**Combo - Blurps - # 2**

# Blurps

Exemple d'affichage attendu:

- la liste des catégories d'un site wcs, choisi parmi une liste
- la liste des catégories d'un ensemble de sites wcs, groupées par site
- la liste de toutes les demandes en cours sur un ensemble de sites wcs, triées par
- les informations d'un compte fédéré d'un système externe (ex: compte lecteur)
- flux RSS ; aggrégation de flux RSS

→ ces demandes, à l'exception de celle pour "un système externe", doivent être natives dans combo, et pas "blurps". (d'après moi aujourd'hui, Frédéric, 3 avril 2015)

# Principe

Un "blurp" :

1. obtient des informations (datas) depuis un ensemble de sources
2. génère un contexte (Django) avec ces datas (en liste, en dico, chaînées, combinées, etc.)
3. envoie le contexte à un template

# Configuration

Idées (Thomas) pour déclarer des COMBO_BLURPS dans le settings :

```
# settings.py

COMBO_BLURPS = [
    {
        'slug': 'slug',
        'template': 'slug.html',
        'title': 'Blurp Title',
        'aggregator': methode sources -> context, # 'list', 'dict', 'chain', ...
        'sources': [
            {
                'slug': 'slug1',
                'title': 'Source Title'
                'url': 'https://wcs/myspace?NameId={{ user.username }}',
                'initial_data': { }, # valeur initiale de la ressource si c'est un dict, pourrait être d'un autre type...?
                'signature': '...?',
                'lazy': true;  # les données sont récupérées aussitôt (cf plus bas)
            },
            {
                'slug': 'slug2', ...
            }
            ],
        'sources_choice': false;  # possibiliter de limiter à un sous-ensemble des sources (false par défaut)
    },

    { blurp2 ... },

]
```

# Génération du contexte

On prend chaque source sélectionnée (ou toutes sinon) → construction d'un contexte

```
    # (tendance pseudo code)
    datas = {}
```

```
    for source in sources:
        data = Data(**source, context)    # le contexte sera utilisé pour calculer l'url
        if not source['lazy']:
            data = data()
            context.update(data.as_dict())
        source['data'] = data
```

Note : les sources "non lazy" sont récupérées aussitôt et permettent d'augmenter le contexte (ainsi, un premier webservice peut donnes des infos aux suivants)

A partir du sources qui contient donc des source['data'], on construit le contexte qui sera envoyé au template, selon le type d'aggregator défini dans le blurp:

- si une seule source dans la définition du blurp, on renvoie data directement, pas d'agrégation, bien sûr
- aggregator list

```
context = [data1, data2, data3]
```

- aggregator dict

```
context = {'slug1': data1, 'slug2': data2, 'slug3': data3}
```

- aggregator chain

```
context = [] + data1 + data2 + data3  # non lazy
```

- d'autres idées ?...

Point à éclaircir : le fonctionnement des initial_data

# Instanciation des Blurps

Une cellule BlurpCell indique :

- le slug d'un settings.COMBO_BLURP
- une liste des slug des sources, si ce blurp l'accepte (sources_choice=true)

(si besoin de plus de paramétres => une classe fille)

# Sur un tenant hobo

On construit le settings.COMBO_BLURPS à partir du hobo.json, c'est donc un TenantSettingsBlurps à écrire.

Exemple de blurps automatisables sur un combo lié à "n" wcs:

- les catégories des wcs (tous, ou un sous-ensemble) / aggregator=list
- les formulaires des wcs (tous, ou un sous-ensemble) / aggregator=list
- les demandes en cours / aggregator=chain  (puis tri et limitation dans le template html)
- les infos d'un user