HowDoWeDoTests

- pytest, pytest-django et webtest: OK pour tout le monde
- tests/ à la racine, plutôt que distribués dans les différents répertoires des applications django (ça semble être une tendance). : NOK en django les tests vont dans les apps quand il y en a, OK pour les projets
- en Django prévoir un fichier settings de test pour les applications django (pour les projets le fichier settings normal suffit), ajouter une cible "test" au fichier setup.py qui appelle "django-admin test" ou pytest en installant le fichier de config
- utiliser tox pour tester dans différents environnements (django 1.6, 1.7, etc.. python 2.6, 2.7, 3.2, etc.., avec différentes lib python si il y a des options ou si le paquets est censé fonctionner avec ou sans une certaine dépendance) et plus généralement pour sa capacité à gérer tout seul la création d'un virtualenv adapté ce qui permet de détecter les soucis de packaging (dépendance manquante ou dont la dernière version n'est plus compatible)

Ressources

- mock/requests: http://engineroom.trackmaven.com/blog/real-life-mocking/
- pytest + webtest + Django : http://mathieu.agopian.info/presentations/2013_09_djangocong/

Tests manuels

- Pour tester un callback HTTP ou simplement un appel: http://requestbin.com/ (API carte-bancaire par exemple)
- Pour tester la réaction à des réponses HTTP bizarres: http://httpbin.org/
- Pour tester l'envoi de mail sans pourrir sa boite: https://one-time.email/http://getairmail.com/ (pas pour des trucs dangereux hein ! genre récupérer son compte sur un site en prod :))

Pense bête py.test&Django

Une fois qu'on a tox de configuré pour lancer les tests py.test on peut faire pas mal de choses amusantes, je donne mes exemples dans le contexte du projet authentic2.

- lancer les tests dans un environnement donné (parce qu'on a pas besoin de tout tester là maintenant, ça c'est pour jenkins)

```
$ tox -e dj18-authentic-sqlite
```

- lancer les tests en conservant la base de donnée (ça permet de passer l'étape laborieuse des migrations)

```
$ tox -e dj18-authentic-sqlite -- tests --reuse-db
```

- s'arrêter avec pdb dans le dernier test qui a planté (--lf pour lastfail et --pdb pour ouvrir pdb sur une exception)

```
$ tox -e dj18-authentic-sqlite -- tests --lf --reuse-db --pdb
```

- ne lancer qu'un test en particulier (on peut passer à l'option -k n'importe quelle sous chaîne pertinente du nom du ou des fonctions de test)

```
$ tox -e dj18-authentic-sqlite -- tests -k test_moncul --reuse-db --pdb
```

- recréer la base de test (parce qu'on a ajouté une migration)

```
$ tox -e dj18-authentic-sqlite -- tests --reuse-db --create-db
```

Utiliser un debugger alternatif

Par défaut, utiliser l'option --pdb ou taper import pdb; pdb.set_trace() dans le code ouvrira l'interpréteur pdb de base.
Il est possible d'utiliser un autre interpréteur, par exemple IPython.

L'environnement installé par tox n'inclut pas IPython. Il faut donc injecter globalement la dépendance via un plugin : pip install tox-ipdb-plugin.

Ensuite, il faut dire à pytest d'utiliser cet interpréteur pour pdb. Là encore un peu de gymnastique pour que la configuration soit globale, il faut ajouter deux variables d'environnement dans son .bashrc :

```
export PYTEST_ADDOPTS='--pdbcls=IPython.terminal.debugger:TerminalPdb'
export TOX_TESTENV_PASSENV='PYTEST_ADDOPTS'
```

Voilà, vive l'autocomplétion.