# Néoveille - documentation technique, structure et fonctionnement du site web

Emmanuel Cartier, emmanuel.cartier@lipn.univ-paris13.fr

14 octobre 2018

## Table des matières

# 1 Présentation générale et architecture su système

La figure 1 présente l'architecture générale de Néoveille. Elle comprend une série de programmes en back-end (tout ce qui se trouve au-dessus de la ligne horizontale pointillée) et une série de programmes en front-end (interface web, partie en-dessous de la ligne horizontale).

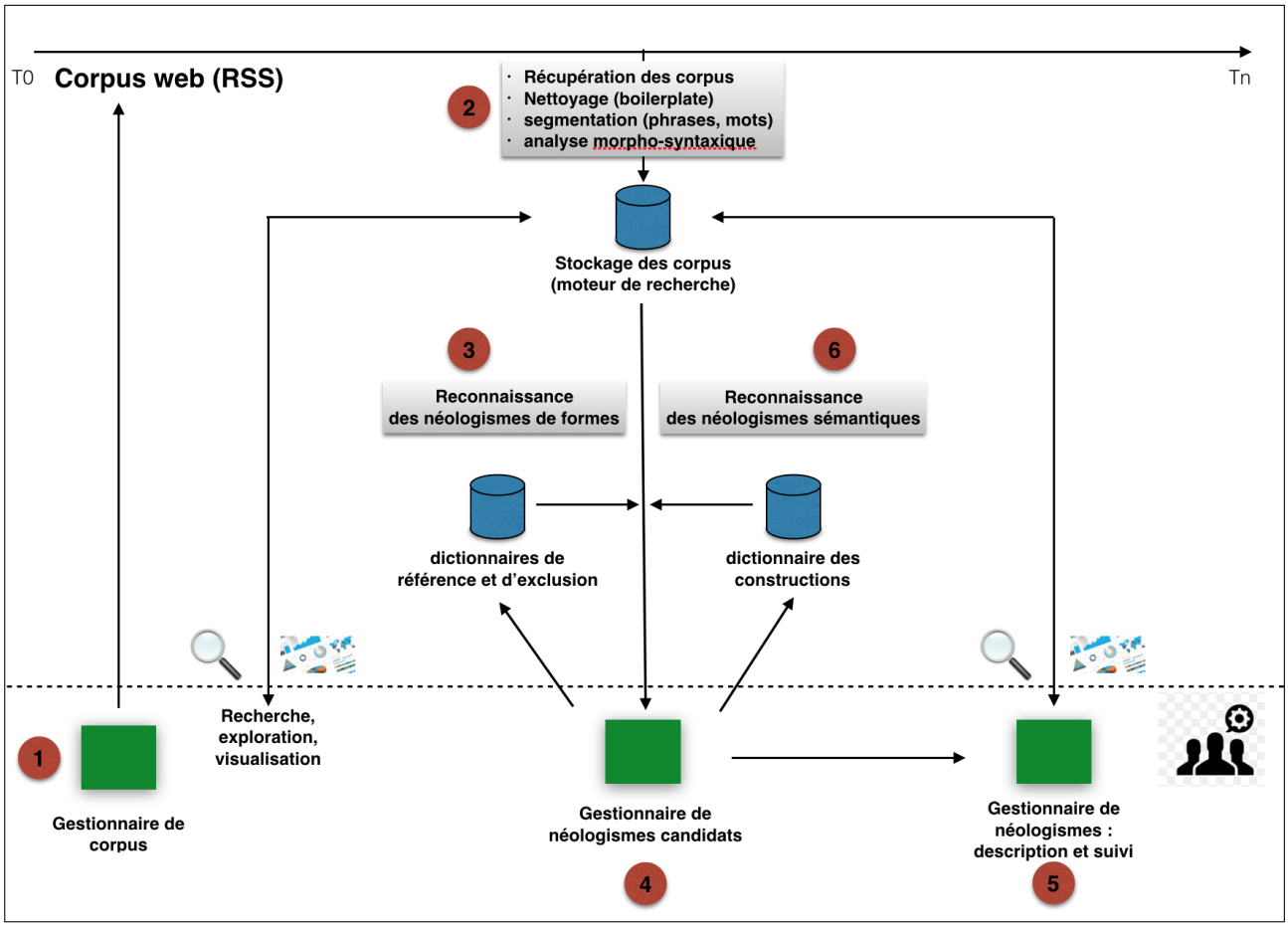


FIGURE 1 – Architecture de Néoveille

Dans le schéma, les cylindres bleus représentent des ressources de deux types : des corpus, stockés dans un moteur de recherche Apache Solr, et des ressources linguistiques de type dictionnaires, stockées dans des bases de données Mysql.

Les deux zones de l'applications (backend et frontend) se distinguent par l'absence d'interaction avec les utilisateurs pour les programmes backend, alors les programmes frontend sont déclenchées par l'interaction avec les utilisateurs. Tous les processus interagissent avec les données stockées.

On détaille l'architecture fonctionnelle dans la figure 2.

Dans ce schéma, en backend, deux processus principaux se produisent en continu (selon la fréquence de déclenchement des deux programmes) :

— **corpus_crawl.py** : récupération des corpus sur le web à partir des sources d'informations définis par les utilisateurs ; ce programme stocke les résultats (articles de presse complets avec les méta-informations glanés), dans le moteur de recherche ;

— **detect_neologisms.py** : analyse morphosyntaxique des textes des articles, détection des néologismes, sauvergarde des néologismes candidats dans la base des candidats néologismes (Candidate Neologisms), sauvegarde des textes analysés morphosyntaxiquement dans Apache Solr ;

En frontend, l'interaction entre les données et les utilisateurs comporte deux familles de processus :

— **Opérations SCRUD (Search, Create, Read, Update, Delete)** : la première série de processus permet aux utilisateurs de rechercher, de lire, de créer, de mettre à jour et de supprimer les données contenus dans les différentes bases de données : sources d'informations, dictionnaires de référence et d'exclusion, candidats néologismes, néologismes validés. ces opérations se produisent sur l'interface à l'aide la librairie javascript datatables.editor (`https://editor.datatables.net/` ;)

— **Visualisation multimodale interactive** : une autre interaction consiste à pouvoir effectuer une recherche combinant les données se trouvant les tables de bases de données (les néologismes par exemple)

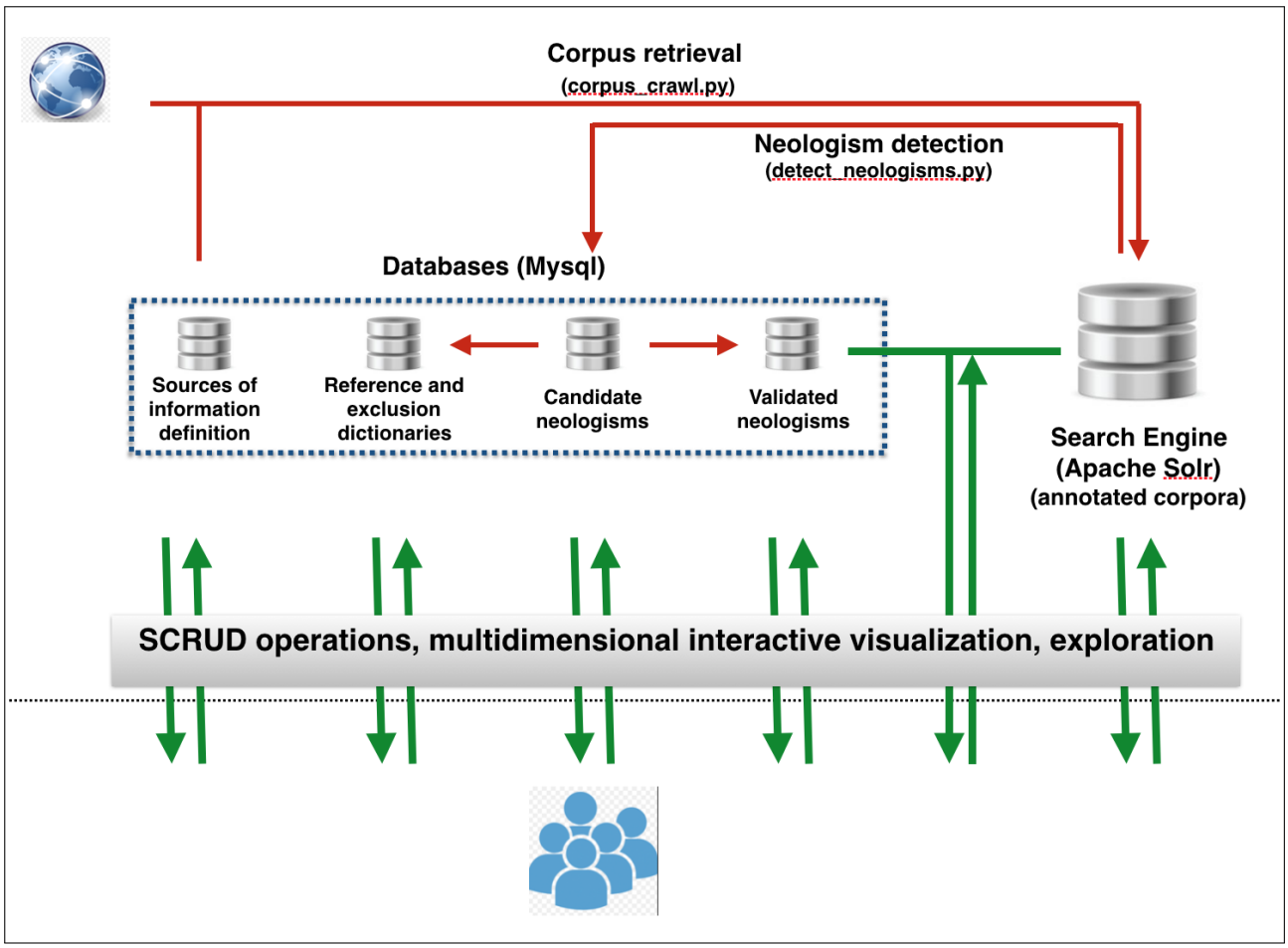


FIGURE 2 – Architecture fonctionnelle de Néoveille

et d'obtenir également les contextes (et les méta-informations associées) dans le moteur de recherche, aboutissant à une exploration visuelle, multidimensionnelle et interactive. Les processus d'aggrégation des données sont effectués grâce à la librairie Javascript dc.js (`https://dc-js.github.io/dc.js/`) ;

Nous détaillons ci-après l'installation des différents programmes, de manière globale puis en distinguant le frontend et le backend..

# 2 Installation globale de l'ensemble des fichiers du programme

Pour avoir tous les fichiers en local, faites un clone du dépôt github :

```
git clone https ://github.com/ecartierlipn/neoveille2016.git
```

Les fichiers pour l'interface web se trouve dans le sous-dossier "frontend". Dans le sous-dossier "frontend", on trouvera tous les programmes qui se déroule de manière totalement automatique. Le dossier resources contients des ressources additionnelles. Le dossier docs contient la documentation détaillée (ce fichier).

# 3 Installation de l'interface web de Néoveille

## 3.1 Prérequis

L'installation a été testée sur une machine Linux Ubuntu 16.04 code Xenial.

**Serveur Apache :** un serveur web Apache 2 doit être installé avec une gestion de PHP (>version 5) et Mysql (> version 5.7).

On peut soit installer séparément ces différents éléments, soit les installer ensemble : voir procédure par exemple ici : `https://www.linode.com/docs/web-servers/lamp/install-lamp-stack-on-ubuntu-16-04/`

**Apache Solr :** le moteur de recherche Apache Solr doit également être installé (> 5.3 (version 5.5 actuellement utilisée), non testé sur les versions 7+) : `http://lucene.apache.org/solr/`

## 3.2 Procédure d'installation

Pour l'installation, il faut effectuer la suite d'opérations suivantes :

1. placer le contenu du dossier frontend dans un répertoire web du serveur Apache (par défaut : /var/www/).
2. modifier les paramètres d'accès au serveur Mysql dans le fichier credentials.php (qui se trouve, en prenant /var/www comme base d'installation des programmes du site ./html/html/crendetials.php), à savoir la variable $*usermysql*, et la variable $*password*.
3. création de trois bases de données dans Mysql : rssdata (contiendra les informations sur les corpus à récupérer), datatables (contiendra les néologismes candidats et les dictionnaires d'exclusion), neo3 (contiendra les néologismes et l'ensemble des descriptions de ceux-ci).
4. lancer les trois scripts sql (qui se trouve à `resources/mysql`) à partir de la racine du clone, pour créer les tables dans les trois bases de données [1].

Ensuite, si le serveur Apache est lancé, il faut se rendre à : `http://localhost/html/html/index.php`. Vous n'avez plus qu'à naviguer sur le site (il y a un id/mot de passe par défaut : admin/neoveille2016).

# 4 Installation des programmes Néoveille en backend

Si ce n'est déjà fait, faites un clone du dépôt github :

```
git clone https ://github.com/ecartierlipn/neoveille2016.git
```

Les programmes du backend se trouvent dans le sous-répertoire backend.

## 4.1 Pré-requis

Pour que les programmes de récupération et d'analyse des néologismes fonctionnent, il faut préalablement avoir installé :

— Apache Solr : une fois le programme installé, il faut créer une collection Solr (par défaut une par langue). Créer un répertoire rss_french dans <répertoire d'installation solr>/server/solr. Allez dans ce répertoire et dézipper le fichier de configuration exemple (resources/apachesolr/solr_configs.zip). Cela va créer un sous-répertoire conf avec les fichiers de configuration (el plus important est schema.xml qui contient les champs d'index). Il faut relancer le serveur Solr et vérifier que la collection a bien été chargée (elle sera vide).

— MySQL avec les bases de données listées ci-dessus

1. Pour lancer les scripts, saisir (exemple de neo3) : $mysql - u < id > -p < password > neo3 < neo3.complete.truct.sql$

— Treetagger pour l'analyse morphosyntaxique du français : voir `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`.
— Hunspell avec les fichiers de la langue traitée. Par défaut installé sur linux. Les fichiers dicos pour le français sont disponibles dans `resources/hunspell`.
— librairies pour Python : **à vérifier en lançant le programme**.

Les programmes Python ont été confectionnés en python 2.7. le portage est possible en python 3.4+ mais nécessite de changer plusieurs librairies, notamment pour la récupération web.

## 4.2 Détails des fichiers du backend

Le répertoire backend contient tous les fichiers du backend. Il contient un sous-répertoire pour crawl des corpus (corpus_crawler) et un sous-répertoire pour la détection des néologismes formels (formal_neology).

La répertoire corpus_crawler contient :
— **corpus_crawler** : contient les fichiers principaux (fichiers .py) pour récupérer les corpus par langue (corpus_<lang>.py), et les .sh pour faciliter l'installation d'une tâche cron (corpus_crawl_<lang>.sh). le fichier URLutils.py est une librairie additionnelle utilisée par les fichiers corpus_<lang>.py.
— **corpus_crawler/log** : contient les fichiers de log qui sont créés chaque fois que les programmes python sont lancés. Voir le nom de ces fichiers de log déclarés en fin des fichiers corpus_<lang>.py ;
— **corpus_crawler/tobeindexed** : contient les fichiers à indexer pour Apache Solr. Répertoire à nettoyer de temps à autres.
— **corpus_crawler/webcrawl** : travail en cours pour récupérer des données sur le web sans le support des fils RSS..

Le répertoire formal_neology contient :
— **fichiers Python** : contient les fichiers principaux (fichiers .py) pour détecter les néologismes par langue (detect_neologisms_<lang>.py, all pour le français), et les .sh pour faciliter l'installation d'une tâche cron (detect_neologisms_<lang>.sh) ;
— **formal_neology/log** : contient les fichiers de log qui sont créés chaque fois que les programmes python sont lancés. Voir le nom de ces fichiers en fin des detect_neologisms_<lang>.py ;
— **formal_neology/tobeindexed** : contient les fichiers à indexer pour Apache Solr. Répertoire à nettoyer de temps à autres.

## 4.3 Récupération dynamique des fichiers sur le web : fichier corpus_fr.py

Le programme *corpus_fr.py* effectue les opérations suivantes : il récupère, pour les sources définies dans la table rssdata/RSS_INFO (dans cette table il récupère toutes les métainfos + le champ NAME_RSS, ie url du fil RSS)) les fichiers qui sont listés dans les fils rss : il récupère également les méta-infos de l'item du fil rss (title, author, description, url, keywords, etc.) puis va chercher l'article complet via le champ url. Il effectue un nettoyage de la page (boilerplate removal, avec justext) pour ne conserver que les zones textes utiles. Ensuite, le programme stocke le texte et toutes les métainformations dans la table rssdata/rss_data2, en donnant au champ IS_INDEXED la valeur 0, indiquant que le fichier doit encore être traité pour repèrer les néologismes. Il stocke les mêmes informations dans le moteur Apache Solr. NB - le stockage dans la table rssdata_2 est temporaire (il faudrait prévoir un nettoyage régulier de cette table, actuellement aucun process ne s'en occupe).

Le programme génère un fichier de log qui doit permettre d'identifier les erreurs. Il se trouve dans le sous-répertoire log.

### 4.3.1 Paramètres à régler dans le fichier

:
— **accès à la base de données MySQL** : à modifier dans trois méthodes : get_corpus_list_fromDB, get_last_indexed_fromDB (dans la classe corpus) et dans save_rsscorpus_to_DB (classe rssfeeds).
— **accès à la collection Apache Solr** : à modifier dans la variable lang_solr2. (par défaut, pour le français, la collection s'appelle *rss_french*).

### 4.3.2 Installation du fichier corpus_fr.py en tâche récurrente

: Le plus simple sous linux est de créer une tâche CRON :

```
1        crontab -e
```

et (par exemple, en utilisant le script shell fourni, pour lancer le programme chaque nuit à 0 :14) :

```
0 14 * * * <chemin vers le fichier>corpus_crawl_fr.sh 2> <chemin vers le fichier>/
    errors_corpus_crawl_all_crontab.txt
```

#### 4.3.3 Structure du fichier corpus_fr.py

Le fichier se compose de variables globales, de deux classes (corpus et rssfeeds) et d'une fonction main() détaillée ci-dessous.

```
for lang in ['Rép. Tchèque','pologne','france']:
        c = corpus(lang, 'db', 'rss') # init. corpus avec info langue, type stockage info
            et type fichiers
        c.get_corpus_list_fromDB() # recup des fils rss à recuperer dans bd
        if str(len(c.rssfeeds))>0:
                c.get_last_indexed_fromDB() # recup des fils rss recuperes (>un jour avant)
                c.retrieve_corpus()# recup des nouveaux articles et stockage dans bd et
                    Apache Solr
        else:
                log.info("No RSS feeds for this language. Check your configuration" )
                log.info("All is done! Quitting program.")
```

### 4.4 Détection des néologismes, stockage des néologismes candidats et mise à jour des articles dans Apache Solr : fichier detect_neologisms_all.py

Le programme **detect_neologisms_all.py** effectue les opérations suivantes (pour le français) : il récupère dans la table datatables/rssdata_2 les articles à analyser, il les analyse morphosyntaxiquement (avec Treetagger), il récupère l'analyse morphosyntaxique du texte qu'il uniformise, y récupère les mots inconnus, fait un passage par Hunspell pour enlever des candidats néologismes les coquilles, ensuite filtre les mots qui se trouvent dans le dictionnaire d'exclusion, enfin stocke les candidats néologismes dans la table datatables/neologismes_fr et met à jour l'article dans Apache Solr avec l'analyse morphosyntaxique.

#### 4.4.1 Paramètres à régler dans le fichier

:

- **accès à la base de données MySQL** : à modifier dans deux fonctions : get_corpus_list_fromDB, et dans add_neologisms_to_db .
- **accès à la collection Apache Solr** : à modifier dans la variable lang_solr2. (par défaut, pour le français, la collection s'appelle *rss_french*).
- **Accès à l'exécutable paramétré de Treetagger** : il s'agit des shell script du Treetagger lançant le programme avec une configuration de langue spécifique. Par exemple pour le français, il s'agit du fichier : '<chemin d'installation Treetagger>/cmd/tree-tagger-french-utf8'. On explicite le chemin vers ce fichier dans la variable : pos_ana['france'] ;
- **Accès aux dictionnaires Hunspell** : indiquer le chemin pour accéder aux dictionnaires Hunspell. Par exemple pour le français : hunspell['france']='/opt/nlp_tools/dictionaries/fr_FR/fr_FR'.

#### 4.4.2 Installation du fichier detect_neologisms_all.py en tâche récurrente

: Voir programme précédent.

#### 4.4.3 Structure du fichier detect_neologisms_all.py

Le fichier se compose de variables globales, d'une classe (document), d'une séries de fonctions utilitaires et d'une fonction main() détaillée ci-dessous.

```
conn=''
langs = ['pologne','france','brésil','Rép. Tchèque']
for lang in langs:
        res = get_corpus_data_fromDB(lang)
        if res:
                load_exclusion_dico(iso[lang])
                i=0
                ### preparation of solrxmlfile
                now = datetime.now().strftime("%d-%m-%y_%H")
                filenameXML = './tobeindexed/update_rss-data.'+ lang_solr[lang] + "." + now
                    + '.xml'
                fout = codecs.open(filenameXML,mode="w+",encoding="utf-8")
```

```
12                  fout.write("<add>")
13                  for data in res: # on parcourt les fichiers à analyser
14                          i+=1
15                          metas={}
16                          source_link =data[0]
17                          metas['title'] =data[1]
18                          metas['subject'] =data[2]
19                          metas['category'] =data[3]
20                          #metas['description'] =data[4]
21                          cts =data[1] + "\n" + data[5]
22                          contents1 = re.sub(r'[ńż]', r' ', unicode(cts), flags=re.UNICODE)
23                          contents = re.sub(r'\s+', r' ', unicode(contents1), flags=re.
                                UNICODE)
24                          metas['ID_RSS'] =data[6]
25                          d = document(source_link, lang, metas,contents)
26                          langd = d.detect_language(contents)
27                          if (langd not in lang_detect[lang]):
28                                  log.info("Problem with language detection for contents : "
                                        + unicode(contents) + "\nAutomatic detection says : [" +
                                         langd + "] whereas expected language is " + str(
                                        lang_detect[lang]) + "\nSkipping analysis for this
                                        document and deleting it from database.")
29                                  continue
30                          d.ling_analyze('pos_tagging') ## to be done : specific return value
                                 if fails
31                          #print str(d)
32                          ### retrieve info so as to create a dict for saving into solr xml
                                format
33                          doc = {}
34                          doc['link']=source_link
35                          if d.postagger==False:
36                                  log.info("Pos tagger has returned false for content")
37                                  continue # go to next document
38                          else:
39                                  doc['lemmes']= d.lemmas
40                                  doc['noms_propres']= d.np
41                                  doc['pos-text']= d.postaggedText # too heavy for apache
                                        solr 15-12-2016
42                                  # neo exclusion list
43                                  if len(d.unk)>0: # if some unknown words
44                                          log.info(d.unk)
45                                          unk2=d.filter_unknown(d.unk) # hunspell and
                                                exclusion dictionaries filtering
46                                          if len(unk2)>0:
47                                                  add_neologisms_to_db(unk2,iso[lang])
48                                                  # update command for apache solr
49                                                  updatecmd={'lemmes':'add','noms_propres':'
                                                        add','pos-text':'set'}
50                                                  res = write_xml_chunk(fout,doc,lang_solr[
                                                        lang],updatecmd)
51                                                  if res:
52                                                          # if solr indexing ok, ùark text as
                                                                1 in db table
53                                                          db_tag_doc_as_processed(source_link
                                                                ,1)
54                                                  else: # else mark as 2
55                                                          db_tag_doc_as_processed(source_link
                                                                ,2)
56                                  # no neologismes : index-update anyway with other
                                        linguistic information (lemmes, noms-propres)
57                                  else:
58                                          log.info("Pas de mot inconnu dans : " + source_link
                                                + ". indexing anyway.")
59                                          updatecmd={'lemmes':'add','noms_propres':'add','pos
                                                -text':'set'}
60                                          res = write_xml_chunk(fout,doc,lang_solr[lang],
                                                updatecmd)
61                                          if res:
62                                                  db_tag_doc_as_processed(source_link,1)
63                                          else:
64                                                  db_tag_doc_as_processed(source_link,2)
65                          # no unknown words but change IS_INDEXED to value 1
66                          else:
67                                  log.info("Pas de mot inconnu dans : " + source_link + ".
                                        indexing anyway.")
68                                  updatecmd={'lemmes':'add','noms_propres':'add','pos-text':'
```

```
                                        set '}
69                                      res = write_xml_chunk(fout,doc,lang_solr[lang],updatecmd)
70                                      #                               continue
71                                      if res:
72                                              db_tag_doc_as_processed(source_link,2)
73                                      else:
74                                              db_tag_doc_as_processed(source_link,2)
75                      fout.write("</add>") # level of for data in res
76                      fout.close()
77                      # now indexing for lang
78                      res = index_data_solr(filenameXML, lang_solr[lang],"xml")
79                      if res:
80                              log.info("Indexation succeeded for file : " + filenameXML)
81                      else:
82                              log.error("Indexation failed for file : " + filenameXML)
83              else:
84                      log.info("No corpus data to process for " + lang)
85                      log.info("All is done. Exiting.")
```

# 5 Détails des fichiers : interface web

## 5.1 Organisation des dossiers et fichiers principaux

Les fichiers sont, à partir de la racine d'installation, répartis dans des sous-dossiers :

```
(racine)
├── css : fichiers de style généraux
├── html : dossier principal contenant index.php
│   ├── css : fichiers de style
│   ├── images : fichiers images
│   ├── php : codes php pour interagir avec les bases de données et Apache Solr
│   └── table : fichiers php contenant les codes pour afficher la partie centrale des pages
├── img
├── js
└── lib
```

Nous présentons ci-après l'organisation des pages web. Nous présentons tout d'abord la structure de la page d'accueil et le mécanisme général d'organisation des fichiers.

## 5.2 Organisation page d'accueil

La page d'accueil (index.php quand on est connecté, login.php ldans les autres cas) se présente sous forme de quatre zones (figure 3). Le bandeau supérieur est couvert par l'élément <nav class="navbar navbar-default...">, le menu gauche par l'élément , le contenu principal par l'élément , et le pied de page par <footer class="app-footer">.

### 5.2.1 Feuilles de style

Les feuilles de style associées sont (à vérifier...) déclarées dans le header, il s'agit d'un mixte entre bootstrap.css et des thèmes définis dans ../css/style.css et ../css/themes/flat-blue.css.

Il serait nécessaire d'uniformiser les styles en ne prenant qu'une librairie basée sur bootstrap.

### 5.2.2 Librairies javascript

Les librairies javascript chargées sont les suivantes (figure 4).

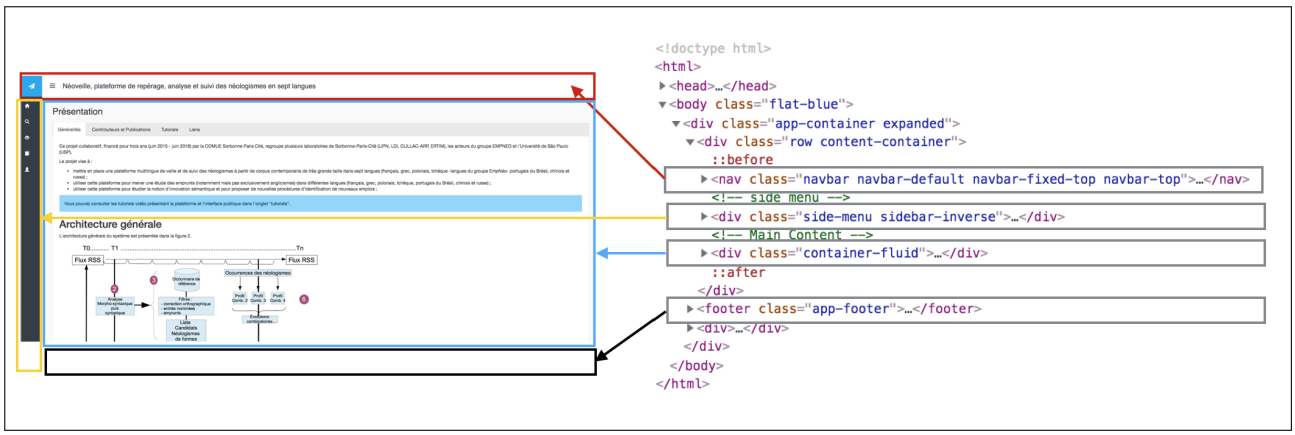


FIGURE 3 – Organisation page d'accueil (login.php et index.php)

```
<!-- Javascript Libs -->
<!-- jquery -->
<script type="text/javascript" src="../lib/js/jquery.min.js"></script>
<!-- bootstrap -->
<script type="text/javascript" src="../lib/js/bootstrap.min.js"></script>
<!-- datatables -->
<script type="text/javascript" charset="utf-8" src="js/dataTables.min.js">
<!-- datatables style bootstrap -->
<script type="text/javascript" src="../lib/js/dataTables.bootstrap.min.js"></script>
<!-- datatables editor -->
<script type="text/javascript" charset="utf-8" src="js/dataTables.editor.min.js"></script>

<!-- librairies spécifiques datatables -->
<script type="text/javascript" charset="utf-8" src="js/editor.title.js"></script>
<script type="text/javascript" src="js/jquery.dataTables.columnFilter.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.3.1/js/dataTables.buttons.min.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.3.1/js/buttons.flash.min.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.3.1/js/buttons.print.min.js"></script>
<script type="text/javascript"
src="https://cdn.datatables.net/buttons/1.3.1/js/buttons.html5.min.js"></script>
<script type="text/javascript"
```

FIGURE 4 – Liste librairies javascript chargées dans login.php et index.php

## 5.3 Bandeau supérieur (<nav class="navbar...")

Le lien entre les composants de l'élement navbar et les zones affichées est donné dans la figure 5.

Lorsque l'utilisateur est connecté, un élément supplémentaire apparaît en haut à droite (<ul class="nav navbar-nav navbar-right"). Les composants de cet élément sont détaillées dans la figure 6, à lire de bas en haut.

## 5.4 Menu gauche ( )

Le menu à gauche est le centre de contrôle pour la navigation entre les pages. Lorsqu'on clique sur un des éléments du menu, on déclenche une action (via une action Javascript décrite dans js/menu.js, voir plus loin) qui aboutit soit : à charger un fichier php qui va se placer dans le  dans la zone d'affichage principal, soit à rendre visible un élément div contenu dans le fichier index.php ou login.php, dans cette même zone centrale.

### 5.4.1 Action Javascript associée aux éléments de menu

Lorsqu'utilisateur clique sur un élément de menu, l'action associée à l'id de l'élément est déclenchée. Par exemple, dans la figure 8, nous avons l'exemple du javascript déclenché sur l'id "" qui aboutit ) charger dans le <div class="" le contenu du fichier php. le Javscript associe à l'id "sources-dev", les actions suivantes : changer

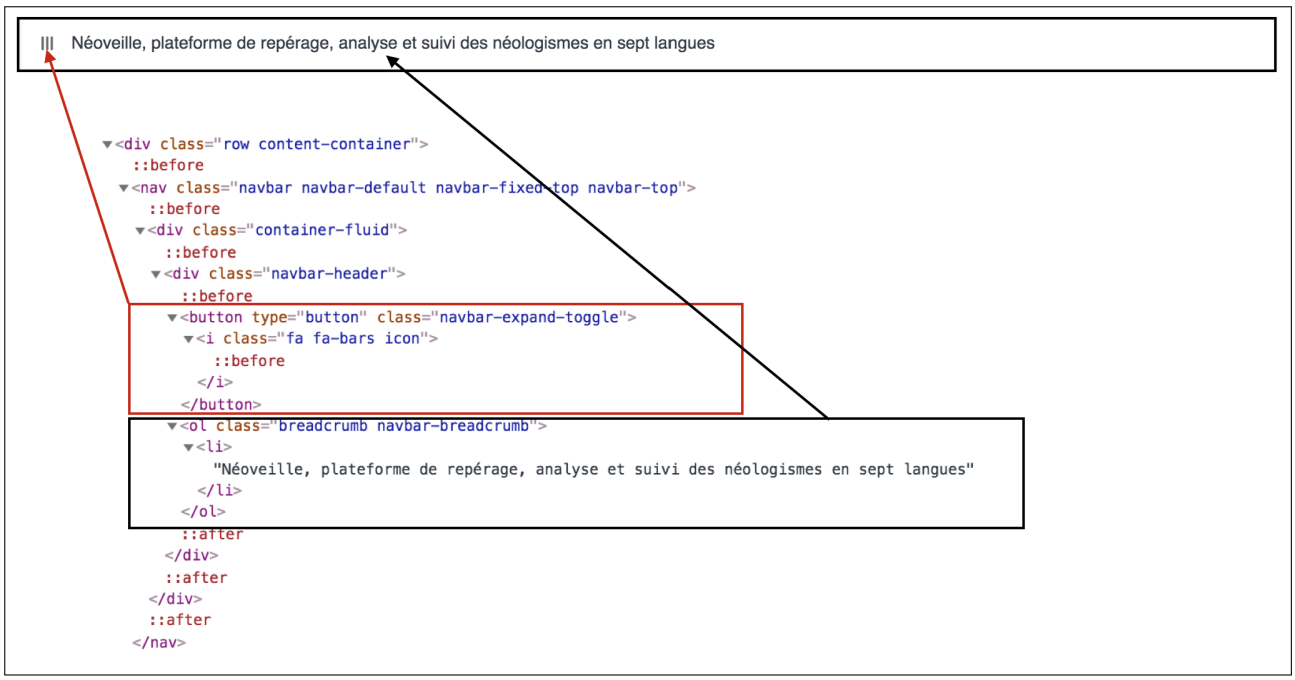


FIGURE 5 – Organisation Barre de navigation (login.php et index.php)

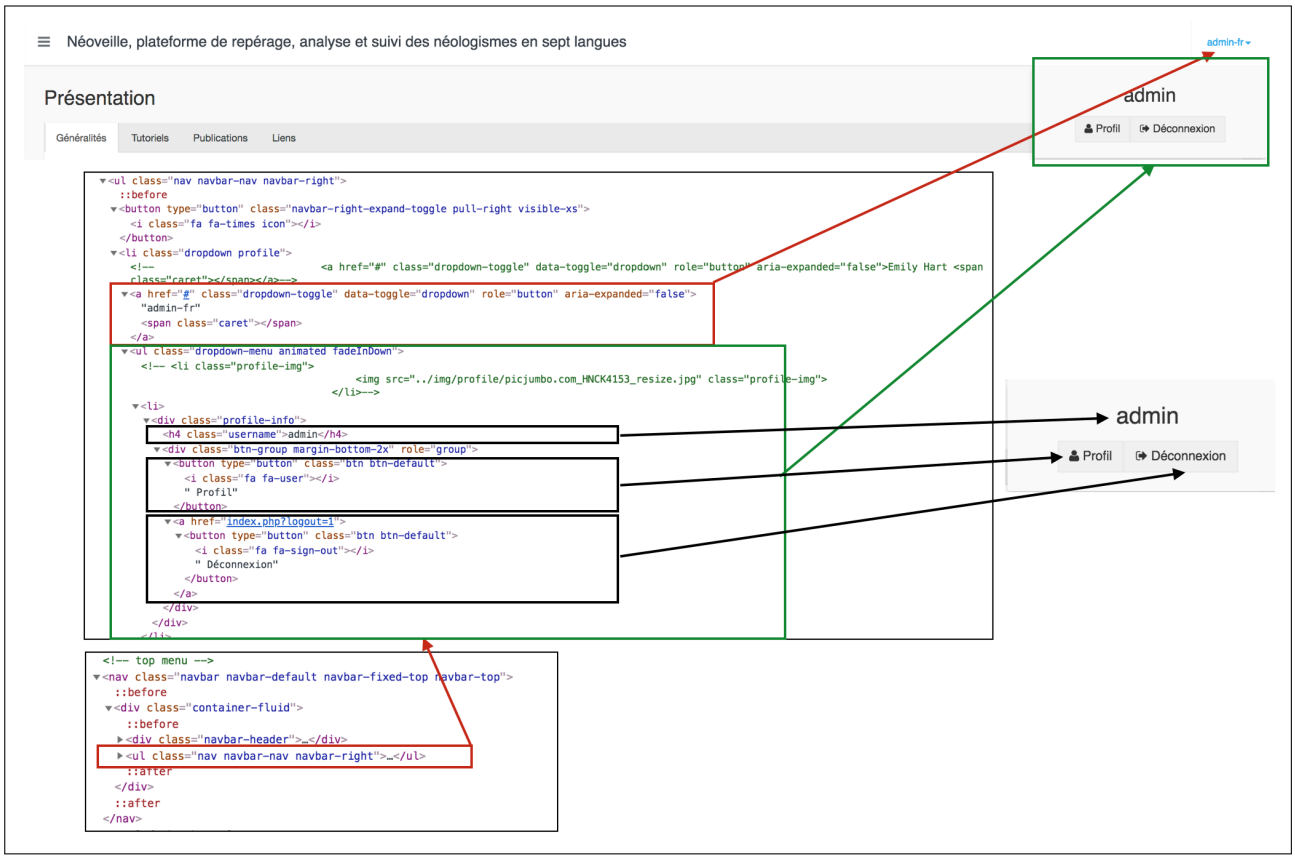


FIGURE 6 – Élément d'administration du compte dans le bandeau supérieur (index.php)

les valeurs des éléments li (les éléments de menu) de "active" à "inactive" (correspondant à une colorisation des éléments de menu), et à mettre la valeur du li courant à "active" (ie le coloriser en blanc). Ensuite on charge (load) dans l'élément contenant l'id "container-fluid (le div contenant la zone centrale), le contenu du fichier "table/datatable-corpus-dev.php". Ce fonctionnement est général pour tous les éléments de menu.

Deux autres fonctionnements plus simples consistent :

— à associer le clic d'un élément de menu au chargement d'un élément  déjà écrit dans le fichier

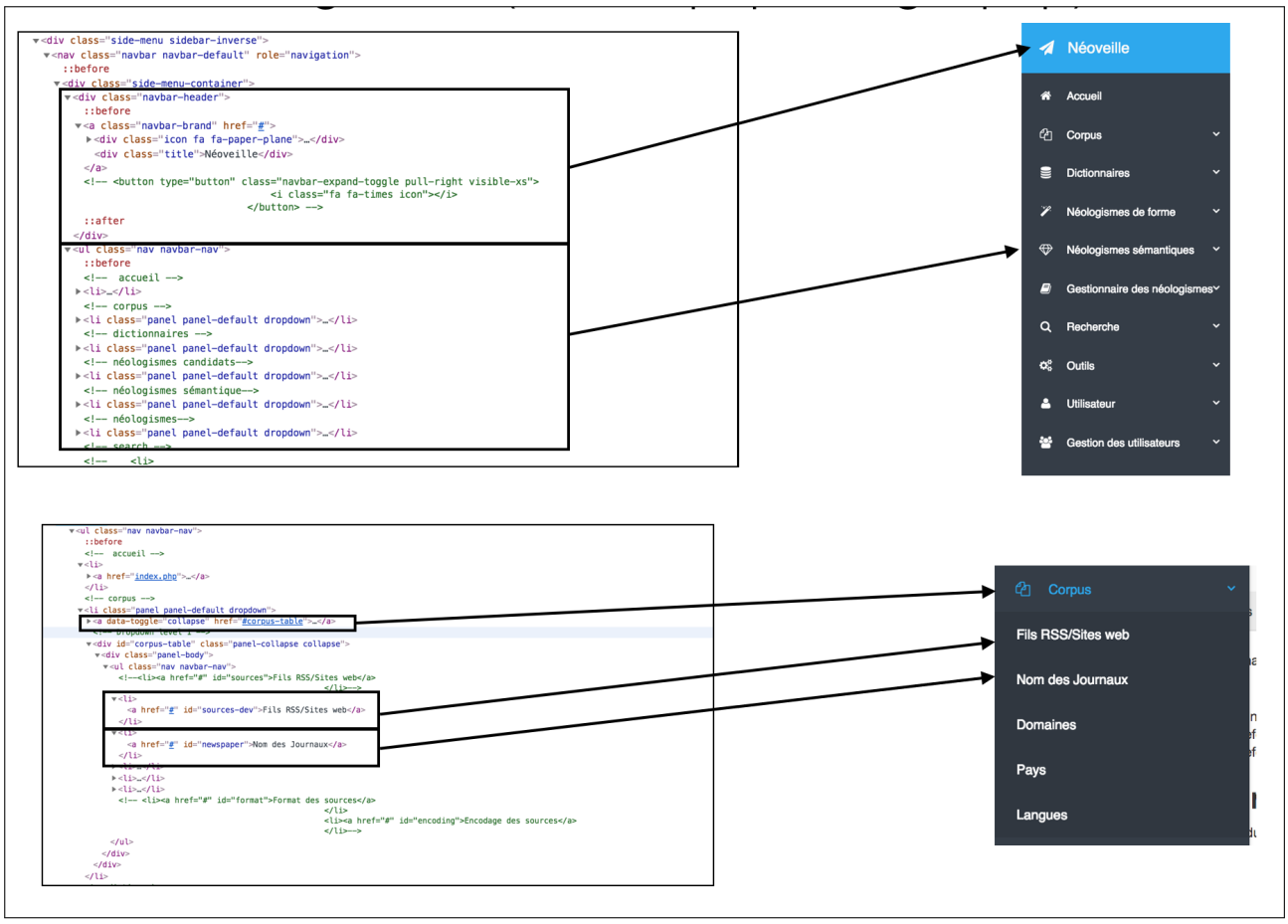


FIGURE 7 – Organisation Menu gauche (login.php et index.php)

index.php (ou login.php), via le javascript suivant (contenu directement dans ). Par exemple, pour revenir à la page d'accueil, le javascript est directement inclus dans l'élément de départ (figure 9) ;

— à déclencher une action javascript associée à l'id de l'élément de menu, le javascript étant décrit dans les scripts (cela concerne le fichier login.php). Par exemple, pour ouvrir la page de recherche (voir figure 10).

## 5.5 Page de contenu ( )

Les pages de contenu ont un affichage par défaut (page d'accueil) soit sont chargées via les actions liées aux éléments de menu. La structure générale de l'élément  est lié au framework bootstrap. La figure 11 montre la structure de base de cet élément ( ). dans le <div class="page-title" on inclut le titre, et dans le <div class="row" on inclut le contenu lui-même.

Lorsque le contenu est dans un fichier php (qui se trouve toujours dans le sous-répertoire "table") il est généralement lui-même lié à deux autres fichiers : un fichier javascript qui va prendre en charge les différentes actions associées et la création des tables éditables (via les librairies datatables.js et datatables.editor.js), cette librairie s'occupant de charger les données à afficher/éditer, et de mettre en place les éléments d'interaction (tri des tables, recherche, navigation, édition).

**Remarque** : on peut retrouver les fichiers liés à chaque des éléments de menus, avec les outils de développements web sous Google Chrome, dans l'onglet Network. Pour plus d'infos sur l'utilisation de ces outils : `https://www.malekal.com/chrome-firefox-outils-de-developpement/`.

## 5.6 Gestionnaire de corpus

Le menu Corpus/Fils RSS/Sites web permet d'accéder à l'interface présentée dans la figure **??**. Cette page est chargée en chargeant le fichier **table/datatable-corpus-dev.php**, qui lui-même charge le fichier javascript **js/table.RSS_INFO-dev.js**, lui-même chargeant le fichier **php/table.RSS_INFO.php**. Le rôle respectif de ces trois fichiers est détaillé ci-dessous ;

— **table/datatable-corpus-dev.php** : fichier chargent les éléments html à placer dans la zone principale d'affichage.

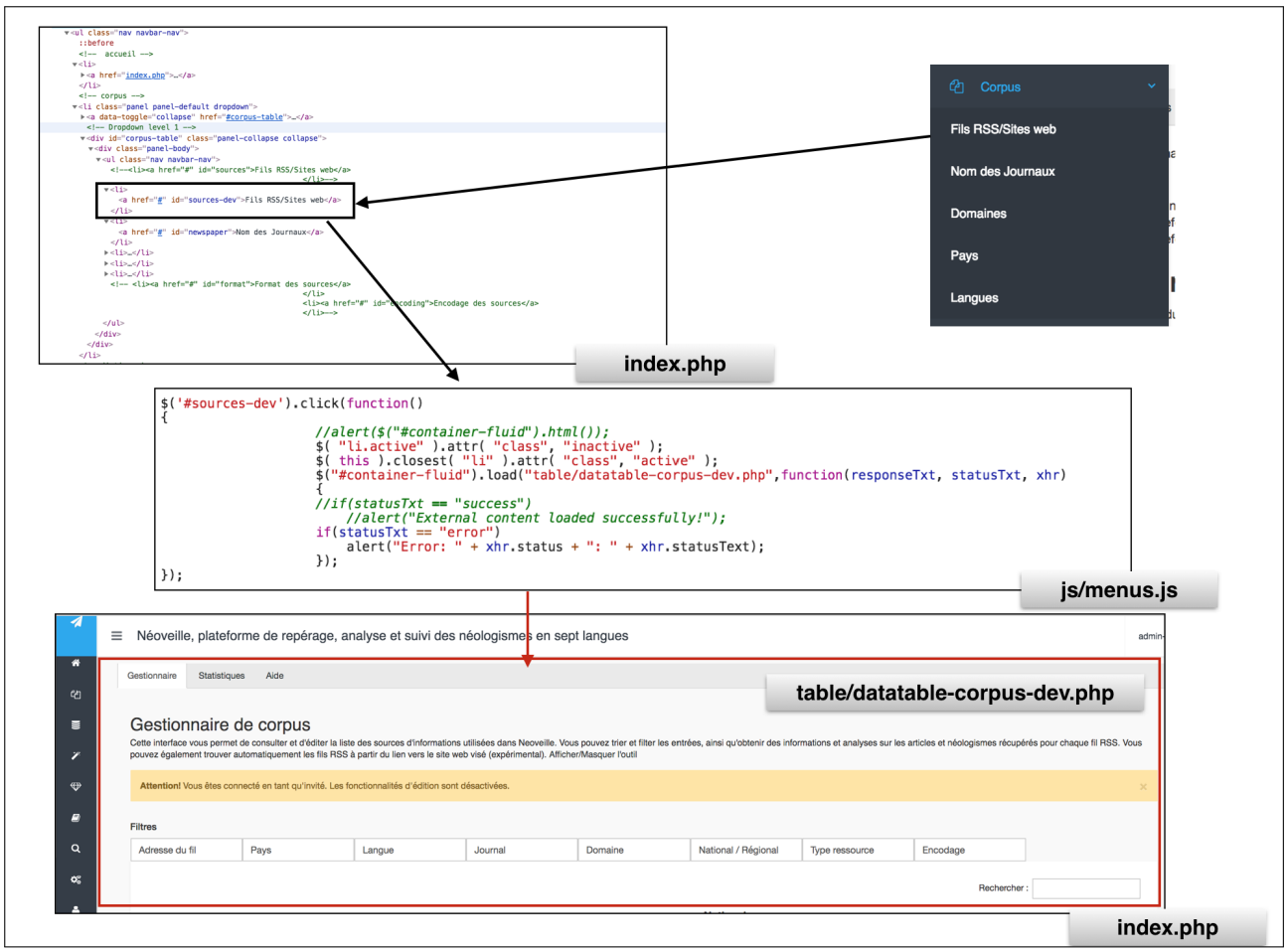


FIGURE 8 – Détail de l'action de menu (exemple corpus)

```
▼<li>
  ▶<a href="#" id="accueil" onclick="jQuery('div#neovalsynth').hide();jQuery('div#signupbox').hide(); jQuery('div#loginbox').hide();
  jQuery('div#neo-search2').hide();jQuery('div#neoval').hide();jQuery('div#presentation-gen').show();">…</a>
  </li>
```

FIGURE 9 – Action de menu directement dans l'élément (exemple accueil)

```
$('#neo-search2').click(function()
                        {
                                        //alert($("#container-fluid").html());
                                        jQuery('div#signupbox').hide();
                                        jQuery('div#loginbox').hide();
                                        jQuery('div#presentation-gen').hide();
                                        jQuery('div#neoval').hide();
                                        jQuery('div#neovaldemo').hide();
                    jQuery('div#neovalsynth').hide();
                    jQuery('div#neo-search2').show();
                    var languageW ='fr';
                        $("div#neo-search2").load("table/datatable-search-demo.php",function(responseTxt, statusTxt, xhr)
                        {
                                //if(statusTxt == "success")
                                //alert("External content loaded successfully!");
                                if(statusTxt == "error")
                                alert("Error: " + xhr.status + ": " + xhr.statusText);
                                });
});
```

FIGURE 10 – Action javascript de menu intégré au fichier login.php (exemple search)

— **js/table.RSS_INFO-dev.js** : fichier chargé via le précédent et contenant la définition initiale de la grille à afficher (datatables.editor), et les actions/fonctions associées aux éléments.
— **php/table.RSS_INFO.php** : fichier qui assure l'interaction entre datatable.editor et les données elles-mêmes, stockées dans une base de données (en l'occurrence la table rssdata/RSS_INFO et les tables liées). Le fichier php fait partie du framework datatables.editor.

```
<!-- Main Content -->
▼<div class="container-fluid" id="container-fluid">
  ::before
  ▼<div class="side-body"> == $0
    ▶<div class="page-title">…</div>
    ▶<div class="row">…</div>
    </div>
    ::after
  </div>
  ::after
</div>
```

FIGURE 11 – Structure globale de l'élément 

Les librairies javascript pour la visualisation interactive des résultats sont également chargées par le fichier table/datatable-corpus-dev.php.

- **../js/d3.js** : librairie pour créer des graphes en svg dans des pages html ;
- **../js/dc.js** : librairie utilisant d3.js pour générer plusieurs graphes en interaction ;
- **../js/crossfilter.js** : librairie pour agréger les données brutes provenant de d3.js et permettant l'interaction entre les graphes résultants ;

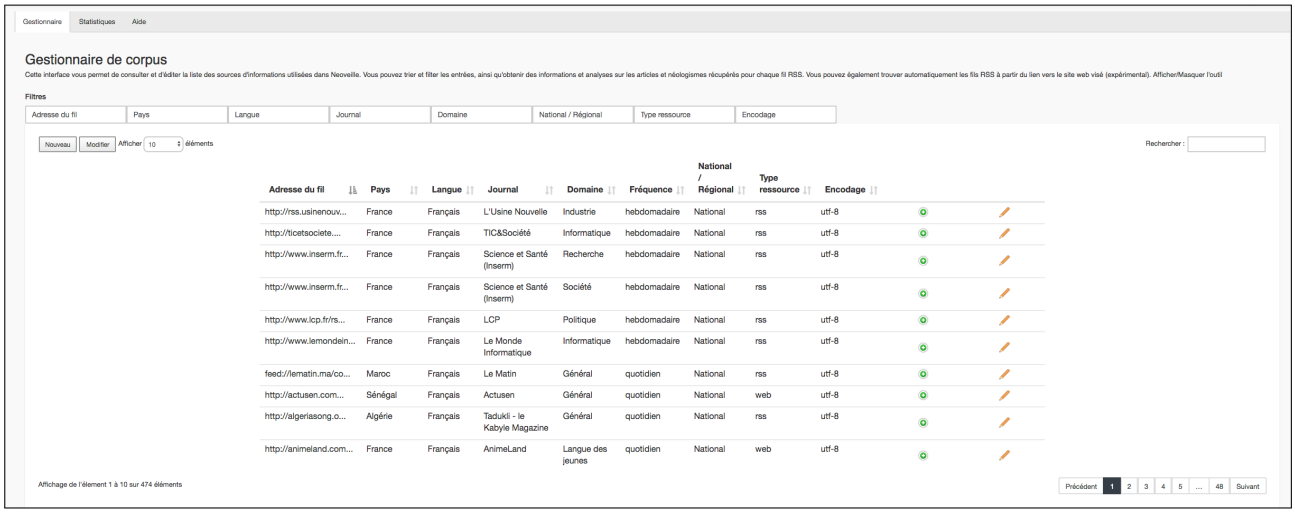


FIGURE 12 – Exemple de contenu dans le fichier table/datatable-corpus-dev.php

### 5.7 Gestionnaire des néologismes candidats

### 5.8 Gestionnaire des néologismes sémantiques

### 5.9 Gestionnaire des néologismes

### 5.10 Moteur de recherche

## 6 Détails des bases de données

Trois bases de données permettent de stocker les différentes informations :

- **Les sources d'informations** : base de donnée rssdata.
- **Les néologismes candidats et les dictionnaires de référence "utilisateurs" et d'exclusion** : base de données datatables.

— **La base de description des néologismes** : base de données neo3.

## 6.1 Détails de la base de données rssdata (sources des données)

Cette base de données est dédiée à la représentation des sources de données (corpus). Elle est organisée autour de la table RSS_INFO, avec des clés étrangères liées aux différentes informations (voir figure 13). Ce modèle est extensible, si l'on souhaite ajouter de nouvelles métadonnées aux sources d'information.

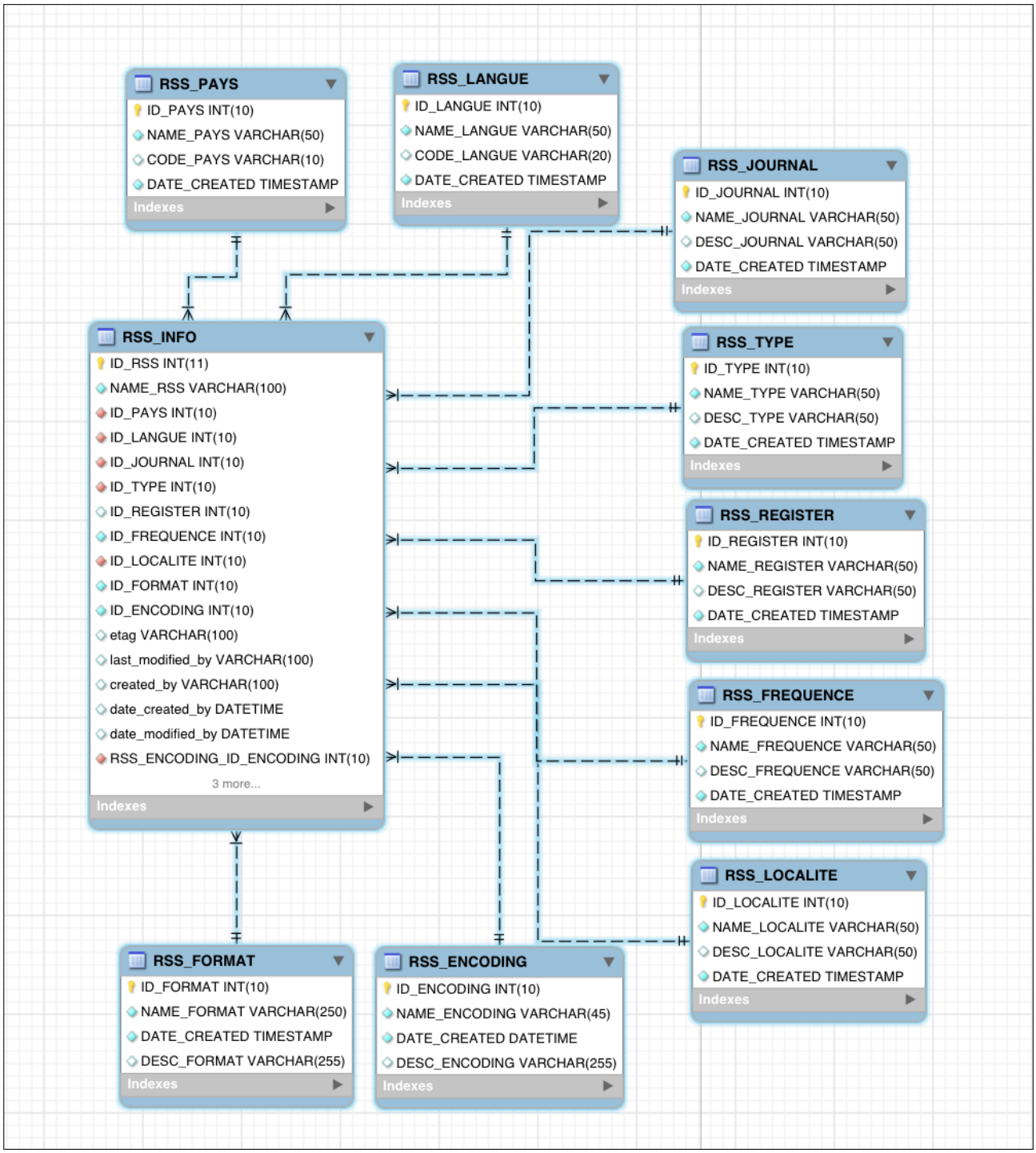


FIGURE 13 – Modèle de données MySQL pour rssdata

Actuellement, le fichier RSS_INFO comprend les sources d'informations pour toutes les langues (voir champ langue).

### 6.2 Détails de la base de données datatables (néologismes candidats)

Cette base de données est dédiée à la représentation des néologismes candidats repérés par le programme backend (detect_neologisms.py), d'une part (table neologismes) et à la représentation des dictionnaires de référence "utilisateurs" (dico simple, dico composé, dico termino et dico prefixes et suffixes, ces deux derniers non-utilisés actuellement mais qui pourraient l'être pour détecter les affixations) et dictionnaires d'exclusion (excluded_fr). La même structure est suivie pour chaque langue (avec le suffixe _<lang>).

### 6.3 Détails de la base de données neo3 (base des néologismes validés)

## 7 Détails des collections Apache Solr

Les corpus récupérés sont stockées dans une collection dans Apache Solr. nous présentons ci-dessous l'architecture générale de ce système et le format des collections (susceptible d'évoluer). La version d'Apache Solr utilisée est actuellement la 5.3.

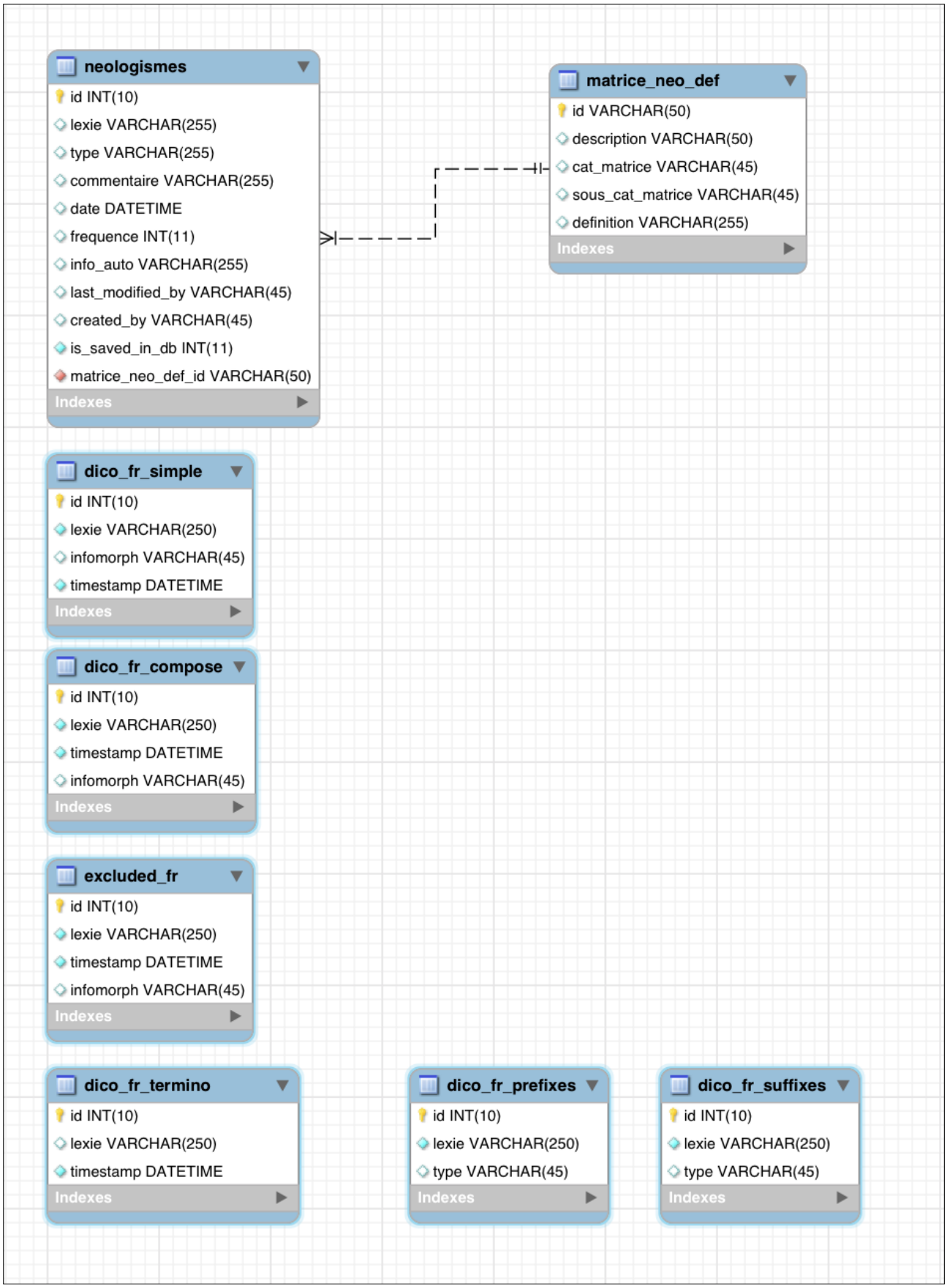


FIGURE 14 – Modèle de données MySQL pour datatables

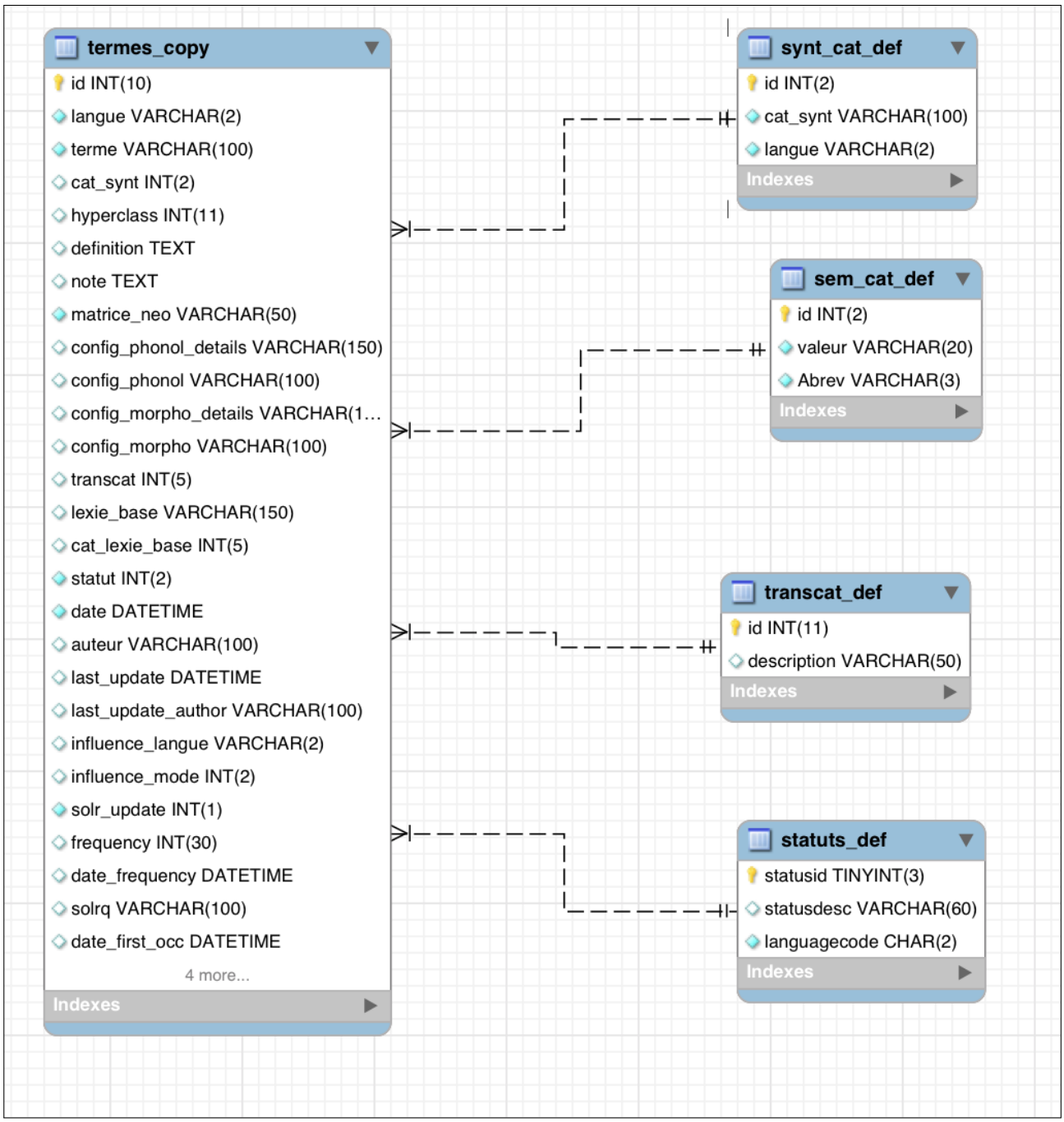


FIGURE 15 – Modèle de données MySQL pour neo3