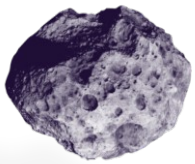
MAKINA CORPUS

GEO TREK



Synthèse 2J d'étude sur les performances

API v2 / Rando V3

Septembre 2022

Paris

Nantes

Toulouse

Bruxelles

Siège social
11 rue du Marchix
44000 Nantes

Makina Corpus

Établissement de Toulouse
52 Rue Jacques Babinet
31100 Toulouse

+33 (0)9 70 33 21 50

MAKINA CORPUS

Table des matières

1 Contexte

Les différentes itérations successives sur le développement de l'api v2 ont révélé des faiblesses sur la rapidité d'accès aux données.

Utilisées en direct par des serveurs comme Geotrek-rando v3, ces données doivent être rapidement accessibles. Une session de développement de 10 jours a été commandé par le Parc National des Écrins afin de commencer les travaux. Ces 10 j ont débuté par 2 j d'analyse dont voici la restitution.

2 Rappel des problématiques

- Vitesse et accessibilité de l'API V2
- Volumétrie des données, notamment dans le cas des itinérances
- Utilisation de l'API par Geotrek-rando V3

Suite à cette étude de 2 jours, nous allons lister ici tous les points à prendre en compte pour définir la feuille de route des développements :

2.1 Les points faibles actuels de l'API

- Calculs à chaque accès
  - Il n'y a pas de stratégie de cache. Chaque appel de l'API sollicite le serveur Geotrek-admin, avec des calculs plus ou moins complexe. Par exemple, le calcul de la liste des villes traversées dans l'ordre est non négligeable, surtout pour une itinérance. De plus, par défaut, la configuration de Geotrek-admin réserve seulement 4 processus pour l'API (interne et V2). Lors d'appels successifs, une file d'attente se crée au niveau du serveur, et les réponses sont de plus en plus longues à arriver.
- Optimisation des requêtes SQL
  - Certains points d'accès de l'API nécessitent d'être optimisés afin de mieux gérer les jointures avec l'ORM Django et diminuer le nombre de requêtes SQL exécutées.
  - La pagination nécessite de ré-executer la requête SQL afin de déterminer le nombre d'élément à inclure dans la réponse. Dans le cas de calculs complexe (zones sensibles décrit plus loin), cela peut doubler le temps de réponse.
- Points d'accès aux calculs complexes.
  - Les zones sensibles : Le calcul pour déterminer les zones sensibles présentes (moins de 500m par défaut) d'un itinéraire peut être extrêmement long en fonction du nombre de zones sensibles, de leur complexité géographique et de la longueur de l'itinéraire / itinérance. De plus, l'inclusion du nombre d'élément dans la pagination des réponses double le temps d'exécution qui devient trop important.
  - Les thèmes : La liste des thèmes est calculée à partir de plusieurs requêtes très longues, car il s'agit de la somme des thèmes utilisés dans des éléments publiés issus de plusieurs tables de données. Plusieurs niveaux de complexités s'ajoutent, la notion de « publié »

est déjà complexe en fonction des langues. Le cas de l'itinérance rajoute de la complexité pour le calculs des thèmes des itinéraires, et idem pour les activités outdoor ou la structure SQL de la hiérarchie employée (liste adjacente) est très complexe en lecture.

2.2 Côté Geotrek-admin

L'accès à une itinérance dans Geotrek admin devient de plus en plus long en fonction de plusieurs facteurs :

- la longueur de l'itinérance
- le nombre de modules activés (zones sensibles, outdoor …)
- le nombre d'éléments liées (zonages, POI, contenus touristiques etc
- le nombre e ces éléments à afficher sur la carte

Le système d'affichage de Geotrek-admin se base sur l'inclusion automatique des modules Mapentity, qui par défaut, n'optimise pas les requêtes SQL et ne cache pas les réponses. Plus il y a de modules et de données, plus c'est long.

Contrairement à l'API qui ne renvoie que la liste des villes traversées dans l'ordre, l'admin renvoi la liste des Villes, Secteurs, Zones réglementaires traversées dans l'ordre. Ces calculs rallongent considérablement l'affichage notamment sur les instances utilisant beaucoup de secteurs et de zones réglementaires.

2.3 Le comportement de Geotrek-rando v3

Voici quelques éléments de réflexion sur le comportement de Geotrek-rando v3 sur la consultation d'une itinérance :

- Appel de l'itinérance
- Appel de sa géométrie
- Appel de chaque étape
- Appel de la géométrie de chaque étape
- Appel des pages statiques
- Appel de toutes les informations nécessaires à l'envoi d'un signalement (4 Appels API)

J'ai remarqué que :

- Chaque point d'accès d'API est appelé 2 fois, du soit à une mauvaise conception, soit à un bug de Next JS (https://github.com/vercel/next.js/issues/37715)

- Chaque appel est mal formaté (pas de « / » à la fin), ce qui déclenche une redirection côté serveurs, mais qui prend du temps dû en partie à la saturation et à la file d'attente formée sur le serveur. Donc chaque appel déclenche 4 appels API
- Les informations et les géométries pourraient être appelées en même temps
- Avec un filtre supplémentaire sur les itinéraires, ou en utilisant le point d'accès /*api*/v2/tour, toutes les informations des étapes pourraient être récupérées en un seul appel
- 4 appels (donc 16) d'API sont déclenchés pour effectuer un signalement, même si l'utilisateur ne souhaite pas signaler de problème sur le parcours.
- Les pages statiques sont chargées à chaque fois

3 Conclusion

D'après l'analyse effectuée, voici les actions à entreprendre, dont la pertinence et la priorité seront soumises à discussion:

1. Optimisation des requêtes SQL
    1. Passe sur toutes les jointures à optimiser par point d'accès API
    2. Zones sensibles : Requête ST_DWTHIN complexe. Soit remplacer par un ST_INTERSECTS, soit calculer et stocker le buffer des zones sensibles à l'ajout / edtion afin d'utiliser ST_INTERSECTS sur eux.
    3. Thèmes : garder en cache les liste des thèmes, mise à jour a chaque publication / dépublication

2. La mise en cache des données
    1. Mise en cache 'bas niveau' des éléments de zonages traversés dans l'ordre.
    2. Mise en cache des appels API
        1. clés de cache : md5 hexdigest de l'url, paramètres d'appels triés dans l'ordre, date de modifications
        2. listes : Dernière date de modification → requêtes SQL supplémentaires à chaque accès
        3. détail : Date de modification de l'objet (et des objets liés inclus – pièces jointes par exemple) → requêtes SQL supplémentaires à chaque appel
        4. cas des thèmes : garder en cache les listes des thèmes publiés par table, mise à jour a chaque publication / dépublication
    3. Problématiques :
        1. Date de modification
            - Toutes les tables n'ont pas de date d'ajout / date de modification. Il va falloir généraliser ça et mettre à jour tous les fichiers SQL pour leur affecter une date par défaut au niveau SQL (https://github.com/GeotrekCE/Geotrek-admin/issues/3008)
            - Pour les tables gérant le 'no delete' (données non supprimées mais marquées comme supprimées en base de données), la dernière date de

modification est mise à jour lors d'une suppression. Pour les autres, elle peut ne pas refléter la suppression d'un objet et donc ne pas invalider le cache. Il va falloir soit généraliser cet usage, soit trouver une alternative, par exemple en créant une nouvelle table contenant la date de dernière suppression de chaque table de données, et déterminer si cela doit être mis à jour par déclencheur SQL ou dans le code python.

- La structure des certains point d'accès API (ex : */api/*v2/pois/?trek=1000) ne permettra pas de détecter un changement en cas de modification de l'itinéraire en question (géométrie changée ou POIS exclus) car seul son ID sera utilisé pour la mise en cache. Pour que cela fonctionne, il faudra impérativement créer de nouveaux points d'accès (ex : /api/v2/treks/1000/pois/) et qu'ils soient utilisés par Geotrek-rando. C'est complexe à mettre en place, mais ça permettra aussi d'éventuellement supprimer la pagination, et dans ce cas le comptage des éléments pourra aussi être optimisé (compter les éléments dans la requête SQL déjà exécutée au lieu de la ré-exécuter)

3. Modifications côté Geotrek-rando
    1. bien formater les requêtes pour utiliser un « / » à la fin des urls
    2. trouver / corriger le bug des doubles appels API
    3. Étudier la faisabilité de récupérer les geojson et les propriétés en un seul appel.
    4. Étudier la possibilité d'optimiser la récupération des infos de signalements
    5. Étudier la possibilité d'optimiser la récupération des infos des pages statiques
    6. Étudier la possibilité de récupérer les étapes d'itinérances en un seul appel
    7. Mettre en place les nouveaux appels pour les objets liés sur les points d'accès d'api en cache (/api/v2/trek/1000/pois/)

4 Quelques alertes

- Il faut prioriser et définir quelles actions peuvent être réalisées dans les 8j restants

5 Suite de l'étude

Nous allons attendre vos retours sur les conclusions afin de déterminer le plan d'action.