Aabu

Geste de savoir

Préparer l'avenir en découplant les modules

19 avril 2022

Table des matières

Introduction 1
1. Aperçu de l'architecture du backend 1
2. Le problème des notifications 2
2.1. Architecture du système de notification 2
2.2. Quand les soucis surviennent 3
3. Solution du problème et perspectives 3
Conclusion 4

Introduction

Zeste de Savoir est un projet au long cours, avec presque 8 ans depuis la sortie officielle du site, encore plus depuis le début du développement, et plus encore quand on considère que la base de code est issue d'un projet antérieur. Avec le temps et la succession des développeurs, il n'est pas rare de voir apparaître quelques signes d'obsolescence, qu'il est de bon ton de corriger au fur et à mesure pour maintenir le code dans un état le plus optimal possible, sur la durée.

Cette série de billet intitulée Archéocode vise à montrer comment le code de Zeste de Savoir tente de garder une certaine jeunesse, loin des nouvelles fonctionnalités visibles pour les utilisateurs.

Dans ce cinquième billet, je raconte comment des modules qui se sont retrouvés couplés ont été de nouveau découplés, ce qui permet de gagner en flexibilité, en facilité de test et prépare ainsi l'avenir.

1. Aperçu de l'architecture du backend

Le *backend* d'un site en Django peut être conçu soit de manière monolithique, ou tout est vu comme un seul grand ensemble formant le site web, soit de manière modulaire sous forme d'*applications* Django, qui sont des gros modules relativement indépendants les uns des autres et essentiellement auto-contenus.

Le code de Zeste de Savoir est structuré sous formes d'applications interdépendantes. On peut citer par exemple les applications suivantes:

— la gestion des membres,
— la gestion des galeries d'images,
— les messages privés,
— le forum,
— les notifications,

— et quelques autres.

Ces applications dépendent les unes des autres de diverses manières. Par exemple, le forum et les messages privés ont besoin de la gestion des membres, ou encore les notifications interagissent avec les forums et les MP.

Le plus souvent, il s'agit de dépendances assez légères. Par exemple, un message de forum a un auteur et donc a besoin de connaître le module de membres, mais on peut aussi avoir des dépendances plus fortes. Un exemple de couplage assez fort est celui du module de notification, du module de forum et de celui de MP : le module de notification doit être informé quand il se passe des événements liés aux MP et forums et les deux modules ont été conçus de manière rapprochée.

2. Le problème des notifications

Le module de notification était (initialement) très couplé avec le code des MP et celui du forum et dans une moindre mesure, celui des tutos. Ce couplage ne l'empêchait pas de marcher, mais est très handicapant pour le faire évoluer.

2.1. Architecture du système de notification

Les fonctionnalités du module de notifications sont conceptuellement simples à décrire: il s'agit pour le module de notifications de réagir à différents événements en provenance des modules produisant des notifications:

— créer un abonnement (qui émettra ensuite des notifications) lorsqu'un membre s'abonne à un forum, un tag, etc. ;
— supprimer cet abonnement si nécessaire (suppression du contenu notamment) ;
— émettre les notifications correspondant aux abonnements (nouveaux commentaires, messages de forum, MP, sujets, etc).

Pour limiter les imports croisés de module Python (ce qui peut conduire à des imports cycliques et donc une erreur), le module de notification communique avec les autres modules grâce à des *signaux*. Le principe des signaux est qu'un module qui veut notifier quelque chose émet un signal et un module qui veut agir à ce moment va l'écouter. Le système de signaux de Django s'occupe ensuite en sous main de faire passer le message entre les émetteurs et les récepteurs. C'est un système en mode *publish/subscribe* assez classique.

Là où le problème intervient, c'est que la définition des signaux est faite *dans le module de notification*. Pour émettre ses signaux, le module de forum, par exemple, va importer la définition des signaux qui l'intéresse. De l'autre côté, tous les récepteurs sont définis dans le module de notifications, et un certain nombre ont besoin de connaître les modèles émetteurs pour faire leur job correctement. On a donc des imports croisés entre *apps*.

2.2. Quand les soucis surviennent

La plupart du temps, ça fonctionne sans imports cycliques, parce que nos modules de plus haut niveau ne sont pas importés directement, mais seulement ceux des niveaux inférieurs. On importe `zds.notifications.signals` d'un côté et pas `zds.notifications` et de l'autre côté, on importe par exemple `zds.mp.models` et pas `zds.mp`.

Le problème, c'est que c'est aussi très dur à tester proprement en l'état. Quand un module de test cherche à importer ce qu'il faut des deux côtés… badaboum, import cyclique potentiel. Le module de notifications est d'ailleurs très fragile: on peut casser quelque chose par accident [1]footnote:1, et les tests ne cassent pas à tous les coups, y compris quand les notifications sont effectivement cassées. Tests manuels obligatoires. J'ai essayé de corriger de manière *ad hoc* à un moment, mais sans succès.

Ça pose aussi des soucis lorsque le système de signaux de Django découvre automatiquement les signaux et récepteurs à l'initialisation du backend. On a par exemple un import à la toute fin d'un fichier qui s'il est mis ailleurs casse le code, de mémoire sur un import cyclique:

```
1 # used to fix Django 1.9 Warning
2 # https://github.com/zestedesavoir/zds-site/issues/3451
3 from . import receivers  # noqa
```

On remarquera au passage le fait que ça corrigeait initialement un *warning* d'une version de Django qu'on n'utilise plus. Une alternative à cet import serait d'enregistrer explicitement les signaux dans l'initialisation de l'*app*, mais c'est laborieux et le système de découverte automatique est censé justement éviter de devoir faire ça.

Toutes ces choses font qu'il est très pénible, voire impossible, de faire évoluer le système de notifications, notamment pour introduire des notifications des événements sur les contenus (validation par exemple), pour éviter d'utiliser abusivement le système de MP à cette fin et possiblement notifier plus de petits événements (changement de tags, par exemple).

3. Solution du problème et perspectives

La solution adoptée est une redéfinition plus propre des interfaces en définissant les signaux au bon endroit.

Tout d'abord, ce sont *les modules qui émettent des signaux qui les définissent*. En effet, qui mieux que le module lui-même pour savoir quels événements valent la peine d'être transmis (avec évidemment une prise en compte des besoins de notifications)?

Une fois ceci fait, on a donc un module avec des flux d'événements qu'on peut écouter si on le souhaite ou pas. Le module n'a pas même pas besoin de savoir si une autre partie du code s'en sert, c'est totalement invisible pour lui.

1. [2]footnote:1 Notamment, en touchant au code des MP qui a ses propres curiosités, j'ai remarqué que l'ordre de lignes en apparence indépendantes fait qu'on ne reçoit plus certaines notifications…

C'est aussi très facile à tester, grâce aux outils de tests habituels. On peut vérifier qu'un signal a bien été émis et combien de fois. Par exemple, si un formulaire de forum est envoyé correctement, on doit compter une seule émission du signal "nouveau message dans tel sujet". Mais si le formulaire contient une erreur, on doit constater qu'il n'y a pas d'émission. Pas besoin de s'intéresser aux notifications à ce stade.

De l'autre côté, le module de notifications ne change pas tant que ça. Il perd les définitions transférées dans les autres modules, mais garde toute la responsabilité de gérer les abonnements et d'émettre des notifications aux moments importuns. On peut facilement rajouter des récepteurs pour de nouveaux événements existants sans devoir changer le code qui émet l'événement (ce qui n'était pas toujours possible avant).

Les modifications dans le code ne sont même pas compliquées! Pour les curieux, voilà les PR correspondantes: #5982 , #5971 et #5976 .

J'ai déjà un potentiel usage futur en tête: avoir des notifications pour la deuxième itération du futur journal d'événements des publications , qui utilise déjà des signaux et qui pourront donc être notifiés très facilement (ou presque, il y a quand même du boulot étant donné la fragilité des notifications)!

Conclusion

Voilà, de la poussière en moins sous le tapis! Dans le cas présenté dans ce billet, on peut retenir qu'il existe des techniques pour faire communiquer des modules entre eux de manière très flexible (bien plus flexible que les appels de fonctions), sans trop rajouter de complexité et tout en restant testable de manière unitaire.

À bientôt, je l'espère, pour parler d'une autre vieillerie découverte dans le code de Zeste de Savoir et du coup de ménage associé!

Liens utiles

— Dépôt Git de Zeste de Savoir , sur Github.
— Tous les billets de la série Archéocode .