Publik - Development #32194

Paiement : transaction à validation manuelle sur payzen

11 avril 2019 10:43 - Emmanuel Cazenave

Statut:	Fermé	**Début:**	11 avril 2019
Priorité:	Normal	**Echéance:**	
Assigné à:	Emmanuel Cazenave	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Non	**Club:**	Non
Planning:	Non		

Description

Un peu la suite de #26914, la combinaison de 'Paiement différé à une date arbitraire' et 'validation manuelle' donnant une 'caution'.

Je pensais qu'il n'y avait pas de soucis pour la validation manuelle, mais en fait si.

- cette information est portée par une régie Lingo : vads_validation_mode
- dans un contrat Payzen, on a une boutique unique, sur laquelle on doit renseigner l'URL de notification de paiement
- moralité un contrat payzen = une régie unique lingo (alors que mon plan était d'avoir deux régies Lingo, une en validation manuelle, une en validation automatique)

Donc il va falloir que le mode de validation soit porté par un BasketItem, passé à eopayment, etc.

Tickets combo et eopayment à venir.

PS : vérifié au téléphone avec le support Payzen, "oui c'est possible d'avoir une deuxième boutique, il suffit de souscrire à un deuxième contrat" (sic).

Demandes liées:

Lié à EOPayment - Development #32224: créer une nouvelle API guess() sur les ...	**Fermé**	**12 avril 2019**
Lié à Combo - Development #32441: Nouvel objet PaymentBackend	**Fermé**	**18 avril 2019**

Historique

#1 - 11 avril 2019 12:31 - Benjamin Dauvergne

On reçoit vads_site_id et vads_trans_id sur le retour ça devrait nous permettre via une URL unique de différencier des régies et les transactions (et donc enregistrer plusieurs fois le même raccordement avec des paramétrages différents), je ne dis pas que c'est la bonne solution et ça demande du boulot pour pouvoir demander à eopayment d'analyser n'importe quelle requête pour savoir si c'est du payzen ou du paybox, mais c'est plus simple pour l'appelant.

#2 - 11 avril 2019 18:56 - Frédéric Péters

On aurait une API type :

> transaction_id = eopayment.guess(query_string)

qui itérerait sur les backends ?

Peut-être que pour la différence payzen/paybox, il pourrait y avoir un paramètre supplémentaire, avec la liste des backends à interroger, qu'on mettrait genre backends=[x.backend for x in Regie.objects.all]; ça me semble assez simple pour tout le monde.

mais peut-être qu'arriverait un jour avec un backend où le transaction_id n'est pas directement accessible, du coup quelque chose comme ça :

> transaction_id = eopayment.guess(query_string, backends=[get_eopayment_object(requet, regie) for regie in Regie.objects.all()])

#3 - 12 avril 2019 09:38 - Benjamin Dauvergne

Frédéric Péters a écrit :

> On aurait une API type :

> transaction_id = eopayment.guess(query_string)

qui itérerait sur les backends ?

Je pensais qu'on pourrait s'en passer mais effectivement vu que systempay et payzen ont la même API on va avoir un souci, alors plutôt laisser le boulot d'itération à lingo en ayant une interface de ce genre :

```
for regie in Regie.objects.all():
     eopayment = get_eopayment_backend(request, regie)
     transaction_id = backend.guess(method=request.method, query_string=request.META.get('QUERY_STRING', ''),
headers=headers(request), body=request.body)
     if transaction_id:
         break
else:
     raise RegieNotFound()
```

L'intérêt c'est que le backend saura comparer la valeur de vads_site_id par exemple.

> Peut-être que pour la différence payzen/paybox, il pourrait y avoir un paramètre supplémentaire, avec la liste des backends à interroger, qu'on mettrait genre backends=[x.backend for x in Regie.objects.all]; ça me semble assez simple pour tout le monde.
>
> mais peut-être qu'arriverait un jour avec un backend où le transaction_id n'est pas directement accessible, du coup quelque chose comme ça :
>
> > transaction_id = eopayment.guess(query_string, backends=[get_eopayment_object(requet, regie) for regie in Regie.objects.all()])

Voilà, mais je garderai le boulot de recherche dans lingo, finalement c'est son taf, c'est lui qui intègre.

#4 - 12 avril 2019 09:38 - Benjamin Dauvergne

Et pour redresser les entêtes HTTP voir https://stackoverflow.com/questions/3889769/how-can-i-get-all-the-request-headers-in-django

#5 - 12 avril 2019 09:41 - Benjamin Dauvergne

- Lié à Development #32224: créer une nouvelle API guess() sur les backends pour permettre de détecter un backend depuis une réponse ajouté

#6 - 12 avril 2019 09:41 - Benjamin Dauvergne

J'ai créé #32224 sur eopayment pour cela; mais au passage je me rends compte que .response() suffirait si on sait quelles exceptions attendre.

#7 - 16 avril 2019 11:38 - Emmanuel Cazenave

Je suis pas très à l'aise avec votre plan.

Dans le cas où on a deux régies de même types, on se retrouve avec deux régies qui vont répondre toutes les deux que oui la transaction les concerne, la première aura gagné. Avec des sources d'erreurs potentielles si la réponse fait appel à des paramètres de configuration de la régie (c'est le cas de sips2), et que différences de configuration entre les deux régies.
Ce qui est possible puisqu'on doit dupliquer toute la configuration, avec dans mon cas seulement un paramètre qui doit changer. En dehors de cas c'est quand même un peu dommage cette duplication, j'ai l'impression qu'on ouvre la porte à des cas d'embrouilles.

Et donc je me dis qu'on serait plus à l'aise à maintenir cette relation one to one bien compréhensible une régie = un backend de paiement. Qu'on pourrait donner la possibilité dans l'UI de déclarer des configuration additionnelle pour une régie, qu'on pourrait appeler ça des 'types de paiement', que add-basket-item accepte un slug de type de paiement en paramètre, qu'il stocke sur un nouveau champ json du BasketItem la config en plus, que get_eopayment_object accepte cette config en plus, que PayMixin.handle_payment en tire parti. Ça me parait aller plus dans la direction 'il y a pas forcément besoin d'un dev pour configurer une démarche de paiement'.

#8 - 16 avril 2019 14:50 - Benjamin Dauvergne

Emmanuel Cazenave a écrit :

> Je suis pas très à l'aise avec votre plan.
>
> Dans le cas où on a deux régies de même types, on se retrouve avec deux régies qui vont répondre toutes les deux que oui la transaction les concerne, la première aura gagné. Avec des sources d'erreurs potentielles si la réponse fait appel à des paramètres de configuration de la régie (c'est le cas de sips2), et que différences de configuration entre les deux régies.

eopayment va te dire que c'est le bon type de regie et te filer le transation_id, ensuite tu dois utiliser le transaction_id pour vérifier que la transaction appartient bien à cette régie, sinon tu passes à la suivante:

```
for regie in regies:
```

```
    backend = regie.get_backend()
    try:
        response = backend.response(query_string)
    except Exception:
        continue
    try:
        transaction = regie.transaction.get(transaction_id=response.transaction_id)
    except Transaction.DoesNotExist:
        continue
    break
else:
    return template('not-found.html')
```

#9 - 16 avril 2019 19:25 - Emmanuel Cazenave

Soit pour le bout de solution, je passe sur les détails.

Mais sur l'aspect dupliquer toute un configuration de régie pour y changer un seul paramètre ? Nobody's chocked ?
Vraiment preneur de retours sur mon plan, j'ai déjà un peu commencé mais si ça soulève de grosses réserves...

#10 - 16 avril 2019 19:40 - Frédéric Péters

De mon côté clairement il y a des trucs qui sont portés côté régie qui n'ont rien à y faire, basiquement tout ce qui suit "Régie par défaut", et tout ça appelle restructuration etc. mais je ne suis pour autant pas ok avec juste dire "régie = backend de paiement", il y a aussi toute une part de configuration qui vient avec le backend.

Le truc est que dans eopayment les paramètres figés (genre secret_key) sont sur le même pied que les paramètres volatiles (genre capture_day), ce qui fait que pour une variation légère (genre capture_day) il y a des bouts de json copiés/collés à l'identique (et qu'il manque le patch discuté ici pour retrouver une régie à partir d'une transaction); mais je pense quand même cette situation meilleure que celle qui serait l'inverse, à faire porter sur le basketitem ces paramètres.

#11 - 17 avril 2019 16:14 - Emmanuel Cazenave

Frédéric Péters a écrit :

> mais je ne suis pour autant pas ok avec juste dire "régie = backend de paiement"

Je parlais des backend physiques pas des backends de eopaypement, je voulais insister sur le fait que devoir créer plusieurs régies pour un même compte chez payzen me parait être un problème.

> Le truc est que dans eopayment les paramètres figés (genre secret_key) sont sur le même pied que les paramètres volatiles (genre capture_day), ce qui fait que pour une variation légère (genre capture_day) il y a des bouts de json copiés/collés à l'identique

Oui c'est pour ça que je propose de pouvoir en déporter ailleurs, sur des objets 'types de paiement' qu'on lierait à une régie.
Sur ces objets j'imaginais un champ json avec de la config comme sur les régies mais ça pourrait être quelque chose de plus haut niveau, genre je pourrais commencer par une colonne booléenne 'Validation manuelle'.
Un BakestItem accepterait une foreign key vers un Type de paiement, les vues qui génèrent des transaction passeraient un nouveau paramètre manual_validation, tiré du 'type de paiement' au payment.request des backend, charge à eux d'envoyer ce qu'il faut au backend physique.

> (et qu'il manque le patch discuté ici pour retrouver une régie à partir d'une transaction);

Est-ce ça a un autre intérêt que de contourner le manque de souplesse actuel sur la configuration des paiements ? (je n'en vois pas c'est pour ça que je pousse dans une autre direction)

> mais je pense quand même cette situation meilleure que celle qui serait l'inverse, à faire porter sur le basketitem ces paramètres.

Ce que viens de décrire me semble plus acceptable de ce point de vue, ne pas répandre des paramètres bruts de configuration de régie jusqu'au BasketItem,
c'était bien l'esprit de ta remarque ?

Pas d'expérience avec d'autres backends que payzen, jamais vu le cas 'régies distantes', je passe peut-être complètement à coté de trucs.

#12 - 17 avril 2019 16:28 - Frédéric Péters

> Un BakestItem accepterait une foreign key vers un Type de paiement, les vues qui génèrent des transaction passeraient un nouveau paramètre manual_validation, tiré du 'type de paiement' au payment.request des backend, charge à eux d'envoyer ce qu'il faut au backend physique.

Parce que l'existant associe basketitem et régie et que ça se trouve dans des appels webservice configurés à mille endroits, je détricoterais dans

l'autre sens, BasketItem → Regie → PaymentBackend, avec PaymentBackend qui serait les champs "service de paiement" et "options du service de paiement".

Et cet objet Régie détricoté pourrait avoir un attribut capture_mode, idéalement proposé uniquement si le backend l'autorise, avec uniquement les choix possibles pour celui-ci. (author_capture, immediate, validation, traduits de manière intelligibles).

Ça me semble, pris dans l'autre sens (pour faciliter la compatibilité avec l'existant), ce que tu proposes, correct ?

~~

> Pas d'expérience avec d'autres backends que payzen, jamais vu le cas 'régies distantes', je passe peut-être complètement à coté de trucs.

Pas vraiment important je pense mais je ne suis pas sûr de ce que tu appelles "régies distantes", ce serait une référence au webservice de récupération des factures ?

#13 - 17 avril 2019 17:51 - Emmanuel Cazenave

Frédéric Péters a écrit :

> Parce que l'existant associe basketitem et régie et que ça se trouve dans des appels webservice configurés à mille endroits, je détricoterais dans l'autre sens, BasketItem → Regie → PaymentBackend, avec PaymentBackend qui serait les champs "service de paiement" et "options du service de paiement".
>
> Et cet objet Régie détricoté pourrait avoir un attribut capture_mode, idéalement proposé uniquement si le backend l'autorise, avec uniquement les choix possibles pour celui-ci. (author_capture, immediate, validation, traduits de manière intelligibles).
>
> Ça me semble, pris dans l'autre sens (pour faciliter la compatibilité avec l'existant), ce que tu proposes, correct ?

Je proposais de rajouter la possiblité d'un BasketItem → 'Type de paiement' → Regie mais en autorisant encore BasketItem → Regie, et du coup je vois pas de problème de compat sur les appels webservice, l'API de AddBasketItem gagnait juste un paramètre optionnel, genre le slug d'un 'Type de Paiement'.

C'est assez proche tout ça. Dan ton plan on déporte d'un coup sur PaymentBackend tout ce qui doit l'être ? Ou on vit un moment avec des régies à l'ancienne qui portent toute la configuration et des nouvelles régies épurées qui ont gagné un PaymentBackend ?

J'ai l'impression que mon plan permet une transition plus progressive, que 'Type de paiement' pourrait absorber petit à petit ce qui doit l'être de la Regie.

> Pas vraiment important je pense mais je ne suis pas sûr de ce que tu appelles "régies distantes", ce serait une référence au webservice de récupération des factures ?

Oui.

#14 - 17 avril 2019 18:00 - Frédéric Péters

> Dan ton plan on déporte d'un coup sur PaymentBackend tout ce qui doit l'être.

Avec "tout ce qui doit l'être" que je définis dans le commentaire précédent comme étant : "service de paiement" et "options du service de paiement".

> Dans ton plan on déporte d'un coup sur PaymentBackend tout ce qui doit l'être ?

Oui; migration qui crée PaymentBackend et ajoute un payment_backend_id dans Regie, migration qui crée des PaymentBackend et rempli les payment_backend_id, migration qui retire service et service_options de Regie.

Et tout ça ne change rien pour les appelants, on ne se trouve pas non plus à avoir une documentation couvrant plusieurs situations ou une documentation qu'on décide de garder simple et omettant les situations intermédiaires ou passées, etc.

#15 - 17 avril 2019 18:23 - Emmanuel Cazenave

Frédéric Péters a écrit :

> on ne se trouve pas non plus à avoir une documentation couvrant plusieurs situations ou une documentation qu'on décide de garder simple et omettant les situations intermédiaires ou passées, etc.

Hihi sur la doc de cette partie on d'autres problèmes, ce sera l'occasion de la reprendre anyway.

Mais ok je commence à mieux voir le truc et ça semble plus propre, je laisse infuser un peu dans mon cerveau et je fais les tickets.

#16 - 18 avril 2019 10:08 - Benjamin Dauvergne

Emmanuel Cazenave a écrit :

> Soit pour le bout de solution, je passe sur les détails.
>
> Mais sur l'aspect dupliquer toute un configuration de régie pour y changer un seul paramètre ? Nobody's chocked ?

Il y a un premier point que je pense utile c'est n'avoir qu'une seule URL à déclarer pour les retours de paiement, ça n'empêche pas de conserver l'URL spécifique pour le cas où en aurait besoin (rien ne sert de casser ce qui marche); ça simplifiera la doc sur les paiements, actuellement on est obligé de dire aux gens d'aller voir le numéro de la régie dans l'URL ou au mieux d'aller voir l'écran de configuration qui leur donnera l'URL.

Voilà déjà pour mon commentaire sur ce point.

#17 - 18 avril 2019 10:11 - Frédéric Péters

> Il y a un premier point que je pense utile c'est n'avoir qu'une seule URL à déclarer pour les retours de paiement (...)

C'est même indispensable vu les services de paiement qui n'en autorisent qu'une, à configurer dans leur backend.

#18 - 18 avril 2019 10:12 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Parce que l'existant associe basketitem et régie et que ça se trouve dans des appels webservice configurés à mille endroits, je détricoterais dans l'autre sens, BasketItem → Regie → PaymentBackend, avec PaymentBackend qui serait les champs "service de paiement" et "options du service de paiement".

Techniquement je n'ai rien contre cela, mais ça va à l'encontre de la logique métier des collectivités où une régie = un compte en banque = une abonnement terminal de payement électronique = un backend eopayment, si on conitnue d'appeler les choses régie et si on a dans l'esprit qu'un jour ce sera suffisamment clair pour qu'une collectivité configure cela toute seule, ça ne va pas dans le bon sens (après peut-être qu'on ne devrait pas appeler ça régie justement).

#19 - 18 avril 2019 10:16 - Frédéric Péters

Absolument zéro attachement aux libellés présents dans l'UI; le seul truc que je souhaite c'est ne pas devoir aller sur des workflows remplacer des ?regie_id=foo par des ?whatever_id=foo.

#20 - 18 avril 2019 10:27 - Benjamin Dauvergne

Emmanuel Cazenave a écrit :

> Je proposais de rajouter la possiblité d'un BasketItem → 'Type de paiement' → Regie mais en autorisant encore BasketItem → Regie, et du coup je vois pas de problème de compat sur les appels webservice, l'API de AddBasketItem gagnait juste un paramètre optionnel, genre le slug d'un 'Type de Paiement'.

Ce que ça complexifie c'est l'après, l'affichage du basketitem dans le panier et le déclenchement du paiement; parce que des éléments qui nécessitent des paramètres de paiement différents (pour un même backend) ne pourront pas être payés ensembles; c'est ça le fond du problème. Comme on a déjà la logique qui permet de ne pas pouvoir payer des BasketItem reliés à des régies différentes je pense que l'idée de Fred c'est de rester sur cette mécanique en faisant en sorte que des paiements avec une paramétrage différentes soient rattachés à des objets régies différents (mais là toujours le fait de nommer ça régie est gênant pour avoir les idées claires).

> C'est assez proche tout ça. Dan ton plan on déporte d'un coup sur PaymentBackend tout ce qui doit l'être ? Ou on vit un moment avec des régies à l'ancienne qui portent toute la configuration et des nouvelles régies épurées qui ont gagné un PaymentBackend ?
>
> J'ai l'impression que mon plan permet une transition plus progressive, que 'Type de paiement' pourrait absorber petit à petit ce qui doit l'être de la Regie.

De toute façon je pense qu'on perd tout le monde avec ce terme régie, on a des trucs à payer, les BasketItem (c'est pas terrible comme nom non plus mais comme c'est plus invisible restons-en là), des types de paiement, à terme, à déclenchement manuel, etc..; et des abonnements TPE qu'on pourra nommer PaymentBackend.
Ce serait bien de pouvoir soumettre une transaction de paiement avec toute sorte de paramètres pour différents paiements mais ça n'existe pas.

Ce qui fait qu'en fait je suis plus sur l'architecture de Manu, on pourrait inventer un PaymentType optionnel pour un BasketItem, si absent on prend le PaymentType par défaut de chaque régie, progressivement on découpe la configuration des régies en deux, la partie statique dans Regie et la partie "dynamique" dans PaymentType (avec un PaymentType par défaut, vide et en fait implicite). La mécanique du front pour regrouper les BaskestItem se baserait désormais sur les PaymentType et plus sur les Regie.

#21 - 18 avril 2019 10:29 - Benjamin Dauvergne

Frédéric Péters a écrit :

> Absolument zéro attachement aux libellés présents dans l'UI; le seul truc que je souhaite c'est ne pas devoir aller sur des workflows remplacer des ?regie_id=foo par des ?whatever_id=foo.

Par rapport à mon dernier commentaire mon idée ce serait de dire si on a juste regie_id, on prend le PaymentType par défaut de la régie (celui qui est implicite, pas forcément un modèle en base), sinon on un payment_type=manuel on prend ce payment type pour la régie concernée. Ainsi les URLs existantes ne varient pas mais on peut en avoir de nouvelles qui font plus.

#22 - 26 avril 2019 11:16 - Emmanuel Cazenave

- Lié à Development #32441: Nouvel objet PaymentBackend ajouté

#23 - 19 juillet 2019 15:04 - Stéphane Laget

On ferme ?

#24 - 19 juillet 2019 15:05 - Stéphane Laget

- Statut changé de Nouveau à En cours

- Assigné à mis à Emmanuel Cazenave

#25 - 29 juillet 2019 10:23 - Emmanuel Cazenave

- Statut changé de En cours à Fermé

C'est maintenant ok via #32967 et #32441.