Création d’un Parseur

Pré-requis

- **Disposer d’un environnement sous Linux**
- **Disposer d'une instance locale d’ezPAARSE à jour**
- **Avoir de bonnes notions de programmation et d’usage de la ligne de commande**
- **Avoir à disposition la page d’analyse ANG de la plateforme qui va être traitée (wiley dans les exemples ci-dessous).**

Avant toute chose, ouvrez un terminal et assurez-vous d'avoir accès à **Node.js**. Pour cela, placez-vous à la racine du projet ezPAARSE et lancez la commande suivante :

```
~/ezpaarse$ . ./bin/env
```

Cette commande charge l'environnement d'ezPAARSE et donne accès à la version de Node.js embarquée avec le logiciel. Placez-vous ensuite dans le dossier des plateformes et mettez-le à jour à l'aide de **Git** :

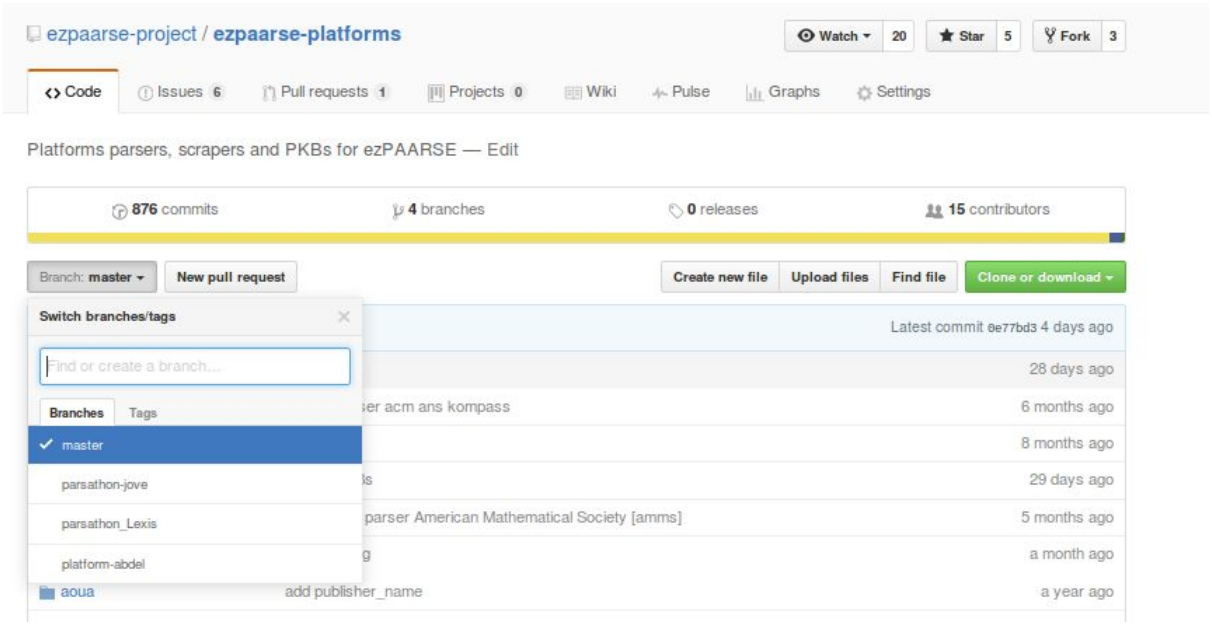
```
~/ezpaarse$ cd platforms
~/ezpaarse/platforms$ git pull
```

Enfin, lancez la commande suivante pour installer les outils nécessaires au travail sur les plateformes :

```
~/ezpaarse/platforms$ make install
```

Les ajouts ou modifications que nous allons effectuer nécessitent une validation, nous allons donc créer une **"branche"** que nous pourrons soumettre par la suite. Son nom devrait être en rapport avec les changements effectués, nous allons donc l'appeler **"wiley"**.

Vérifiez qu'une branche avec le même nom n'existe pas déjà, afin de ne pas créer de confusion. La liste des branches est visible sur la page du dépôt GitHub, onglet "branches".



Une fois assuré que le nom est disponible, créons la branche localement :

```
~/ezpaarse/platforms$ git checkout -b wiley
Basculement sur la nouvelle branche 'wiley'
```

Notre branche locale est maintenant créée et sélectionnée. Vous pouvez le vérifier avec la commande suivante :

```
~/ezpaarse/platforms$ git branch
  master
* wiley
```

La dernière étape consiste à publier cette nouvelle branche afin de la faire apparaître sur **GitHub**. Pour cela, lancez la commande suivante en prenant soin de remplacer **wiley** par le nom de la branche (vos identifiants GitHub vous seront demandés) :

```
~/ezpaarse/platforms$ git push --set-upstream origin wiley
Username for 'https://github.com': nojhamster
Password for 'https://nojhamster@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/ezpaarse-project/ezpaarse-platforms.git
 * [new branch]      wiley -> wiley
La branche 'wiley' est paramétrée pour suivre la branche distante 'wiley' depuis
'origin'.
```

I - Initialisation du parseur

Commencez par lancer un terminal. Depuis la racine du projet ezPAARSE, chargez l'environnement qui va permettre d'utiliser les commandes de l'application :

```
~/ezpaarse$ . ./bin/env
```

Une fois l'environnement chargé, placez-vous dans le dossier **platforms/** et lancez la commande d'initialisation d'un parseur :

```
~/ezpaarse$ cd platforms
~/ezpaarse/platforms$ make init
```

Le programme va vous poser une série de questions. Ces dernières vont servir à créer le dossier dédié à la plateforme, ainsi que la plupart des fichiers nécessaires à son fonctionnement.

Avant toute chose, on vous demande d'entrer l'URL de l'analyse, puis de confirmer l'identifiant automatiquement extrait de l'URL. Cette étape permet au programme de pré-remplir les questions suivantes en utilisant les informations déjà présentes sur Trello et ANG, ce qui accélère grandement la saisie. Nous donnons ici l'adresse de l'analyse de **Wiley** :

```
? What's the URL of the analysis?
http://analyses.ezpaarse.org/platforms/53ccdeecd5f7b93cbe52da1c
? Confirm analysis ID 53ccdeecd5f7b93cbe52da1c
 ✔ Fetch data from Trello
 ✔ Fetch analyses from ANG
```

Viennent ensuite le nom complet de la plateforme ainsi que son "nom de code". Ce dernier doit être court, simple, et en minuscule. S'il s'agit d'une plateforme avec un nom déjà concis (comme **Wiley**), vous pouvez l'utiliser comme nom de code. Dans le cas d'une plateforme avec un nom plus long, ou composé, nous recommandons d'utiliser des acronymes (exemple : The New England Journal of Medicine = nejm).

```
? What's the full name of the platform? Wiley
? What's the abbreviated name of the platform? wiley
```

On vous demande alors une petite description du parseur, et une liste de contributeurs susceptibles de pouvoir aider les personnes intéressées par la plateforme. Dans le cas de **Wiley**, il y a du monde ! Pour un contact plus aisé, vous pouvez renseigner les adresses mail professionnelles.

```
? How would you describe this parser? Recognizes the accesses to the platform Wiley
? Who's the contact for this parser? Dominique Lechaudel, Yannick Schurter, Stéphane
Gully, Thomas Jouneau, Abderrazzak Loukili, Rousseau, Ferez, Frederic Truong inist,
Anne-Claire, Sylvie Franck, laure Gouneaud, FOURNERET Philippe, Gerard Bruere, Zouari
Nicole, crepin, Pamphile ISCH, SRatsimandrava, Sonia Launay, Adeline Batailler, Jean
Loup Bruschet, eperez, Florence Barre, Bleux, Anne Cecile Bories, Farida Zerari,
Cécile Etesse, Thomas Porquet, coordnd Couperin, Odile Jullien Cottart, Cécile Girbon,
fxboffy, Laurent Lhuillier, Pascale Pauplin, Marine Rigeade
```

S'il est prévu que la plateforme utilise une base de connaissance (PKB), répondez "**Y**" à la question suivante. Il vous sera alors demandé si un champ de la base de connaissance contient des domaines à associer au parseur. Ce cas est surtout valable pour les plateformes agrégateurs, comme **Highwire**.

```
? Does it have a knowledge base? (y/N)
```

La dernière étape consiste à lister les noms de domaine qui seront associés au parseur. Dans un premier temps, on vous propose de cocher (avec la barre espace) parmis les domaines présents dans analyses rédigées sur ANG :

```
? Check the domains that should be handled (Press <space> to select, <a> to toggle
all, <i> to inverse selection)
>◉ onlinelibrary.wiley.com
 ◉ agupubs.onlinelibrary.wiley.com
```

S'il manque des noms de domaine, vous pouvez ensuite les entrer un par un :

```
If some domains are missing, enter them now.
(empty answer to stop)
? Enter a domain name
```

Et voilà !

```
Ready... go!

  ✔ Create root directory
  ✔ Create test directory
  ✔ Create pkb directory
  ✔ Create scrapers directory
  ✔ Create manifest.json
  ✔ Create parser.js
  ✔ Create test file

Parser generated at /home/ezpaarse/platforms/wiley
```

Un dossier est créé automatiquement. Il contient :

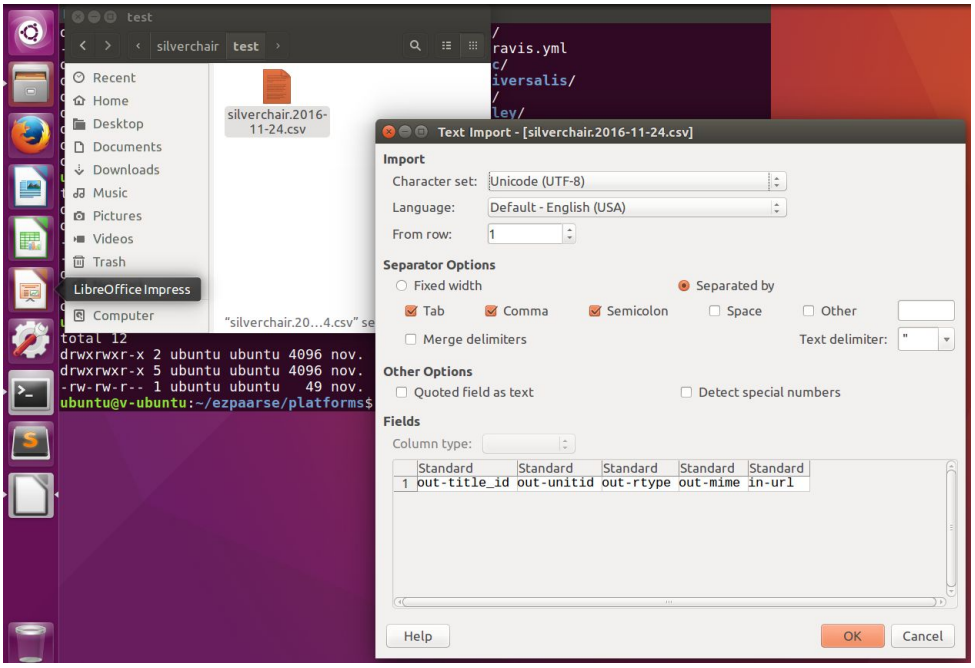
- Un fichier **manifest.json** décrivant le parseur.
- Un fichier **parser.js** contenant la logique du parseur, à compléter.
- Un dossier **pkb** où seront stockées (si besoin) les bases de connaissance.
- Un dossier **scrapers** pour les programmes de récupération automatisée des PKBs.
- Un dossier **test** contenant les fichiers permettant de tester le parseur. Celui-ci sera pré-complété à l'aide des analyses présentes sur ANG.

Vous pouvez visualiser le contenu du dossier avec la commande "**ls -l *nom_court_de_la_plateforme***"

```
~/ezpaarse/platforms$ ls -l wiley
total 24K
-rw-r--r-- 1 ubuntu  ubuntu   533 déc.   1 09:34 manifest.json
-rwxr-xr-x 1 ubuntu  ubuntu  7,9K déc.   1 09:34 parser.js*
drwxr-xr-x 2 ubuntu  ubuntu  4,0K nov.  30 17:09 pkb/
drwxr-xr-x 2 ubuntu  ubuntu  4,0K nov.  30 17:09 scrapers/
drwxr-xr-x 2 ubuntu  ubuntu  4,0K nov.  30 17:10 test/
```

II - Préparation du fichier test

Dans le dossier **test/** se trouve un fichier csv qui va nous servir à vérifier le bon fonctionnement du parseur. Si vous avez initialisé le parseur à partir d'une analyse existante, celui-ci devrait déjà être rempli à partir des URLs présentes sur ANG. Cependant, une étape de vérification et/ou complétion est toujours nécessaire.



Exemple :

out-title_id	out-unitid	out-doi	out-rtype	out-mime	in-url
	(ISSN)1600-5724	10.1111/(ISSN)1600-5724	TOC	MISC	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/journal/10.1111/%28ISSN%29160
AAR	aar.2012.83.issue-1	10.1111/aar.2012.83.issue-1	TOC	MISC	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1111/aar.2012.83.issue-1/is
	j.1600-0390.2012.00514	10.1111/j.1600-0390.2012.0	ABS	MISC	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1111/j.1600-0390.2012.005
ANIE	anie.201209878	10.1002/anie.201209878	ABS	MISC	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1002/anie.201209878/abstr
ACV	acv.12024	10.1111/acv.12024	ARTICLE	HTML	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1111/acv.12024/full
	j.1600-0390.2012.00514	10.1111/j.1600-0390.2012.0	ARTICLE	PDF	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1111/j.1600-0390.2012.005
ANIE	anie.201209878	10.1002/anie.201209878	ARTICLE	PDF	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1002/anie.201209878/pdf
9781118268117	9781118268117	10.1002/9781118268117	BOOK_SEC	MISC	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/book/10.1002/9781118268117
9781118268117	9781118268117.ch3	10.1002/9781118268117.ch	BOOK_SEC	PDF	http://onlinelibrary.wiley.com.passerelle.univ-rennes1.fr/doi/10.1002/9781118268117.ch3/p

Le fichier de test est au format csv, séparé par des points-virgules (;), attention donc si vous le modifiez sous LibreOffice. Pour chaque ligne, les champs des colonnes **"in-"** sont donnés au parseur, et le résultat est comparé aux champs des colonnes **"out-"**.

III - Tester le parseur

La logique du parseur se trouve dans le fichier **parser.js**. Ce dernier a été généré avec un code minimal et quelques exemples. Il est déjà possible d'exécuter ce parseur, même s'il ne reconnaît pas encore les URLs du fichier de test.

Pour le tester, positionnez-vous à la racine des plateformes et lancez la commande suivante :

```
~/ezpaarse/platforms$ make test wiley
```

Cette commande lit les fichiers du dossier **test/**, et vérifie pour chaque ligne que le parseur retourne les résultats attendus. Si une ligne échoue au test, une erreur sera affichée sous cette forme :

```
Wiley
    1) works


  0 passing (14ms)
  1 failing

  1) Wiley works:
     Error: result does not match
            input: {
  "url": "http://onlinelibrary.wiley.com:80/doi/10.1002/cpa.3160270102/pdf"
}
            result: {}
            expected: {
  "unitid": "cpa.3160270102/",
  "rtype": "ARTICLE",
  "mime": "PDF"
}
```

L'erreur contient trois parties :

- **Input** : données fournies au parseur (dans la majorité des cas, uniquement l'URL)
- **Result** : résultats retournés par le parseur
- **Expected** : résultats attendus

Comme nous n'avons encore rien fait, ces erreurs sont normales. Le parseur ne reconnaît aucune URL présente dans notre fichier de test.

IV - Coder le parseur

Un parseur consiste généralement en une suite d'expressions régulières permettant d'identifier les structures d'URLs présentes dans le fichier de test. Certains peuvent nécessiter des méthodes plus avancées, mais ce ne sont pas les plus fréquents.

Le parseur est constitué d'une fonction nommée **analyseEC()**, qui reçoit en paramètre un objet correspondant à une URL décomposée. Ses principales propriétés sont :

- **pathname** : le chemin de l'URL
- **query** : les paramètres de la requête, sous forme d'objet
- **hostname** : le nom de domaine

En pratique, ces propriétés se récupèrent de cette manière :

```
module.exports = new Parser(function analyseEC(parsedUrl) {
  let path  = parsedUrl.pathname;
});
```

Le chemin de l'URL est ainsi stocké dans la variable **path**. C'est sur cette partie que nous allons appliquer les expressions régulières.

Exemple d'expression régulière en Javascript :

```
/^\/doi\/(10\.[0-9]+\/([a-z]+)\.[0-9]+)\/full$/i
```

Cette expression identifie ce chemin :

```
/doi/10.1111/acv.12024/full
```

`^` : indique le commencement de l'expression et c'est facultatif.
`$` : indique la fin de l'expression et c'est facultatif.
`[a-z]+` = expression contient au moins une lettre entre a et z en minuscule.
`[0-9]+` = expression contient au moins un chiffre.
`[A-Z]+` = expression contient au moins une lettre entre a et z en majuscule.
`[a-zA-Z0-9]+` = expression qui contient à la fois au moins une lettre entre a et z en minuscule et en majuscule et un chiffre.

Le `i` à la fin de l'expression régulière rend celle-ci insensible à la casse. En d'autres termes, `/doi\/[a-z]+/i` correspond aussi bien à **doi/acv** qu'à **Doi/ACV**.

Les **parenthèses** permettent de capturer des parties de l'URL.

Les caractères spéciaux `!^$()[]{}?+*./\|` doivent être précédés d'un antislash s'ils sont utilisés comme de simples caractères, à moins de se trouver dans une expression entre crochets `[ ]` (à l'exception du crochet fermant `]`). Exemple : `/\/article\.pdf/`

Un cours plus complet est disponible sur OpenClassrooms pour vous aider à comprendre le fonctionnement des expressions régulières.

Dans le code du parseur, nous recommandons de commenter les expressions régulières avec un ou plusieurs exemples d'URLs reconnues. Celles-ci étant parfois difficiles à lire, il est pratique d'avoir des exemples sous les yeux lorsque l'on parcourt le parseur, comme sur l'exemple ci-dessous :

```
if ((match = /^\/doi\/(10\.[0-9]+\/(([^.]+)\.[0-9]+))\/full$/i.exec(path)) !== null) {
   // /doi/10.1111/acv.12024/full
   result.doi      = match[1];
   result.unitid   = match[2];
   result.title_id = match[3].toUpperCase();
   result.rtype    = 'ARTICLE';
   result.mime     = 'HTML';
 }
```

À chaque nouvelle URL implémentée, vous pouvez tester de nouveau le parseur en rejouant l'étape III, afin de vérifier que les éléments attendus sont bel et bien retournés.

L'objectif est de procéder par itérations. Au début, le test échoue car le parseur est encore vide. On commence par écrire une expression régulière pour identifier les éléments de la première ligne du fichier test. Une fois la première ligne reconnue, le test échoue sur la ligne suivante. On passe alors à la deuxième, et ainsi de suite jusqu'à ce que tout soit reconnu. La commande de test affichera alors la mention **"passing"** :

```
~/ezpaarse/platforms$ make test wiley
cd .lib && EZPAARSE_PLATFORM_TO_TEST="wiley" npm test

> ezpaarse-platforms@1.0.0 test
/home/ADS.INTRA.INIST.FR/schurter/dev/ezpaarse/platforms/.lib
> mocha test/*-test.js


  Wiley
    ✓ works


  1 passing (15ms)

make: « wiley » est à jour.
```

V - Vérification du code

Avant de pousser votre code sur le dépôt GitHub, il va falloir vérifier sa conformité avec les règles syntaxiques du projet. Ces règles nous permettent d'homogénéiser le code venant de nos différents contributeurs. Vous pouvez lancer la vérification syntaxique avec la commande :

```
~/ezpaarse/platforms$ make lint
```

La commande affiche les éventuelles erreurs sous cette forme :

```
/home/ezpaarse/platforms/wiley/parser.js
  162:80  error  Unnecessary escape character: \.              no-useless-escape
  192:23  error  Strings must use singlequote                  quotes
  214:1   error  Expected indentation of 4 spaces but found 6  indent
```

Pour chaque anomalie, on retrouve le numéro de ligne et de colonne où elle a été détectée, son type et sa description. Les anomalies peuvent être des **erreurs**, auquel cas vous devez la corriger, ou des **warnings**, dont la correction est conseillée mais non bloquante.

Sachez que s'il est possible de lancer la vérification syntaxique manuellement, cette dernière est de toute façon lancée au moment de valider vos modifications. En cas d'erreur, la validation ne pourra pas se faire, ce qui vous évite de soumettre par erreur du code non conforme.

Une fois toutes les erreurs corrigées, il est temps de passer à la validation !

VI - Valider les modifications

Avant de valider vos modifications, vous pouvez vérifier l'état de votre dépôt local avec la commande `git status -u`. Dans le cas d'un nouveau parseur, les fichiers seront tous listés comme "non suivis", comme vous pouvez le voir ci-dessous :

```
~/ezpaarse/platforms$  git status -u

Sur la branche master
Votre branche est à jour avec 'origin/master'.

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

      wiley/manifest.json
      wiley/parser.js
      wiley/test/wiley.2017-11-30.csv

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents
(utilisez "git add" pour les suivre)
```

Nous allons commencer par ajouter ces fichiers aux modifications à valider. Si vous n'avez pas encombré le dossier de fichiers superflus, vous pouvez ajouter directement le dossier en utilisant la commande suivante :

```
~/ezpaarse/platforms$  git add wiley
```

Dans le cas contraire, vous devrez ajouter les fichiers un par un :

```
~/ezpaarse/platforms$  git add wiley/manifest.json wiley/parser.js
~/ezpaarse/platforms$  git add wiley/test/wiley.2017-11-30.csv
```

On valide ensuite les modifications en créant un **"commit"**, auquel on adjoint une courte description :

```
~/ezpaarse/platforms$  git commit -m 'add wiley platform'
```

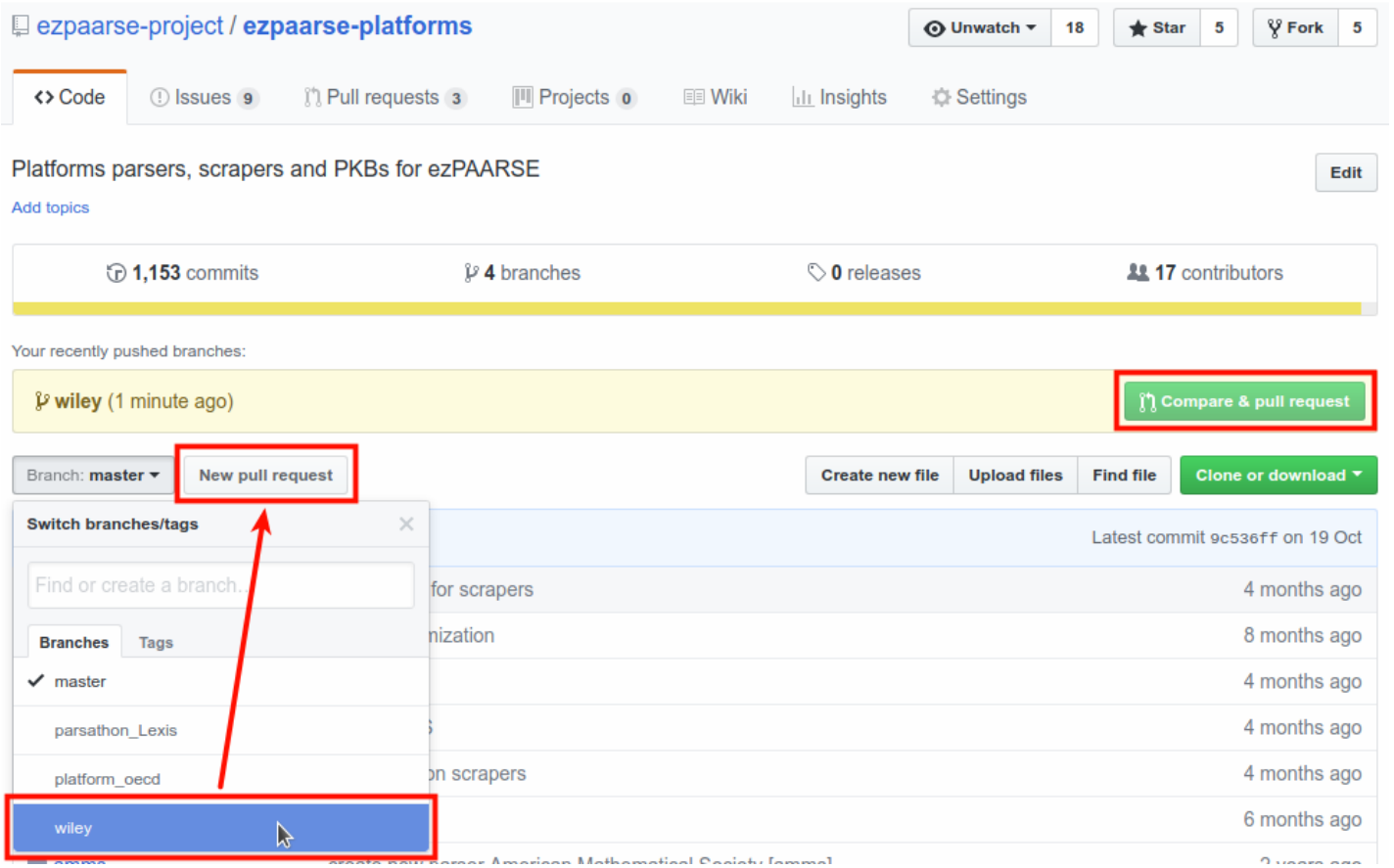
Puis on pousse notre commit sur Github avec la commande suivante :

```
~/ezpaarse/platforms$  git push
```

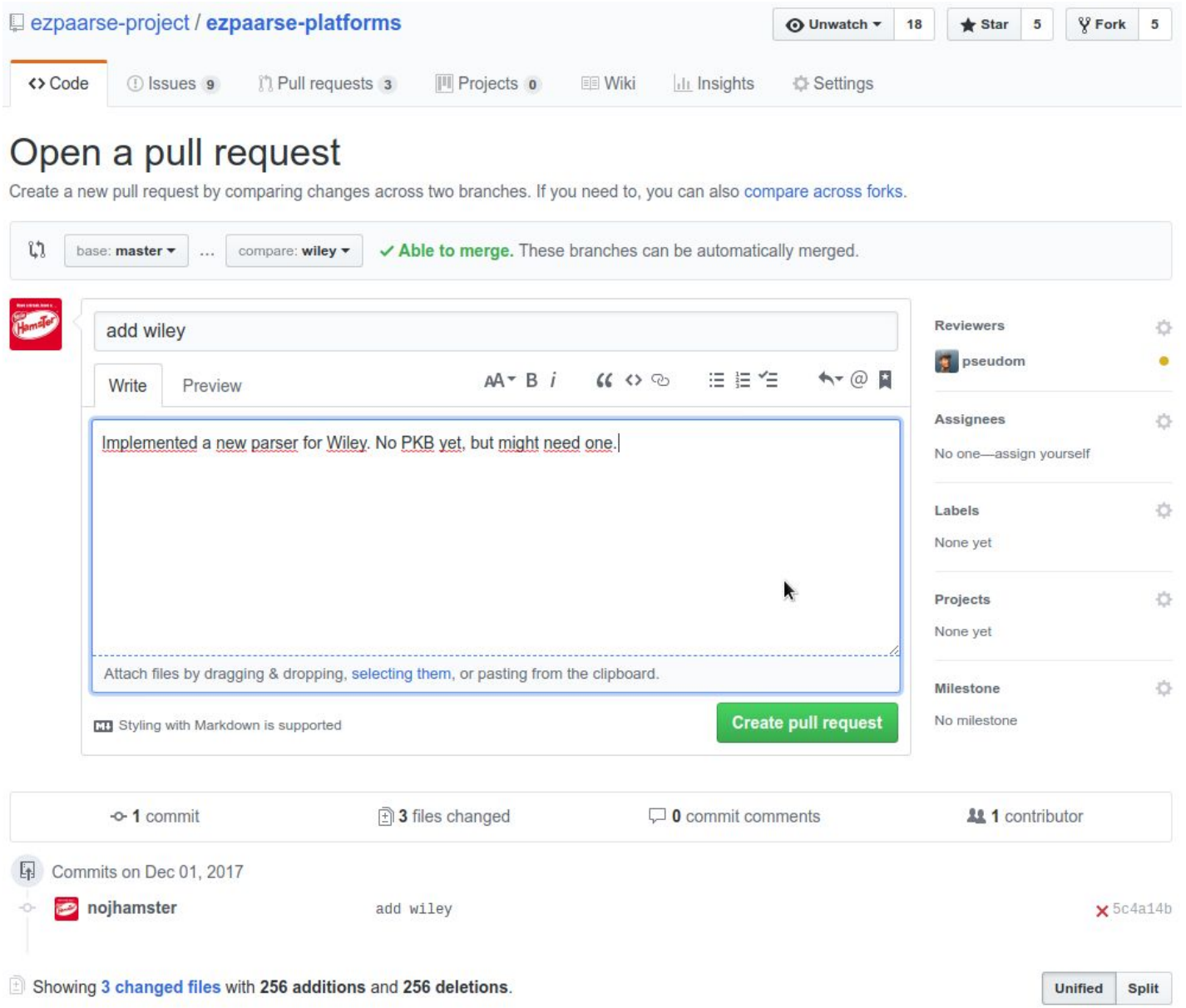
Vous pouvez créer d'autres commits si vous sentez le besoin d'opérer de nouveaux changements.

VI - Reversement

Une fois toutes les modifications validées et envoyées sur GitHub, il nous reste à créer une **"Pull Request"** afin de demander à fusionner votre branche avec la branche principale. Rendez-vous sur le dépôt GitHub, sélectionnez votre branche, et cliquez sur **"new pull request"**. Un raccourci peut également apparaître en évidence, auquel cas vous pouvez directement cliquer sur **"compare & pull request"**.



Remplissez le titre, ajoutez éventuellement une description plus détaillée de vos modifications (vous pouvez écrire en Markdown), puis cliquez sur **"Create pull request"**.



Et voilà ! Votre pull request est désormais en attente, un membre de l'équipe se chargera de la passer en revue. Tant qu'elle n'est pas validée, vous pouvez ajouter des commits à votre branche. Ils seront automatiquement ajoutés à votre Pull Request. Si nous estimons que des ajustements sont nécessaires avant l'intégration, nous vous inviterons à ajouter de nouveaux commits.

Une fois validée, votre pull request est intégrée au dépôt principal. Vos modifications sont alors accessibles à toutes les instances d'ezPAARSE, après mise à jour de leur dépôt local.