ARTRAGIS
FIRM1

[chronique]Zest Of Dev 8

15 janvier 2019

Table des matières

1. Mise à jour, feuille de route

Nous vous l'avions annoncé dans le dernier zest of dev , les mises à jour du site seront bien plus fréquentes. Nous avons donc appliqué notre promesse :

— la v27.2 est en production, elle corrige quelques bugs graphiques et ajoute une favicon spéciale lorsque vous avez une nouvelle notification ;
— un nombre assez important de modifications est testable sur la beta , notamment une mise à jour de django vers la dernière LTS (django 1.11). Plusieurs améliorations sont aussi à suivre, vous avez la liste sur cet historique .

i Appel à contribution

Je profite de ce billet pour remercier toutes les personnes qui ont fait la QA (c'est à dire les tests) des PR ces derniers temps mais aussi les personnes qui sont passé sur la béta pour s'assure que tout allait bien.
N'hésitez pas à y aller, et à remonter d'éventuels bugs dans les commentaires de ce billet, c'est le meilleur moyen pour que vous ayez un site agréable à utiliser !
J'aimerais aussi savoir si parmi nos lecteurs il y avait des personnes qui sont utilisateurs de navigateurs "oraux", c'est-à-dire qui vous lisent le site plutôt que les navigateurs grand public pour les voyants. Avoir votre retour serait utile pour favoriser l'accessibilité du site !

Durant le zest-meeting, le besoin d'améliorer la documentation a été clairement établi. Notamment, améliorer la feuille de route a été requis. J'ai donc complété cette page et j'attends vos retours pour l'améliorer.

2. Projecteur sur l'API

Zeste de Savoir est une application web qui fonctionne sur le modèle du "rendu côté serveur", c'est-à-dire que la page que vous voyez quand vous naviguez, c'est le serveur qui l'a générée pour vous.

Cette méthode n'est pour autant pas suffisante. Zeste de Savoir gère beaucoup de données qui peuvent être présentées autrement, qui peuvent avoir d'autres utilités que celles qu'on leur donne à l'heure actuelle.

Il y a plusieurs années maintenant, @Andr0 a donc proposé de faire de Zeste de Savoir, un service en plus d'une simple application web. Et pour cela il faut passer par une **API**.

> API, ça signifie *Interface de Programmation d'Application* (Application Programming Interface). Voyez ça comme un contrat signé entre des développeurs : d'un côté on a les développeurs du *service* qui disent "on a cette donnée là, avec telle structure" et de l'autre les développeurs d'une application (le zds-notificateur par exemple) qui disent "moi j'ai besoin de cette donnée-là". L'API c'est simplement la *méthode de communication* que les développeurs du service vont utiliser pour que les développeurs de l'application obtiennent les données qui les intéressent.
> En soi les langages de programmation sont des API : les développeurs qui créent le langage vous disent "pour faire telle chose, il faut utiliser telle fonction, telle structure et vous obtiendrez tel résultat".

Lorsque le service est un service *web* comme zeste de savoir, les API ont tendance à suivre un certains ensembles de normes ou de protocoles. Pour zds, on a choisi REST pour concevoir l'API et OAUTH pour sécuriser l'authentification.

invitation

Je vous invite à lire les différents articles à ce propos qu'on trouve sur zds, d'ailleurs :
- Comprendre OAuth 2.0 par l'exemple
- La théorie REST, RESTful et HATEOAS
- Pour ceux qui aiment le PHP : Créez votre API REST avec Symfony 3

Pour zeste de savoir, notre API se développe petit à petit au fur et à mesure de nos besoin et de ceux des applications qui existent déjà.

Nous avons donc :

- une API pour les membres (les lister, avoir quelques informations...)
- une API pour les messages privés (lister, répondre, créer...)
- une API pour voter sur les messages
- une API pour les notifications
- une API pour les tags

Et trois gros morceaux sont en cours de développement :

— une API pour les galleries, qui permet à la fois de gérer ces dernières et surtout d'envoyer des images, c'est un prérequis pour l'amélioration de l'upload d'image quand on écrit un message.
— une API pour les contenus : les créer, les éditer, les lister.
— une API pour les forums.

Au fur et à mesure que nous créer ces API, la documentation de cette dernière s'affiche dans la page dédiée [1].

Cela peut permettre à ceux que cela intéressent de faire des petites applications qui utiliseront la connaissance partagée sur zds pour aider le monde !

3. Témoignage de Firm1 : Comment j'ai QA la PR de pierre-24 sur l'API des images ?

Je ne suis pas un habitué des témoignages, mais je me suis dit, qu'un retour d'expérience d'une QA pouvait intéresser certains.

3.1. Quelques mots sur la QA

Tout d'abord commençons par briser des mythes pour certains, ou enfonçons des portes ouvertes pour d'autres : **faire de la QA revient juste à tester que le code qu'un contributeur demande d'ajouter fait bien ce qu'il prétend faire, sans casser ce qui marche déjà.**

Dans le processus de développement de ZdS, les choses s'articulent ainsi :

Contenu masqué n°1

Quand je fais de la QA, j'ai donc deux volets en tête :

1. Le bug a bien été corrigé ? La fonctionnalité marche comme voulu ?
2. Aucune régression ne se cache dans ce nouveau code ?

C'est dans cet état d'esprit que j'ai décidé de me lancer dans la QA de la proposition de pierre-24 sur l'API des images.

1. Pour voir l'ensemble de l'API il faut être connectée, en effet le logiciel qui affiche la documentation swagger s'assure que vous avez les autorisations avant de tout vous montrer.

3.2. La découverte de LaPéAir

Une PR peut prendre plus ou moins de temps, selon sa taille et selon les choses à tester. En ce qui me concerne, quand je vois arriver une PR à tester, je vais déjà voir (en lisant la description de la PR) si je peux la tester rapidement ou s'il va me falloir beaucoup de temps.

Voici a quoi ressemble l'exercice quand je me rends sur la description de la PR :

3.2.1. Le titre

> Création de l'API des images

Le contenu de la PR a été bien annoncé, je peux imaginer ce qu'il y a dedans.

3.2.2. La description

> Puisqu'il s'agit d'un prérequis à l'existence de *drag and drop*, je m'attaque à ça. La spécification suivie sera (est ?) la spécification de (feu) la ZP-45 . On verra si ça tient la route

Je comprend que cette PR est un prérequis à une autre fonctionnalité qui est très attendue, donc ça augmente un peu son niveau d'urgence. Par contre, je ne suis pas rassuré quand je vois que la spec sur laquelle se base l'API date de 2016 :euh :, d'autant plus que l'auteur de la PR fini sa description par "On verra si ça tient la route" . On a connu plus rassurant comme message ^^.

3.2.3. Les instructions du contrôle qualité (QA)

> — `make fixtures` (j'ai rajouté une *fixture* pour créer une application) ;
> — *to be continued.*

Là aussi on comprend qu'il faut charger des données de tests avant de tester et que l'auteur de la PR a pris la peine de rajouter ces données pour faciliter le travail du testeur. Il a l'air cool et ça rassure un peu.

3.2.4. Les stats de la PR

> — Nombres de fichiers modifiés par la PR : 
>
> — Différence entre les lignes ajoutés et les lignes supprimées : 
>

Oula ! On a affaire à un gros morceau. On ajoute une grosse fonctionnalité ici, il peut y avoir des régressions, donc il faut redoubler de vigilance et tester les choses bien comme il faut.

3.2.5. Les tests automatiques

On constate que l'auteur de la PR a rajouté beaucoup de tests dans sa PR, ce qui est un bon, signe. Et travis (notre outil d'intégration continue qui déroule les tests automatiques) nous signale que les tests passent bien.

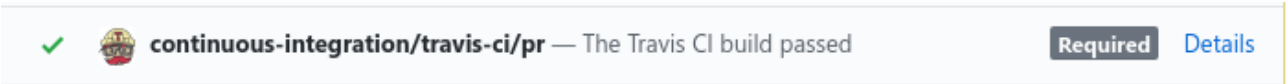

FIGURE 3. – Les tests automatiques sont OK pour travis

Bon point aussi pour la PR, mais rien ne vaut les tests d'une vrai humain.

> En gros, en découvrant la PR, je constate que ça va être un peu long a tester tout ça, mais j'ai le sentiment au vu des premiers indicateurs que l'on aura pas beaucoup a redire sur le code, et si jamais, l'auteur de la PR semble réactif.

Je préviens l'auteur que je compte tester sa PR et je m'organise.

3.3. Ecrire mes scénarri de tests

> En général je me lance directement sur les tests, sans écrire les scénarii avant. Mais de ce que j'ai compris, on a affaire ici à une grosse PR, donc il vaut mieux préparer les tests, ne serait-ce que pour être capable de les rejouer.

Pour s'assurer de couvrir tout les cas lors de mes tests, je dois au préalable définir mon mini plan de test. Avec tout les scénarii qui me semblent utiles.

On parle de tester une API, donc ça signifie essentiellement qu'on va tester les nouvelles urls (ce qu'on appelle les routes) rajoutées.

A grosse maille, mon plan de tests ressemble donc a ça :

— Tester l'ajout, modification, consultation et suppression, d'une galerie d'images
— Tester l'ajout, modification, consultation et suppression, d'une image dans une galerie

Penser à bien tester les cas tordus, du genre : quand une galerie est vide, quand une galerie est remplie, les différents formats d'images (png, jpeg, svg, archive, etc.) ou encore si une galerie est liée a un contenu.

Une fois mes scénarii de test plus ou moins imaginés, j'ouvre un éditeur de texte et je commence a écrire mes requêtes de tests à l'API[2].

Pour savoir comment utiliser l'API de ZdS, je me base essentiellement sur la documentation du projet , ainsi que sur les nouvelles routes de l'API accessible depuis la documentation . Comme un utilisateur normale le ferait en temps normal.

3. Témoignage de Firm1 : Comment j'ai QA la PR de pierre-24 sur l'API des images ?

Concrètement, mes notes ressemblent à la liste de commandes ci-dessous :

> Contenu masqué n°2

3.4. Tester réellement

Maintenant que le travail de préparation est réalisé, il ne reste plus qu'à ouvrir la PR en question sur mon poste en local.

Je développe avec Pycharm, ça m'évite de devoir retenir des commandes git par coeur.

Une fois le projet zds-site installé sur mon poste, il me suffit de faire un *checkout* de la branche de pierre-24 (l'auteur de la PR), et de charger les nouvelles *fixtures* comme demandées.

```
## j'ajoute le dépôt distant de pierre-24 dans la liste de mes dépôts distants
git remote add pierre-24 https://github.com/pierre-24/zds-site
## je récupère en local les informations sur le dépot que je vient de rajouter
git fetch all
## je passe sur de l'API des images
git checkout -b api_images pierre-24/api_images
## je charge les nouvelles fixtures
python manage.py loaddata fixtures/*.yaml
## je démarre zds
python manage.py runserver
```

Le reste de l'opération consiste a dérouler mes cas de tests, et voir si le résultat est concluant.

J'ai fais ça en plusieurs fois (ce qui n'a été possible que parce que mes cas de tests étaient écrit à l'avance).

— Première partie : mon rapport de QA
— Deuxième partie : mon rapport de QA
— Troisième partie : mon rapport de QA

Au bout du compte, on a donc plusieurs problèmes sur cette PR. Sur 44 cas testés, j'ai 8 cas d'erreurs. Des erreurs 500 que renvoient l'API, ou encore des erreurs 403 qui ne devraient pas être ainsi. Il ne reste plus qu'à attendre patiement que pierre-24 corrige ces points et on reviendra dessus un peu plus tard pour rejouer nos cas de test.

EDIT : il a déjà fait les corrections (fix1 , fix2 , fix3) qui s'imposent (qu'elle efficacité).

2. J'utilise curl pour mes tests

Conclusion

Voilà un peu comment je m'y prend pour faire de la QA. Comme vous l'avez vu, a aucun moment j'ai eu besoin de savoir faire du Python/Django. Je me contente de tester que les modifications font bien ce qu'elles sont censées faire.
Cette QA étaient particulièrement longue (j'ai mis environs quatre heures à tout faire en cumulé), parce que la PR est grosse, mais la plupart du temps, une QA, je la fais entre 10 et 20 min. Une fois que le projet est installé en local.

Merci de m'avoir lu, j'espère que mon témoignage vous encouragera à vous lancer dans la QA. C'est très formateur, et on apprend aussi beaucoup sans prendre de risque.

4. Liste des PR

4.1. zmarkdown

Cette semaine quelques petits bugs ont été découverts sur zmarkdown, ils ont été corrigés et @cepus a – comme promis – immédiatement déployé la correction.

— correction d'un bug qui faisait planter les grid-tables quand elles étaient suivies par une ligne avec simplement des espaces (https ://github.com/zestedesavoir/zmarkdown/commit/c021f248ec225ff4b12e867eb7f91029d41924d7)
— correction d'un bug qui faisait que lorsque des espaces étaient mis à l'extérieur des grid tables le parsing se faisait mal (https ://github.com/zestedesavoir/zmarkdown/commit/f8c31b2cfde30d5d759a34570549ea02a87d0f4b)
— Une contribution externe à la documentation de remark-caption https ://github.com/zestedesavoir/zmarkdown/pull/267

D'autres améliorations sont en cours :

— Un bug de gestion des espaces dans le parsing des smiley https ://github.com/zestedesavoir/zmarkdown/pull/272
— Une fonctionnalité : faire que zmarkdown tourne dans les navigateurs pour proposer un aperçu en temps réel (merci @heziode) https ://github.com/zestedesavoir/zmarkdown/pull/268

4.2. zds-site

— Retire le message "contenu modéré" sur les articles et tuto (https ://github.com/zestedesavoir/zds-site/pull/5036 @gcodeur)
— Enlève la coloration en vert des messages masqués qui étaient marqués comme "utile" (https ://github.com/zestedesavoir/zds-site/pull/5023 @gcodeur)
— Affiche la description du groupe dans la page d'une casquette (https ://github.com/zestedesavoir/zds-site/pull/5018 @gcodeur)
— Améliore les "fixtures" pour faciliter la QA de la page de contact (https ://github.com/zestedesavoir/zds-site/pull/5021 @firm1)

— Améliore l'interface de modération des billets (https ://github.com/zestedesavoir/zds-site/pull/4994 @pierre-24)
— Corrige quelques info techniques dans la doc (https ://github.com/zestedesavoir/zds-site/pull/5015 @firm1)
— Corrige quelques bugs de redirection lorsqu'on se connecte (https ://github.com/zestedesavoir/zds-site/pull/5011 @gustavi)

Contenu masqué

Contenu masqué n°1

Le contributeur	Le testeur (QA)	Le *reviewer* de code	Les membres du site	Le Release Manager
Il développe sa fonctionnalité ou son correctif				
Une fois que le travail lui semble bon, il soumet une PR				
-	Il voit arriver la PR dans la liste des PR du dépôt, et choisi de la tester	Il voit arriver la PR dans la liste des PR du dépôt, et choisi de *reviewer* le code		
-	Il teste la PR, et commente ses dernières lorsqu'il remarque des choses qui se passent mal	Il lit le code la PR, et le commente lorsqu'il remarque des choses qui ne vont pas		
Il corrige sa PR selon les remarques du testeur et du reviewer				
-	Une fois que la PR est OK, il déclare dans le commentaire de la PR : QA OK	Une fois que la PR est OK, il déclare dans le commentaire de la PR : Review OK		

-		La PR est acceptée et le code rentre officiellement dans la base de code du site		
-				Le code est déployé sur la beta
-			Les membres du site vont tester en masse les nouveautés sur la beta	
-				Dès que les tests grandeur nature sont concluants, le code est déployé sur la prod

Retourner au texte.

Contenu masqué n°2

```
# je récupère mes tokens
curl -X POST -H "Content-Type: application/json" -d
    '{"username": "user","password": "user","grant_type": "password","client_id
    http://localhost:8000/oauth2/token/

# je crée un galerie normale
curl -X POST -H 'Content-Type: application/json' -H
    "Authorization: TOKEN" -d
    '{"title": "Ma gallerie normale", "subtitle": "Mon sous titre"}'
    http://localhost:8000/api/galeries/

# je crée un galerie avec un titre vide
curl -X POST -H 'Content-Type: application/json' -H
    "Authorization: TOKEN" -d
    '{"title": "", "subtitle": "Mon sous titre"}'
    http://localhost:8000/api/galeries/

# je crée un galerie sans attribut titre
curl -X POST -H 'Content-Type: application/json' -H
    "Authorization: TOKEN" -d '{"subtitle": "Mon sous titre"}'
    http://localhost:8000/api/galeries/
```

Contenu masqué

```
12
13 # je crée un galerie avec un sous titre vide
14 curl -X POST -H 'Content-Type: application/json' -H
       "Authorization: TOKEN" -d
       '{"title": "Ma gallerie", "subtitle": ""}'
       http://localhost:8000/api/galeries/
15
16 # je crée un galerie sans attribut sous titre
17 curl -X POST -H 'Content-Type: application/json' -H
       "Authorization: TOKEN" -d '{"title": "Ma gallerie"}'
       http://localhost:8000/api/galeries/
18
19 # ...
```

Retourner au texte.

Liste des abréviations

PR Pull Request / Demande de changement dans le code. 1, 3–9

QA Assurance Qualité. 1, 3–8