Ministère de l'enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Rapport

Module : Traitement automatique du langage naturel

Master 1 SII

Mini projet

Réalisation d'unn outil d'aide au développement d'un dictionnaire de la langue arabe sur des bases historiques

- Réalisé par :

  BENHADDAD Wissam
  BOURAHLA Yasser
  MOHAMEDI Haroune
  LAHBIB Abdelghani

10-12-2018

Table des matières

Table des matières 2

Table des figures 3

1 Motivations et problematique 4
- 1.1 Introduction . . . 4
- 1.2 Définitions . . . 5
  - 1.2.1 DataSet et Corpus . . . 5
  - 1.2.2 TALN . . . 5
  - 1.2.3 Dictionnaire historique . . . 5
- 1.3 Conclusion . . . 5

2 Conception du système 6
- 2.1 Introduction . . . 6
- 2.2 Schéma global du système . . . 6
- 2.3 Les modules du système . . . 7
  - 2.3.1 Aspirateur de sites web . . . 7
  - 2.3.2 Organisateur de corpus . . . 7
  - 2.3.3 Corpus reader . . . 8
  - 2.3.4 Base de données . . . 9
  - 2.3.5 Application (Front-end et Back-end) . . . 10
- 2.4 Déploiement des modules dans le cloud . . . 10
- 2.5 Conclusion . . . 10

3 Réalisation de l'application 11
- 3.1 Introduction . . . 11
- 3.2 Environnement de travail et outils utilisés . . . 11
  - 3.2.1 Python . . . 11
  - 3.2.2 JavaScript . . . 11
  - 3.2.3 NLTK . . . 11
  - 3.2.4 VueJS . . . 12
  - 3.2.5 Django . . . 12
  - 3.2.6 PostgreSQL . . . 12
  - 3.2.7 Google Cloud Platform . . . 12
- 3.3 Présentation de l'application . . . 13
  - 3.3.1 Interface principale . . . 13
  - 3.3.2 Ajouter corpus . . . 13
  - 3.3.3 Explorateur de corpus . . . 13
  - 3.3.4 Ajouter entrée . . . 14
  - 3.3.5 Dictionnaire . . . 15
  - 3.3.6 Dictionnaire historique . . . 15
  - 3.3.7 Statistiques . . . 16
- 3.4 Fonctions supplémentaires . . . 16
  - 3.4.1 Mode-automatique . . . 16
- 3.5 Conclusion . . . 17

4 Conclusion générale 18
- 4.1 Objectifs atteints . . . 18
- 4.2 Limites du système . . . 18
- 4.3 Perspectives futures . . . 19

Table des figures

2.1 Schéma global du système 6
2.2 Schéma de la base de données 9

CHAPITRE 1

MOTIVATIONS ET PROBLEMATIQUE

Introduction

Depuis son apparition (au IIe siècle), la langue arabe n'a cessé d'évoluer, donnant naissance à de nouveaux mots, ou modifiant le sens de mots existants. Cette évolution a particulièrement enrichi le vocabulaire de la langue de l'islam, en conséquent et au fil du temps, plusieurs ouvrages destiné a recenser les différentes définitions et sens d'un mot ont vu le jour, chacun durant sa période. néanmoins, il est primordial de garder une trace des différents changement qui ont eu lieu sur ces mots, et cela depuis leurs émergence. C'est avec cette idée en tête que les lexicographes des temps modernes en eu l'initiative d'entamer la construction de dictionnaires historiques afin de regrouper toutes les nuances des mots a travers les ages.

La création d'un tel ouvrage n'est pas chose facile, en effet elle demande d'une part une grande connaissance sur les différentes périodes historiques de la langue, ainsi que sur la langue en elle même durant ces périodes. Chercher et regrouper des écrits, documents et ouvrages des différentes auteurs durant ces périodes est une tâche qui est en elle même très ardue, cela peut prendre plusieurs décennies pour créer une collection de documents asses représentative de chaque période. Analyser le contenue de tout ces documents est la phase qui dure le plus de temps, une vérification minutieuse de chaque information ajoutée au dictionnaire finale doit être faite, puis sujette à l'approbation de plusieurs experts du domaine.

C'est avec l'avènement de l'informatique, de l'intelligence artificielle et plus récemment avec l'explosion du volume de données présent sur internet, que l'idée d'utiliser ces technologies pour faciliter et accélérer le processus de création d'un dictionnaire historique de la langue arabe à émergé. En effet la grande quantité et diversité de documents présente sur internet pourrait être exploité par un lexicographe pour ne pas s'attarder sur fastidieuse tâche de collecte des données, et cela en utilisant des techniques de traitement automatique du langage (TALN), de recherche d'information(RI) et d'intelligence artificielle.

Ce besoin d'un outillage informatique est la principale motivation derrière ce mini-projet, avec suffisamment de données et une bonne conception, la réalisation d'un tel outillage pourrait faire gagner énormément de temps aux lexicographes du monde arabe.

Le but de projet étant maintenant établi, nous allons maintenant passer à la schématisation de ce rapport. Nous commencerons d'abord par de petites définitions pour se situer dans la suite du rapport, nous enchaînerons ensuite sur la conception du système pour expliquer le travail réalisé, viendra ensuite la présentation de notre application, enfin nous finirons par une conclusion générale comportant un bilan du projet, des critiques sur notre système ainsi que les perspectives envisagées.

Définitions

DataSet et Corpus

Un jeu de données (DataSet) est un ensemble de données traité et organisé dans un schéma spécifique aux besoins d'un système, un dataset peut être une base de données relationnelle, un ensemble de fichiers texte, une banque d'images/videos ...

Dans notre cas nous nous intéresserons plus particulièrement à un type de dataset appelé Corpus, informellement un corpus est un dataset principalement utilisé dans le domaine du TALN, il est constitué d'un ensemble de fichier texte (annotés ou pas) qui représentent un domaine, une thématique, un(ou des) type(s) d'ouvrages ...

Un corpus est un composant essentiel pour la l'application des techniques de TALN, la taille et la qualité d'un corpus est donc un facteur primordial pour assurer une bonne performance d'un système.

TALN

Le Traitement Automatique du Langage Naturel (TALN) est un sous domaine de l'intelligence artificielle qui vise à analyser et à modéliser les composants du langage humain, que ce soit du point de vue syntaxique, sémantique ou pragmatique. l'aspect principal du TALN est le fait de permettre aux machine de traiter les séquence de texte non plus comme une simple suite de symboles, mais comme des entités informationnelles. Des connaissances sur la langue sont un prérequis essentiel pour le développement d'un système utilisant le TALN, ainsi que la disponibilité d'un grand ensemble de données pour faire de l'apprentissage automatique.

Le TALN est découpé en un ensemble de techniques et opérations à appliquer sur du texte, une multitude de domaine d'application existent pour l'utilisation de ces derniers. Dans ce projet nous nous intéresserons principalement aux techniques suivantes :

Lemmatisation

La lemmatisation est un terme désignant l'analyse lexicale d'un texte dans le but de regrouper les mots d'une même famille. Les mots d'une même famille sont donc réduits en une unique entité appelée « **lemme** ». Ainsi la lemmatisation consiste à regrouper les différentes flexions d'un mot unique.[1]

Segmentation

La segmentation d'un texte est l'opération de découpage de ce dernier en composantes linguistiques plus petites(des phrases, des groupes nominaux, des mots ...), c'est un processus non-trivial car chaque langue dispose de règles spécifiques en ce qui concerne les marqueurs de fin de phrases.

Étiquetage morphosyntaxique (PoS-Tagging)

il consiste à identifier pour chaque mot sa classe morphosyntaxique(Nom,Verbe,Nom pluriel, ...) à partir de son contexte dans un corpus ou texte ,ainsi que de connaissances lexicales de la langue.

Dictionnaire historique

Informellement, un dictionnaire historique est un ouvrage qui rassemble, sous forme d'un liste d'entrées, un ensemble de mot d'une langue donnée avec leurs définitions et/ou des exemples d'utilisation selon des périodes historiques prédéfinies (en rapport avec la langue ou pas).

Conclusion

Au terme de ce chapitre, nous avons une idée plus claire sur le travail qui doit être réaliser, nous allons donc attaquer l'aspect conceptualisation, il s'agira principalement de définir les composants de notre système.

CHAPITRE 2

CONCEPTION DU SYSTÈME

Introduction

Comme mentionné précédemment, nous allons nous intéresser dans ce chapitre à la conception que nous avons réalisé, nous présenterons un schéma global du système, puis nous nous détaillerons le rôle de chaque composant, en donnant un exemple d'utilisation et/ou du flux de donnés qui entre/sort de ce dernier, nous parlerons ensuite du déploiement du système dans une plateforme serverless(dans le cloud), principalement car c'est un aspect important de l'expérience d'utilisation(UX).

Schéma global du système

Notre système se compose essentiellement de deux parties(elles même subdivisées en plusieurs modules) :

— **Récupération et pré-traitement des données** : principalement, c'est depuis des sites web que le système cherche des données, puis il se charge d'organiser les fichiers téléchargés dans un espace de stockage.

— **Exploitation et mise à jour des données récupérées** : c'est la partie où les données qui sont maintenant structurées et organisées seront utilisées par l'application, qui dans notre cas se trouve être une application web hébergé dans le cloud.

Le schémas suivant explicite un peu plus l'explication précédente :

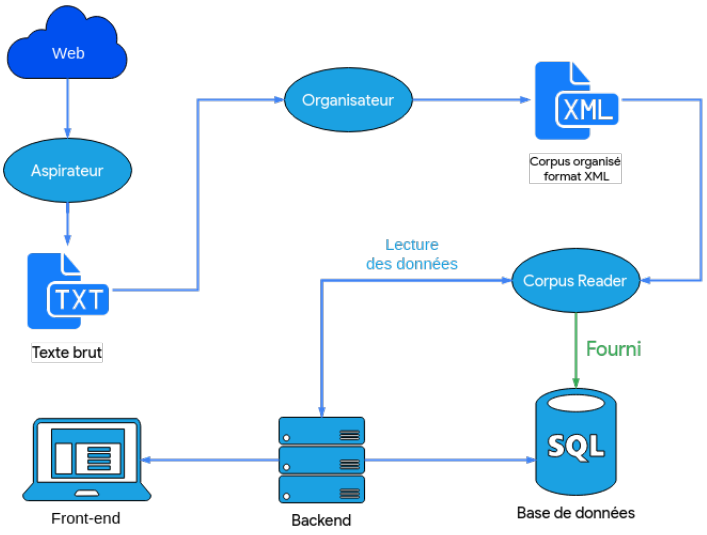


Figure 2.1 – Schéma global du système

Les modules du système

Cette section se consacre à la description en détails des différentes modules du système, avec éventuellement des exemples d'utilisation de chacun.

Aspirateur de sites web

Le module d'aspiration est composé de plusieurs petits scripts, le premier d'entre eux **initializer.py** , son rôle est d'initialiser certaines variables globales (comme les dates de début/fin des périodes historiques de la langue arabe (العصر الجاهلي ، عصر صدر الإسلام ...), ou encore les chemins d'accès des différents fichiers récupérés et des fichiers xml produits. Il permet aussi la création dynamique d'une arborescence de répertoire où placer chaque fichier aspiré selon son époque et son genre.

Les autres scripts sont destiné à aspirer les documents d'internet en faisant appel aux fonctions définies par le module précédent afin que l'organisation des fichiers brutes se fait de la même manière pour tous les scripts. Nous pouvons citer les différents **"scrapeur"** [1] utilisé :

- **books_scrape.py** Il a pour but d'aspirer le contenu du site alwaraq.net, qui est un recueil de livres arabes.
- **chi3r_scrape.py** Son but est d'aspirer le contenu du site aldiwan.net, qui est une collection de poèmes arabes organisés par auteurs, périodes, pays ...,
- **islamicbook_scrape.py** Son but est d'aspirer le contenu du site islamicbook.ws, qui est un recueil d'ouvrages dédiés à l'islam.
- **news_scrape.py** Son but est de télécharger les corpus déjà organisés depuis le site aracorpus.e3rab.com qui est une banque de blogs rédigés en arabe.
- **quran_corpus_builder.py** par respect pour les verset coraniques, nous avons décidé de télécharger un corpus déjà organisé sous format xml et certifié comme étant authentique par le groupe **Tanzil Project**(voir tanzil.net), puis nous l'avons ré-organisé en utilisant le script en une structure **XML** (que nous définirons plus bas).

Il est à noter que tous ces scripts utilisent des fonctions d'organisation communes, pour placer chaque fichier acquis dans son répertoire adéquat dans l'arborescence créée par **initializer.py**, cela a pour but de faciliter l'accès ultérieur aux données brutes, pour les nettoyer et construire les corpus au format **XML**.

Autre point important, durant l'aspiration des données, on s'est souvent retrouvé dans le cas où l'information sur la période historique manquait, pour palier à ce problème, et puisque tous les documents disposent d'un auteur, nous avons utilisé l'API Wikipédia pour rechercher les informations liées à l'auteur (Date de naissance,mort,nationalité ...), déterminant ainsi la période dans la quelle l'ouvrage a été rédigé.

Organisateur de corpus

Ce module est composé principalement d'un seul script qui effectue deux tâches, une fonction de nettoyage **"cleaner"**, et un constructeur de fichiers **convertScrapedToXml** pour générer le fichier **XML** associé à chaque texte brut extrait. Le format des fichiers XML que nous avons choisi est le suivant :

```
<root encoding="utf-8">
 <metadata>
  <book_name>ألا حي بالبردين دارا ولا أرى</book_name>
  <era>Umayyad</era>
  <author>
   <name>جرير</name>
   <birth>650</birth>
   <death>728</death>
   </author>
  <id>116</id>
  <type>شعر</type>
  <size>66</size>
 </metadata>
  <doc>
   <sentence>ألا حي بالبردين دارا ولا أرى</sentence>
   ...
   <sentence>كريما ولم تعلق عنانا يقيمها</sentence>
```

1. Terme anglais pour désigner un aspirateur de site web

```
        </doc>
    </root>
```

Remarque : Pour les textes coraniques, chaque **sura** (سورة) sera représenté par un fichier **XML**, dont les balises **<sentence>** contiendront ses versets (آيات السورة)

Chaque document du corpus sera donc divisé en deux ensembles de balise, une partie méta-données contenant une variété d'information sur cet élément du corpus (Nom de l'auteur, période historique, id , type ...), puis une partie données, qui contiendra plusieurs phrases qui composent le document.

Après avoir construit l'arborescence, l'initialisation du système sera presque complète (il faudra remplir les tables de la base de donnés, nous verrons le schéma relationnel plus tard), il s'agira maintenant de passer la main au module suivant pour l'exploitation de ce corpus.

Corpus reader

Ce module joue le rôle d'une interface d'abstraction entre les fichiers XML et l'utilisateur qui désire récupérer ces données, ce dernier devra instancier un objet de la classe **HistoricalCorpus**, cette classe hérite directement de la classe **nltk.corpus.XMLCorpusReader**, en redéfinissant les méthodes approprié, nous pourrons donc accéder au corpus et ses composants en invoquant des méthodes comme :

```
def fileids(self,era=None,category=None):
```

Cette fonction retourne une liste de tous les documents du corpus, ou une partie selon les valeurs des paramètres qui lui sont passer (période,catégorie...).

```
def sents(fileid=None,start=None,end=None,era=None,category=None):
```

Cette fonction retourne une liste de phrase du corpus tout entier, ou d'une portion selon les valeurs des paramètres qui lui sont passer (période,catégorie...).

```
def tagged_sents(fileid=None,start=None,end=None,era=None,category=None):
```

Retourne une liste de phrases avec leurs étiquetages morpho-syntaxique.

```
def lemma_sents(self,fileid=None,start=None,end=None,era=None,category=None):
```

Retourne une liste de phrases auxquelles une lemmatisation à été appliquée.

```
def words(fileid=None,start=None,end=None,era=None,category=None):
```

Retourne une liste de tous les mots du corpus (ou d'une partie du corpus).

```
def tagged_words(fileid=None,start=None,end=None,era=None,category=None):
```

Retourne une liste de tuples (mots,Étiquette-Morpho-Syntaxique) de tout les mots du corpus (ou d'une partie du corpus).

```
def lemma_words(self, fileid=None,start=None,end=None,era=None,category=None):
```

Retourne une liste de de tout les lemmes (forme infléchie des mots) du corpus (ou d'une partie du corpus)

```
def word_apparitions_gen(self,dictionarySet,fileid=None,era=None,category=None,stop_words=None,
get_sentences=False,context_size=5,
lemma=True):
```

Méthode utilisant un Generator(voir MMMM) pour parcourir le corpus, en extraire les portions de texte qui contiennent un des mots passé en paramètre ($dictionarySet$), cette méthode sera utilisée dans le mode-automatique (VOIR HHSDJHSD).

Remarque La force majeure de ce module est qu'il fait appel a plusieurs "Generator"[2], en effet, par souci de performance, sachant que nous disposant d'un très gros volume de données sur le disque, charger toutes ces données dans la mémoire à chaque appel d'une des fonctions méthodes du corpusReader serait très coûteux en terme de temps et d'espace, l'utilisation de Generators permet donc de palier à ce problème en ne chargeant en mémoire que le peu d'information dont on a besoin, c'est une génération de valeurs **à la demande**, évitant ainsi une sur-utilisation de l'espace mémoire.

Il reste donc à voir comment le serveur remplit la base de données.

Base de données

Le but d'avoir cette base de données et principalement de stocker les différentes méta-données sur les documents du corpus, ainsi que les relations entres les contenus des documents, cela facilitera la tâche au serveur de ne considérer que le niveau le plus abstrait des données, et laisser la tâche de récupérer et manipuler ses données au CorpusReader.

Nous avons choisis le schéma relationnel suivant pour la base de données :

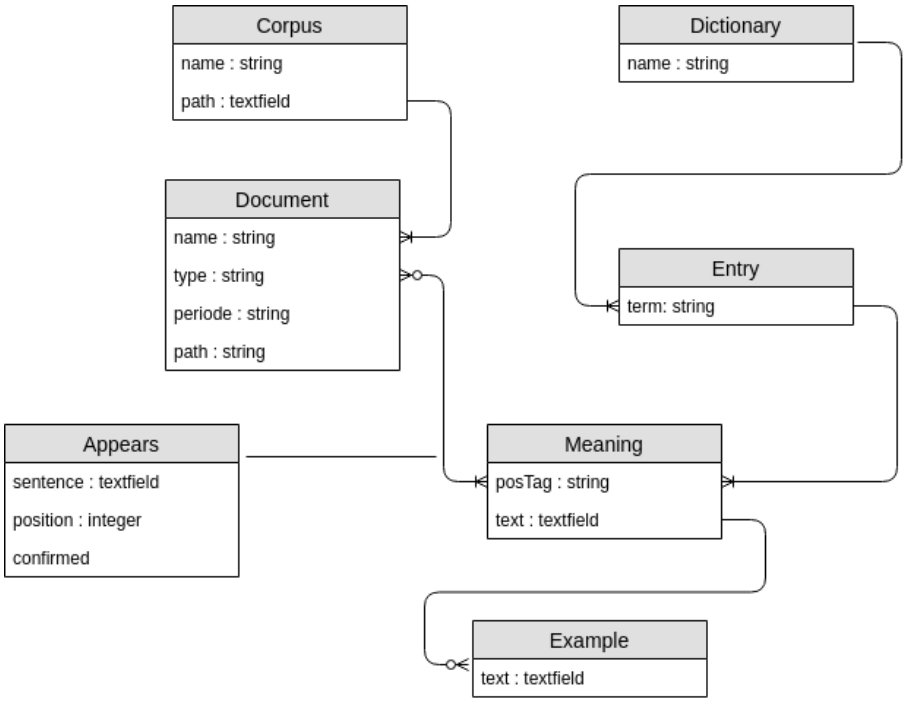


Figure 2.2 – Schéma de la base de données

La table **Dictionary** est rempli avant la fin de la phase d'initialisation, nous avons pour cela utilisé les données provenant de deux sources :

— المعجم الرائد

— المعجم الجامع

— المعجم الوسيط

2. Les Generators dans python sont des fonctions qui se comportent comme un Iterator, c'est ç dire qu'il peuvent être utilisé dans une boucle

Application (Front-end et Back-end)

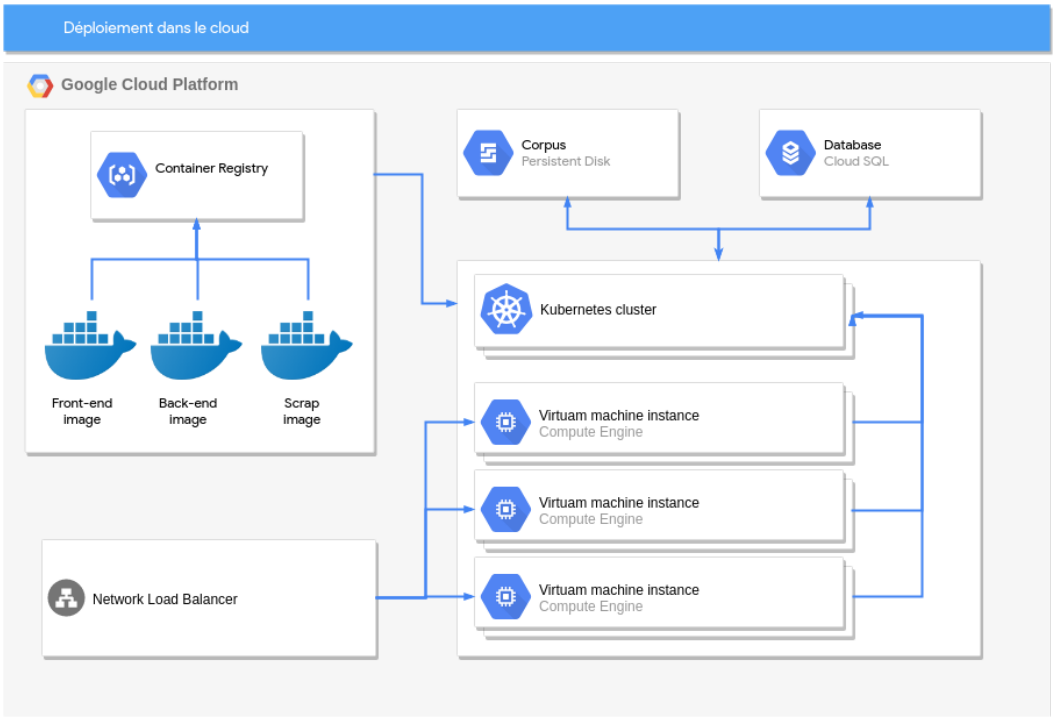
Back-end : C'est une application web qui expose une api REST-ful , elle est accessible a partir de certains **end-points**[3] qui seront exploités avec des requêtes **http**, l'avantage d'une telle architecture est qu'elle est 100% inter-opérable (elle ne dépends ni du système, ni des technologies ni du langage du serveur ou des clients).

Front-end le front-end utilise ces **endpoints** pour récupérer de la donnée et l'afficher à travers des requêtes lancée selon les besoins de l'utilisateur (en manipulant l'interface par exemple). On peut aussi développer une application mobile ou desktop de la même façon.

Déploiement des modules dans le cloud

Nous avons choisi de déployer notre application suivant une architecture server-less dans le but de faciliter son utilisation, de la rendre publique et disponible à tout ceux qui voudraient la tester, donner un feedback ou proposer une amélioration en consultant le dépôt sur Github

Le diagramme suivant résume le processus de communication entre les composantes du système dans le cloud :



Conclusion

Au terme de chapitre, nous avons donc fais le tour des principales composantes du système, sans trop rentrer dans les détails, nous introduirons les composantes non-citées dans le chapitre suivant lors de la présentation des fonctionnalités de l'application si besoin est.

3. Emplacement dans le serveur renvoyant une réponse à une requête

CHAPITRE 3

RÉALISATION DE L'APPLICATION

Introduction

Durant ce chapitre, nous allons présenter l'aspect pratique de notre système, à savoir l'application qui lui est dédiée, nous commencerons par une description des outils utilisés, suivit d'une présentation de l'interface d'utilisation pour enfin introduire les fonctionnalités supplémentaire non-apparente dans cette dernière.

Environnement de travail et outils utilisés

Python

Python est un langage de programmation scripté très puissant, de par sa simplicité d'utilisation et de sa syntaxe très intuitive, il est aussi un des langages préférés des adeptes du TALN, pour la raison qu'un grand nombre de bibliothèques y sont disponibles. Nous l'avons choisi principalement pour des traitement de TALN mais aussi dans la partie back-end du système.

JavaScript

Très populaire chez les développeur web, JavaScript(JS) est un outil très puissant pour un développement rapide d'application web rapide, puissantes et élégantes.

NLTK

Natural Language Toolkit (NLTK) est un framework (ensemble de librairies) implémentant un grand nombre d'algorithmes et modèles du TALN, écrite en python elle sera donc parfaitement compatible avec notre back-end qui lui sera développé avec **DJANGO** (voir suite).

VueJS

Vue.js est un framework écrit en JavaScript pour le développement d'interface web, basé sur le principe de **components**, Vue donne au développeurs une grande liberté et flexibilité pour la création d'interface dynamiques et **reponsive**.

Django

Django est un framework écrit en Python qui facilite le développement du back-end d'un site web, il s'occupe de gérer les couches basses de ce dernier (sessions, sécurité...) et peut même générer une interface d'administration automatiquement. L'objectif de Django est de proposer un développement plus efficace et plus rapide d'un site web dynamique tout en maintenant sa qualité.

PostgreSQL

PostgreSQL est un système de gestion de base de données relationnelle et objet (SGBDRO). C'est un outil libre disponible selon les termes d'une licence de type BSD.

Google Cloud Platform

Google Cloud Platform est une plateforme de cloud computing fournie par Google, proposant un hébergement sur la même infrastructure que celle que Google utilise en interne pour des produits tels que son moteur de recherche1. Cloud Platform fournit aux développeurs des produits permettant de construire une gamme de programmes allant de simples sites web à des applications complexes.

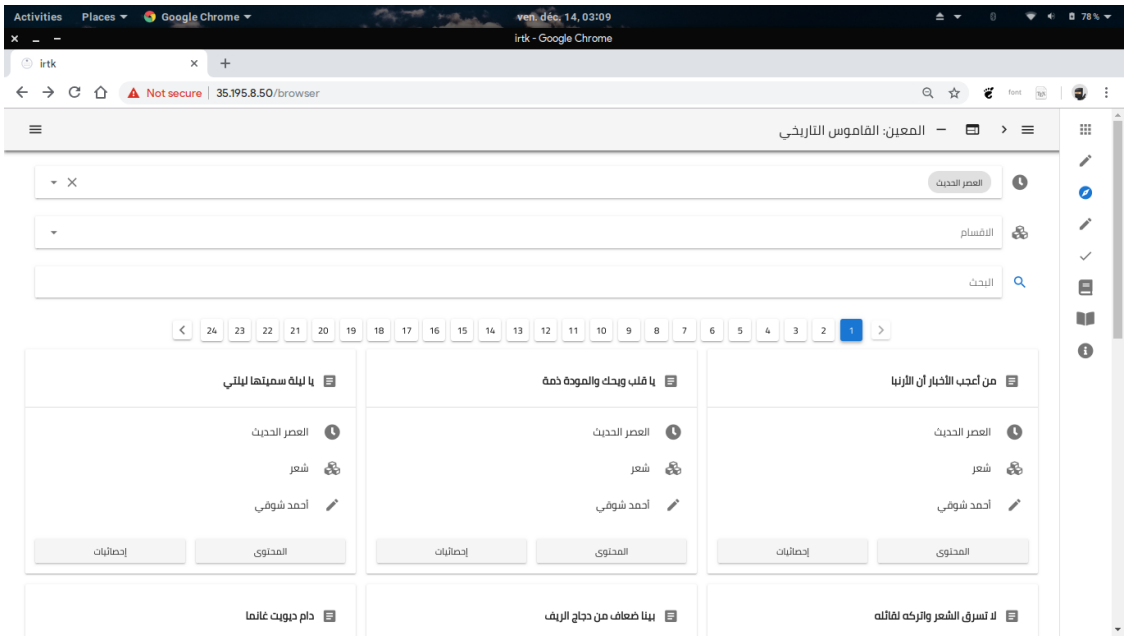
Présentation de l’application

Interface principale

L’application se présente comme composé de trois partie, une barre d’outils en haut, un panneau d’onglet latéral permettant d’accéder au différentes section de l’application, et une zone d’affichage principale pour afficher le contenu de chaque section.

Explorateur de corpus

تصفح الملفات

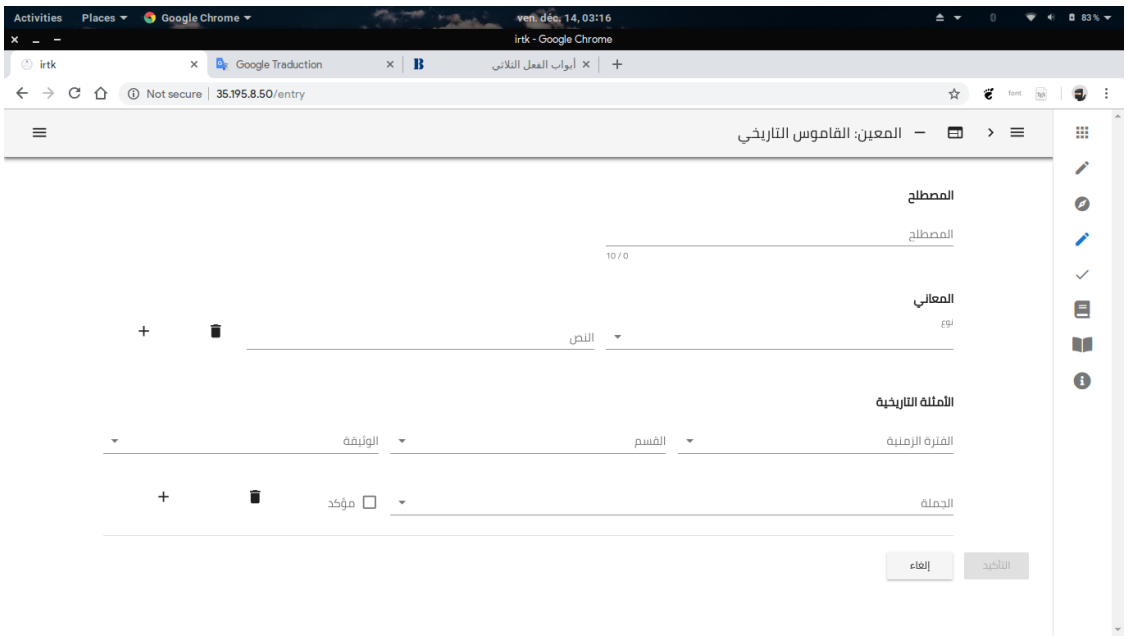


Dans cette section, l’utilisateur pourra parcourir chaque composant du corpus, le choix lui est donné pour filtrer sa recherche, que ce soit par période (الفترة الزمنية), par section (قسم) ou par mots clés, le résultat est une succession de pages dynamiquement ajoutées à la mémoire (grâce au corpusReader), chaque page est composé de tuile contenant les informations relatives au documents pertinents selon le filtre de recherche.

En cliquant sur le bouton **content** (المحتوى), une nouvelle section est affichée, elle permet à l’utilisateur de lire le fichier sélectionné. De plus s’il désire ajouter une nouvelle entrée au dictionnaire historique, il lui suffit de sélectionner la partie du texte qui l’intéresse , une liste déroulante s’affiche lui proposant d’ajouter un des mots précédemment sélectionnes au dictionnaire (nous verrons les détails de cette opération das la section suivante) en remplissant automatiquement les camps nécessaire.

Ajouter entrée

أضف مصلح



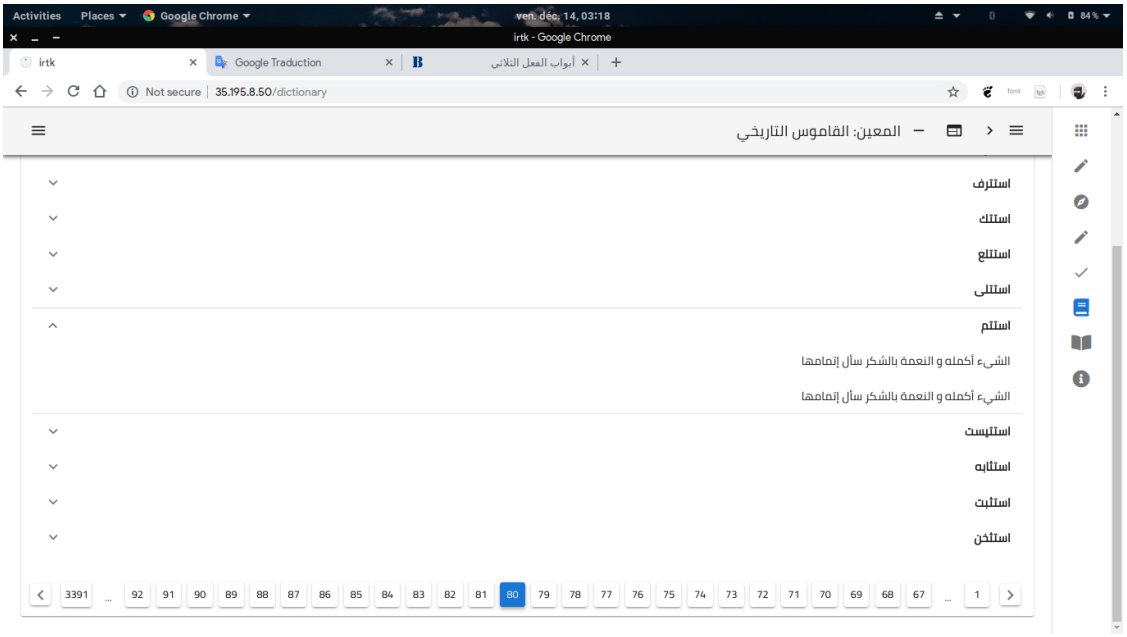
Cette section donne la main au lexicographe pour ajouter une nouvelle entrée dans le dictionnaire historique, pour ce faire il faut entrer le terme à ajouter (المصطلح), il faudra ensuite remplir un (ou plusieurs) champs selon le besoin de l'utilisateur :

— Une liste de lignes réservées aux sens du termes à ajouter, il faudra fournir une étiquette morpho-syntaxique parmis une liste déroulante de choix possible (النوع), suivit d'un extrait de texte où le terme apparaît (النص). L'utilisateur pourra répéter ce processus autant de fois qu'il désire ajouter de sens au terme.

— Une liste de lignes réservées aux information chronologiques (الأمثلة التاريخية), il faudra fournir la période historique (الفترة الزمنية), la section visée (القسم) , le document (الوثيقة) puis finalement un exemple d'utilisation (الجملة), un champ **confirmé** est à cocher pour valider les informations saisies, sinon l'ajout n'est pas autorisé.

Dictionnaire

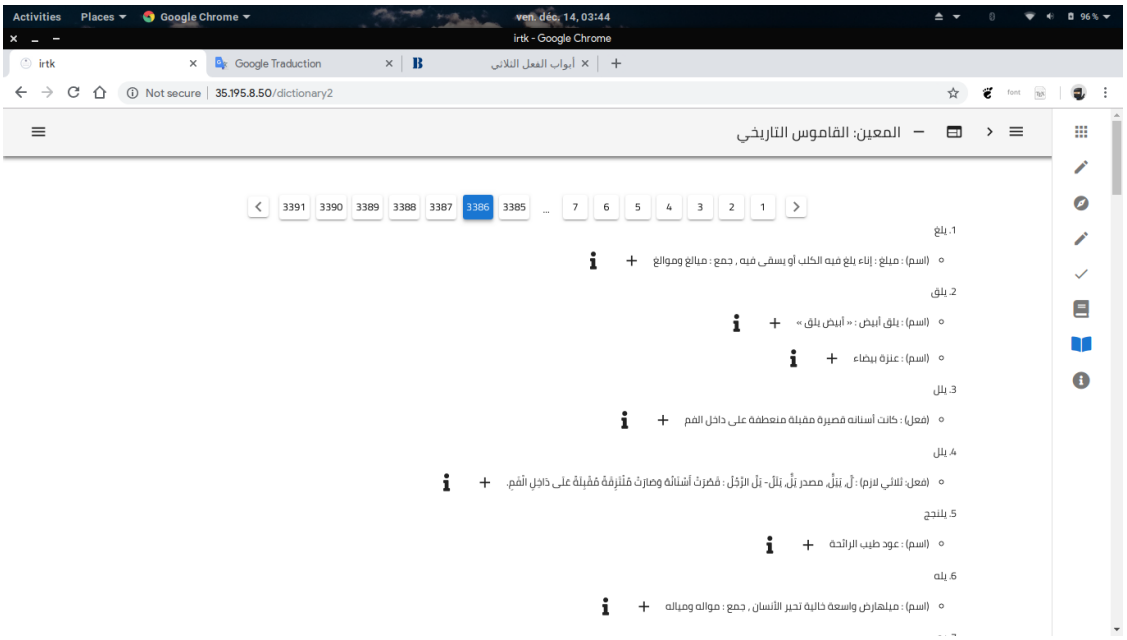
القاموس



Tout simplement cette section récupère tout les mots du dictionnaire classiques, et affiche les différents sens liés au mots auquel on s'intéresse.

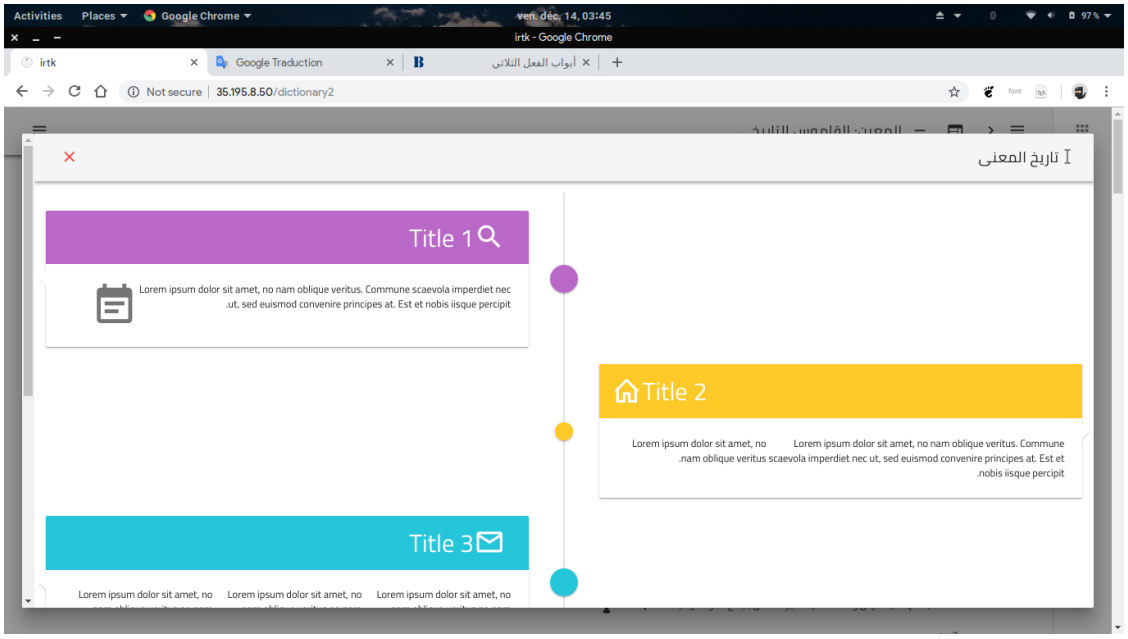
Dictionnaire historique

القاموس التاريخي



Dans cette section, nous trouverons notre dictionnaire historique, organisé comme un dictionnaire classique avec des entrées(des mots) et leurs définitions, mais avec l'option de consulter les différentes utilisation du mot lors des siècles en cliquant sur un bouton, une timeline[1] s'affiche en donnant un récapitulatif chronologique du mot :

1. Ligne chronologique passant par les périodes historiques



L'option **+** permet de modifier cette entrée, elle renvoie à la section **add-entry** mais avec le champs **terme** déjà rempli avec le terme courant à modifier.

Statistiques

إحصائيات

Display different statistics for some words

Fonctions supplémentaires

Mode-automatique

Le but de ce mode, est l'ajout automatique de nouvelles entrées dans le dictionnaire historique, en recevant en entrée une liste de mot, et en matchant les apparitions de ces mots dans le corpus, récupérant ainsi les méta-données nécessaires pour créer (résp. modifier) une entrée.

Conclusion

Au terme de projet, nous avons donc présenté les principales fonctionnalités de notre application, néanmoins, par des soucis de concision et de brièveté, des fonctionnalité internes n'ont pas été mises en avant, la documentation (manuel d'utilisation fournira plus de détails sur ces dernières).

CHAPITRE 4

CONCLUSION GÉNÉRALE

Objectifs atteints

Durant ce projet, nous avons tenté de proposer une contribution à la résolution d'une problématique datant de plusieurs siècles, qui est l'élaboration d'outils pour accélérer le processus de construction d'un recueil lexicographique historique de notre langue maternelle. Ayant un tel objectif en tête ainsi qu'en travaillant avec des données réelles tout en utilisant des technologies récentes, les adaptant à notre problématique, a permit de mieux apprécier le fruit du travail fournit pour la réalisation de ce projet.

Néanmoins cela reste un modeste travail, très incomplet et présentant beaucoup de lacunes que nous allons mentionner dans la section suivante.

Limites du système

Notre décision de déployer l'application sur une plateforme enligne nous a poussé a revoir nombre de fois notre conception, essayant d'optimiser les opération internes, de minimiser le temps de latence pour q'un utilisateur profite d'une expérience fluide et agréable, cependant, n'ayant pas à notre disposition un serveur assez puissant (en y ajoutant les possibles erreurs de conception pour la gestion des ressources), en y ajoutant le très grand volume de données à traiter, le système peut souffrir d'un temps de réponses très instable.

Un autre point à soulever est la simplicité du mode automatique, le matching se faisant d'une manière rudimentaire, consomme trop de temps et ne prend pas en compte le contexte d'apparition des mots.

Perspectives futures

Parmi les améliorations qui serait possibles, nous citerons :

— l'utilisation de techniques de TALN ou NLU(Natural Langage Understanding) pour améliorer le filtrage des mots dans le mode automatique, ajouter une couche sémantique permettrait de catégoriser les mots selon leurs sens.

— Monopoliser plus de ressources (du coté du serveur) pour améliorer les performances lors de l'utilisation.

— Continuer l'intégration de nouvelles sources de données dans le corpus, pour enrichir ce dernier avec de la quantité, de la qualité et de la variété.

BIBLIOGRAPHIE

[1] “La lemmatisation : Optimiser le seo avec la lemmatisation.” https ://www.yakaferci.com/lemmatisation-seo/. (Accessed on 12/13/2018).