ionyweb Documentation

Release 0.4dev

Ionyse

November 13, 2013

Contents

A CMS based on Django with a powerfull Javascript UI based on a REST API.

CHAPTER 1

More informations

- The code : https://github.com/ionyse/ionyweb/
- The documentation : http://ionyweb.readthedocs.org/
- IRC Chan : Ask #ionyweb@irc.freenode.net

1.1 Contents

1.1.1 Getting Started

1. Get the code

The development version

To be able to hack on the code:

```
$ cd ~/git/
$ git clone https://github.com/ionyse/ionyweb.git
$ cd ionyweb
$ mkvirtualenv ionyweb
(ionyweb)$ python setup.py develop
```

To install the last stable version

Using pip:

```
$ pip install ionyweb
```

or:

```
$ pip install -e git+git://github.com/ionyse/ionyweb.git#egg=ionyweb
```

2. Starts a new Ionyweb project

```
$ workon ionyweb
(ionyweb)$ ionyweb-quickstart <NewProject>
# Configure your MySQL database autoconfigure in your settings.py
(ionyweb)$ cd <NewProject>
(ionyweb)$ make syncdb
(ionyweb)$ make runserver
```

After that, you will have to set-up a themes for your website.

1.1.2 Working with Themes and Layouts

Some themes and layouts are provided with ionyweb.

In the code they are on *ionyweb/contrib/*.

In your ionyweb project, you can also put some local themes and layout.

Don't forget to define the *LAYOUTS_DIR* and *THEMES_DIR* in you settings.

```
LAYOUTS_DIRS = (
    os.path.join(ABSOLUTE_PATH, 'layouts'),
    os.path.join(get_ionyweb_path(), 'contrib', 'layouts'),
)

THEMES_DIRS = (
    os.path.join(ABSOLUTE_PATH, 'themes'),
    os.path.join(get_ionyweb_path(), 'contrib', 'themes'),
)
```

Themes

Files tree of a theme

On a theme, you must create a *MANIFEST.json*:

```
{"title": "Jungleland",
 "description": "A stylish warm brown design suitable for environmental community and nature business
 "author": "styleshout",
 "website": "http://www.styleshout.com/",
 "date": "01/09/2009",
 "preview": "jungleland.jpg",
     "editable": false
}
```

You must define a preview image that will give a visual overview of your theme.

A theme can have multiple styles, if you need only one, you must call it `default`.

Then you must to have a *templates* directory with all your template inside. They will be find there by the *ionyweb.loaders.themes_templates.Loader* when you are looking for *'themes/notmyidea/index.html'*

All the other file outside the *templates* dir are static files and will be collected.

For example:

```
themes/
-- notmyidea
    -- default
    |   -- css
    |   |   -- main.css
    |   |   -- reset.css
    |   |   -- styles.css
    |   |   -- typogrify.css
    |   |   -- wide.css
    |   -- images
    |   |   -- icons
    |   |       -- delicious.png
    |   |       -- lastfm.png
    |   |       -- linkedin.png
    |   |       -- rss.png
    |   |       -- twitter.png
    |   -- templates
    |       -- index.html
    |       -- navigation.html
    -- favicon.ico
    -- MANIFEST.json
    -- preview.png

6 directories, 15 files
```

Create a custom theme

Most of the time, you just need to inherit from `templates/themes/html5.html` file:

```
{% load ionyweb_tags i18n %}
{# Minimum HTML5 template #}
<!DOCTYPE html>
{% get_current_language as LANGUAGE_CODE %}
<html lang="{% firstof LANGUAGE_CODE "en" %}">
  <head>
    {% admin_toolbar_medias %}

    {% block head %}
            {% block header_title %}
                    <title>{% title %} - {{ request.website.title }}</title>
            {% endblock %}
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <!-- Le HTML5 shim, for IE6-8 support of HTML5 elements -->
      <!--[if lt IE 9]>
    <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
      <![endif]-->
      {% render_favicon %}
      {% block theme-design-css %}{% endblock %}
      {% block theme-design-js %}{% endblock %}
    {% endblock %}

    {% render_meta_kw %}
    {% render_meta_description %}
    {% render_medias %}
  </head>
```

```
  {% block body-tag %}
  <body>
  {% endblock %}

    {% admin_toolbar %}

    {% block top_body %}{% endblock %}

    {% block body %}

      <!-- Header -->
      {% block header %}
        <header>{% render_layout "header" %}</header>
      {% endblock %}
      <!-- End of Header -->

      <!-- Navigation -->
      {% block navigation %}
        <nav>{% render_navigation %}</nav>
      {% endblock %}
      <!-- End of Navigation -->

      <!-- Content -->
      {% block content %}
        <div id="main-content">
          {% block page %}{% render_page %}{% endblock %}
        </div>
      {% endblock %}
      <!-- End of Content -->

      <!-- Footer -->
      {% block footer %}
        <footer>{% render_layout "footer" %}</footer>
      {% endblock %}
      <!-- End of Footer -->
      {% render_clipboard %}

      {% block render_default %}
        {% render_default %}
      {% endblock %}

    {% endblock %}

    {% googleanalytics %}
    {% block extra-medias %}{% endblock %}

    {% block bottom_body %}{% endblock %}

  </body>

</html>
```

This file is the minimum working Ionyweb theme.

The `ionyweb_tags` templatetags provide quite a few tags:

Load the admin interface

- `{% admin_toolbar_medias %}` that will load the static file for the admin interface

- `{% admin_toolbar %}` that will load the html needed for the admin interface

Load the header

- `{% render_favicon %}` : Load the favicon of the theme.
- `{% render_meta_kw %}` : Render the page meta keywords
- `{% render_meta_description %}` : Render the page meta description

Load the content

- `{% render_medias %}` : Render the medias needed by plugins and page's apps
- `{% render_layout "header" %}` : Render layout that contains plugins that will be displayed on all pages. You can create as many layout as you want by changing its name.
- `{% render_page %}`: Render the content of the page. Plugins specific to this page and the Page App content
- `{% render_navigation %}` : Load the navigation and generate the menu.
- `{% render_clipboard %}` : Display the clipboard that allows you to change a plugin from one page to another.
- `{% render_default %}` : This is the garbage collector for plugins. Since some layout have more placeholders than other, when changing between layouts, some plugins may disapear. They will be displayed here if the placeholder they used to be linked to is not there anymore.
- `{% googleanalytics %}` : Display the google analytics code.

Then there is blocks that you can extends to add some functionnality to your theme.

It is a good idea to extends this file, so that if a modification is needed, you theme will still be compatible with future ionyweb versions.

Using different templates for a theme Most of the time, a theme will be an index.html file then everything else is manage with layouts.

But is certains circumstances, you need more.

Hopefully Ionyweb is flexible.

You can define in your MANIFEST.json a list of template that you will be able to select in the page form:

```
{"title": "Eurochene",
 "description": "Thème pour la Scierie Eurochêne",
 "author": "Agence Esprit Nomade",
 "website": "http://www.agence-esprit-nomade.com/",
 "date": "26/06/2012",
 "preview": "eurochene.png",
 "editable": false,
 "templates": [
     {"title": "Standard", "file": "index.html", "preview": "index.png"},
     {"title": "Home page", "file": "home.html", "preview": "home.png"},
     {"title": "Which Product", "file": "quel-produit.html", "preview": "which-product.png"},
     {"title": "Slideshow History", "file": "story.html", "preview": "story.png"},
     {"title": "Slideshow Metiers", "file": "metiers.html", "preview": "metiers.png"},
     {"title": "Slideshow Certification", "file": "certifications.html", "preview": "certification.pn
     {"title": "Slideshow International", "file": "international.html", "preview": "international.png
     {"title": "Slideshow Chêne", "file": "chene.html", "preview": "chene.png"},
     {"title": "Slideshow Qualités et Normes", "file": "qualites_normes.html", "preview": "qualites.p
     {"title": "Slideshow Nos Références", "file": "references.html", "preview": "references.png"},
```

```
     {"title": "Slideshow Produits innovants", "file": "Produits-innovants.html", "preview": "produit
     {"title": "Slideshow Hêtre", "file": "hetre.html", "preview": "hetre.png"},
     {"title": "Slideshow Approvisionnement", "file": "approvisionnement.html", "preview": "approvisi
     {"title": "Slideshow Feuillus Divers", "file": "feuillus_divers.html", "preview": "feuillus-dive
}
```

The template variable is a list of available templates defined with a title a file and a preview image. The preview should be about 200px high.

Each template file will be used to load the specific page template. Here is the `certification.html` template:

```
{% extends 'themes/eurochene/default/index.html' %}
{% load static from staticfiles %}
{% load i18n %}

{% block slideshow %}
    <img src="{% static 'themes/eurochene/default/images/certifications/eurochene-scierie-certificati
         alt="{% trans "Eurochêne sawmill, Beech Forest French" %}"
         title="{% trans "Eurochêne sawmill, Beech Forest French" %}" />
    <img src="{% static 'themes/eurochene/default/images/certifications/eurochene-scierie-bandeau-for
         alt="{% trans "Eurochêne sawmill, Beech Forest French" %}"
         title="{% trans "Eurochêne sawmill, Beech Forest French" %}" />
{% endblock %}
```

Customize the navigation

One of the tricky thing you want to change each time you create a menu is the navigation.

With ionyweb, the navigation is rendered with `{% render_navigation %}`

The default navigation template `templates/themes/navigation.html` looks like this:

```
{% load mptt_tags %}

<ul>
  {% recursetree menu %}
  <li class="{% if page.lft >= node.lft and page.rght <= node.rght and page.tree_id == node.tree_id %
    <a href="{{ node.get_absolute_url }}">{% firstof node.menu_title node.title %}</a>
    {% if children %}
    <ul class="submenu">
      {{ children }}
    </ul>
    {% endif %}
  </li>
  {% endrecursetree %}
</ul>
```

And it is loaded by the `templates/theme/html5.html` with `{% render_navigation %}`.

If you need to change it, you can create a `themes/YOUR_THEME/default/templates/navigation.html` file which will improve this. As an example, you can create this file:

```
<ul class="nav">
{% for m in menu %}
  {% if m.level == 0 %}
    {% if m.app_page_type.model != 'pageapp_contact' %}
        <li{% if page.lft >= m.lft and page.rght <= m.rght and page.tree_id == m.tree_id %} class="ac
    {% else %}
    </ul>
```

```
    <ul class="contact">
            <li><a href="{{ m.get_absolute_url }}">{% firstof m.menu_title m.title %}</a></li>
    {% endif %}
  {% endif %}
{% endfor %}
</ul>
```

Layouts

Files tree of a layout

The layout works the same way but the MANIFEST.json is much simpler.

```
{"title": "100% - 1 placeholder",
 "preview": "icon-layout.png"
}
```

The preview must show the placeholder organization of the layout.

The layout is defined in the *layout.html* file. Every other file in this directory is considered as a staticfiles and may be access with a browser.

You may access them using */static/layouts/<slug>/file.png*

To access the layout template you can use *layouts/<slug>.html* or even *layouts/<slug>/*.

For example, see the contrib layouts of ionyweb:

```
layouts/
-- 100
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 100_25-25-25-25_100
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 100_33-33-33_100
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 100_50-50_100
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 100_66-33_100_33-66_66-33_33-33-33
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 100_66-33_100_33-66_66-33_50-50
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 33-66
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 50-50_33-33-33_50-50
```

```
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 66-33
|   -- icon-layout.png
|   -- layout.html
|   -- MANIFEST.json
-- 66-33_33-66_66-33_33-66
    -- icon-layout.png
    -- layout.html
    -- MANIFEST.json

10 directories, 30 files
```

Each slug of the list represents the structure of the layout. The char '_' is a new row and the char '-' represents a column of the current row. The values used represent the width of each placeholder, as a percentage of the width of the site.

For example, layout *50-50_33-33-33_50-50* is a layout of three lines, first with two cells of 50% each, second with 3 cells of 33% and last one with two cells of 50% each.

Create customs layouts

To add your customs layouts, create a new dir in your layouts project dir. The name of the dir will be the slug of the layout.

Now, create a *layout.html* file which contains the structure of the layout, i.e. the number of placeholders you want.

Ionyweb contains 10 default structures for layouts.

For example, this is the standard *layout.html* file to create a layout with 5 placeholders:

```
{% extends "layout/5-placeholders.html" %}
```

You can use the default structure *x-placeholders.html* file, where *x* is between 1 and 10. The rendered template looks like this

```
{% extends "layout/base.html" %}
{% load page_extras %}

{% block layout %}

{% render_placeholder "1" %}
{% render_placeholder "2" %}
{% render_placeholder "3" %}
{% render_placeholder "4" %}
{% render_placeholder "5" %}

{% endblock layout %}
```

You can also define a custom structure file on the same schema. You must extend the *layout/base.html* and load the *page_extras* templatetags. Then, overlaod the block *layout* with your own code and use the *render_placeholder* tag to define a placeholder area.

Then, you MUST create the *layout.css* to design the placeholders.

If your design file is empty, each placeholder will be a 100% line of the layout.

For example, this file discribes a layout with a first line of 2 columns (50%-50%), 1 line of 1 column (100%) and 1 third row with 2 placeholders (65%-35%):

```
#placeholder-1 { width: 49%; float: left; }
#placeholder-2 { width: 49%; float: right; }
#placeholder-3 { clear: both; }
#placeholder-4 { width: 64%; float: left; }
#placeholder-5 { width: 34%; float: right; }
#footer { clear: both; }
```

Then you have to create a *MANIFEST.json* file than will give some informations about your layout:

```
{"title": "100% - 1 placeholder",
 "preview": "icon-layout.png"
}
```

By default, the title will be the directory slug of the layout and the preview will load */static/layouts/icon-layout.png*.

Just define the *LAYOUTS_DIRS* in your personnal settings, and now, you can configure your pages with your new layout !

1.1.3 Creating your first app

You have some app defined in the ionyweb code:

- page_blog
- page_redirect
- page_text
- page_agenda
- page_book
- page_gallery_images
- page_sitemap

You can use this apps as an example or directly in your project.

Create the app skeleton

```
$ cd YOUR_PROJECT
$ ionyweb-manage startapp YOUR_APP_NAME
Starting creation of : YOUR_APP_NAME

App dir created.
App Models file created.
App Views created.
App Forms created.
App Urls created.
App Admin created.
App Templates created.
App Locale dir created.


Now just define your models,
Custom the default template : 'index.html',
Add your app to your INSTALLED_APPS : 'page_YOUR_APP_NAME'
Synchronise the database.
 => Your app is fully configured !
```

```
$ python manage.py schemamigration plugin_YOUR_APP_NAME --initial
    $ python manage.py migrate plugin_YOUR_APP_NAME
```

You can now create page of your new type. But it is as empty as possible.

Configure the app

Let's say we want to create a list of music groups.

We will need this kind of data:

- Country with country code
- Music style
- Group informations

Country and Music gender are not related to the page but we want to be able to create different list of group for each pages.

Create the app models

If we open the generated `page_group/models.py` file we have:

```
# -*- coding: utf-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from ionyweb.page.models import AbstractPageApp


class PageApp_Group(AbstractPageApp):

    # Define your fields here

    def __unicode__(self):
        return u'Group #%d' % (self.pk)

    class Meta:
        verbose_name = _(u"Group")
```

This is the minimum to define a Ionyweb Page App.

We will add some other models:

```
class Country(models.Model):
    """A list of countries."""
    code = models.CharField(_(u'code'), max_length=2, primary_key=True,
                            help_text=_(u"See <a href='http://nephi.unice.fr/"
                                        u"codes_iso_pays.php' target='_blank'>"
                                        u"the country code list</a>."))

    name = models.CharField(_(u'name'), max_length=75)

    def __unicode__(self):
        return u'%s' % self.name

    class Meta:
        verbose_name = _(u'Country')
```

```
        verbose_name_plural = _(u"Countries")


class MusicStyle(models.Model):
    """A list of music type."""
    name = models.CharField(_(u'name'), max_length=30, unique=True)

    def __unicode__(self):
        return u'%s' % self.name


class Group(models.Model):
    app = models.ForeignKey(PageApp_Group, related_name="groups")

    music_style = models.ForeignKey(MusicStyle, related_name="groups")
    countries = models.ManyToManyField(Country, related_name="groups")

    code = models.CharField(_(u'code'), max_length=5, help_text=_(u"Exemple C002 ou MA201"))
    photo =  models.CharField(_("photo"), max_length=200, blank=True)

    name = models.CharField(_(u'name'), max_length=100)
    description = models.TextField(_(u'description'), blank=True)

    class Meta:
        ordering = ('code',)

    def __unicode__(self):
        return u"%s : %s" % (self.code, self.name)

    def class_css(self):
        style_class = re.search('^[a-zA-Z]+', self.code)
        return style_class.group(0)
```

Create the app view

Next we want to display the group list on our page.

If we open the generated `page_group/views.py` file we have:

```
# -*- coding: utf-8 -*-

from django.template import RequestContext
from ionyweb.website.rendering.utils import render_view

# from ionyweb.website.rendering.medias import CSSMedia, JSMedia, JSAdminMedia
MEDIAS = (
    # App CSS
    # CSSMedia('page_group.css'),
    # App JS
    # JSMedia('page_group.js'),
    # Actions JSAdmin
    # JSAdminMedia('page_group_actions.js'),
    )

def index_view(request, page_app):
    return render_view('page_group/index.html',
                       { 'object': page_app, },
```

```
                        MEDIAS,
                        context_instance=RequestContext(request))
```

You can provide some medias specific to your app views and to your app administration.

Note: Don't forget the context_instance parameter of the `render_view` if you want the media to be displayed.

The index view is the default. It is defined in the urls.py:

```
# -*- coding: utf-8 -*-

from django.conf.urls import patterns, url
from views import index_view

urlpatterns = patterns('',
                        url(r'^$', index_view),
                        )
```

Lets modify the template `page_group/templates/page_group/index.html` file we have:

```
<p>That the app Group.</p>
```

We will change it for:

```
<h1>My list of groups</h1>
<ul>
    {% for group in object.groups.all %}
    <li>{{ group }}</li>
    {% empty %}
    <li>No groups yet</li>
    {% endfor %}
</ul>
```

Creating the administration

Create the urls

We need to create a `page_group/wa_actions_urls.py` file:

```
# -*- coding: utf-8 -*-
from ionyweb.administration.actions.utils import get_actions_urls

from models import Country, MusicStyle, Group
from forms import CountryForm, MusicStyleForm, GroupForm

urlpatterns = get_actions_urls(Country, form_class=CountryForm)
urlpatterns += get_actions_urls(MusicStyle, form_class=MusicStyleForm)
urlpatterns += get_actions_urls(Group, form_class=GroupForm)
```

We will also create basic forms that we will be able to improve `page_group/forms.py`:

```
# -*- coding: utf-8 -*-
import floppyforms as forms
from ionyweb.forms import ModuloModelForm
from models import PageApp_Group, Country, MusicStyle, Group
```

```
class PageApp_GroupForm(ModuloModelForm):

    class Meta:
        model = PageApp_Group


class CountryForm(ModuloModelForm):
    class Meta:
        model = Country


class MusicStyleForm(ModuloModelForm):
    class Meta:
        model = MusicStyle


class GroupForm(ModuloModelForm):
    class Meta:
        model = Group
```

Create the js UI

Then we will create the js to display the form.

We need to create the former file `page_group/static/admin/js/page_group_actions.js`:

```
admin.page_group = {

    edit_countries: function(relation_id){
    admin.GET({
        url : '/wa/action/' + relation_id + '/country_list/',
    });
    },
    edit_country: function(relation_id, country_pk){
    admin.GET({
        url : 'wa/action/' + relation_id + '/country/' + country_pk + '/',
    });
    },
    edit_music_styles: function(relation_id){
    admin.GET({
        url : '/wa/action/' + relation_id + '/musicstyle_list/',
    });
    },
    edit_music_style: function(relation_id, music_style_pk){
    admin.GET({
        url : 'wa/action/' + relation_id + '/musicstyle/' + music_style_pk + '/',
    });
    },
    edit_groups: function(relation_id){
    admin.GET({
        url : '/wa/action/' + relation_id + '/group_list/',
    });
    },
    edit_group: function(relation_id, group_pk){
    admin.GET({
        url : 'wa/action/' + relation_id + '/group/' + group_pk + '/',
    });
```

```
    },
}
```

In the `page_group/views.py`, we need to activate the AdminJSFile:

```
from ionyweb.website.rendering.medias import CSSMedia, JSMedia, JSAdminMedia
MEDIAS = (
    # App CSS
    # CSSMedia('page_group.css'),
    # App JS
    # JSMedia('page_group.js'),
    # Actions JSAdmin
    JSAdminMedia('page_group_actions.js'),
    )
```

Configure the UI actions

In the models file, we will configure the PageApp actions:

```
class PageApp_Group(AbstractPageApp):

    # Define your fields here

    def __unicode__(self):
        return u'Group #%d' % (self.pk)

    class Meta:
        verbose_name = _(u"Group")

    class ActionsAdmin:
        title = _(u"Group App")
        actions_list = (
            {'title':_(u'Edit countries'),
             'callback': "admin.page_group.edit_countries"},
            {'title':_(u'Edit music styles'),
             'callback': "admin.page_group.edit_music_styles"},
            {'title':_(u'Edit groups'),
             'callback': "admin.page_group.edit_groups"},

            )
```

That's it, now we will be able to add our groups, countries and music styles to the app. Don't hesitate to read the code of the other app to improve the basic UI.

Improve the Group's form

We want to improve the UI.

- In the photo field we want to be able to select an image at the right size from the FileManager.
- The description field should be a HTML edit.

Let change the `page_group/forms.py`:

```
# -*- coding: utf-8 -*-
import floppyforms as forms
```

```
from ionyweb.forms import ModuloModelForm
from ionyweb.file_manager.widgets import FileManagerWidget
from ionyweb.widgets import TinyMCELargeTable

from models import PageApp_Group, Country, MusicStyle, Group

# [...]

class GroupForm(ModuloModelForm):
    class Meta:
        model = Group
        exclude = ('app',)
        widgets = {
            'photo': FileManagerWidget,
            'description': TinyMCELargeTable(attrs={'style': 'width: 100%; height: 300px;', }),
            }
```

Create another view to the page app

Now that we have our list of group, we would like to display another page to see the group's details.

This will be done by using defindig other urls:

```
# -*- coding: utf-8 -*-

from django.conf.urls import patterns, url
from views import index_view, detail_view

urlpatterns = patterns('',
                       url(r'^$', index_view),
                       url(r'^(?P<pk>[\w-]+)/$', detail_view),
                       )
```

Then we will create another view:

```
from django.shortcuts import get_object_or_404

def detail_view(request, page_app, pk):
    obj = get_object_or_404(page_app.groups.get(pk=pk))

    return render_view('page_group/detail.html',
                       { 'object': obj, },
                       MEDIAS,
                       context_instance=RequestContext(request))
```

And another template:

```
<h1>{{ object.name }}</h1>
    {{ object.description|safe }}
```

We will also provide a get_absolute_url to our `Group` object:

```
class Group(models.Model):
    # [ ... ]

    def get_absolute_url(self):
        return u'%sp/%s/' % (self.app.get_absolute_url(),
                             self.pk)
```

So we can change the `index.html` template:

```
<h1>My list of groups</h1>
<ul>
    {% for group in object.groups.all %}
    <li><a href="{{ group.get_absolute_url }}">{{ group }}</a></li>
    {% empty %}
    <li>No groups yet</li>
    {% endfor %}
</ul>
```

That's it !

1.1.4 Creating your first plugin

You have some app defined in the ionyweb code:

- plugin_contact
- plugin_map
- plugin_website_title
- plugin_fb_likebox
- plugin_slideshow
- plugin_blog_entries_list
- plugin_image
- plugin_text
- plugin_breadcrumb
- plugin_links_list
- plugin_video

You can use this plugins as an example or directly in your project.

Create the plugin skeleton

```
$ cd YOUR_PROJECT
$ ionyweb-manage startplugin YOUR_PLUGIN_NAME
Starting creation of : inscription

Plugin dir created.
Plugin Models file created.
Plugin Views created.
Plugin Forms created.
Plugin Admin created.
Plugin Templates created.
Locale dir created.


Now just define your models,
Custom the default template : 'index.html',
Add your plugin to your INSTALLED_APPS : 'plugin_YOUR_PLUGIN_NAME'
Synchronise the database.
 => Your plugin is fully configured !
```

```
$ python manage.py schemamigration plugin_YOUR_PLUGIN_NAME --initial
    $ python manage.py migrate plugin_YOUR_PLUGIN_NAME
```

You can now create your plugin.

Configure the plugin

As an example, we will create an inscription plugin that will take the name and email address of a guest and save it in the database.

Create the plugin model

Let's edit the `plugin_inscription/models.py` file:

```
# -*- coding: utf-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from ionyweb.plugin.models import AbstractPlugin


class Plugin_Inscription(AbstractPlugin):

    # Define your fields here

    def __unicode__(self):
        return u'Inscription #%d' % (self.pk)

    class Meta:
        verbose_name = _(u"Inscription")


class Guest(models.Model):
    name = models.CharField(_('name'), max_length=100)
    email = models.EmailField(_('email'))

    def __unicode__(self):
        return u'%s' % self.name
```

Edit the plugin manifest

Since a plugin is a python package, we can edit its plugin_info in the `__init__.py` file:

```
# -*- coding: utf-8 -*-
from django.utils.translation import ugettext as _

PLUGIN_INFO = {
    'NAME': _(u"Inscription"),
    'CATEGORY': 'socialnetwork',
    'VERSION': "1.0",
    'SHORT_DESCRIPTION': _(u"Inscription form."),
    'DESCRIPTION': _(u"Guest inscription form."),
}
```

You can choice category in the list:

- text
- picture
- audio
- video
- socialnetwork
- ads
- other

By default, it will be other.

Add your plugin

A plugin always has a title that you can decide to hide.

By default the template is `plugin_inscription/templates/plugin_inscription/index.html`:

```
<p>That the plugin Inscription</p>
```

Create the inscription form

Let's defined our form `plugin_inscription/forms.py`:

```
# -*- coding: utf-8 -*-
import floppyforms as forms
from ionyweb.forms import ModuloModelForm
from models import Plugin_Inscription, Guest


class Plugin_InscriptionForm(ModuloModelForm):

    class Meta:
        model = Plugin_Inscription


class Guest(ModuloModelForm):
    class Meta:
        model = Guest
```

Load the form in the view

We modify a little bit the default view, to manage the form:

```
# -*- coding: utf-8 -*-
from django.template import RequestContext
from django.utils.translation import ugettext_lazy as _
from ionyweb.website.rendering.utils import render_view
from forms import GuestForm

# from ionyweb.website.rendering.medias import CSSMedia, JSMedia, JSAdminMedia
MEDIAS = (
    # App CSS
    # CSSMedia('plugin_inscription.css'),
```

```
    # App JS
    # JSMedia('plugin_inscription.js'),
    # Actions JSAdmin
    # JSAdminMedia('plugin_inscription_actions.js'),
    )

def index_view(request, plugin):
    form = GuestForm()
    message = None

    if request.method == "POST" and not request.is_admin_url:
        # Check if we submit this specific form.
        if int(request.POST['inscription_form']) == plugin.pk:
            form = GuestForm(request.POST)
            if form.is_valid():
                form.save()
                message = _(u'Inscription saved')
                form = GuestForm()
            else:
                message = _(u'There is some errors in your form.')

    return render_view('plugin_inscription/index.html',
                       {'object': plugin,
                        'form': form,
                        'message': message},
                       MEDIAS,
                       context_instance=RequestContext(request))
```

We create the template:

```
{% load i18n %}
<div class="iw-plugin-inscription">

  {% include 'themes/plugin_app_title.html' %}

  {% if message %}
  <div class="alert {% if form.errors %}alert-error{% else %}alert-success{% endif %}">
    {{ message }}
  </div>
  {% endif %}

  <form action="" method="post">
    {{ form.as_p }}
    <input type="hidden" name="inscription_form" value="{{ object.pk }}" />
    <div><button type="submit" class="btn">{% trans "Save" %}</button></div>
  </form>

</div>
```

Create the administration

We will create the administration to be able to see our inscription list.

First create the `wa_actions_urls.py` file:

```
# -*- coding: utf-8 -*-
from django.conf.urls import patterns, url
```

```
from ionyweb.administration.actions.utils import get_actions_urls

from models import Guest
from forms import GuestForm

# Generic Action View
urlpatterns = get_actions_urls(Guest,
                               form_class=GuestForm)
```

Then create the `static/admin/js/plugin_inscription_actions.js` file:

```
admin.plugin_inscription = {
    edit_guests : function(relation_id){
            admin.GET({
                    url : '/wa/action/' + relation_id + '/guest_list/',
            });
    },
}
```

Then add the action to the `Plugin_Inscription` class:

```
class Plugin_Inscription(AbstractPlugin):

    # Define your fields here

    def __unicode__(self):
        return u'Inscription #%d' % (self.pk)

    class Meta:
        verbose_name = _(u"Inscription")

    class ActionsAdmin:
        actions_list = (
            {'title':_(u'Edit guests'),
             'callback': "admin.plugin_inscription.edit_guests"},
            )
```

Also load the `JSAdminFile` with the view:

```
from ionyweb.website.rendering.medias import JSAdminMedia
MEDIAS = (
    # Actions JSAdmin
    JSAdminMedia('plugin_inscription_actions.js'),
    )
```

1.1.5 Admin API

Describe informations about the admin API.

Status Code Response

1. **GET** Method :

- *200 - OK* : The resource has been found and the response contains datas.
- *400 - Bad Request* : Indicates a bad request (e.g. wrong parameter).
- *404 - Not Found* : The asked resource does not exist.

2. **PUT** Method :

- *200 - OK* : The new resource has been created and the response contains datas.
- *400 - Bad Request* : Indicates a bad request (e.g. wrong parameter).

3. **POST** Method :

- *200 - OK* : The existing resource has been updated the response contains datas.
- *202 - ACCEPTED* : The existing resource has been updated the page must be reloaded. The json contains the location of the redirection.
- *400 - Bad Request* : Indicates a bad request (e.g. wrong parameter).
- *404 - Not Found* : The asked resource for updating does not exist.

4. **DELETE** Method :

- *200 - OK* : The existing resource has been deleted.
- *400 - Bad Request* : Indicates a bad request (e.g. wrong parameter).
- *404 - Not Found* : The asked resource does not exist.
- *409 - Conflict* : A BDD conflict has been occured during deletion (e.g. resource is used by another resource).

Default Data Response

- **Message** - `msg` :

By default, all responses contains the `msg` parameter which will be displayed in the interface.

The *200* and *400* status displays the message with standard message design (`admin.messages.alert()` method). For *404*, *409* and *500* status, the interface displays the message using standard error message design (`admin.messages.error()` method).

- **Content** - `html` :

By default, only *200* and *400* response contains the `html` parameter.

This parameters contains the new HTML content which will use by the admin to refresh the center panel.

Custom actions

If you want custom the admin actions for a particular status code, just define a new function for this status in your request method.

Be careful about data object in the response :

> If the request is successful (*200* and *400* status codes), the status code functions take a `json` parameter which contains the data returned from the server; if it results in an error (*404*, *409* and *500* status codes) they take `jqXHRan` paramater which have to be transformed with `admin.xhr2json()` method before to be used.
>
> See the jQuery doc for more precisions.

For example, sending a POST request and override the *400* and *500* status codes actions:

```
admin.POST({
    url: '/my/custom/url/',
    data: {'first_data': "Value #1", },
    statusCode: {
```

```
        400: function(json) {
            // Put your actions here...
            admin.messages.alert('My custom action !!');
        },
        500: function(json) {
            // Convert jqXHRan object
            json = admin.xhr2json(json);
            // Put your actions here...
            admin.messages.alert(json.myData);
    }
});
```