Authentic 2 - Development #46424

API de recherche doublons avant création

07 septembre 2020 17:04 - Frédéric Péters

Statut:	Fermé	**Début:**	07 septembre 2020
Priorité:	Normal	**Echéance:**	
Assigné à:	Valentin Deniaud	**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	Non

Description

De #41431#note-5,

> une API dans Authentic à qui seraient envoyées toutes les infos qu'on enverrait à l'API de création d'utilisateur et dont le retour serait la liste de potentiels doublons.

Ça pourrait permettre de commencer avec une recherche sur prénom/nom et email et de laisser possible l'extension à des détections plus avancées comme ce qui peut exister dans Zoo.

Révisions associées

Révision e6b2e5db - 01 octobre 2020 13:29 - Valentin Deniaud

api: add find duplicate users endpoint (#46424)

Historique

#2 - 16 septembre 2020 17:21 - Valentin Deniaud

- Assigné à mis à Valentin Deniaud

#3 - 22 septembre 2020 16:32 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

- Statut changé de Nouveau à Solution proposée

- Patch proposed changé de Non à Oui

Pas tout à fait fini mais je préfère montrer ce que j'ai fait avant d'aller plus loin.

J'ai choisi l'option simple : aucun mix avec la full text search, je fais les choses ad hoc dans mon coin ; aucune volonté d'exploiter tous les paramètres envoyés lors de la création d'un utilisateur, je me limite à nom/prénom/date de naissance.

#4 - 22 septembre 2020 17:23 - Valentin Deniaud

En relisant la description je vois

Frédéric Péters a écrit :

> une recherche sur prénom/nom et email

Et j'avais fait ça dans une première version du patch mais il me semble que ce n'est justement pas ce qu'on veut : jean dupont s'est créé un compte avec supersayan42@pouet.com et bgdu92@hop.com, il ne faut surtout pas prendre en compte l'email lors de la recherche de doublon sinon on passe à côté. Le bon critère important en plus de nom/prénom ça me semble être la date de naissance.

D'ailleurs pour la date de naissance, étant donné qu'elle est stockée sous la forme d'une chaîne de caractère %Y-%m-%d, la manière d'avoir une recherche robuste aux erreurs de saisie ne passe pas par les trigrammes (imo). Par contre la distance de lenvenshtein ça me paraît très adapté, c'est ce que j'ai fait.

#5 - 22 septembre 2020 21:42 - Benjamin Dauvergne

Pour la date de naissance on ne fait pas de recherche approximative, si la date est présente, un match donne un bonus dans le calcul de match (genre 60% + 30% * score-trigram-nom/prenom), si pas présent on ne cherche que via trigram sur le nom complet (c'est comme ça que j'ai fait Nanterre, avec en plus pour les doublons des heuristiques spécifiques parce qu'ils ont importés des personne à un moment ayant uniquement l'année de naissance ou alors tous né le 1er janvier 1970, on aura pas ça ici).

Pour l'email c'est pareil, il faut gérer un score variable, si ça matche bonus sinon on revient sur du standard nom/prénom. Et généralement il est difficile de régler ces facteurs une fois pour toute, c'est mieux si c'est configurable.

#6 - 22 septembre 2020 21:50 - Benjamin Dauvergne

J'ai relu, plusieurs remarques :

- ne pas matcher sur nom et prénom mais sur la concaténation des deux, ça donne un score plus facilement utilisable (sauf si l'un des deux n'est pas fourni, mais on peut ignorer ce problème je pense),
- je ne vois pas de création d'index, sans index sur cette concaténation ça ne marchera juste jamais; il faut utiliser RunSQL pour le créer en Django 1.11 je ne pense pas que l'API Meta des modèles permettent de créer un index aussi compliqué,
- levenshtein ne peut pas utiliser d'index, oublier, c'est un gadget pas vraiment utilisable en temps réel
- j'imaginais plutôt une API ayant exactement la même forme que l'API de création, charge ensuite pour le endpoint de voir les champs qu'il utilise pour la recherche de doublon; on peut facilement cabler que si un champ de type birthdate est présent, on s'en sert (qu'il s'appelle birthdate, "date-de-naissance" ou quoi que ce soit).
- au niveau de la requête il faut arriver à refiler le calcul du score complètement à postgresql qui s'en sortira mieux que nous en python surtout si on veut utiliser LIMIT pour ne récupérer que les n premiers résultats, voir le code de zoo.

#7 - 23 septembre 2020 10:25 - Valentin Deniaud

- Statut changé de Solution proposée à En cours

Merci pour les remarques, je me doutais bien que ça n'allait pas convenir du premier coup et je vais reprendre. Je laissais aussi la création d'index de côté en attendant d'être plus sûr des requêtes à faire.

Benjamin Dauvergne a écrit :

> - levenshtein ne peut pas utiliser d'index, oublier, c'est un gadget pas vraiment utilisable en temps réel

J'ai l'impression qu'un problème analogue se pose avec unaccent, non ?

> - j'imaginais plutôt une API ayant exactement la même forme que l'API de création, charge ensuite pour le endpoint de voir les champs qu'il utilise pour la recherche de doublon

Et c'est ce qu'il se passe, si on fournit tous les attributs de création ils seront simplement ignorés.

> on peut facilement cabler que si un champ de type birthdate est présent, on s'en sert (qu'il s'appelle birthdate, "date-de-naissance" ou quoi que ce soit).

Ça par contre non, effectivement... Mais je suis quand même davantage pour avoir un truc carré, à mes yeux le petit confort qu'apporte de rebalancer tous les paramètres sans se poser de question ne contrebalance pas 1) la simplicité du code 2) la sûreté du fonctionnement, on ne confond pas la date de naissance avec une autre date éventuellement présente 3) le fait que l'appelant sait ce qui sera utilisé pour déterminer un doublon (parce que du coup il aura dû lire la doc et vu les paramètres réellement utiles).

#8 - 23 septembre 2020 11:25 - Benjamin Dauvergne

Valentin Deniaud a écrit :

> Merci pour les remarques, je me doutais bien que ça n'allait pas convenir du premier coup et je vais reprendre. Je laissais aussi la création d'index de côté en attendant d'être plus sûr des requêtes à faire.
>
> Benjamin Dauvergne a écrit :
>
> > - levenshtein ne peut pas utiliser d'index, oublier, c'est un gadget pas vraiment utilisable en temps réel
>
> J'ai l'impression qu'un problème analogue se pose avec unaccent, non ?

Comme je n'ai pas conscience du problème que tu as en tête, je vais dérouler ce que j'ai fait sur zoo :

- pour pouvoir utiliser un index sur une expression complexe il faut que l'index soit fait sur l'expression complexe elle même donc si recherche trigram sur une chaîne sans accent en minuscule il faut CREATE INDEX ... ON LOWER(UNACCENT(value)) ... (je ne montre pas les modifieurs pour les trigrams c'est dans la doc postgres). Mais souci on ne peut utiliser que des fonction "immutable" ou pure pour utiliser la désignation en programmation fonctionnelle, i.e. qui ne dépendent pas de l'environnement or unaccent dépend de la locale en cours (lower aussi je pense si c'est bien fait) la solution que j'ai trouvé a été de créer une fonction déclarée arbitrairement immutable qui fait le LOWER(UNACCENT(...)), on peut voir ça là : http://git.entrouvert.org/zoo.git/tree/zoo/zoo_meta/migrations/0002_auto_20161214_1545.py#n35

Je suis aller un peu plus loin avec immutable_normalize qui en plus virer les caractères spéciaux par des 'x' (c'est assez arbitraire mais appliqué des deux coté de l'égalité ou de la recherche trigram ça marche bien).

immutable_timestamp c'est pour pouvoir indexer des dates dans un document JSON (via CREATE INDED ... ON

immutable_timestamp(document->>'date')).

- j'imaginais plutôt une API ayant exactement la même forme que l'API de création, charge ensuite pour le endpoint de voir les champs qu'il utilise pour la recherche de doublon

> Et c'est ce qu'il se passe, si on fournit tous les attributs de création ils seront simplement ignorés.

Sauf qu'il n'y a pas d'attribut birthdate en standard, il peut s'appeler n'importe comment, autant réutiliser BaseUserSerializer et simplement cherche si un champ de type "birthdate" existe.

> > on peut facilement cabler que si un champ de type birthdate est présent, on s'en sert (qu'il s'appelle birthdate, "date-de-naissance" ou quoi que ce soit).
>
> Ça par contre non, effectivement... Mais je suis quand même davantage pour avoir un truc carré, à mes yeux le petit confort qu'apporte de rebalancer tous les paramètres sans se poser de question ne contrebalance pas 1) la simplicité du code 2) la sûreté du fonctionnement, on ne confond pas la date de naissance avec une autre date éventuellement présente 3) le fait que l'appelant sait ce qui sera utilisé pour déterminer un doublon (parce que du coup il aura dû lire la doc et vu les paramètres réellement utiles).

Sauf qu'authentic est générique et pas ton code, et on aura beaucoup de mal à l'étendre pour dire qu'on cherche aussi les doublons via l'adresse ou le numéro de mobile, je ne dis pas qu'il faut faire tout ça maintenant, mais introduire un serializer et une forme du code inutile qui bloque les évolutions c'est se mettre des batons dans les roues pour rien.

Aussi la recherche par trigram devrait être écrite à coté du code de l'API qu'on ait pas trop de mal à l'utiliser dans le manager mais que ça donne les mêmes résultats (en gros je verrai bien une méthode fuzzy_search sur User ou UserManager qui prend un dico d'attributs ou une instance et l'API n'aurait qu'à l'appeler son boulot étant seulement d'apter le contenu de serializer.validated_data vers ce qu'attend cette méthode, la méthode contenant nos dernières avancée en matière de recherche approchée).

#9 - 24 septembre 2020 17:57 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

- Statut changé de En cours à Solution proposée

Merci pour toutes les infos, je pose un patch wip histoire de dire que ça avance.

J'ai appliqué la plupart des remarques, reste surtout les histoires d'index. C'est le gros bout du boulot mais j'ai bien commencé à comprendre comment ça marchait en général dans postgres et en particulier dans zoo, je devrais avoir un truc lundi, donc ça sera bon pour le prochain cycle.

#10 - 24 septembre 2020 17:57 - Valentin Deniaud

- Statut changé de Solution proposée à En cours

#11 - 24 septembre 2020 21:41 - Benjamin Dauvergne

Ça m'a l'air ok, il faudrait voir en combien de temps ça répond avec une centaine de milliers d'utilisateur (on est à 146 000 à Lyon, 80 000 à Nanterre et environ 50000 dans le Nord pour les ordres de grandeur).

Je viens de faire un petit dév sur le coté pour tester ça sur une base un peu plus réelle (ajouter --slow à pytest), #46988.

#12 - 29 septembre 2020 18:03 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

- Statut changé de En cours à Solution proposée

Bon, je m'en suis sorti mais j'ai galéré. Florilège de problèmes :

- Utiliser TrigramSimilarity et filtrer là dessus ça ne marche pas avec l'index.
  - Pour le filtrage il faut utiliser le lookup __trigram_distance et là l'index est utilisé.
  - Pour le ORDER_BY avec calcul du score il faut utiliser TrigramDistance, parce que c'est un opérateur, alors que TrigramSimilarity est une fonction, et pas d'index pour les fonction.
  - Tout ça matche ce qui est fait dans zoo, je l'ai juste redécouvert à grand peine.
- Utilier Concat et faire un index là dessus ça ne marche pas.
  - Concat de django se traduit en CONCAT de postgres, qui n'est pas immutable contrairement à ||.
  - Donc à la Unaccent, nouveau lookup.
- Créer une extension type unaccent ou trigram avec les opérations fournies par django ne semble pas marcher dans un contexte multitenant.
  - Elles font simplement CREATE EXTENSION, et avec migrate_schemas, l'extension est créée dans le premier schéma authentic_xx alors qu'on la voudrait dans public.
  - Je me retrouve donc à faire des opérations custom pour avoir CREATE EXTENSION ... SCHEMA public.
    - Mais de ce que j'ai lu j'ai la vague impression qu'un comportement par défaut ne se fait pas chez moi.
    - Aussi, je suis obligé de tout namespacer, genre dans la déclaration de l'index au lieu de faire USING gist (... gist_trgm_ops), je doit

faire USING gist (... public.gist_trgm_ops), je ne sais pas si c'est normal, dans zoo en tout cas ça semble marcher sans.
- Pour l'index, brève tentative de créer une classe index qui puisse s'ajouter à Meta, django style, pour ne pas avoir à taper du RunSQL.
  - Peine perdue, rien n'est prévu pour surcharger les méthodes proprement dans ce coin là, tout change entre les versions de django.

Voilà, normalement j'ai fait attention et l'index est bien utilisé.

#13 - 30 septembre 2020 11:19 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

Fait pour la migration.

Et en jouant avec, j'ai découvert un nouveau comportement chelou sur lequel je me casse les dents : RunSQL(reverse_sql='DROP ...') ne droppe rien du tout chez moi, sans erreur aucune (j'enlève bien le IF EXISTS dans mes tests). Si je migrate_schemas -v3, que je copie colle exactement la commande qui est exécutée dans un shell psql, elle marche. J'ai écrit un truc pour préciser le schéma au cas où, DROP INDEX schema_name.index, rien non plus (pareil pour DROP FUNCTION).

#14 - 30 septembre 2020 11:29 - Benjamin Dauvergne

Je relis mais déjà, sous-classe BaseUserSerializer en DuplicateSerializer pour juste y ajouter duplicate_distance, qu'on ait pas "duplicate_distance": null dans les APIs existantes sur les utilisateurs.

#15 - 30 septembre 2020 11:44 - Benjamin Dauvergne

Valentin Deniaud a écrit :

> Fait pour la migration.
>
> Et en jouant avec, j'ai découvert un nouveau comportement chelou sur lequel je me casse les dents : RunSQL(reverse_sql='DROP ...') ne droppe rien du tout chez moi, sans erreur aucune (j'enlève bien le IF EXISTS dans mes tests). Si je migrate_schemas -v3, que je copie colle exactement la commande qui est exécutée dans un shell psql, elle marche. J'ai écrit un truc pour préciser le schéma au cas où, DROP INDEX schema_name.index, rien non plus (pareil pour DROP FUNCTION).

C'est SafeExtensionOperation qui cache une erreur SQL mais les opérations ne s'exécutant pas dans une transaction individuelle, ça tue la migration (je pense), plutôt vérifier si l'extension est déjà présente via select * from pg_extension where extname = '<extname>'.

#16 - 30 septembre 2020 12:33 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

Valentin Deniaud a écrit :

> Fait pour la migration.

Non, j'avais très mal recopié ce qui était fait dans combo, patch corrigé pour permettre une discussion claire.

Cette approche de ne pas planter si l'extension ne s'installe pas vient de #40024, et fred sur le salon :

```
18:11:56 - fpeters: il y aura des endroits où l'utilisateur n'a pas les permissions pour créer l'extension, c'
est à gérer.
18:12:10 - fpeters: (en format minimal c'est ne pas planter la migration)
```

Mais en fait la situation n'est pas la même que dans combo, où la migration ne sert qu'à installer l'extension. Ici on l'installe et on l'utilise tout de suite après dans la création d'un index et d'une fonction, il ne sert donc à rien d'éviter de planter à la création de l'extension, vu que ça va planter juste après.

Alors la question c'est de quoi on a besoin et quelle situation on veut gérer.
Soit on fait l'hypothèse que cet endpoint ne sera utilisé que marginalement, notamment pas sur les systèmes où pg est vieux/mal configuré, et qu'il vaut mieux zapper la migration, entièrement. Si on veut en bénéficier par la suite, il faudra entrer les commandes de création d'extension et d'index et de fonction à la main. Downside, si on modifie l'index ou whatever dans une migration suivante, ça va devenir compliqué de rattraper le coup à la mano.
Soit on ne la fait pas, alors il me semble qu'il faut planter, laisser les gens aller installer l'extension, revenir jouer la migration et ça passe.

Benjamin Dauvergne a écrit :

> Valentin Deniaud a écrit :
>
> > RunSQL(reverse_sql='DROP ...') ne droppe rien du tout

> C'est SafeExtensionOperation qui cache une erreur SQL mais les opérations ne s'exécutant pas dans une transaction individuelle, ça tue la migration (je pense)

Bien joué c'était ça, merci !

> plutôt vérifier si l'extension est déjà présente via select * from pg_extension where extname = '<extname>'.

Ça par contre je ne vois pas trop le rapport. Le fix c'est juste de mettre le DROP EXTENSION à l'origine de l'erreur dans un transaction.atomic().

#17 - 30 septembre 2020 12:36 - Frédéric Péters

> Soit on ne la fait pas, alors il me semble qu'il faut planter, laisser les gens aller installer l'extension, revenir jouer la migration et ça passe.

Il faut commencer dès maintenant alors à contacter les endroits où on n'a pas les permissions suffisantes sur la db pour leur demander d'exécuter telles instructions, ça ne peut pas planter un jeudi de mise à jour.

#18 - 30 septembre 2020 13:30 - Benjamin Dauvergne

Frédéric Péters a écrit :

> > Soit on ne la fait pas, alors il me semble qu'il faut planter, laisser les gens aller installer l'extension, revenir jouer la migration et ça passe.
>
> Il faut commencer dès maintenant alors à contacter les endroits où on n'a pas les permissions suffisantes sur la db pour leur demander d'exécuter telles instructions, ça ne peut pas planter un jeudi de mise à jour.

Le souci c'est que sur combo il me semble que l'ajout des index/extensions vont juste jouer sur les performances, ici si on a pas les nouveaux opérateurs trigram le code ne marche juste pas, et on pourra pas la rejouer une fois qu'elle sera passée sans défaire des migrations qui viendraient après, il faudra joué à la main toutes les commandes et on se prendra le problème à chaque fois dans le futur pour des nouveaux tenant (mais pas au moment d'une mise en prod sûr). Je n'ai pas vraiment de solution mais on ne peut juste pas ignorer des migrations qui foirent dans tous les cas, sinon on le ferait déjà tout le temps.

Valentin comme je ne vois pas de moyen d'avancer en faisant plaisir à tout le monde, je dirai de faire la totalité de la migration en une seule opération manuelle (database_forward()) d'ignorer toutes les erreurs (même sur la création de l'index ou de la fonction) pour ne pas tout péter un jeudi et d'accepter que dans ce cas find_duplicates() plantera, si on se met à l'utiliser sur ces installations on saura d'où ça vient. Mais il faudra donc donner tout le code de la migration à exécuter à la main, pas juste le CREATE EXTENSION....

À noter qu'il me semble qu'on peut poser des questions pendant les migrations ("Cannot create extension pg_trgm and index for duplicate search, do you want to continue ?") ça me semble acceptable pour un jeudi et ça évite que ça passe inaperçu à des endroits où on peut corriger le souci immédiatement.

#19 - 30 septembre 2020 13:58 - Frédéric Péters

> À noter qu'il me semble qu'on peut poser des questions pendant les migrations ("Cannot create extension pg_trgm and index for duplicate search, do you want to continue ?") ça me semble acceptable pour un jeudi et ça évite que ça passe inaperçu à des endroits où on peut corriger le souci immédiatement.

Les migrations sont lancées via le job systemd qui ne posera pas de question.

#20 - 30 septembre 2020 16:30 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

OK j'ai fait un truc qui ignore les erreurs, modulo

Benjamin Dauvergne a écrit :

> faire la totalité de la migration en une seule opération manuelle (database_forward())

J'ai essayé de garder des opérations séparées en remplaçant RunSQL par un RunSQLIfExtension, c'est bizarre mais ça marche (prouvé par un test).

#21 - 30 septembre 2020 18:15 - Benjamin Dauvergne

```
            with transaction.atomic():
                try:
```

```
                    schema_editor.execute('CREATE EXTENSION IF NOT EXISTS %s SCHEMA public' % self.name)
                except (OperationalError, ProgrammingError):
```

Ça ne peut logiquement pas marcher, si tu interceptes une exception de l'ORM elle ne passera pas dans le contextmanager atomic() et donc ne provoquera pas l'envoie de la commande ROLLBACK[1].

[1] troisième exemple après https://docs.djangoproject.com/fr/3.1/topics/db/transactions/#django.db.transaction.atomic et le bloc d'aide qui suit

PS: ou plutôt, si ça marche, c'est qu'en fait le atomic() ne sert à rien.

#22 - 30 septembre 2020 18:27 - Valentin Deniaud

Benjamin Dauvergne a écrit :

> ne provoquera pas l'envoie de la commande ROLLBACK[1].
>
> [1] troisième exemple après https://docs.djangoproject.com/fr/3.1/topics/db/transactions/#django.db.transaction.atomic et le bloc d'aide qui suit

Où l'on lit

```
This is mostly a concern for DatabaseError and its subclasses such as IntegrityError. After such an error, the
 transaction is broken and Django will perform a rollback at the end of the atomic block. If you attempt to ru
n database queries before the rollback happens, Django will raise a TransactionManagementError.
```

Ici on a rien à rollback d'une part, rien à exécuter ensuite d'autre part, il me semble qu'on est pas concerné.
Btw cette partie a tourné dans combo sans problème, c'est juste un copier coller de #40024.

#23 - 30 septembre 2020 18:27 - Valentin Deniaud

Benjamin Dauvergne a écrit :

> PS: ou plutôt, si ça marche, c'est qu'en fait le atomic() ne sert à rien.

Il sert pour la InternalError je pense.

#24 - 01 octobre 2020 11:46 - Valentin Deniaud

- Fichier 0001-api-add-find-duplicate-users-endpoint-46424.patch ajouté

Rebasé sur #46988 et ajout de tests qui l'utilise, voir #46988#note-3 pour les comparaisons de temps avec/sans index, qui prouvent que l'index marche.

#25 - 01 octobre 2020 12:46 - Benjamin Dauvergne

- Statut changé de Solution proposée à Solution validée

#26 - 01 octobre 2020 13:30 - Valentin Deniaud

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

```
commit e6b2e5dbf4e956461d36528cae4c0760edd50e31
Author: Valentin Deniaud <vdeniaud@entrouvert.com>
Date:   Mon Sep 21 15:11:52 2020 +0200

    api: add find duplicate users endpoint (#46424)
```

#27 - 03 octobre 2020 11:16 - Frédéric Péters

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

Fichiers

0001-api-add-find-duplicate-users-endpoint-46424.patch	8,63 ko	22 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	8,84 ko	24 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	13,9 ko	29 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	13,8 ko	30 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	14,1 ko	30 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	15,8 ko	30 septembre 2020	Valentin Deniaud
0001-api-add-find-duplicate-users-endpoint-46424.patch	17 ko	01 octobre 2020	Valentin Deniaud