Analyse de données

http://pandas.pydata.org/

généralisation d'un code de suivi

L'idée.

Chaque formdata possède un attribut tracking_code.

Ce code est disponible sous la variable [form_tracking_code]

Création du tracking_code

Le code de suivi est créé par une action de workflow, typiquement lors de la toute première étape.

Cette action de workflow a des paramètres :
- chaine de caractère "format du code de suivi" = une chaine avec n pour un nombre, l pour une lettre, et on autorise "-" et "." . si rien, on utilise nnnnnnn (7 chiffres). Les chiffres sont 2-9 (jamais de 1 et 0 pouvant être confondus avec I et O). Les lettres sont majuscules, en éliminant O,I et autres lettres qui ressemblent à des nombres.
- case à cocher "effacer l'ancien code si présent" (=False par défaut)
- case à cocher "autoriser l'accès frontoffice même si le formulaire appartient à un autre utilisateur" (=False par défaut) ou "accès aux formulaires anonymes seulement" (=True par défaut)

Utilisation en frontoffice

Un formulaire est présent pour taper un code, on tape le code de suivi et on accède au formulaire comme si on était l'expéditeur.

Si on est loggué, on dispose d'un lien permettant de devenir vraiment l'expéditeur du formulaire (il n'est alors plus anonmye) (="rattacher de formulaire à mon compte")

Utilisation en backoffice

...

Réalisation

Une classe TrackingCode similaire à Token, avec _name = 'trackingcode' et les restrictions sur le format (LLNNLLNNLL). Dans le *init* , on passe la référence vers le formulaire (formdef.id et formdata.id) qui sont placés dans le contenu de l'objet, ainsi que les paramétrages selon l'action de workflow (accès anonyme seulement).

Pour éviter un risque de réutilisation, le code de suivi reste en place même si le formdata est archivé/effacé. Il est cependant marqué "obsolète", et on peut prévoir une date d'expiration via un cron (6 mois ?).

Une URL /trackingcode/?code=xxxxx permet d'obtenir l'accès via le code de suivi. Cette URL est protégée contre une attaque brute-force (throttle). Bien sûr, l'accès n'est validé que si le formdata cible a bien le trackingcode attendu et que le formdata est anonyme (sauf si "accès anonyme seulement" est faux).

...

Django

Selon les installations, le démarrage de wcs "pur" a été bloqué via update-rc.d ou via un exit 0 posé dans /etc/default/wcs.

Avant la mise à jour, faut virer ce "exit 0" et restaurer le paramétrage. → update-rc.d wcs remove && update-rc.d wcs defaults

Passage de wcs-au-quotidien à wcs (utilisateur, répertoires, etc)

Les paquets étaient précédemment prévus pour pouvoir faire tourner plusieurs variantes de wcs en même temps (auquo, pollo), on arrête ça, wcs tourne dans /var/lib/wcs/ et avec wcs:wcs comme uid/gid, auquotidien y est chargé comme une extension.

- psql postgres → alter user "wcs-au-quotidien" rename to wcs;
- mv /var/lib/wcs /var/lib/wcs.old && mv /var/lib/wcs-au-quotidien /var/lib/wcs && find /var/lib/wcs -user wcs-au-quotidien -exec chown wcs:wcs {} \;
- modifier les vhosts nginx
  - remplacer les références à /var/lib/wcs-au-quotidien/ pour les fichiers statiques
  - peut être fait en amont, remplacer :

```
        location /static { alias /var/lib/wcs-au-quotidien/collectstatic/; }
```

```
        location /themes {
            root /;
            try_files /var/lib/wcs-au-quotidien/$host$uri
                      /usr/share/wcs/$uri
                      =404;
        }
```

par :

```
        location ~ ^/static/(.+)$ {
                root /;
                try_files /var/lib/wcs/$host/static/$1
                          /var/lib/wcs/$host/theme/static/$1
                          /var/lib/wcs/collectstatic/$1
                          /var/lib/wcs-au-quotidien/collectstatic/$1
                          =404;
        }
```

```
        location /themes {
            root /;
            try_files /var/lib/wcs/$host$uri
                      /var/lib/wcs-au-quotidien/$host$uri
                      /usr/share/wcs/$uri
                      =404;
        }
```

- remplacer le scgi_pass par un proxy_pass http://unix:/var/run/wcs/wcs.sock;

- faire un hobo_deploy pour mettre en place le lien du thème :
  - sudo -u wcs wcsctl -f /etc/wcs/wcs-au-quotidien.cfg hobo_deploy --redeploy
  - attention, thèmes tiers qui utilisent "overlay" (cas au SICTIAM) → lien supplémentaire nécessaire (de /var/lib/wcs/$tenant/static vers le static de l'overlay.

- ~~fin de l'utilisation de /etc/wcs/wcs-au-quotidien.cfg pour passer à /etc/wcs/wcs.cfg ?~~ (je dis que oui (thomas))
  - c'était mon (fred) intention à un moment mais le wcs-au-quotidien.cfg doit avoir une section [extra] (pour charger auquo pour les commandes "legacy" (dans wcs/ctl), pour ne pas devoir y joindre --extra), et du coup c'est plus facile de garder ce fichier séparé
  - l'idée étant que rapidement les commandes de wcs/ctl/ soient remplacées par des commandes de management normales django, et le wcs(-au-quotidien).cfg totalement retiré

Et aussi :

- des références à wcs-au-quotidien dans le code de hobo (#17984)

Notes particulières aux installation EO pour recette/prod

Références à wcs-au-quotidien dans puppet (obsolètes, #18054)

Sur auquo.entrouvert.org

- cron cg14 à adapter pour passer en user wcs : /etc/cron.d/calvados-remontee-apa
- gestion des bounces (en panne depuis, mais bon) : exim4/conf.d/transport/50_wcs_bounces a beaucoup de référence à wcs-au-quotidien
- partition /var/lib/wcs-au-quotidien-uploads contient les uploads, renommer et refaire les liens de /var/lib/wcs
- références à wcs-au-quotidien dans /etc/munin

Sur auquo.test

- sans doute pareil que la prod

Installation rapide de w.c.s. pour aperçu & tests

Notes pour les gens qui oublient tout trop vite, comme moi, ou pour ceux qui veulent voir à quoi ressemble la version de développement de w.c.s. rapidement...

Installation du virtualenv et des modules nécessaires :

```
virtualenv ve
cd ve
. bin/activate

apt-get install python-dev

pip install http://quixote.ca/releases/Quixote-2.7.tar.gz
pip install scgi

pip install -U git+git://repos.entrouvert.org/wcs.git
```

Lancement du mini serveur HTTP intégré avec :

```
wcsctl.py start --http --port=8080 --data-dir=$PWD/share/wcs/ --app-dir=$PWD/wcs-data
```

Installation des traductions

Installer les fichiers wcs.mo et auquotidien.mo dans ve/share/locale/fr/LC_MESSAGES

Initialisations au premier lancement :

1. aller sur http://localhost:8080/admin/settings/identification et cocher "Simple local username / password"
2. créer un premier compte, qui sera le compte administrateur
3. créer un rôle (par exemple "test")

Mise à jour de wcs :

```
pip install -U git+git://repos.entrouvert.org/wcs.git
```

Ajout de l'extra « Au Quotidien »

```
pip install vobject
pip install -U git+git://repos.entrouvert.org/auquotidien.git
```

Démarrage avec :

```
wcsctl.py start --http --port=8080 --data-dir=$PWD/share/wcs/ --app-dir=$PWD/wcs-data --extra=$PWD
/lib/python*/site-packages/extra/
```

L'apparence de w.c.s. est contrôlée par deux aspects : le squelette et le thème. Le squelette est la structure (« l'ossature ») des pages, le thème en est la présentation. Ils correspondent au couple HTML / CSS.

Squelette

Le squelette du site peut être modifié dans « administration / paramètres / squelette », un bouton « restaurer le squelette par défaut » existe à tout moment pour revenir au squelette de base. Le squelette est une simple page HTML pouvant, en plus du balisage HTML, contenir un certain nombre de variables, écrites entre crochet.

Les différentes variables, qui seront remplacées lors du rendu de la page par leurs valeurs respectives, sont les suivantes :

- title : titre de la page affiché en haut de celle-ci ;
- site_name : nom du site ;
- page_title : le titre reprenant nom du site et titre de la page, a vocation à être affiché dans la barre de titre, via une balise <title> ;
- css : le nom du fichier contenant la feuille de style ;
- script : javascripts utilisés pour certaines fonctionnalités comme le tri des listes ;
- onload : instructions en javascript déclenchées lors d'un évènement. Cette variable a vocation à être placée comme valeur de l'attribut onload dans la balise <body> ;
- breadcrumb : fil d'Ariane ;
- body : contenu principal de la page, habituellement situé entre le titre et le pied de page ;
- lang : langue (code deux lettres) du site ;
- user : nom de l'utilisateur (quand identifié) ;
- root_url : adresse de la racine du site.

Il est possible de tester qu'une variable est remplie en utilisant la syntaxe suivante : [if-any nom-de-la-variable]...[end]. Exemple : [if-any title][title][else][site_name][end] affichera le titre de la page s'il existe, le nom du site sinon.

Thème

Le thème du site peut être modifié dans « administration / paramètres / thème », les thèmes se basent pour la plupart sur la structure du squelette par défaut.

Il est facile, les compétences en HTML et CSS acquises, de créer son propre thème; un thème est un répertoire se composant au minimum de deux fichiers : desc.xml et un fichier CSS. Le fichier desc.xml est un fichier XML comprenant quelques informations de bases sur le thème : son nom, sa version, sa description, son auteur.

<?xml version="1.0"?> <theme name="alto" version="1.0"> <label>Foobar</label> <desc>Thème Foobar</desc> <author>Moi</author> </theme>

Le fichier CSS (qui pour le squelette par défaut doit s'appeler wcs.css) est une feuille de style standard.

Pour illustrer le thème dans la page de sélection, un thème peut également contenir un fichier icon.png, cette image doit avoir une hauteur et une largeur de 30 pixels.

Un thème peut également venir avec son propre squelette par défaut, dans un fichier appelé template.ezt, celui-ci a la structure d'un squelette vue plus haut.

L'installation du thème au niveau système se fait en plaçant le répertoire sous /usr/share/wcs/themes/, il est aussi possible d'installer le thème au niveau d'un site particulier, en plaçant le répertoire sous /var/lib/wcs/**site**/themes/.

Le répertoire du thème peut aussi être placé dans un fichier zip et uploadé depuis la page de sélection de thème.

Annexe sur Au quotidien

Au quotidien ajoute deux variables aux squelettes :

- gauche : liens utiles, étapes du formulaire, autre chose, cette variable contient ce qui se trouve dans la colonne de gauche dans le thème par défaut d'Au quotidien ;
- bigdiv : qualificateur pour le type de page, les valeurs possibles sont profile, new_member, member, info, accessibility, contact, help, rub_service, rub_consultation, rub_annonce et rub_agenda.

Discussion Jabber sur l'implémentation des annonces SMS

- jschneider: alors tout d'abord sur la page : announces/sms; je dois mettre quoi ? un formulaire avec juste une check box
- bdauvergne: je pense simplement un formulaire à un seul champ "Numéro de mobile" et deux boutons "Recevoir les annonces sur mon mobile", "Annuler"
- jschneider: ok c'est ce que j'allez mettre et les annonces à envoyer par sms je les recup ou ?
- bdauvergne: si la soumission du formulaire est bonne ("Recevoir les annonces sur mon mobile" a été cliqué), tu valides le numéro (tu vire points et espaces et tu vérifies que ça matche la regexp ***?33)?6[0-9]{8,8})*** si c'est le cas tu recherche l'object **AnnounceSubscriber** correspondant à l'utilisateur, si tu le trouves pas tu le créés et dans son champ 'sms' tu mets le numéro.
- jschneider: ok merci
- bdauvergne: si le numéro est pas valide selon cette regexp, tu rajoutes un message d'erreur au champ "Numéro de mobile" et tu retourne au code path d'affichage normal du formulaire (ça va réafficher le formulaire avec un message d'erreur expliquant que le numéro a pas le bon format) ensuite dans la classe ***Announce*** (***extra/modules/announces.py***) il faut ajouter une méthode sms (à construire sur le modèle 'email') ensuite encore il faut regarder dans announces_ui.py, où il y a une méthode email appelé par un bouton émis par la méthode ***_q_index*** de la classe AnnounceDirectory, là il faut rajouter un bouton "Envoyer par SMS", qui comme pour la méthode email ajoute un ***afterJob*** qui envoye les messages par SMS -- un_*afterJob*_ c'est un bout de code qui est exécuté après que la réponse à la requête en cours ait été renvoyée, voir encore la méthode "email" de l'objet ***AnnounceDirectory*** -- après on verra à simplifier ça (3 checkbox "RSS", "SMS", "Email" sur la page de creation d'une annonce et un seul bouton "Envoyer" et pas deux bouton "envoyer par mail"/"envoyer par sms") l'affichage du bouton "envoyer par mail" est conditionné par le contenu du champ ***sent_by_email_time*** de la classe ***Announce***, voir si on remplace ce champ par un simple champ ***sent_time*** ou si on ajoute un champ ***sent_by_sms_time*** pour faire la distinction, à toi de voir.

WebServices w.c.s.

Cette page décrit les modes de communication de w.c.s. avec l'extérieur.

Généralités

Protocole général : REST, données en JSON.

Gestion des accès via des API Users et des ACLs (sur les URLs REST).

API w.c.s. Objects

Permet de créer, lire, mettre à jour ou effacer des formulaires (CRUD sur formdata).

- Accès aux formdef en lecture seule : **/api/v1/formdef/X**
- Accès aux formdata :
  - **/api/v1/formdef/X/formdata/Y** : CRUD
  - **/api/v1/formdef/X/formdata/Y/data** uniquement sur les données, read/update
  - **/api/v1/formdef/X/formdata/Y/wf** sur le workflow : read/update. En read on obtient le statut actuel et les suites possibles (dont les données attendues) ; en write on envoie les éventuelles données demandées pour un changement de statut choisi
  - **/api/v1/regie/<régie>/invoice/Y** : CRUD

Cas d'usage

- Orléans: connecteur facturation
- Noyelles-Godault: externalistion du connecteur Abelium compte famille et facturation

API w.c.s. Form (réponse à distance)

Permet de remplir un formulaire pas à pas, en suivant éventuellement les pages et en obtenant une réponse à la fin (première étape du workflow)

> POST sur **/api/v1/formdef/X/page/Y**

Retourne des codes 200, 300, 400, 500... selon ce qui doit se passer pour continuer ou reprendre la saisie, selon le cas.

API w.c.s. DataSources

Utilisé par w.c.s. :

- pour renseigner les valeurs possibles d'un select
- pour le préremplissage d'un champ en fonction d'autres données

w.c.s. interroge la source distance avec une requête de type "JSONSQL" (à trouver ou à inventer)
Du genre :

```
{
  "columns": [ "a", "b", "c"],
  "where": ['or', ['a', '<', 1], ['b', '=', 'xyz']],
  "offset": 5,
  "limit": 2
}
```

Retourne un JSON :

```
[
 { "a": ..., "b": ..., "c": ... },
```

```
 { "a": ..., "b": ..., "c": ... }
]
```

Cas d'usages

- CG14 recherche d'homonymie
- Montpellier & CG14 complétion des adresses

API w.c.s. Notify

Utilisé par w.c.s. pour envoyer des messages. Peut recevoir une réponse en synchrone ou asynchrone (via une URL de retour).

Requête :

```
{ "data": ...,
  "replyto": url
}
```

Réponse attendue : (selon l'URL de replyto ?)

```
{ "data": ...,
}
```

Cas d'usage

- Vincennes: faire interagir les workflows W.C.S. avec autre chose genre K2

w.c.s.

w.c.s. is a web application which allows to design and set up online forms. It gives a user the ability to create web forms easily without requiring any other skill than familiarity with web surfing.

See: http://wcs.labs.libre-entreprise.org/

- Personnalisation de l'apparence * SMS

Fichiers

admin-wcs.png	13,3 ko	21 mars 2012	Pierre Cros

Wcs & gunicorn

Extrait du ticket #2522

Juste pour mémoire, ça marche avec gunicorn et le fichier wsgi.py suivant:

```
> import quixote.wsgi
> from .publisher import WcsPublisher
> from wcs.qommon.http_request import HTTPRequest
>
> class QWIP(quixote.wsgi.QWIP):
> request_class = HTTPRequest
>
> application = QWIP(WcsPublisher.create_publisher())
>
```

Ça ne marche pas avec les conteneurs WSGI qui définissent la variable d'environnement wsgi 'wsgi.multithread' (i.e. apache en mode worker ou uwsgi). Il y a plein d'autres problèmes (on a plus la main sur tout ce qui se configure en ligne de commande, --extra, --app-dir, --data-dir). C'est juste un POC.