

# Retour sur le développement d'un site à gros volume de données

*Par Stéphane « Twidi » Angel*

*Djangocong 2012*

# Intro : me, my & myself

Dev web depuis 14 ans

- Perl (trop longtemps)
- Php (malheureusement)
- Python (c'est un serpent)
  
- + frontend
- + admin site (vite fait)

*[@twidi](#) sur Twitter (et un peu partout sur internet en fait)*

# Intro : me, my & myself

Passé notamment par :

- Gandi
- Libération

+ nombreux projets perso dont...

# REPOS.IO

« Search and manage code repositories »

- Recherche sur plusieurs plateformes (providers) : github, bitbucket, google code...
- Organise tout ce qui est *favori / follow / watch*
- Pour chaque utilisateur :
  - **Flags** (favoris, « check later »)
  - **Tags** (simple, « places » & « projects »)
  - **Notes descriptives**
- **Flux/notifications** (plus tard)

# Besoins

- Recherche rapide (car à la base du fonctionnement du site)
- Tagging automatique (aide à la recherche)
- Nombre croissant de providers
- Mise à jour (très) fréquente des données
- Automatisation (workers, scaling...)

# Problématique(s)

- Multiple providers
- Beaucoup de données
  - Github : 2.000.000 projets
  - Bitbucket : 90.000
  - Google code : 260.000
  - Sourceforge : 450.000
  - Pypi : 20.000
- Certains objets mis à jour très souvent
- Différentes API (voire pas du tout, cf Google)

# Résolution

- Mes solutions (ou tentatives) pour :
  - Stockage
  - Récupération des données
  - Recherche
  - Optimisations frontend

# Stockage

- 2 modèles principaux : Repository & Account
  - Un modèle abstrait commun
  - PostgreSQL
- Backends à la « django-social-auth »
  - Base commune (interface, pour normaliser les appels) + un par provider
- Identifiants de projets
  - « user/repository » sur Github
  - « repository » sur Google code
  - Résolution
    - Pas de « unique » au niveau de la DB
    - Mais au niveau du code : backend + « identifiants »
    - Identifiants => tuple en fonction du backend (sert aussi pour les urls)
- Redis (compteurs « xx a yy followers »)



# Récupération des données

- Asynchrone (of course)
- Pas de celery (machinerie trop lourde)
- Redis (listes bloquantes)
- Un script (python) par type de worker :
  - Chaque worker lié à une liste Redis (ou plusieurs pour gérer la priorité)
  - Récupération chez le provider
  - Mise à jour moteur de recherche
  - Mise à jour des compteurs (dénormalisation, stockés dans redis)
  - + un worker « management » qui gère le nombre de workers en fonction des besoins (et le nombre de serveurs si vraiment besoin)
- Le tout via supervisor

# Workflow

- Un utilisateur se connecte pour la première fois
  - => On récupère ses « objets » : repositories, followers... récursivement sur 3 niveaux
- Il revient, pas de mise à jour depuis 2 jours
  - => On recommence, sur 2 niveaux
- Un objet est accédé sur le front, sans mise à jour depuis un certain temps (calculé via date de dernière mise à jour chez le provider, et plus globalement un « score » interne)
  - => On le met à jour
- Possibilité de demander une mise à jour depuis le front

Le tout de façon asynchrone bien évidemment, et avec gestion de priorité (les objets d'un nouvel arrivant seront récupérés en priorité par rapport à une demande sur le front, elle même devant une simple mise à jour automatique)

# La recherche

- Recherche « full text »
- Solr + haystack
- Gestion manuelle (via worker) de la mise à jour des entrées dans Solr lors de la modification d'un objet
- Récupération des IDs uniquement
  - Le template de chaque objet est caché via django-adv-cache-tag et retrouvé via l'ID
- Problème pour filtrer parmi, par exemple, les repositories d'un user (pas encore de solution, sauf l'équivalent d'un « in », lourd)

# Optimisations frontend

- Cache ORM
  - johnny-cache non pertinent (écritures permanentes)
  - django-cache-machine... ou rien, trop de requêtes différentes, mais simples
- Templates
  - Gros fan de « extends » et « include »
  - Moteur de django très lent (clairement le bottleneck)
    - django-template-preprocessor
    - django-adv-cache-tag
    - django-include-strip-tag (pour s'en sortir avec les newlines)

# A propos de Repos.io

- Nouvelle version alpha en ligne depuis quelques jours
- Design « responsive » via css (à l'origine composé de tableaux et de twitter bootstrap)
- Side project depuis Août (SO LONG!)
- Hébergé sur [dotcloud](#)
- Code source dispo sur [github.com/twidi/Repos.io](https://github.com/twidi/Repos.io)
- À venir :
  - Flux/notification de mise à jour des projets
  - « more like this »
  - « explore »
- (et plein de choses super intéressantes à dire sur le frontend, vraiment : html5, history API, css3 à fond, js-or-not...)


*PS : si quelqu'un est intéressé par l'aventure...*

· Full search · Regroup by project & place · Add private tags & notes · Discover stuff · More soon ! ·


Search on **GitHub** and soon also on Bitbucket, Google code and more repositories **people**

twidi 🔍


Return to the list



TWIDI



**REPOS.IO** 2012-02-20  
25

by  TWIDI @ GitHub

The idea of this project came to me when i searched in my **GitHub** watched projects to find something...  
The problem is that Github just give us a list. No description, no way to organize it. Useless.

io github provider site source +

About **Readme** Owner Followers **46** Contributors **1**