

OPL Projet #1

REPUT GIT

Simon Decottignies - David Fitoussi



Table des matières

Introduction	4
Travail technique	5
But?	5
Principe	5
Algorithme	5
Ratio de confiance	5
Taille totale des commits	6
Nombre de PR total et ancienneté	6
Construction du classement	6
Implémentation	8
Architecture	9
Utilisation	10
Evaluation	12
Conclusion	14

Introduction

Sur de gros projets avec beaucoup de contributeurs les intégrateurs sont forcément amenés à recevoir plusieurs Pull Request (PR) par jour. Une Pull Request permet à un contributeur de proposer ses modifications à l'intégrateur. S'il est satisfait du travail de son contributeur il peut directement faire un merge de ses modifications. Mais une PR peut aussi être discutée pendant plusieurs semaines ou tout simplement non vérifiée le jour même, l'intégrateur se retrouve donc souvent avec beaucoup de Pull Request à gérer et ne sait pas forcément par laquelle commencer.

Il perd donc beaucoup de temps sur cette page de PR. Certaines, proposées il y a longtemps, peuvent être oubliées. D'autres proposées par des contributeurs sérieux et dignes de confiance pourrait être validés rapidement mais elles ne sont pas mise en avant. Ou alors des PR très petites, en terme de nombre de lignes de code, pourraient être vérifiées avant toutes les autres pour réduire le nombre de PR et avoir une vision plus claire du travail qu'il reste à faire. Il faut donc améliorer l'expérience de l'intégrateur concernant les PR de son ou de ses projets.

La solution que nous allons vous présenter sera donc un outil de gestion des PR pour un intégrateur. En récupérant tous les PR d'un projet et en les présentant d'une manière nouvelle et plus pratique que celle de GitHub, le travail de l'intégrateur sera plus simple.

1. Travail technique

1.1. But

Le but premier de cet outil est de faire gagner du temps à l'intégrateur, du temps qu'il pourra utiliser pour d'autres tâches et donc être plus productif dans son travail. C'est dans la présentation des PR et dans la façon de les ordonner que notre outil se veut utile.

1.2. Principe

Pour gérer ses PR l'intégrateur a besoin de plus qu'une simple liste triée de la plus récente à la plus ancienne comme sur GitHub. Il lui faut une liste qui priorise certaines PR par rapport à d'autres et qui justifie cet ordre d'apparition. Le plus gros du travail est donc de trouver un système qui trie toutes les PR d'un projet.

Les critères de tri doivent être indiqués à l'utilisateur pour qu'il sache pourquoi une PR est classée première et une autre dernière. Bien sûr ce tri sera un tri de base, et l'utilisateur pourra trier lui même toutes les PR.

1.3. Algorithme

Le tri est effectué selon une méthode très simple. Lorsque nous récupérons les PR, nous faisons en sorte de récupérer l'ensemble de celles-ci, pas seulement celles en état "open". Cela nous permet d'obtenir l'historique de toutes les PR du projet et de toutes les PR des contributeurs. Suite à cela nous utilisons toutes ces données pour trier toutes le PR du projet. Plusieurs critères entre en jeu.

1.3.1. Ratio de confiance

Le premier critère est le ratio du contributeur ou le taux de confiance. Ce critère se base sur les PR acceptées et refusées du contributeur sur le projet courant. Pour le calculer, on utilise une formule simple :

$$ratioConfiance = \frac{PrOK}{PrOK + PrNOK}$$

PrOK : nombre de Pull Request validées dans l'historique du contributeur

PrNOK : nombre de Pull Request refusées dans l'historique du contributeur

On obtient donc un nombre compris entre 0 et 1 qui nous permet de classer la performance d'un contributeur. Plus ce nombre se rapproche de 1 plus l'intégrateur peut faire confiance au contributeur et à son PR car dans le passé il s'est montré digne de confiance.

1.3.2. Taille totale des commits

Suite à cela nous prenons également en compte la taille totale des commits. Pour chaque contributeurs ayant une PR ouverte sur le projet, nous regardons sur l'ensemble de ses commits effectués si ce dernier a eu une contribution conséquente ou non.

$$qualitéCommit = (ajoutLignes + suppLignes)$$

ajoutLignes: nombre d'ajouts de lignes

suppLignes: nombre de suppressions de lignes

Ce critère déterminera quel contributeur est le plus actif sur ce projet et donc lequel a le plus de connaissances sur celui-ci. On a toujours ici une notion de confiance, s'il connaît bien le projet l'intégrateur pourra passer moins de temps à vérifier si son PR est pertinent ou non.

1.3.3. Nombre de PR total et ancienneté

Nous regardons également le nombre total de PR effectuées sur le projet par le contributeur. Ce critère ressemble au précédent mais ne regarde pas la taille du PR. Avec ces deux critères nous ne pénalisons pas un contributeur qui fait des gros PR (en terme de nombre lignes) face à un contributeur qui fait plusieurs des plus petits PR, et inversement.

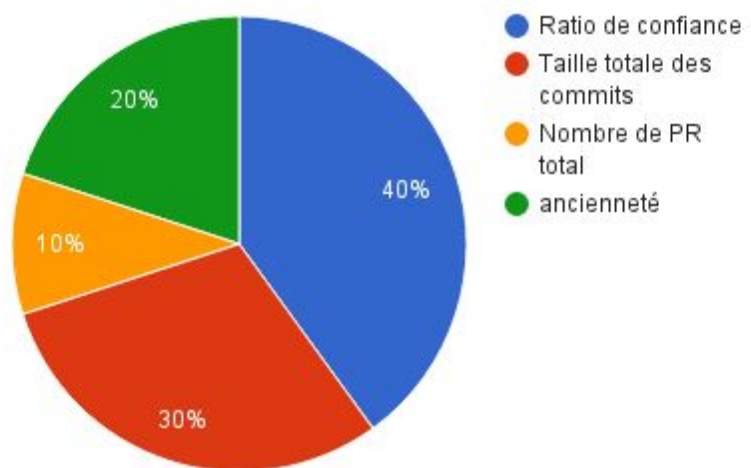
Enfin nous regardons également la date de dernière update de la PR, et nous en sortons un classement allant de la plus vieille à la plus récente. C'est ce critère qui permettra de remonter les "*vieilles PR oubliées*" dans le classement et qui permettra à l'intégrateur de ne pas garder les mêmes PR pendant des mois tout en bas de la liste.

1.3.4. Construction du classement

Après avoir effectué tout ce travail préparatoire, on est en mesure pour chaque critères de sortir une valeur maximum (Ratio de confiance maximum, taille totale des commits maximum, nombre total de PR maximum, Date maximum). Cette valeur maximum nous permet de comparer toutes les PR entre elles et d'appliquer notre formule finale pour ensuite trier les PR.

Nous avons posé comme base un pourcentage s'appliquant aux différents critères :

- ratio de confiance : 40% du score total
- taille totale des commits : 30% du score total
- ancienneté : 20% du score total
- nombre de PR total : 10% du score total



(Fig 1) Graphique de la répartition des différents critères de tri

Ces pourcentages sont là pour classer les différents critères par niveau d'importance. Ici nous considérons que le ratio de confiance doit être vu comme critère le plus significatif par le tri.

Avec cette répartition nous avons établi la formule suivante :

$$score = \left(\frac{ratio}{ratioMax} * 40 \right) + \left(\frac{tailleCommit}{tailleCommitMax} * 30 \right) + \left(\frac{anciennete}{ancienneteMax} * 20 \right) + \left(\frac{PrTotal}{PrTotalMax} * 10 \right)$$

ratio : ratio de confiance

ratioMax : ratio de confiance maximum

tailleCommit : taille totale des commits

tailleCommitMax : taille totale des commits maximum

anciennete : date du dernier update du commit

ancienneteMax : date du commit le plus ancien

PrTotal : nombre de PR total du contributeur

PrTotalMax : nombre de PR total maximum

On obtient donc un score allant de 0 à 100, on se base donc sur ce score pour effectuer le tri. Plus le score d'un PR est élevé, plus elle sera haut dans la liste.

1.4. Implémentation

Les pages Web sont générées en HTML, CSS et Javascript avec le framework ExpressJS pour gérer plus facilement les différents Routes et utiliser des templates (Ici EJS).

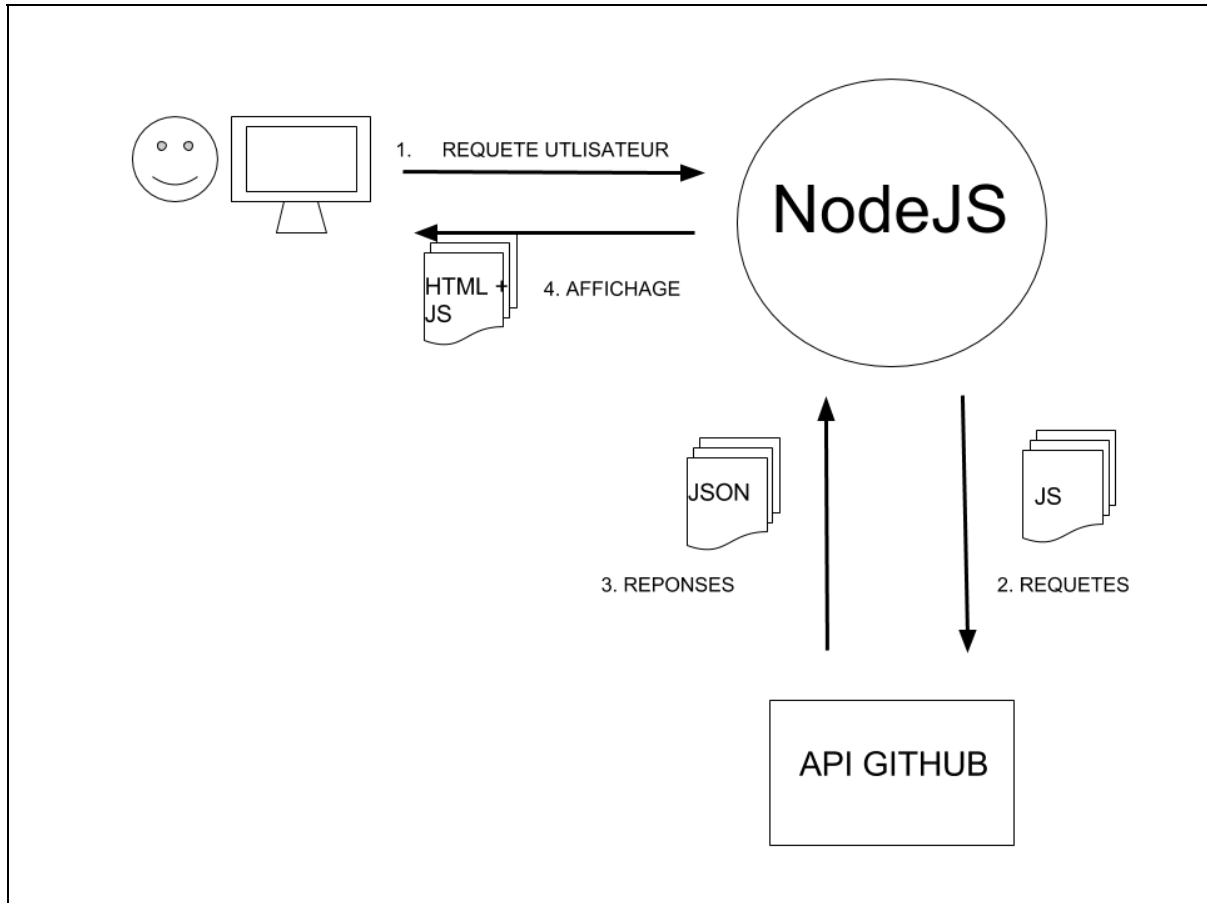
En plus de cela, nous avons incorporé plusieurs plugins issu de node tel que celui de Github proposé par Mike Deboer (<https://github.com/mikedeboer/node-github>) qui nous a permis de gérer plus facilement les appels que nous devons faire à l'Api GitHub.

Nous avons tout d'abord mis en place la base projet via NodeJS. Suite à cela, nous avons intégré l'ensemble des appels à l'api tout d'abord en JS uniquement puis par le module node.

Nous avons par la suite mis en place notre formule dans l'application. Après avoir testé l'application sur des projets de différentes tailles, nous avons mis en place les derniers éléments graphiques.

Le programme présente une complexité importante dû aux nombreux appels effectués sur l'API GitHub. En effet, l'ensemble des informations stockées sur le site sont issues des réponses à nos différentes requêtes. L'impossibilité de passer outre rend le programme dépendant de l'état des serveurs de GitHub ainsi que la taille des repos contenant de nombreuses PR influence sur les performances de l'application.

1.5. Architecture



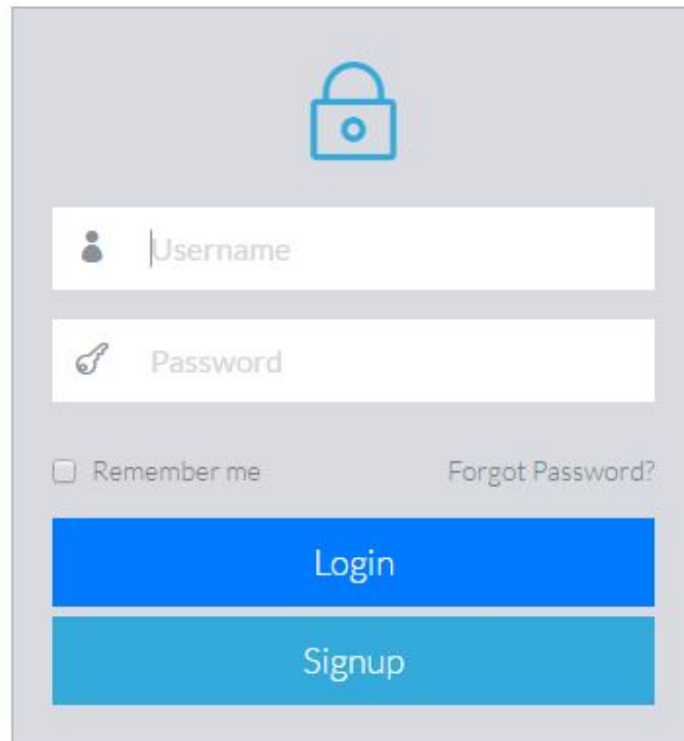
(Fig 2) Schéma de l'architecture de l'outil

Après avoir installé NodeJS et ExpressJS, il suffit d'écrire la commande "npm install" dans le dossier du projet puis "npm start" afin de lancer le serveur de l'application.

Une fois le serveur Node lancés, l'accès à la plateforme se fait simplement via l'adresse localhost:3001 (sauf modifications externes)

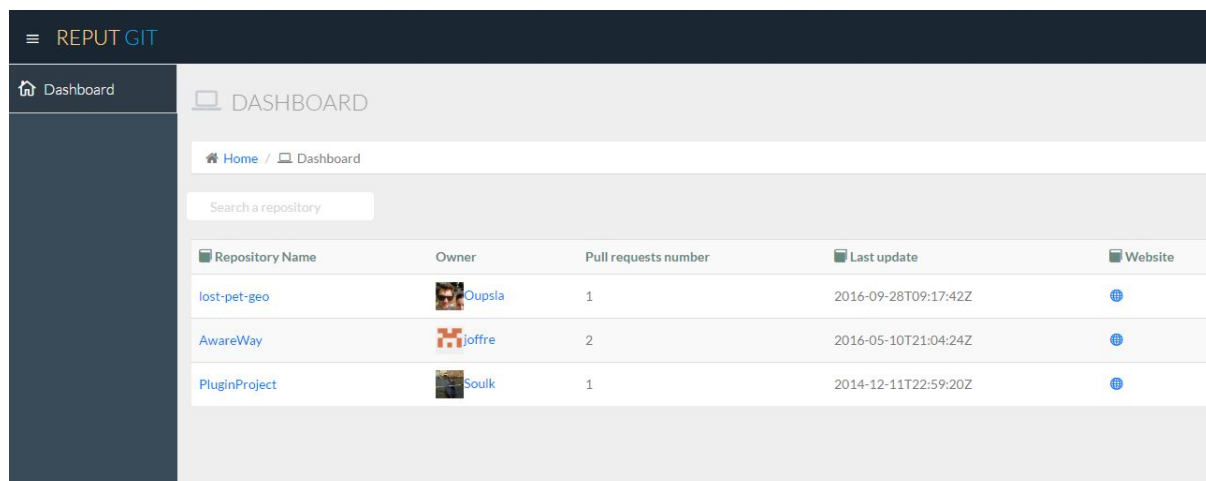
1.6. Utilisation

L'utilisation de l'outil est très simple. La première page est celle de l'authentification (*Fig 3*), une étape obligatoire. Elle est obligatoire car l'API Github n'autorise que 60 requêtes par heure si on ne se connecte pas avec ses identifiants Github, un nombre beaucoup trop faible.

The image shows a login page with a light gray background. At the top center is a blue padlock icon. Below it are two white input fields. The first field has a gray user icon on the left and the placeholder text 'Username'. The second field has a gray key icon on the left and the placeholder text 'Password'. Below these fields are two links: 'Remember me' with an unchecked checkbox to its left, and 'Forgot Password?'. At the bottom are two large buttons: a blue 'Login' button and a teal 'Signup' button.

(Fig 3) Page de login

Si votre authentification est un succès vous êtes redirigé vers la Homepage (*Fig 4*). Après avoir récolté tous les répertoires Github de votre compte contenant au moins une PR, la page vous les affiche. Vous avez un accès direct vers la page Github du répertoire grâce à la colonne "Web", vers le compte utilisateur du propriétaire grâce à la colonne "Owner" et vers la liste des PR (*Fig 5*) grâce à la colonne "Repository name".



(Fig 4) Homepage

La liste du PR (Fig 5) est donc le résultat du tri (rf. 1.3.) effectué pendant le chargement de cette page. Le score de chaque PR est affiché et une justification de ce score est offert à l'utilisateur. Si le tri ne convient pas à l'intégrateur la liste de PR offre un système de tri interne, par colonne. Une recherche parmi toutes les colonnes de toutes les lignes est aussi incluse. Ainsi chacune des PR est trouvable facilement.

Show	10	entries	Search:	
Pull Request (open)	User	Total pull requests(closed or merged)	Last update	Score*
Upgrade Quartz to 2.2.3	arledesma - (50%)	2	2014-06-24T01:50:54Z	69
Add Visual Studio 2013 3rd party test framework service reference	arledesma - (50%)	2	2014-06-24T01:39:16Z	69
Fix #25	schulz3000 - (new%)	new	2016-02-28T17:12:53Z	20
Upgraded .NET version to 4.5 and Quartz version to 2.2.3; did not change...	jkav - (new%)	new	2014-06-13T13:55:03Z	19
Showing 1 to 4 of 4 entries			Previous	1 Next
<p>* Score calculé via une formule comprenant le nombre de PR, la date d'update de la PR, les statistiques d'ajout et de suppression des différents commits et le ratio du collaborateur. Ce score peut atteindre la note maximal de 100, les PR disposant du meilleur score sont celles susceptibles d'être intégrés plus rapidement. A titre d'indice voici le ratio des différentes catégories :</p> <ul style="list-style-type: none"> -Nombre de pr : 10% -Date d'update : 20% -Statistiques d'ajout et de suppression : 30% -Ratio du collaborateur : 40% 				

(Fig 5) Liste des Pull request

L'outil est donc simple d'utilisation. Le but étant de faire gagner des temps à l'intégrateur il ne fallait pas proposer quelque chose de compliqué et qui au final demande plus de temps de compréhension que ne libère du temps à l'utilisateur.

2. Evaluation

Notre évaluation consiste à comparer l'outil de listage de Pull request de github avec le nôtre. Les critères de comparaisons sont le temps gagné, l'ergonomie et les PR qui seront traitées en premières avec ces deux différents outils.

Pour faire cette comparaison nous utilisons un gros projet, qui contient une somme importante de PR non traitées : <https://github.com/mozilla-mobile/firefox-ios/pulls>

Nous obtenons bien deux liste différemment triées (Fig 6 + Fig 7).

Show entries

Search:

Pull Request (open)	User	Total pull requests(closed or merged)	Last update	Score*
No bug - ui tests fixes	fluffyemily - (85%)	201	2016-07-05	89.34
Bug 1304035 - correctly detect whether current page is web page	fluffyemily - (85%)	201	2016-09-22	89.25
Swift 3.0 Migration	sleroux - (87%)	367	2016-09-27	65.79
Bug 1283742 - Share to wechat is invalid	lemonYLX - (100%)	7	2016-07-01	60.09
Bug 1293512 - Support QR Code on iOS	lemonYLX - (100%)	7	2016-09-02	60.02
Bug 1268939 Login records should be removed if user signs out	jacobwhite - (100%)	1	2016-05-24	59.94
Bug 1272229 Add Safari shortcuts alongside to the current Firefox keyboard shortcuts.	jacobwhite - (100%)	1	2016-06-07	59.93
Bug 1109806 - send links to desktop via Handoff	TETRA2000 - (100%)	1	2016-07-29	59.87
Bug 1304456 - Create a wrapper around WKWebView	thebnich - (82%)	223	2016-09-28	58.65
Bug 1306043 - Create UIWebView implementation of shim APIs	thebnich - (82%)	223	2016-09-29	58.64
Bug 1257574 Marketing Screenshots	st3fan - (81%)	212	2016-03-31	58.16
L10N Screenshot Improvements	st3fan - (81%)	212	2016-07-14	58.03
Fixes 1220378 - AVAudioEngine for AuralProgressBar keeps running inbackground	st3fan - (81%)	212	2016-07-21	58.03
Bug 1285664 - Add a text-to-speech mode to Reader View	varkor - (91%)	58	2016-08-23	57.83
Bug 1256470 - Enable Passcode/Touch ID for Private Browsing Mode	varkor - (91%)	58	2016-08-26	57.83
Bug 1238971 - Session navigation history (per tab) lost on session restoration	varkor - (91%)	58	2016-08-22	57.83
Bug 1288465 - Support tab reordering through the top tabs	varkor - (91%)	58	2016-08-26	57.83

(Fig 6) Résultat de l'outil avec un gros projet

40 Open ✓ 2,017 Closed		Author ▾	Labels ▾	Milestones ▾	Assignee ▾	Sort ▾
	Bug 1305574 - Implement Dismiss option for context menu in Activity Stream panel & Bug 1303731 - [iOS] AS highlights: Implement block list ✓					4
	#2143 opened 4 days ago by bkmunar					
	Bug 1283409 - Make sure keyboard is usable when orientation changes. ✓					
	#2141 opened 4 days ago by farhanpatel					
	Bug 1307440 - Clean up schemes. ✗					
	#2140 opened 4 days ago by farhanpatel					
	Bug 1304202 - Detect bookmarked state of highlights and update context menu accordingly ✗					6
	#2130 opened 12 days ago by bkmunar					
	Bug 1306043 - Create UIWebView implementation of shim APIs ✗	do not merge	experiment	wip		2
	#2128 opened 12 days ago by thebnich					
	Bug 1304456 - Create a wrapper around WKWebView ✗	do not merge	experiment			
	#2127 opened 14 days ago by thebnich					
	Bug 1253656 - Enable bookmark keyword searching ✗					13
	#2123 opened 14 days ago by mauryat					
	Bug 1304035 - correctly detect whether current page is web page ✓					
	#2113 opened 18 days ago by fluffymily					
	Bug 1243412 - Open Top Sites in new tabs using contextual press wip					4
	#2106 opened 19 days ago by mauryat					
	Bug #1303790 - Use en-US as the fallback locale ✓					
	#2099 opened 21 days ago by mkaply					
	Bug 1302815 - Use a master JSON file for locale search engines instead of list.txt ✗					17
	#2091 opened 26 days ago by mkaply					
	Bug 1299322 - Option to keep toolbars always open ✓					7
	#2078 opened on 4 Sep by squelart					
	Bug 1265700 - Handle WebViewControllerToolbar while scrolling ✓					3
	#2069 opened on 28 Aug by mauryat					
	Bug 1285664 - Add a text-to-speech mode to Reader View ✓					

(Fig 7) Liste des PR sur Github avec ce même projet

Les PR que l'intégrateur traitera en premières seront celles situées tout en haut de la liste. Il est donc intéressant de comparer celles de la première et de la deuxième liste.

Sur ce projet la première PR de la liste fournie par GitHub est donc forcément la plus récente. Celle de notre liste n'est elle pas la plus récente. On peut voir qu'elle a été mise en avant car justement la date de son dernier update est plus lointaine que les autres PR. Si on additionne le nombre de PR de son auteur et le bon ratio de confiance de celui-ci, on arrive à un score très élevé. Sur la liste de Github cette PR est classée à la 20ème place, elle est donc très loin derrière. On peut donc dire que notre outil va accélérer le traitement de cette PR, qui devenait très ancienne et qui aurait pû être oublié par l'intégrateur.

La troisième PR de notre liste elle était à la 22ème position dans la liste Github. On peut voir cette fois-ci que c'est le ratio de confiance élevé et le nombre de PR total de l'auteur qui ont mené cette PR à cette place, et non pas la date.

Suite à ces résultats nous pouvons confirmer que le contributeur va mettre plus de temps à traiter ces deux PR avec la liste GitHub qu'avec la nôtre. Pourtant ces deux PR sont intéressantes du point de vue de leurs caractéristiques. Elles présentent soit une ancienneté importante ou soit un contributeur digne de confiance. Du point de vue de l'ergonomie, notre recherche de PR dynamique est plus rapide que celle de Github et le tri par colonne est un peu plus pratique.

Une évaluation plus rigoureuse aurait demandée plus de temps. Des feedback d'intégrateurs sur un test de grande durée nous aurait aidé à affiner notre tri et à avoir des résultats plus justes.

Conclusion

Notre outil était au départ une solution pour faire gagner du temps aux intégrateurs pendant le traitement de leurs PR. Les résultats ont été satisfaisants mais la méthode de test et d'évaluation aurait mérité d'être plus fournie et sur une période plus longue. Ainsi de vrais intégrateurs nous auraient aidé à améliorer notre tri.

Ce projet nous a permis de développer nos compétences techniques dans les technologies du Web tel que NodeJS et le framework ExpressJS ainsi que le lien existant avec le module GitHub en accord avec le sujet du projet. En dehors du technique, il nous a permis de mettre en avant une recherche poussée et une réflexion importante de la manière de la mise en place d'une formule efficace.