Compte-rendu – Re2o – Réunion du 16/05/2018

La réunion a eu lieu sur https://meet.jit.si et s'est plutôt bien passé dans l'ensemble malgré des soucis de connexion assez réguliers.

Les personnes présentes lors de la réunion :

- Blupon
- Chirac
- Nanoy
- Klafyvel
- Volgarr42 (quand son téléphone voulait bien)
- MoaMoaK

L'ordre du jour est le suivant :

- Rapide revue des **nouveautés** depuis la dernière réunion (Nocturnes FedeRez 2017)
- Révision du **processus de développement**
- Présentation de la nouvelle **API**
- Présentation de la **traduction**
- Présentation des **tests**
- Présentation et décisions sur la **documentation**
- Redéfinition de **langue** du projet
- Divers sujets plus petits pour finir (Copyright, Groupe Gitlab, Découpage du code en fichiers plus petits, dashboard, Gitlab CI, RGPD, Fichier de requirements, Faire des apps indépendantes
- S'il reste du temps, passage en revue des issues.

Les nouveautés depuis les Nocturnes FedeRez 2017

Le changement majeur est la **refonte des ACL**. Les ACL permettent de mettre en place un système de permissions plus fin et ciblé. Il met à la disposition un grand nombre de fonctions, de décorateurs et de templatetags qui permettent de vérifier si un utilisateur a accès à telle ou telle fonctionnalité ou action. Ainsi, on a pu enlever tous les groupes et permissions qui jusque-là étaient hardcodées.

Le système de groupes et maintenant beaucoup plus flexible et dynamique grâce à l'introduction des ACL. Cependant la sélection des permissions reste assez moche et demanderait d'y retravailler le design et la façon de choisir les permissions. De plus, cette feature nécessiterait de la documentation pour expliquer les effets de chacune des permissions.

Au-delà de ces changements, il s'agit de modifications mineures ou de modification qui seront détaillés par la suite de ce CR. A noter : le payement online (avec Comnpay), l'ajout d'une page « about », le redesign de la navbar, le redesign de la page de profil, l'ajout des informations sur les batiments et baie de brassage dans la topologie, l'ajout de bannière FB et Twitter sur la page d'acceuil.

Le processus de développement

Dans l'ensemble tout va bien, les gens sont à peu près content de la façon de fonctionner avec le Gitlab.

Il faut juste que les gens essayent de faire l'effort de remplir et de maintenir à jour **le board des issues** (https://gitlab.federez.net/federez/re2o/boards) et de s'assigner les issues dont ils s'occupent. Cela permetterait de savoir qui travaille sur quoi actuellement. Cependant le but n'est pas d'y mettre trop d'efforts, juste que ce serait mieux de le faire.

De même une bonne idée serait d'utiliser les **milestones** (https://gitlab.federez.net/federez/re2o/milestones) pour fixer les objectifs importants mais le projet n'est pas encore prêt à fonctionner avec ça : pas de politique de versions défnie.

Les merge requests fonctionnent bien mais il a été demandé de ralentir un peu les merges pour laisser le temps à plus de gens de donner leur avis (et donc éviter de merger des bugs). A partir de maintenant, un MR peut être mergée si et seulement si :

1. Toutes les discussions sont résoutes (pas juste bypass) et personne n'est mécontent
2. L'une des conditions suivante est remplie :
    - C'est un hotfix (= doit être mergé rapidement pour fixer un bug important) et 1 personne au moins valide la modification
    - 3 personnes valident la modification
    - Une semaine s'est passée depuis la demande de merge et au moins 1 personne a validé la modification.

Pour valider une merge request, on peut utiliser **les pouces vers le haut/bas** qui ont l'avantage d'être rapidement visible, on peut le retirer et le rajouter à volonté et on ne peut en mettre qu'un par personne.

Par ailleurs, pour mettre à jour des vieilles branches, on rappelle qu'il est préférable de **les rebaser plutôt que d'y merger master** (s'il est possible de réécrire l'histoire Git sans que cela pose de problème ailleurs). En effet, bien que le résultat final soit le même, cela allège grandement l'historique et facilite la compréhension, et le graphe des commits est bien plus propre. Pour plus d'information, ce lien a été proposé : https://zestedesavoir.com/tutoriels/379/refaire-lhistoire-avec-git/

Le canal de communication principal est très bien sur IRC, mais il faudrait pouvoir afficher plus d'informations venant de Gitlab. Le Crans est en train d'essayer « Irker » qui peut être bien et plus facile à intégrer que le bot actuel. Sinon Metz a dev un bot Télégram qui réagit à des webhooks Gitlab. Donc si des gens du Crans ou de Metz pouvaient se pencher sur la question, ça pourrait faciliter la chose.

La nouvelle API

MoaMoaK à commencer à refaire complétement l'API pour qu'elle permette plus de chose et soit plus accessible. Le but est d'utiliser les fonctionnalités de Django-Rest-Framework (DRF) pour exposer facilement les modèles, gérer l'authentification et les permissions, ainsi qu'adopter les principes de REST.

L'API sera une **app Django indépendante** du reste du projet (dépendante des modèles mais pourra être ajoutée ou retirée sans conséquences). Elle expose des **endpoints pour lister, voir les détails, modifier, créer et supprimer tous les modèles du projet**. Grâce à DRF cela ne prend que quelques lignes par modèle. De plus on utilise aussi le système de **pagination** de DRF qui évite de surcharger le serveur avec une trop grosse requête. Dans un premier temps, les endpoints en seront accessibles qu'en « read-only » (lister et voir) même si le système de permissions devrait empêcher aux personnes non autorisées de faire des modifications.

En effet, DRF propose un **système de permission** qui peut être facilement adapté à nos besoins et aux ACL : on ajoute une permissions « API » qui est nécessaire pour accéder à l'api (quelques soit la requete) et de plus il faudrait être autorisé par le système d'ACL de re2o pour faire une action (= on peut modifier via l'api ssi les ACL l'autorise ssi on peut modifier via l'interface web)

L'authentification aussi a été revue, deux méthodes sont possibles :

- Via un cookie de session fournit par django-auth. Il s'agit du même cookie que pour accéder à l'interface web, ce qui signifie que cette méthode est très pratique pour parcourir l'API sur navigateur. Il suffit de se connecter normalement à Re2o et on a accès à l'API (si on en a les droits)
- Via un token d'authenfication. Cette méthode est plus destinée à un script qui utilise l'API car plus simple à gérer que les cookies. On peut obtenir un token (en fournissant des identifiants valides) via une URL de l'API et après il suffit de fournir ce token dans l'en-tête de la requête pour être authentifié. Pour des raisons de sécurité, ce token expire après un certain temps (réglable via les settings, actuellement 24H) et doit être renouvelé.

Actuellement l'API n'intègre pas les fonctions actuelles de l'API car elles doivent être revues pour s'intégrer proprement au système actuel d'API et utiliser mieux les fonctionnalités de DRF (et enlever les doublons avec la nouvelle API). Comme cela risque fortement de casser l'API actuelle, ce sera l'occasion de **retravailler complétement re2o-tools** pour y intégrer le nouveau système d'auth, de nouvelles architectures des modules, … Le but étant d'avoir une base plus propre et de réécrire certains modules plus proprement.

La traduction

MoaMoaK a commencé la traduction de Re2o, la partie faite (l'app cotisation) a déjà été mergée et les gens trouvent que ça marche plutôt bien. Le choix de la langue est fait par priorité :

1. Par un cookie de langue qui est généré quand on choisit une langue en haut à droite
2. Par le paramètre Accept-Language envoyé par le navigateur
3. La langue du code (définit sur « en » dans les settings)

Le principe est assez simple :

1. On marque toutes les strings de l'interface avec des fonctions de type getttext, gettext_lazy dans les fichiers pythons
2. On marque toutes les strings de l'interface avec des templatetags de type {% trans %}, {% transblock %} dans les templates
3. On lance python3 manage.py makemessages –l <lang_code> pour générer les fichiers .po d'une langue spécifique à partir des strings marquées
4. On remplit les fichiers .po avec la traduction dans la langue concernée
5. On lance python3 manage.py compilemessages pour compiler les fichiers de traduction et crée des fichiers .mo

Le problème de cette méthode est qu'elle **prend beaucoup de temps à faire**. C'est pourquoi ce serait bien d'avoir plus d'une personne pour le faire. Volgarr42 s'est proposé pour filer un coup de main.

Les tests

Chirac a commencé à travailler sur les tests Django. L'app users est quasiment entièrement testée. C'est une tâche très répétitive et longue. C'est pourquoi Klafyvel évoque l'idée de faire du **Test Driven Developpment** (TDD), c'est-à-dire d'écrire des tests avant de coder une nouvelle feature. Ou au moins qu'à partir de maintenant tout le monde écrive des tests lorsqu'il touche à un morceau de code (nouvelle feature -> test pour ça, bug fix -> test pour ça).

Chirac va essayer de reprendre ce projet sous peu mais ce serait bien de filer un coup de main et que chacun écrive les tests **au fur et à mesure**.

La documentation

La documentation du code est longue et pénible à faire en une seule fois, c'est pourquoi, décision a été prise, tout comme pour les tests, qu'à partir de maintenant chacun essaye de compléter **au fur-et-à-mesure** les docstrings des endroits qu'il modifie.

Pour écrire la documentation, la syntaxe n'a pas encore été choisie mais on se dirige soit vers « reST » soit vers « Google ». Un vote est en cours pour déterminer ce qu'on va adopter. En tout cas il faut en adopter une rapidement parce que ce sera pratique pour générer de la documentation facilement.

Un wiki a été créé récement (https://gitlab.federez.net/federez/re2o/wikis/home), il reste plus qu'à le remplir. Pour la partie technique le plus simple est **d'utiliser un outil** (Sphinx a priori) pour générer la doc à partir des docstrings (d'où l'interêt de les écrire proprement). Pour la partie utilisation, c'est surtout un effort important à fournir pour l'écrire à la main. C'est pourquoi là encore, **le mieux est de tout faire au fur-et-à-mesure**. Quand quelqu'un touche à une partie, il écrit la documentation sur comment utiliser en même temps et comme ça on finira par avoir quelque chose de conséquent.

La langue du code

Le code : En anglais

La documentation (wiki, docstrings et commentaires) : En anglais

Le frontend : En anglais (tout passe par la traduction)

Le dev dans Gitlab (issues, MR, commit, …) : On s'en fout (donc probablement en français)

Les réunions : On s'en fout (donc probablement en français)

Divers sujets de discussions

Ce serait plus simple d'avoir **un copyright au nom de Re2o**, ça évite de devoir le changer à chaque nouvel auteur et permetterait d'avoir exactement le même en-têtes. Il faudrait demander sur #FedeRez comment c'est faisable en toute légalité.

La license serait plus adapté à être du **AGPL** mais il faut encore identifier clairement les différences et les conséquences avant de faire ça.

Le projet gagnerait en propreté à être bougé dans **un groupe Gitlab dédié** (« Re2o ») dans lequel on pourrait réunir tous les projets existant ou à venir qui concernent Re2o. Cela éviterait de donner des privilèges sur tous les autres projets FedeRez juste pour pouvoir dev sur Re2o.

Pour alléger le code, une idée est **de découper les fichiers (modules) en dossiers (packages)**. Par exemple users/models.py deviendrait users/models/{user.py|serviceUser.py|….}. Cependant cette idée bien qu'entièrement innofensive en théorie, n'est pas une priorité et va modifier une très grande partie du code (= risque de caser des choses). On attendra donc d'avoir des tests avant de faire ce genre de modifications.

Le **dashboard** serait vraiment une bonne idée à reprendre.

La **CI du gitlab** ne sert à rien actuellement, elle valide même quand pylint relève des erreurs. Il faudrait donc la réétudier et y ajouter les tests à venir.

Pour supporter le **RGPD** (oui on ne sera pas en respect à temps), on a décidé d'utiliser l'API pour sortir toutes les données. Un bouton sur le profil du user (visible uniquement par lui) le redirigera vers une URL spéciale de l'API.

Ce serait une bonne idée d'ajouter **un fichier « requirements »** pour les paquets : pip supporte très bien ce genre d'option (pip –r req.txt) et ce serait intéressant de l'adapter à APT. Cela permettrait d'avoir très rapidement la liste des paquets à installer et aussi de facilité par la suite la mise sous forme de package APT de Re2o.

Il pourrait être intéressant de rendre complétement **indépendantes de Re2o certaines portions du code** (les ACL en particuliers) pour pouvoir les utiliser dans d'autres projets. Toutefois l'idée proposée de d'avoir une app re2o-core (avec grosso modo les modèles), une app re2o-web (avec grosso modo les forms, les vues, les urls et les templates), une app re2o-radius (avec grosso modo l'interface du radius), … est refusée parce qu'elle ne correspond pas à la façon de faire et d'organiser le code de django.