w.c.s. - Development #22236

Sur du saut statique ne pas enregistrer de nouvelle entrée dans l'historique

02 mars 2018 16:01 - Frédéric Péters

Statut:	Fermé	**Début:**	02 mars 2018
Priorité:	Normal	**Echéance:**	
Assigné à:	Thomas Noël	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	

Description

Une construction classique c'est avoir un appel webservice de vérification de statut et un saut automatique basé sur une expiration pour boucler dessus. À chaque itération il y une ligne à l'historique qui est créée et avec le temps ça fait beaucoup. Dans cette situation on pourrait conserver la première entrée dans le statut, puis la dernière, et actualiser celle-ci avec le dernier timestamp.

(j'avais en tête qu'un ticket de la sorte existait déjà mais n'ai pas retrouvé)

Révisions associées

Révision ce94b901 - 16 mai 2018 13:47 - Thomas Noël

do not add an evolution on static jump (#22236)

Historique

#1 - 05 mars 2018 15:30 - Thomas Noël

Je serais presque pour différencier ça dans une action spécifique "rejouer ce statut" (qui serait techniquement une action de saut automatique sans préciser la cible, mais avec tout le reste, condition, trigger, expiration) ; ie mettre cette notion de non-enregistrement dans le journal sur ce type d'action, mais laisser les sauts statiques actuels tels quels (ne pas y ajouter de magie)

#2 - 20 mars 2018 11:48 - Thomas Noël

Idée sur la possible configuration : au lieu d'une expiration (pour indiquer une fréquence) et d'une condition (pour limiter à certaines heures et jours), prévoir plus explicitement de configurer :

- une fréquence : toutes les "H heures M minutes" ; par défaut 2h00
- un créneau horaire journalier : de début à fin ; par défaut 8h00 à 18h00
- les jours de la semaine, à cocher : par défaut lundi à vendredi, samedi et dimanche décochés

#3 - 20 mars 2018 14:10 - Benjamin Dauvergne

Un autre possibilité ce serait de compter les essais et de calculer des durées de réessaie exponentielles par rapport à ce compteur, ce n'est pas normal d'essayer toutes les heures pour toujours, ce ne serait pas extravagant que l'action d'appel de web-service maintienne un compteur (0 sinon la valeur au dernier appel) compteur disponible lors du calcul du temps d'expiration. Mais bon ça fait de plus en plus de python obscure, je serai pour une action "Réessaye" parmis les actions disponibles (avec tout le paramétrage imaginable) et une condition explicite de succès sur l'action appel de web-service.

```
Appel web-service:
URL: ...

Action on 500: réessaie
Action ..
Condition: data.status != "en traitement"
Action on condition failure: réssaie
Action sur réussite: sauter à terminer
Action sur nom de réessaie dépassé: sauter à en échec
Réssaie:
* Temps d'attente : 10 minutes
* Multiplier le temps d'attente à chaque essaie par: 1.5
* Maximum d'essaie: __
* Temps d'essaie maximum: 48 heures
```

avec tout çà on jouera moins dans les workflows.

#4 - 20 mars 2018 15:27 - Thomas Noël

Benjamin Dauvergne a écrit :

> ce n'est pas normal d'essayer toutes les heures pour toujours

Si, si, par exemple pour le retour d'un service technique, ou l'inscription à une NAP, etc...

Il ne s'agit pas de juste re-tester un webservice, on veut faire un appel au webservice, et en fonction de ci ou ça, retenter (mais le "ci ou ça" peut être bien complexe). Pour ma part je préfère que ça soit explicite au niveau du statut (on "verra" que ce statut est une boucle d'attente).

#5 - 20 mars 2018 15:30 - Benjamin Dauvergne

Thomas Noël a écrit :

> Benjamin Dauvergne a écrit :
>
> > ce n'est pas normal d'essayer toutes les heures pour toujours
>
> Si, si, par exemple pour le retour d'un service technique, ou l'inscription à une NAP, etc...

Soit. Ça ne remet pas en cause la façon de faire que je propose de toute façon.

> Il ne s'agit pas de juste re-tester un webservice, on veut faire un appel au webservice, et en fonction de ci ou ça, retenter (mais le "ci ou ça" peut être bien complexe). Pour ma part je préfère que ça soit explicite au niveau du statut (on "verra" que ce statut est une boucle d'attente).

As-tu un exemple concret où une évolution de l'appel de web-service tel que je le propose ne fonctionnerait pas ?

#6 - 20 mars 2018 15:58 - Thomas Noël

Benjamin Dauvergne a écrit :

> As-tu un exemple concret où une évolution de l'appel de web-service tel que je le propose ne fonctionnerait pas ?

Oui, typiquement un appel à un truc qui répond "ok voilà le résultat", mais en fait on teste sur une valeur dans le json obtenu. On envoie la demande dans un statut différent selon la réponse ; tant qu'aucune réponse n'est valide, on boucle (et c'est là que je poserai mon idée d'action "relancer/boucler/younameit")

Exemple plus concret qu'on a en prod, résultat de commission de place en crèche, on interroge le résultat de la commission et on boucle tant que c'est ni "favorable" (saut vers "bonne nouvelle") ni "défavorable" (saut vers "désolé c'est complet"), autrement dit tant que c'est pas décidé.

Le truc étant qu'on ne bouclerait que du lundi au vendredi, de 8h à 18h, parce la nuit et les week-end, le serveur de la commission est souvent HS.

#7 - 20 mars 2018 16:26 - Benjamin Dauvergne

Thomas Noël a écrit :

> Benjamin Dauvergne a écrit :
>
> > As-tu un exemple concret où une évolution de l'appel de web-service tel que je le propose ne fonctionnerait pas ?
>
> Oui, typiquement un appel à un truc qui répond "ok voilà le résultat", mais en fait on teste sur une valeur dans le json obtenu. On envoie la demande dans un statut différent selon la réponse ; tant qu'aucune réponse n'est valide, on boucle (et c'est là que je poserai mon idée d'action "relancer/boucler/younameit")

C'est exactement ce que je décris avec ma condition de réessaye sur l'appel de web-service.

> Exemple plus concret qu'on a en prod, résultat de commission de place en crèche, on interroge le résultat de la commission et on boucle tant que c'est ni "favorable" (saut vers "bonne nouvelle") ni "défavorable" (saut vers "désolé c'est complet"), autrement dit tant que c'est pas décidé.
>
> Le truc étant qu'on ne bouclerait que du lundi au vendredi, de 8h à 18h, parce la nuit et les week-end, le serveur de la commission est souvent HS.

Ça c'est plus sioux, mais bon on peut rêver de tout, comme avoir en plus une condition d'exécution mais moins on a de gestion des WS dans les workflows mieux ce sera pour que nos clients s'en sortent.

#8 - 20 mars 2018 16:37 - Thomas Noël

Benjamin Dauvergne a écrit :

> (...) moins on a de gestion des WS dans les workflows mieux ce sera pour que nos clients s'en sortent.

C'est sûr. En fait, c'est pour ça que je préfère que l'appel webservice ne fasse "que ça" et ne comporte pas de magie de répétition ou autre. Mais je vois bien qu'il y a des cas où ça serait chouette que le webservice "retente tout seul jusqu'à temps que ça marche" (mais j'y suis pas trop favorable).

Mon idée d'action "relancer le statut/boucler" c'est basé sur l'existant, où on a des saut "vers le statut actuel" avec condition et expiration pour faire des boucles, et ça génère des historiques de 10 kilomètres, et ça va pas. Remplacer ces sauts par cette action qui ne fera pas grandir l'historique, et en simplifier la configuration, ça me semble jouable.

#9 - 20 mars 2018 20:05 - Benjamin Dauvergne

Thomas Noël a écrit :

> Mon idée d'action "relancer le statut/boucler" c'est basé sur l'existant, où on a des saut "vers le statut actuel" avec condition et expiration pour faire des boucles, et ça génère des historiques de 10 kilomètres, et ça va pas. Remplacer ces sauts par cette action qui ne fera pas grandir l'historique, et en simplifier la configuration, ça me semble jouable.

Je ne vois pas comment remplacer un simple paramétrage (dans 90% des cas, une seule case à cocher, réessayer!, par défaut on devrait même la cocher) par une action magique dont il faudra décrire le comportement améliore les choses... Ça va sans dire que je trouve mon idée bien meilleure que la tienne ;-) (en même temps ça fait des années que je dis que l'action appel de WS devrait être de plus haut niveau et ne pas demander au gestionnaire des workflows de comprendre comment gérer un système distribué).

#10 - 21 mars 2018 09:29 - Thomas Noël

Benjamin Dauvergne a écrit :

> Thomas Noël a écrit :
>
> > Mon idée d'action "relancer le statut/boucler" c'est basé sur l'existant, où on a des saut "vers le statut actuel" avec condition et expiration pour faire des boucles, et ça génère des historiques de 10 kilomètres, et ça va pas. Remplacer ces sauts par cette action qui ne fera pas grandir l'historique, et en simplifier la configuration, ça me semble jouable.
>
> Je ne vois pas comment remplacer un simple paramétrage (dans 90% des cas, une seule case à cocher, réessayer!, par défaut on devrait même la cocher) par une action magique dont il faudra décrire le comportement améliore les choses...

On peut actuellement tout faire dans un saut "sur le même statut" avec des conditions alambiquées pour ne le faire que le jour et du lundi au vendredi, MAIS ça fait progresser l'historique pour rien, et c'est l'amélioration souhaitée dans ce ticket. Il s'agit juste d'avoir un système de saut qui ne fasse pas grossir l'historique.

#11 - 21 mars 2018 11:32 - Benjamin Dauvergne

Ben justement je propose un non-saut, pas un saut vers le même statut, qui s'appellerai "Ré-essayer" avec tout plein d'options qui n'ont de sens que dans le cas d'un appel à un WS (si on rajoute une action j'ai peur qu'elle ne serve qu'à ça en fait) et qui fonctionnerait via le cron de w.c.s. comme peut le faire wcs.wf.jump dans la fonction _apply_timeout() (mais là ça fait un saut je te l'accorde,

```
                        if x.must_jump(formdata):
                            jump_and_perform(formdata, x.status)
                            break
```

)

Le cron il s'occupe d'exécuter l'appel (faut voir comment ne pas se marcher sur les pieds entre le CRON et l'appel immédiat en arrivant sur le statut, le plus simple ce serait peut-être de ne plus faire d'appel en entrée, mais bon c'est déjà vrai d'un tas de trucs comme les trigger) si ça foire (i.e. le résultat pointe sur l'action ré-essayer) il ne fait rien (à la rigueur il log un truc pour pas que ce soit perdu) sinon il applique les actions habituelles.

#12 - 21 avril 2018 15:11 - Thomas Noël

(relecture dominicale de la discussion, parce que le problème existe toujours et qu'il faudrait le régler)

On a deux approches différentes ici :

- Benjamin propose un système dédié aux appels webservices, lié à notre expérience actuelle du problème. Si on un webservice échoue, on stoppe dessus et un cron viendra le retenter.
- Je propose quelque chose de plus générique, où le webservice qui plante indique juste qu'il faut s'arrêter en cas de plantage (on a déjà ça) et à côté, une action "Relancer ce statut après xx temps" qui va relancer tout le statut.

Les deux solutions n'ont pas le même but.

Évidemment je préfère la mienne qui est plus générique, pourrait s'appliquer dans d'autres situations (au pif, un statut avec trois actions mail à l'agent + bouton "ok" + relancer chaque jour = attendre l'action d'un agent et lui redire par mail tous les jours qu'il est attendu).

L'action "relancer le statut après xx temps" sera effectivement cantonnée à des usages spécifiques, la plupart du temps ça sera juste pour reprendre

des webservices plantés. Mais ça peut aussi être des webservices qui ont bien fonctionné, mais dont la réponse n'est pas celle attendue pour continuer ; typiquement un parapheur qui indique que le document n'a pas encore été soumis à signature, ou une place en crèche dont la commission n'a pas encore eu lieu, etc. Après l'appel webservice, on regarde quelle réponse a été reçue, on saute vers la suite dans tels ou tels cas, pour tous les autres cas on reste sur place et on attend la prochaine tentative (action "relancer").

De plus dans la solution de Benjamin, si l'appel webservice échoue, il faut retenir sur quelle action on a planté, pour ne relancer que celle-ci puis les suivantes en cas de succès. Ça me parait à la fois moins générique et un peu plus compliqué à mettre en place (à coder). On serait aussi dans une situation inconnue aujourd'hui, d'une demande plantée dans **telle** action (et non pas "tel statut").

#13 - 25 avril 2018 17:56 - Frédéric Péters

Pour tenter ma lecture, l'action "relancer le statut après xx temps", ce serait "saut sur place avec expiration" modulo l'écriture dans l'historique ?

Cas pratique, https://eservices.montpellier3m.fr/backoffice/workflows/17/ , on y a des statuts qui miment le statut de l'application métier,

- statut : Transmis au service (= statut qui existe dans l'application métier)
  - action 1 : appel webservice "récup du statut"
  - action 2 : si statut != "transmis au service" → saut vers "dispatch vers nouveau statut" (statut qui refait un appel webservice, un peu du gaspillage ok, puis envoie vers le statut qui a le bon nom).
  - action 3 : saut sur place avec expiration

L'action 3 serait remplacée par "relancer etc.", c'est bien ça ?

#14 - 26 avril 2018 08:15 - Frédéric Péters

(oh zut commentaire perdu, c'était plus élaboré dans sa première version)

La proposition de Benjamin, appliquée à ce cas, il me semble que ça donne :

- statut : Transmis au service
  - action 1 : appel webservice "récup du statut"
    - condition de succès "métier" : statut != "transmis au service"
    - gestion des erreurs
      - action en cas d'échec "méter" : réessai
    - gestion des réessai
      - temps d'attente, etc.

Là-dessus je pense rejoindre Thomas, ça sonne trop comme prévu pour la gestion d'erreurs techniques, détourné dès le début pour de la gestion "métier".

~~

À réfléchir à un autre cas cité, l'attente en commission, je vois une autre particularité, quand on entre dans le statut "attente commission", on n'a pas besoin de faire l'appel webservice tout de suite (vu que la commission ne se réunit que des mois plus tard); ce qu'on approche ici c'est l'envie de statut configurés en "exécution régulière".

- statut "Attente de commission"
  - exécution régulière :
    - condition : > 30 mars
    - régularité : tous les jours
  - action 1 : appel webservice "statut commission"
  - action 2 : saut si statut != "attente".

Fonctionnellement je sépare ici la notion d'exécution régulière, ça répond peut-être à "une action magique dont il faudra décrire le comportement", même si techniquement c'est implémenté pareil.

Mais si ça se comprend sous forme d'action, ça peut avoir un avantage, dans la séparation des actions via leur ordre d'exécution, exemple :

- statut "Attente de commission"
  - action 1 : envoi d'un email "votre dossier est en attente, la commission va se réunir le 30 mars".
  - action 2 : "tâches régulières"
    - condition : > 30 mars
    - régularité : tous les jours
  - action 3 : appel webservice "statut commission"
  - action 4 : saut si statut != "attente".

Et lors de l'exécution de ces tâches, on ne considère que les actions qui suivent celle-ci.

#15 - 26 avril 2018 10:29 - Benjamin Dauvergne

Je trouve la proposition d'un paramétrage exécution régulière sur le statut plus proche de ce que je propose et j'aime bien, je n'aime pas trop la dernière proposition, c'est mieux de conserver le principe d'une exécution atomique de toutes les actions d'un statut, inventer un curseur là dedans me semble juste augmenter la complexité sans apporter grand chose, si on a besoin d'une action avant l'exécution régulière on la met dans le statut précédent.

En fait on pourrait déprécier totalement le paramétrage timeout sur les sauts pour ne le conserver que sur les statuts, le statut redeviendrait ainsi

vraiment l'unité d'exécution.

Maintenant dans ma proposition et les demandes de Thomas deux trucs passent à l'as avec ces solutions purement basées sur nos timeout actuels:

- le calcul exponentiel de la durée de réessaie, alors on peut proposer un fonction =exponential_timeout() qui ferait toute sorte de magie, ça me semble plus clair d'avoir des champs explicite pour cela
- les périodes ouvertes aux essaies, Thomas indiquait vouloir pouvoir préciser que le service ne doit être qu'en semaine entre 8h et 17h
  On pourrait parfaitement intégrer ça à un timeout++ sur les statuts (et on ne touche pas au timeout old school de l'action de saut).

Pour rester dans le format proposé par Fred et ne pas trop augmenter la difficulté du paramétrage ça pourrait donner ça:

- exécution régulière:
  - condition: cron('* 8-17 * * 1-5')
  - période: toutes les 20 minutes (augmentation exponentielle jusqu'à 24h)

À nous de définir une liste de période utiles un peu partout, on pourrait aussi définir des conditions prédéfinies dans un panneau de configuration: { "Heures ouvrables": "cron('* 8-17 * * 1-5')" }.

#16 - 26 avril 2018 10:45 - Thomas Noël

Frédéric Péters a écrit :

> Cas pratique, https://eservices.montpellier3m.fr/backoffice/workflows/17/ , on y a des statuts qui miment le statut de l'application métier, ...
> L'action 3 serait remplacée par "relancer etc.", c'est bien ça ?

Exactement.

À noter qu'avec Brice on se disait que ça serait aussi bien utile hors webservice, par exemple pour relancer un agent tous les 3 jours, genre :

Statut "en attente de l'agent"

- envoyer un mail à l'agent
- afficher les bouton "ok" "ko" pour l'agent
- relancer chaque 3 jours

  À réfléchir à un autre cas cité, l'attente en commission, je vois une autre particularité, quand on ent
  Mais si ça se comprend sous forme d'action, ça peut avoir un avantage, dans la séparation des actions via leur ordre d'exécution, exemple :

  - statut "Attente de commission"
    - action 1 : envoi d'un email "votre dossier est en attente, la commission va se réunir le 30 mars".
    - action 2 : "tâches régulières"
      - condition : > 30 mars
      - régularité : tous les jours
    - action 3 : appel webservice "statut commission"
    - action 4 : saut si statut != "attente".
      Et lors de l'exécution de ces tâches, on ne considère que les actions qui suivent celle-ci.

Le petit bémol que j'y vois c'est que ça ajoute un nouveau "concept" dans nos workflows, le fait d'en reprendre l'exécution d'un bout seulement, de le "couper en deux" (ou en trois, avec deux actions, etc). C'est ceci dit plutôt séduisant. Je suis juste pas certain qu'on soit bien à l'aise dans le code actuel pour faire cela.

On peut regarder cette faisabilité, et si on bloque, il reste l'honorable sortie de secours des "statuts configurés en exécution régulière".

#17 - 01 mai 2018 12:26 - Thomas Noël

- Fichier 0001-do-not-add-an-evolution-on-static-jump-22236.patch ajouté

- Statut changé de Nouveau à En cours

- Patch proposed changé de Non à Oui

En jouant à code cela, je propose ce premier patch qui revient à l'idée de départ : sur un saut statique, ie si on ne change pas de statut et que la dernière évolution est vide de contenu, alors on n'ajoute rien dans l'historique. On se contente de mettre à jour la date de la dernière évolution.

Le patch triture le test «test_form_worklow_multiple_identical_status» pour son bien, parce qu'il en teste maintenant les différents cas possibles.

Le code ainsi ajouté va permettre de programmer plus facilement l'action dont nous avons parlé plus haut.

#18 - 01 mai 2018 13:04 - Frédéric Péters

Détail, le "if not self.evolution" est désormais dupliqué, la version passée peut être retirée.

Sur le fond, je me demande s'il est convenable de mettre à jour la date/heure, j'ai l'impression qu'à l'affichage on s'attend à voir la date/heure

d'arrivée dans le statut, qu'on perd ça. Bien sûr on gagne en facilité sur le calcul de l'expiration après mais je pense préférer l'ajout d'un nouvel attribut pour la date/heure de dernière fois où le saut sur place a eu lieu. Et dans must_jump, le diff serait genre :

```
        if self.timeout:
            timeout = int(self.compute(self.timeout))
            if formdata.evolution:
-                last = formdata.evolution[-1].time
+                last = getattr(formdata.evolution[-1], 'last_jump_time', None) or formdata.evolution[-1].time
            else:
                last = formdata.receipt_time
            if last:
                diff = time.time() - time.mktime(last)
                must_jump = (diff > timeout) and must_jump
```

#19 - 01 mai 2018 17:43 - Thomas Noël

Frédéric Péters a écrit :

> je pense préférer l'ajout d'un nouvel attribut pour la date/heure de dernière fois où le saut sur place a eu lieu

J'y ai pensé (à ça, et aussi à stocker le nombre de sauts) ; ça demande des modifs entre autre au niveau du stockage SQL (ajout d'un système de migration sur les _evolutions) ou inventaire des usages de evolution[-1].time, et j'avais pas trop envie de m'y lancer. Mais c'est aussi le bon moment, après tout, pour éclaircir un peu la gestion de formdata.evolution.

#20 - 01 mai 2018 17:43 - Thomas Noël

- Assigné à mis à Thomas Noël

- Patch proposed changé de Oui à Non

#21 - 01 mai 2018 18:45 - Thomas Noël

- Fichier 0001-do-not-add-an-evolution-on-static-jump-22236.patch ajouté

- Patch proposed changé de Non à Oui

Voici donc avec l'ajout d'un last_jump_time sur les evolution[].

#22 - 01 mai 2018 19:02 - Frédéric Péters

Dans stats_resolution_time ça doit rester l'heure d'entrée dans le statut.

Côté migration il faudra faire attention au rebase, #23511 aussi en ajoutait une.

D'un côté l'uniformité, de l'autre côté avancer sur le sujet, j'hésite quant à avoir last_update_time comme étant un datetime.

#23 - 01 mai 2018 23:27 - Thomas Noël

- Fichier 0001-do-not-add-an-evolution-on-static-jump-22236.patch ajouté

Frédéric Péters a écrit :

> Dans stats_resolution_time ça doit rester l'heure d'entrée dans le statut.

Effectivement.

> Côté migration il faudra faire attention au rebase, #23511 aussi en ajoutait une.

On verra qui c'est qui gagne ;)

> D'un côté l'uniformité, de l'autre côté avancer sur le sujet, j'hésite quant à avoir last_update_time comme étant un datetime.

Allez, lançons-nous, last_update_datetime.

#24 - 15 mai 2018 13:37 - Thomas Noël

- Fichier 0001-do-not-add-an-evolution-on-static-jump-22236.patch ajouté

Rebasé.

#25 - 16 mai 2018 11:08 - Frédéric Péters

À l'usage (un peu), pour nous permettre plus facilement le debug, je proposerais d'ajouter evolution.last_jump_datetime à l'affichage de l'historique.

Après #23856, ce serait :

```
-           <span class="time">{{evolution.datetime}}</span>
+           <span class="time">{{evolution.datetime}}
+                   {% if evolution.last_jump_datetime and user.is_admin %}
+                   ({% trans "last check:" %} {{ evolution.last_jump_datetime }})
+                   {% endif %}</span>
```

#26 - 16 mai 2018 11:55 - Thomas Noël

Le "is_admin" je le trouve restrictif, limité à une ou deux personnes qui ne sont pas en général des utilisateurs réels. J'imaginerais bien afficher ça à tout le monde :

```
<span class="time">{{evolution.datetime}}{% if evolution.last_jump_datetime %} … {{ evolution.last_jump_dateti
me }}{% endif %}</span>
```

(et en plus, même pas besoin d'attendre un autre patch, ah ah ah)

#27 - 16 mai 2018 12:08 - Frédéric Péters

À afficher à tout le monde venait davantage la question du libellé à mettre, last check / dernière évaluation (sonne trop technique je trouve)…

Aussi, peut-être, limiter ça à la dernière étape ?

```
                    {% if evolution.last_jump_datetime and forloop.last %}
                    <span class="last-jump">({% trans "last check:" %} {{ evolution.last_jump_datetime }})</spa
n>
                    {% endif %}</span>
```

(ah oui aussi, pour se permettre de jouer sur le style si on veut ça affiché à tout le monde, l'info dans un).

#28 - 16 mai 2018 13:46 - Thomas Noël

- Fichier 0001-do-not-add-an-evolution-on-static-jump-22236.patch ajouté

Finalement dans un premier temps d'acclimatation, j'ai préféré laisser le is_admin, et toujours afficher, pas seulement sur le forloop.last:

Donc même patch que précédent, avec en plus :

```
--- a/wcs/templates/wcs/formdata_history.html
+++ b/wcs/templates/wcs/formdata_history.html
@@ -13,7 +13,11 @@
        {% if evolution.status %}
          <div class="evolution-metadata">
            <span class="status">{{evolution.get_status_label}}</span>
-           <span class="time">{{evolution.datetime}}</span>
+           <span class="time">{{evolution.datetime}}
+             {% if evolution.last_jump_datetime and user.is_admin %}
+             <span class="last-jump">({% trans "last check:" %} {{ evolution.last_jump_datetime }})</span>
+             {% endif %}
+           </span>
          </div>
        {% endif %}
        <div class="msg">
```

#29 - 16 mai 2018 13:46 - Frédéric Péters

Ok pour moi, go.

#30 - 16 mai 2018 13:47 - Thomas Noël

- Statut changé de En cours à Résolu (à déployer)

```
commit ce94b901791b50a19d466d8eda5f648be40f162c
Author: Thomas NOEL <tnoel@entrouvert.com>
Date:   Tue May 1 12:22:19 2018 +0200

    do not add an evolution on static jump (#22236)
```

#31 - 23 décembre 2018 14:42 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

Fichiers

0001-do-not-add-an-evolution-on-static-jump-22236.patch	4,95 ko	01 mai 2018	Thomas Noël
0001-do-not-add-an-evolution-on-static-jump-22236.patch	15,8 ko	01 mai 2018	Thomas Noël
0001-do-not-add-an-evolution-on-static-jump-22236.patch	15 ko	01 mai 2018	Thomas Noël
0001-do-not-add-an-evolution-on-static-jump-22236.patch	15,4 ko	15 mai 2018	Thomas Noël
0001-do-not-add-an-evolution-on-static-jump-22236.patch	16,2 ko	16 mai 2018	Thomas Noël