',
            'footer': Sum('numero_piante_sintomatiche'),
            'datalist_column': '''{id: "numero_piante_sintomatiche", serverFilterType:
↪"icontains", header: ["{{_("Numero piante sintomatiche")|escapejs}}", {content:
↪"serverFilter"}], adjust: "all", sort: "server"}'''
        },
    ]
    extra_header = {}

    enable_json_loading = False
    pk_field = None
    title = None
    actions_style = None
    enable_column_copy = True
    enable_column_delete = True
    enable_row_click = True
    type_row_click = 'single'
    enable_actions = True
    remove_disabled_buttons = False

    # permission custom
    only_superuser = False

    def get_list_display(self, request=None):
        pass

    def is_webgis_enable(self):
        pass

    def is_webix_filter_enable(self):
        pass

    def get_model_perms(self, request):
        return {
            'add': self.has_add_permission(request),
```

(continues on next page)

(continued from previous page)

```
            'change': self.has_change_permission(request),
            'delete': self.has_delete_permission(request),
            'view': self.has_view_permission(request),
        }

    def has_add_permission(self, request):
        if self.get_queryset(request=request).exists():
            return False
        else:
            return True

    def get_failure_add_related_objects(self, request):
        return []

    def get_failure_change_related_objects(self, request, obj=None):
        return []

    def get_failure_delete_related_objects(self, request, obj=None):
        return []

    def get_failure_view_related_objects(self, request, obj=None):
        return []

    def get_info_no_add_permission(self, has_permission, request):
        if not has_permission:
            return [_("You haven't add permission")]
        return []

    def get_info_no_change_permission(self, has_permission, request, obj=None):
        if not has_permission:
            return [_("You haven't change permission")]
        return []

    def get_info_no_delete_permission(self, has_permission, request, obj=None):
        if not has_permission:
            return [_("You haven't delete permission")]
        return []

    def get_info_no_view_permission(self, has_permission, request, obj=None):
        if not has_permission:
            return [_("You haven't view permission")]
        return []

    def get_queryset(self, request):
        qs = super().get_queryset(request=request).filter(anno=request.session.get(
↪'anno', 0))
        if request.user.is_nucleo():
            qs = qs.filter(nucleo__userprofile__user=request.user)
        return qs.distinct()

    def get_add_view(self):
        AddView = super().get_add_view()

        def pre_forms_valid(self, form=None, inlines=None, **kwargs):
            form.instance.anno = self.request.session['anno']
            form.instance.nucleo = self.request.user.get_profilo().nucleo
        AddView.pre_forms_valid = pre_forms_valid
```

(continues on next page)

(continued from previous page)

```
        return AddView
```

1.5 Advanced Usage

1.5.1 Inline Templates

```
from django_webix.formsets import WebixTabularInlineFormSet, WebixStackedInlineFormSet
from django_webix.views import WebixCreateWithInlinesUnmergedView,␣
↪WebixUpdateWithInlinesUnmergedView

from <app_name>.forms import MyModelForm
from <app_name>.models import MyModel, InlineModel


class InlineModelInline(WebixStackedInlineFormSet):
    model = InlineModel
    fields = '__all__'


class MyModelCreateView(WebixCreateWithInlinesUnmergedView):
    model = MyModel
    inlines = [InlineModelInline]
    form_class = MyModelForm


class MyModelUpdateView(WebixUpdateWithInlinesUnmergedView):
    model = MyModel
    inlines = [InlineModelInline]
    form_class = MyModelForm
```

1.5.2 Inline QuerySet

```
from django_webix.formsets import WebixStackedInlineFormSet
from django_webix.views import WebixCreateWithInlinesView, WebixUpdateWithInlinesView,
↪ WebixDeleteView

from <app_name>.models import InlineModel


class InlineModelInline(WebixStackedInlineFormSet):
    model = InlineModel
    fields = '__all__'

    def get_queryset(self):
        return self.inline_model.objects.filter(**filters)
```

1.5.3 Custom FormSet Class

```
from django_webix.formsets import BaseWebixInlineFormSet


class CustomInlineFormSet(BaseWebixInlineFormSet):
    # ...


class InlineModelInline(WebixStackedInlineFormSet):
    # ...
    custom_formset_class = CustomInlineFormSet
    # ...
```

1.5.4 Add custom action with user input request

Write custom ui to django-webix acton with input parametrs asked to users.

Python view

In python view write the action as usual with *@action_config*

```
@action_config(action_key='my_action_name',
               response_type='json',
               short_description=_("Action title"),
               allowed_permissions=[])
def my_action(self, request, qs):
    _elements_count = int(qs.count())
    # ...
    return JsonResponse({
        "status": True,
        "message": _("{elements_count} elements succesfully updated").format(
            elements_count=_elements_count
        ),
        "redirect_url": self.get_url_list(),
    })
```

Javascript template

In javascript list template overwrite the *toolbar_list_actions* block

1. First of all write a new function with the action user interface definition, usually with some user input widgets (choices, text)
    - This function show the user input widgets windows, and then, on confirm callback, runs the action_execute function with the input params as last optional argument (all the other arguments are not changed)
    - The input params shoud be an object like: {'my_choice': 'choice_value'}

```
var my_custom_action_ui = function (action, ids, all, response_type,␣
↪short_description, modal_title, modal_ok, modal_cancel) {
    webix.ui({
        view: "window",
```

(continues on next page)

(continued from previous page)

```
        width: 300,
        modal: true,
        position: "center",
        head: {
            view: "label",
            label: short_description,
            align: "center",
        },
        body: {
            view: "form",
            rules: {
                "my_choices_name": webix.rules.isNotEmpty,
            },
            elements: [
                // INPUT TITLE
                {template: "Select the desired value", borderless: true,
↪css: {"text-align": "center"}, autoheight: true},
                // INPUT WIDGETS
                {
                    cols: [
                        {},
                        {
                            view: "richselect",
                            label: "Choices label",
                            labelWidth: 100,
                            width: 250,
                            name: 'my_choices_name',   // name serve per
↪rules, validate e get form elements
                            invalidMessage: "Required to select a value",
                            options: [
                                {id: 'option_1', value: "Option 1"},
                                {id: 'option_2', value: "Option 2"}
                            ]
                        },
                        {}
                    ]
                },
                {height: 5},
                // FOOTER WITH CENETERD BUTTONS: | Cancel | Send |
                {
                    margin: 5,
                    cols: [
                        {},
                        {
                            view: "button",
                            width: 100,
                            value: modal_cancel,
                            click: function () {
                                // returns the top parent view, for
↪element in window: window
                                this.getTopParentView().hide();
                            }
                        },
                        {
                            view: "button",
                            width: 100,
                            value: modal_ok,
```

(continues on next page)

(continued from previous page)

```
                            css: "webix_primary",
                            click: function () {
                                if (this.getFormView().validate()) {
                                    this.getTopParentView().hide();
                                    var params = {'my_choice': this.
↪getFormView().elements["my_choices_name"].getValue()}
                                    _{{ view_prefix }}action_execute(
                                        action, ids, all, response_type,␣
↪short_description, modal_title, modal_ok, modal_cancel, params
                                    )
                                }
                            }
                        },
                        {}
                    ]
                }
            ]
        }
    }).show();
};
```

2. Then overwrite the *toolbar_list_actions* block to use *my_custom_action_ui*

```
{% block toolbar_list_actions %}
    {% if is_enable_actions %}

        var {{ view_prefix }}actions_list = [
            {% for action_key,action in actions.items %}
                {id: '{{ action_key }}', value: '{{ action.short_
↪description }}'}{% if not forloop.last %}, {% endif %}
            {% endfor %}
        ];

        function {{ view_prefix }}actions_execute(action, ids, all) {
            {% for action_key, action in actions.items %}
                {% if action_key == 'my_action_name' %}
                    if (action == '{{ action_key }}') {
                        my_custom_action_ui(
                            '{{ action_key }}',
                            ids,
                            all,
                            '{{ action.response_type }}',
                            '{{ action.short_description }}',
                            '{{ action.modal_title }}',
                            '{{ action.modal_ok }}',
                            '{{ action.modal_cancel }}'
                        )
                    } {% if not forloop.last %} else {% endif %}
                {% else %}
                    if (action == '{{ action_key }}') {
                        _{{ view_prefix }}action_execute(
                            '{{ action_key }}',
                            ids,
                            all,
                            '{{ action.response_type }}',
                            '{{ action.short_description }}',
                            '{{ action.modal_title }}',
```

(continues on next page)

(continued from previous page)

```
                            '{{ action.modal_ok }}',
                            '{{ action.modal_cancel }}'
                        )
                    } {% if not forloop.last %} else {% endif %}
                {% endif %}
            {% endfor %}
        }
    {% else %}
        var {{ view_prefix }}actions_list = undefined;
        var {{ view_prefix }}actions_execute = undefined;
    {% endif %}
{% endblock %}
```

Conclusions

Finally you can access the input parametrs as request POST data in the action method *my_action(self, request, qs)*

```
params = json.loads(request.POST['params'])
choice_value = params['my_choice']
```

1.6 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.6.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/MPASolutions/django-webix/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

Django Webix could always use more documentation, whether as part of the official Django Webix docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/MPASolutions/django-webix/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.6.2 Get Started!

Ready to contribute? Here's how to set up *django-webix* for local development.

1. Fork the *django-webix* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-webix.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-webix
$ cd django-webix/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 django_webix tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, and 3.4, 3.5, 3.6, 3.7 and for PyPy. Check https://travis-ci.org/MPASolutions/django-webix/pull_requests and make sure that the tests pass for all supported Python versions.

1.6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_django_webix
```

1.7 Credits

This package was developed by MPA Solutions

1.7.1 Development Lead

- Alessandro Regolini <regolini@mpasol.it>
- Alessio Bazzanella <bazzanella@mpasol.it>

1.7.2 Contributors

None yet. Why not be the first?

1.8 Class Reference

1.8.1 Decorators

`django_webix.utils.decorators.`**`script_login_required`**(*view*, *login_url=None*)

1.8.2 Template Tags

`django_webix.templatetags.django_webix_utils.`**`do_if_has_tag`**(*parser*, *token*, *negate=False*)

The logic for both `{% if_has_tag %}` and `{% if not_has_tag %}`. Checks if all the given tags exist (or not exist if `negate` is `True`) and then only parses the branch that will not error due to non-existing tags. This means that the following is essentially the same as a `{% comment %}` tag:

```
{% if_has_tag non_existing_tag %}
    {% non_existing_tag %}
{% endif_has_tag %}
```

Another example is checking a built-in tag. This will always render the current year and never FAIL:

```
{% if_has_tag now %}
    {% now "Y" %}
{% else %}
    FAIL
{% endif_has_tag %}
```

django_webix.templatetags.django_webix_utils.**friendly_load**(*parser*, *token*)

Tries to load a custom template tag set. Non existing tag libraries are ignored. This means that, if used in conjunction with `if_has_tag`, you can try to load the comments template tag library to enable comments even if the comments framework is not installed. For example:

```
{% load friendly_loader %}
{% friendly_load comments webdesign %}
{% if_has_tag render_comment_list %}
    {% render_comment_list for obj %}
{% else %}
    {% if_has_tag lorem %}
        {% lorem %}
    {% endif_has_tag %}
{% endif_has_tag %}
```

django_webix.templatetags.django_webix_utils.**get_value_from_dict**(*dict_data*, *key*)

usage example {{ your_dict|get_value_from_dict:your_key }}

django_webix.templatetags.django_webix_utils.**getattr**(*obj*, *args*)

Try to get an attribute from an object.

Example: {% if block|getattr:"editable,True" %}

Beware that the default is always a string, if you want this to return False, pass an empty second argument: {% if block|getattr:"editable," %}

django_webix.templatetags.django_webix_utils.**if_has_tag**(*parser*, *token*)

Do something if all given tags are loaded:: {% load friendly_loader %} {% friendly_load webdesign %} {% if_has_tag lorem %}

{% lorem %}

{% else %} Non dummy content goes here!

{% endif_has_tag %}

When given multiple arguments each and every tag in the list has to be available. This means that the following will render nothing:

```
{% if_has_tag now nonexisting_tag %}
    {% now "Y" %}
{% endif_has_tag %}
```

django_webix.templatetags.django_webix_utils.**ifnot_has_tag**(*parser*, *token*)

Do something unless any given tag is loaded:: {% load friendly_loader %} {% friendly_load comments %} {% ifnot_has_tag render_comment_list %}

Comment support has been disabled.

{% else %} {% render_comment_list for obj %}

{% endifnot_has_tag %}

In the case of multiple arguments, the condition will trigger if any tag in the list is unavailable. This means that the following will still render the current year:

```
{% ifnot_has_tag now nonexisting_tag %}
    {% now "Y" %}
{% endifnot_has_tag %}
```

1.8.3 Forms

1.8.4 Forms Mixin

1.8.5 Formsets

1.8.6 Views

1.8.7 View Mixins

class django_webix.views.generic.base.**WebixPermissionsMixin**

class django_webix.views.generic.base.**WebixUrlMixin**

class django_webix.views.generic.base.**WebixBaseMixin**

1.8.8 Utils

1.8.9 Admin Config

class django_webix.admin_webix.apps.**SimpleAdminWebixConfig**(*app_name*, *app_module*)

Simple AppConfig which does not do automatic discovery.

ready()

Override this method in subclasses to run code when Django starts.

class django_webix.admin_webix.apps.**AdminWebixConfig**(*app_name*, *app_module*)

The default AppConfig for admin which does autodiscovery.

ready()

Override this method in subclasses to run code when Django starts.

1.8.10 Admin Decorators

django_webix.admin_webix.decorators.**register**(**models*, *site=None*)

Register the given model(s) classes and wrapped ModelWebixAdmin class with admin site: @register(Author) class AuthorAdmin(admin.ModelAdmin):

pass

The *site* kwarg is an admin site to use instead of the default admin site.

Parameters

- **models** –

- **site** –

Returns

1.8.11 Admin Options

class django_webix.admin_webix.options.**ModelWebixAdmin**(*model*, *admin_site*)

get_model_perms(*request*)
Return a dict of all perms for this model. This dict has the keys `add`, `change`, `delete`, and `view` mapping to the True/False for each of those actions.

1.8.12 Admin Sites

class django_webix.admin_webix.sites.**AlreadyRegistered**

class django_webix.admin_webix.sites.**NotRegistered**

class django_webix.admin_webix.sites.**AdminWebixSite**(*name='admin_webix'*)

admin_view(*view*, *cacheable=False*)
Decorator to create an admin view attached to this `AdminWebixSite`. This wraps the view and provides permission checking by calling `self.has_permission`. You'll want to use this from within `AdminWebixSite.get_urls()`:

class MyAdminWebixSite(AdminWebixSite):

def get_urls(self): from django.urls import path urls = super().get_urls() urls += [

path('my_view/', self.admin_view(some_view))

] return urls

By default, admin_views are marked non-cacheable using the `never_cache` decorator. If the view can be safely cached, set cacheable=True.

each_context(*request*)
Return a dictionary of variables to put in the template context for *every* page in the admin site. For sites running on a subpath, use the SCRIPT_NAME value if site_url hasn't been customized.

get_app_list(*request*)
Return a sorted list of all the installed apps that have been registered in this site.

has_permission(*request*)
Return True if the given HttpRequest has permission to view *at least one* page in the admin site.

index(*request*, *extra_context=None*)
Display the main admin index page, which lists all of the installed apps that have been registered in this site.

is_registered(*model*)
Check if a model class is registered with this *AdminWebixSite*.

login(*request*, *extra_context=None*)
Display the login form for the given HttpRequest.

logout(*request*, *extra_context=None*)
Log out the user for the given HttpRequest. This should *not* assume the user is already logged in.

password_change(*request*, *extra_context=None*)
Handle the "change password" task – both form display and validation.

password_change_done(*request*, *extra_context=None*)
Display the "success" page after a password change.

password_reset(*request*, *extra_context=None*)
Handle the "reset password" task – both form display and validation.

password_reset_done(*request*, *extra_context=None*)
Handle the "reset password" task – both form display and validation.

register(*model_or_iterable*, *admin_class=None*, ***options*)
Register the given model(s) with the given admin class. The model(s) should be Model classes, not instances. If an admin class isn't given, use ModelWebixAdmin (the default admin options). If keyword arguments are given – e.g., list_display – apply them as options to the admin class. If a model is already registered, raise AlreadyRegistered. If a model is abstract, raise ImproperlyConfigured.

unregister(*model_or_iterable*)
Unregister the given model(s). If a model isn't already registered, raise NotRegistered.

class django_webix.admin_webix.sites.**DefaultAdminWebixSite**

1.9 Change Log

All notable changes to this project will be documented in this file.

The format is based on KeepAChangelog and this project adheres to SemanticVersioning.

1.9.1 [Unreleased]

1.9.2 [1.4.0] 2021-03-02

Added

- Added exact_in serverFilterType to use serverMultiComboFilter into datatables
- Added OTF filters
- Added *django-filtersmerger* integration
- Added *django-dal* integration
- Added *datarange* filter on datatables
- Added *FormView*
- Allow create/update views witout form_class

Changed

- Changed *from_dict_to_qset* params to check fields type

Removed

- Removed deprecated *WebixCreateWithInlinesView* class
- Removed deprecated *WebixCreateWithInlinesUnmergedView* class
- Removed deprecated *WebixUpdateWithInlinesView* class
- Removed deprecated *WebixUpdateWithInlinesUnmergedView* class

Fixed

- Fixed webgis templatetag loads
- Fixed geo column without webgis
- Fixed lists without form
- Fixed admin_webix logout template
- Fixed readonly fields
- Fixed *get_initial_queryset* method
- Fixed list actions with _blank response

1.9.3 [1.3.0] 2020-09-17

Added

- Stable admin subpackage with multiple functionalities
- *Select all* button in multicombo widget
- Multicombo in PostgreSQL ArrayField with options
- Added middleware to limit browser version
- Added possibility to remove uploaded images and files
- Added browser history urls with ajax requests
- Added PasswordInput widget support
- Added get_queryset_initial on list for initial queryset

Changed

- Add optional argumenent input_params to action_execute function (used to send input paremetr in POST request)
- Admin subpackage improvements
- jQuery version update
- login and logout pages improvements

Fixed

- Object title in create and update forms
- Fix ModelMultipleChoiceField and ModelChoiceField autocomplete with *to_field_name* different from pk
- Fixed duplicate set middleware on static
- Fixed buttons permissions in form view
- Fixed checkbox dimension
- Fixed readonly DateTime format
- Fixed DateTime with timezone in forms

1.9.4 [1.2.1] 2020-01-08

Changed

- Documentation

1.9.5 [1.2.0] 2020-05-28

Added

- Added *admin* subpackage
- Added auto localizated fields
- Added new translations
- Added *delete* confirmation message
- Added extra title for *WebixUpdateView*
- Added overlay container in settings with default *webix_container_id*
- Added signal in each view when some instance change
- Added name for toolbar
- Added *delete* action on list
- Added paging on list
- Added settings to set url of *fontawesome*
- Added param to allows different *dataType* with *load_js*
- Added option to specify which nested models will be show on delete page
- Added string fields config on *WebixListView* by default
- Added default *abort* for all base ajax requests
- Added *decorator* for identify user not authenticated and popup to login
- Added pk field option if *pk_field* different from 'id'
- Added *ordering* into *get_queryset* for standard generic views

Changed

- *InlineForeignKey* separated from control
- Split utils into multiple file
- Changed prefix in *WebixListView* templates
- Add extra ajax params to *load_js* function
- Header borderless

Removed

- remove empy choices

Fixed

- Fixed *SimpleArrayField* initial
- Fixed *DateField* initial
- Fixed *delete* and *copy* functions
- Fixed translations and adjust indentations
- Fixed inline stacked js
- Fixed list queryset
- Fixed post delete valid
- Fixed list without actions and list ordering
- Fixed upload label background
- Fixed delete action
- Fixed list without fields
- Fixed *get_url_create* with kwargs
- Fixed tag trans with escapejs
- Fixed choice for action style type
- Fixed autocomplete fix IE11
- Fixed *WebixListView* with paging and without
- Fixed url on is_popup
- Fixed *WebixListView* and *WebixTemplateView* without model
- Fixed *delete* and *copy* columns
- Fixed function before send expecially for csrf
- Fixed form send custom widget

1.9.6 [1.1.0] 2020-01-08

Added

- Added kwargs params on create for reverse url
- Added header inlines option
- Added post with parameters for redirect
- Added create and delete permission on formsets
- Added *ArrayField* of date on forms
- Added multiple file support
- Added option to put inline not in standard place
- Added webix *overlay* container id
- Added *geometry field* hidden
- Added initial by post on add

Changed

- Better button for add row on inlines

Removed

- Removed console.log

Fixed

- Fixed toolbar extra params
- Fixed template toolbar nav
- Fixed create/update template style
- Fixed stacked inline without rows
- Fixed delete button
- Fixed inline id
- Fixed readonly and autocomplete for inlines
- Fixed autocomplete fields
- Fixed default function post save form before inlines
- Fixed post form save before save inlines on update
- Fixed overlay only if exists
- Fixed *BaseWebixModelForm* with Django <= 2.0
- Fixed *FileField*
- Fixed import geos
- Fixed *InlineForeignKeyField*

- Fixed file input
- Fixed toolbar navigation escapejs

1.9.7 [1.0.0] 2019-10-07

Added

- Added translations
- *WebixUrlMixin* parent class of all django-webix views
- Set *permissions* into django-webix views to use django permissions (default True: use django permissions)
- Set *logs* into django-webix views to use django log entries
- *style* variable in *WebixCreateView WebixUpdateView* with possible values: *merged* and *unmerged*
- Added all permission types in context of all django-webix views
- Added urls in context of all django-webix views
- Added *model* and *model_name* in context of all django-webix views
- Added *CreateUpdateMixin*
- Added hedermenu, generic title, excel datatable webix export
- Added *TemplateListView* class view
- Added inline_id into inline forms and hook for custom js function for each inline
- Added true to checkbox boolean field
- Added disabled list actions
- Added *django_type_field* to identify original formfield
- Added model unique together validation into generic views

Changed

- *get_model_name*, *get_url_list*, *get_url_create*, *get_url_update*, *get_url_delete* moved to *WebixUrlMixin* as methods
- Changed permissions check in templates
- Separated generic views
- Improve copy list function

Removed

- Removed *get_model_name* from *GenericModelWebix*
- Removed *get_url_list* from *GenericModelWebix*
- Removed *get_url_create* from *GenericModelWebix*
- Removed *get_url_update* from *GenericModelWebix*
- Removed *get_url_delete* from *GenericModelWebix*

Fixed

- Check if *django.contrib.admin* is installed before add log entry
- Tests postgres database name
- Init *WebixModelForm* and *BaseWebixMixin* fix
- Forms *clean* method fix
- Fixed delete get_failure_delete_related_objects method
- Fixed initial values for inlines
- Fixed *JSONField*

Deprecated

- *GenericModelWebix* will be removed in a future release
- *WebixCreateWithInlinesView* has been renamed to *WebixCreateView*
- *WebixCreateWithInlinesUnmergedView* has been renamed to *WebixCreateView*
- *WebixUpdateWithInlinesView* has been renamed to *WebixUpdateView*
- *WebixUpdateWithInlinesUnmergedView* has been renamed to *WebixUpdateView*

1.9.8 [0.2.2] - 2019-08-06

Added

- Tree of nested object before delete an instance
- Prevent to delete an instance if has at least one nested object

Changed

- Django-extra-view updates
- *get_model_name* change separator between app_label and model_name from _ to .

Fixed

- Add new line in inline forms with filefield

1.9.9 [0.2.1] - 2019-08-05

Added

- Compatibility with Django 2.2

Changed

- Renamed templatetag *utils_getattr* to *django_webix_utils*

Fixed

- FileField download button
- FileField autoWidth
- Create new inline from empty form

1.9.10 [0.2.0] - 2019-02-26

Added

- Compatibility with Webix 6
- Added RadioSelect widget
- Added empty choice in select widget
- Form fields type checked with isinstance method

Changed

- Changed static path

1.9.11 [0.1.5] - 2018-10-11

Added

- JSONField postgresql support

Fixed

- Fix empty form fields initial values on clean validation error

1.9.12 [0.1.4] - 2018-10-02

Fixed

- Fix delete button click ajax data

1.9.13 [0.1.3] - 2018-10-01

Changed

- Hide tabbar without inlines

Fixed

- Fix readonly dates

1.9.14 [0.1.2] - 2018-10-01

Changed

- Static files updates and include fixes

1.9.15 [0.1.1] - 2018-09-26

Fixed

- Serializer encoder fix

1.9.16 [0.1] - 2018-09-26

Added

- First release on PyPI.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index