Université de Bordeaux
Master 1 Informatique

Projet de Programmation

Le Clicodrome pour Lexique Electronique

par Lionel CLEMENT

Cahier des Besoins

réalisé par
MEHDAOUI Abdelamine
ROUSSEL Damien
EL GUERCH Souhail
DAOUDI Yassir
DIALLO Abdoul Ghadiri

Enseignant responsable : Philippe NARBEL

4 février 2020

Table des matières

1 Introduction

1.1 Présentation du projet

Dans le cadre de notre Cursus de Master 1 Informatique à l'Université de bordeaux, nous devons au compte de l'unité d'enseignement Projet de Programmation réalisé par groupe de cinq étudiants un projet de développement informatique. Nous travaillons avec Monsieur **Lionel CLEMENT** notre client et monsieur **Simon ARCHIPOFF** chargé de l'encadrement de nos travaux dirigés.

Lionel CLEMENT, spécialisé dans le domaine de la linguistique formelle et du traitement automatique des langues est un enseignant chercheur au Laboratoire Bordelais de Recherche en Informatique(**LABRI**). Il a réalisé avec **Benoit SAGOT** [8] chercheur à l'Institut National de Recherche en Informatique et Automatique(**INRIA**) le Lexique électronique des formes fléchies du français, qu'ils ont appelé **LeFFF**[2][3].

Le français est une langue, une langue peut-être définie comme un ensemble de signes oraux et écrits qui permettent à un groupe d'individus de communiquer. Les mots qui forment le français se repartissent en neuf classes : les noms, les déterminants, les adjectifs qualificatifs, les pronoms, les verbes, les adverbes,les prépositions, les conjonctions de coordination et les interjections. Partant de cette classification on peut dire que les formes fléchies d'un mot correspondent aux différentes déclinaisons de ce mot (sa conjugaison pour un verbe, son singulier et son pluriels pour certaines catégories de mots, ses synonymes,...)
Dans le **LAROUSSE**, un **lexique** est un dictionnaire spécialisé et généralement succinct concentrant des connaissances dans un domaine particulier, c'est une ressource complexe constituée de *lexèmes*, de *vocables*, de *catégories* et *sous-catégories syntaxiques*, de *catégories grammaticales*, de *règles de flexion*, de *valence*, de *phraséologie*, de *fonctions lexicales*,...

Habituellement un dictionnaire est utilisé pour trouver la signification d'un mot qu'on connaît et il n'offre pas la possibilité de faire l'inverse, c'est à dire qu'à partir d'explications ou d'une liste de mots trouver le mot correspondant (ce qui offrirait un gain de temps considérable).

L'objectif de ce projet est donc d'implémenter une application Web liée à une base de données (qui contiendra les mots du lexique) pour faciliter les interactions avec le lexique (consultation, ajout, suppression,...).

1.2 Description de l'existant

1.2.1 Le lexique : leFFF

Le **LeFFF** est un fichier texte contenant une base de mots, mais il n'existe aucun outil qui en facilite l'accès et la compréhension encore moins sa modification ou son enrichissement. Le **LeFFF** actuel souffre d'un manque de traçabilité et n'offre aucune garantie pour contrôler les modifications. Nous utiliserons donc ce fichier contenant une base très importante de mots de français implémentant la structure du **FFF**. Ce fichier nous permettra d'avoir une base de mots, ainsi qu'une structure rendant plus simple à implémenter les algorithmes de recherche du **lexique** en utilisant des **transducteurs** et des **unificateurs**.

Dans le **LeFFF**, il y a deux formats :

— **le format Intensif** dont les entrées sont dans le format suivant :

```
afficheur nc-eur 100;Lemma;nc;<Objde:(de-sinf|de-sn),Objà:(à-sinf)>;cat=nc;
                    %default
afficher v-er:std 100;Lemma;v;<Suj:cln|sn,Obj:cla|sn>;cat=v;
                    %actif,%passif,%ppp_employé_comme_adj
```

Ce format a une structure plus ou moins identique pour chaque catégorie grammaticale, et ne contient que le radical des mots, ces radicaux seront transformés pour donner toutes les formes du mot, donnant le second format.

— **le format Extensif** qui est donc la version compilée du format **intensif**, contient donc tous les mots sous toutes leurs formes. Ce format est obtenu par l'application d'un transducteur pour chaque mot formant toutes ses formes selon sa catégorie.

Nous utiliserons le **format intensif** car seul le radical importe pour le sens. Nous utiliserons des **transducteurs** afin de pouvoir générer le radical des mots recherchés, puis nous utiliserons un **unificateur** pour pouvoir lier les mots entre eux et pouvoir obtenir le sens de l'expression formée

1.2.2 Le formalisme PFM

Dans la documentation [6] de **Olivier Bonami** et **Gilles Boyé**, le formalisme Paradigm Function Morphology(**PFM**) est une théorie explicite de la morphologie flexionnelle qui présente une conclusion à partir d'un fait, d'une situation. Les affixes ne sont pas traités comme des signes, mais comme des résultats de l'application d'une règle liant les caractéristiques morphosyntaxiques à une fonction phonologique qui modifie une base. Dans le formalisme PFM, le système flexionnel d'un langage est modélisé par une fonction de paradigme. Les fonctions de paradigme prennent en entrée une racine et un ensemble de fonctions, et retournent un ensemble phonologique.

La forme générique des fonctions paragigmes est : **PF(l, σ) = IV o III o II o I (l,σ)(ϵ)**

— l : lemme auquel on souhaite appliquer le formalisme

— σ : série de tags que l'on souhaite appliquer au lemme

— I, II, III, IV : Niveaux d'applications des règles

De manière plus formelle, le formalisme indique pour un lemme donné avec une série de tags, une règle pour chaque niveau d'application. Pour chaque niveau, uniquement la règle correspondant aux tags renseignés va être appliqué au lemme.
Si pour un niveau d'application donné, aucune règle ne correspond, le formalisme passe au niveau d'application suivant.
Si pour un niveau d'application, plusieurs règles correspondent, celle qui s'applique sera celle qui comporte le plus de tags (la règle la plus spécifique donc la plus forte). Le **format** de base des **règles** est le suivant : **n, X, t $\Longrightarrow$ f(X)**

— **n** : niveau d'application de la règle

— **X** : lemme

— **t** : combinaison de tags pour laquelle la règle s'applique (Tous les tags doivent être renseignés par l'appel du formalisme pour que la règle puisse s'appliquer

— **f(X)** : la forme flexionnelle

Prenons un **exemple** concret du formalisme PFM avec les règles suivantes :

— I, chaîne,f $\Longrightarrow$ chaîne + "ne"

— II, chaîne, p $\Longrightarrow$ chaîne + "s"

On applique le formalisme sur le lemme "Mien" pour en obténir le féminin pluriel :
PF(Sien, f,p) = IV o III o II o I (Sien, f,p)(ϵ) *Application de la règle I*
= IV o III o II (Mien, f,p)(**Mienne**) *Application de la règle II*
= IV o III (Mien, f,p)(**Miennes**) *Aucune règle à appliquer*
= IV (Mien, f,p)(**Miennes**) *Aucune règle à appliquer*

= **Miennes**

Dans cet exemple, on souhaite appliquer le féminin et le pluriel au lemme "Mien". La syntaxe montre l'application des règles de niveau 1 avant l'application des règles de niveau 2. Cette **hiérarchie** des niveaux d'application est nécessaire pour **éviter les incohérences**. Ce système garantie que le résultat sera "Miennes" et non "Miensne".
D'autres exemples sont disponible dans la documentation de Olivier Bonami [1].

2 Besoins fonctionnels

Lors de nos différents échanges avec le client nous avons pu identifier ses besoins pour ce projet. En effet, la réalisation de ce projet nécessite plusieurs parties :

2.1 Partie Langue

Mettre en place une base de données qui utilise le niveau intensif du **LeFFF** pour former les différents mots du **lexique**.

2.1.1 Génération des formes fléchies - Importance 5/5

Ce besoin est le **besoin principal** de notre projet. Si notre base de données devait contenir en plus des mots du lexique, toutes les expressions possibles qui vont avec, le projet aurait été vraiment très simple (c'est à dire un site internet qui permet d'enregistrer et de manipuler des informations). En effet, une **génération dynamique** des formes fléchies permettra une **réduction** considérable de la **taille du lexique** à enregistrer dans la base de données. Pour remédier à ce problème, il nous faut donc mettre en place un outil qui respecte les **règles linguistique** pour générer les formes fléchies.

2.1.2 Base de Données - Importance 5/5

Nous allons utiliser une **base de données relationnelle**. A première vue il aurait été plus judicieux d'utiliser une base de données non relationnelle du fait du nombre volumineux de données à traiter. Néanmoins le choix d'utiliser ici une base de données relationnelle nous permettra une plus **simple** gestion des données en particulier pour la recherche. Afin **d'optimiser** la **recherche** nous utiliserons une ou plusieurs de ces 3 solutions :

- Des **fichiers triés**, rapide pour la recherche (recherche dichotomique), mais lents pour l'insertion et la suppression.
- Des **fichiers hachés** efficaces pour les sélections avec égalité.
- Des **index** (pour chaque lettre de l'alphabet) pour permettre d'améliorer certaines opérations sur un fichier.

2.1.3 Rechercher un lexique - Importance 5/5

Nous offrirons la possibilité de rechercher un **lexique** de la même manière qu'un dictionnaire, mais aussi en permettant de le chercher en fonction de son **lemme** et **lexème** (sa définition ou synonyme). Par exemple l'entrée en recherche "content", nous pourrons retrouver en sortie "content", "heureux", "être à la fête", "être aux anges", "être gai comme un pinson",... Par la suite, nous pourrons afficher toutes les formes possibles pour chacun des résultats retournés comme par exemple singulier, pluriel, masculin, féminin ou temps conjugués pour les verbes. Un exemple de test réalisable serait de rechercher un mot et de refaire une recherche avec l'un des résultats obtenus pour comparer les deux recherches.

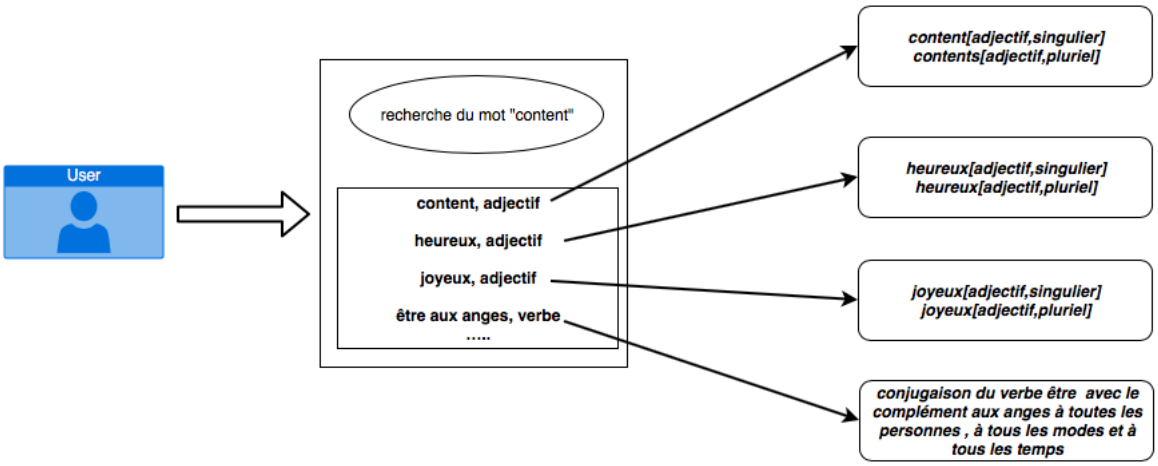

FIGURE 1 – Scénario d'utilisation pour la recherche d'un mot

2.1.4 Transducteur - Importance 5/5

La composition des **transducteurs** nous permettra de construire un **lexique** spécifique à la nature de l'entrée, nous réaliserons des **transducteurs** qui vont **conjuguer** des verbes, d'autres qui vont **transformer** du masculin au féminin, du singulier au pluriel, ou **inversement**. Les algorithmes de construction des transducteurs pourront prendre en compte des **filtres** afin d'affiner la recherche et la rendre plus rapide.Chaque **arcs** correspondra à une **spécification** du mot (symboles signifiant la nature et la morphologie d'un mot).

Il faut aussi mettre en place un **transducteur** qui aura pour fonction de **récupérer** le **radical** d'un mot fourni pour se faire nous utiliserons plusieurs automates finis afin de détecter sa catégorie grammaticale, pour ensuite connaître sa forme dans le but de retourner son radical.

Inversement, à partir d'un autre transducteur, celui ci devra retourner toutes ses formes.

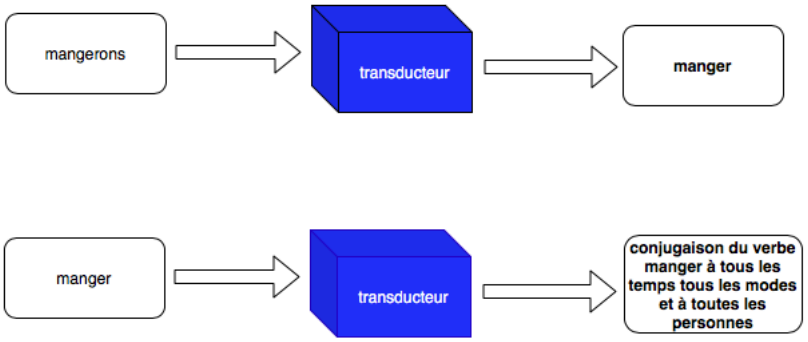

FIGURE 2 – transducteur

2.1.5 Unificateur - Importance 5/5

L'unificateur est l'algorithme qui va nous permettre de **lier** plusieurs **mots** afin d'en tirer le **sens**, pour ce faire, il faut faire **l'union des informations** sur ces mots. S'il y a une **contradiction** avec ces informations, alors la phrase est **mal formée** et dans ce cas aucun sens ne sera retourné, sinon on pourra voir quel est le sens de l'expression formée.

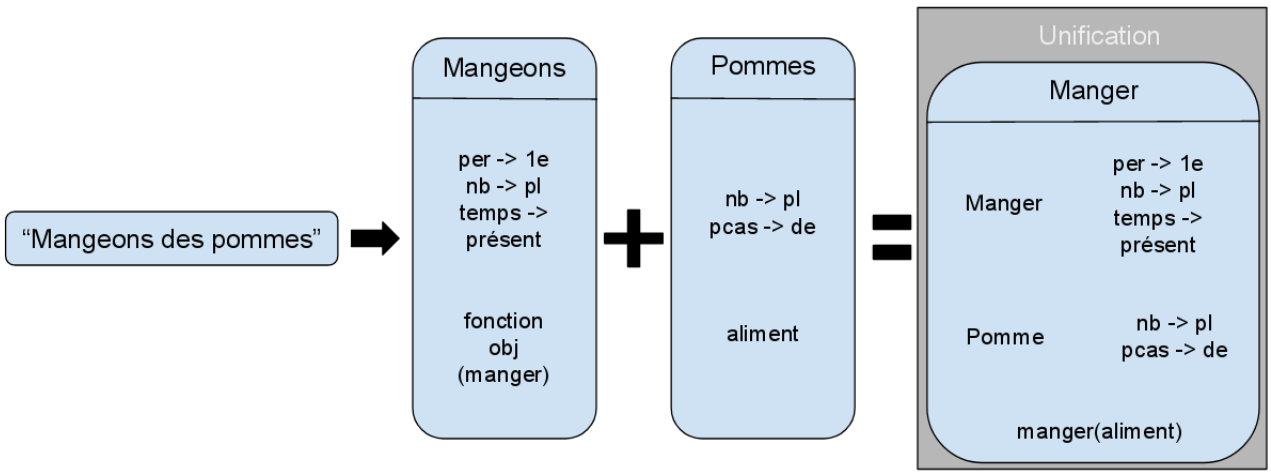

FIGURE 3 – Unificateur

Les mots utilisés sur **l'unificateur** utiliserons une **sous-structure commune** afin de faciliter et de rendre sensée cette opération, puis à l'aide de la **théorie de substitution des ensembles**, nous pourrons **optimiser** la construction de nos formules. Celles-ci nous permettrons par la suite d'**améliorer** la **rapidité** de l'algorithme de recherche, ainsi que la **mémoire** utilisée.

2.2 Partie Web

Le développement d'une **application Web** permettant aux utilisateurs de facilement interagir avec le **lexique**.

2.2.1 Ajouter un mot au lexique - Importance 4/5

Nous offrirons la possibilité d'ajouter un mot au **lexique** en rentrant les données de la structure du **FFF**. Seules les **radicaux** des mots seront a rajouter, les différentes formes seront automatiquement générées par les transducteurs. Pour ajouter un mot, nous devrons entrer des informations sur celui-ci comme son sens et sa catégorie grammaticale, pour pouvoir ensuite être reconnu par l'algorithme de recherche demandé ci-dessus. Un exemple de test réalisable serait de voir si on trouve bien les éléments rajoutés avec la recherche de lexique.

2.2.2 Modifier un mot - Importance 4/5

Nous offrirons la possibilité de modifier un mot du **lexique** en modifiant un ou plusieurs champs de la structure du **FFF** de ce mot utilisés lors de l'ajout de celui-ci. Cette fonction permettra entre autre de corriger des erreurs d'ajout dans le lexique comme par exemple faute

d'orthographe, mauvaise catégorisation,... Un changement de sens devra être répercuté dans tout le lexique du fait de sa génération automatique. Il sera possible de tester cette fonctionnalité en recherchant l'élément modifié directement et de constater ses changements (toutes les possibilités de changement seront donc a tester).

2.2.3 Supprimer un mot du lexique - Importance 1/5

Nous offrirons la possibilité de supprimer un mot du **lexique**, afin de les enlever de toute la base de données utilisée, et donc de faire en sorte à ce que l'algorithme de recherche ne le trouve plus. Cette fonction ne sera normalement que très peu utilisée car l'ajout et la modification du lexique est contrôlée par une gestion des rôles, rendant cette fonctionnalité peu utile. Un exemple de test réalisable serait de faire une recherche et par la suite de supprimer un des résultats, pour enfin réitérer la recherche sur tous les résultats pour vérifier que l'élément supprimé est bien supprimé.

2.2.4 Gérer les rôles - Importance 2/5

Nous mettrons en place un système à trois **types d'utilisateurs** :

— **Utilisateur** : peut **s'inscrire, valider, supprimer** son compte, et peut **consulter** et extraire le lexique.
— **Contributeur** : peut faire ce que fait un utilisateur, ainsi que **modifier** le lexique.
— **Administrateur** : peut faire ce que fait un modérateur, ainsi que **gérer les droits** des autres utilisateurs (faire passer certains utilisateurs contributeurs et vis versa).

Cette façon de gérer les rôles reste simple mais efficace qui permettra de limiter au maximum les erreurs lors d'ajout d'éléments dans le **lexique**.

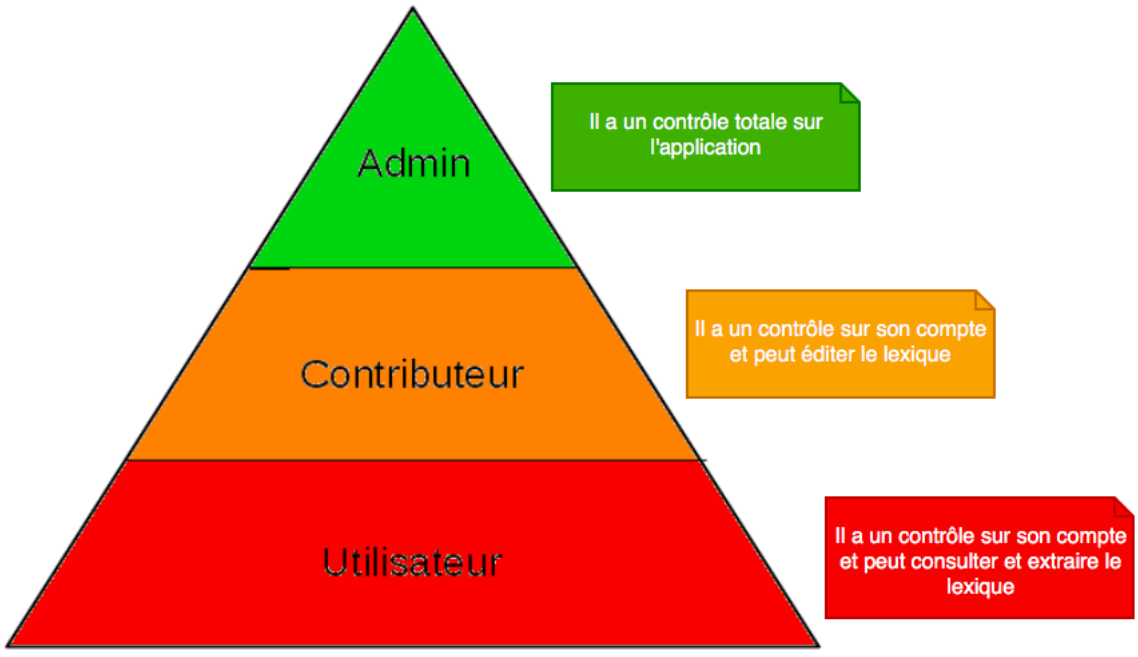

FIGURE 4 – Gestion des rôles des utilisateurs

2.2.5 Exporter le LeFFF - Importance 3/5

Nous offrirons la possibilité d'exporter le **lexique** sélectionné à l'aide d'un **filtre** pour que l'utilisateur puisse **télécharger** un document contenant tous les détails de la recherche, il sera au format **Intensif** du **LeFFF**. En effet, celui-ci par rapport au format extensif permet une plus grande optimisation du fait de leur taille. Deux choix de formats seront possible, soit **xml**, soit **ilex**. Pour ce qui est des **filtres**, nous pourrons en avoir plusieurs types comme une **catégorie** de mot ou un sens (verbe, nom, adjectif,..., singulier, pluriel,..., définition, lexique,...). Il sera possible de tester cette fonction en exportant sans filtre le **LeFFF** au format ilex et par la suite vérifier que le fichier téléchargé est bien identique au **LeFFF**.

3 Les besoins non fonctionnels

3.1 Sécurité

Le système doit offrir un **accès personnalisé** pour nos utilisateurs (chacun a son rôle), de plus doit utiliser des **protocoles de sécurité** ou des certificats HTTPS

3.2 Format de fichier

Nous offrirons la possibilité de gérer le téléchargement du **LeFFF** ou d'une partie du **lexique** sous deux formats au choix **xml** ou **ilex**, contenant le radical des mots c'est à dire une version intensive du **LeFFF**.

3.3 Performances

Nous devrons faire en sorte à ce que la **base de données** soit **optimisée** afin qu'une recherche ou exportation par de multiple personnes en même temps ne prenne par un temps trop grand.

Nous sommes dans l'obligation de manipuler la **gestion de la mémoire** d'une manière très **efficace** dans le but d'assurer le fonctionnement des besoins fonctionnels dans tous les scénarios d'utilisation possible du logiciel.

3.4 Ergonomie

L'ergonomie touche à la **qualité** et au **confort d'utilisation** de la navigation web, dans notre cas le client nous a donné une liberté dans le choix de l'agencement de la page, la police, les couleurs, le design. Nous avons choisi de concevoir un site web simple qui sera d'une utilisation facile et efficace par tous types de visiteurs, en utilisant des instructions claires et basiques.

3.5 Complexité

Lors de la **recherche** d'un mot dans la base de données , l'algorithme de recherche devra **parcourir** la base de données. En effet, le fait de parcourir **naïvement** toute la base de données, rendra la recherche d'un mot ou d'une expression **coûteux** en terme de temps et de mémoire. Il faudra donc **éviter** d'utiliser un algorithme qui aura une **complexité exponentielle**. Du fait d'un temps d'attente non maîtrisé et qui risque de rendre l'utilisation du site web peu plaisante. Afin de répondre à cette problématique, privilégier un parcours rapide et intelligent qui aura une **complexité au plus polynomiale**. La complexité sera une grande partie dans la réalisation du projet.

4 Planning du projet

Pour élaborer ce planning prévisionnel, nous utilisons le planning donné en cours par Monsieur **Philip NARBEL** et les tâches indispensables qu'on a identifiés jusque là. Ce planning et la distribution des tâches peut donc changer au cours de la réalisation du projet.

Tâches	**Semaines**	**développeurs**
Rédaction du cahier des besoins	1 à 4	TOUS
Mise en place de l'architecture	5 à 6	TOUS
Importation du LeFFF dans la base de données	5 à 6	MEHDAOUI Abdelamine
Conception du compilateur	5 à 6	DIALLO Abdoul
Conception de l'application web	7 à 8	DAOUDI Yassir
Développement du compilateur	8 à 12	EL GUEURCH Souhail
Développement de l'application web	8 à 12	ROUSSEL Damien
Interfaçage entre l'application web, le compilateur et la base de données	12	TOUS
Phases de tests	12 à 14	TOUS
Soutenance	15	TOUS

5 Bibliographie

Références

[1] Olivier Bonami. A theory of inflectional periphrasis at the morphology-syntax interface. http://www.. Accessed on 2020-02-03. 2015.

[2] Benoit Sagot Lionel Clément. Liste des étiquettes utilisées dans LeFFF. http://www.labri.fr/perso/clement/lefff/public/data/lefff-tagset-0.1.pdf. Accessed on 2020-01-31. 2003.

[3] Bernard Lionel Clément Benoit Sagot. Morphology based automaticacquisition of large-coverage le http://www.labri.fr/perso/clement/lefff/public/lrec04ClementLangSagot-1.0.pdf. Accessed on 2020-01-31. 2010.

[4] Pierre Boullier Lionel Clément Éric Villemonte de la Clergerie. "The Lefff 2 syntactic lexicon for French : architecture, acquisition, use". In : Université de Bordeaux (2004). http://www.labri.fr/perso/clement/lefff/public/LREC06b.pdf,,consulté le 25/01/2020, p. 1–4.

[5] Guillaume NEDELEC. Maquettes. https://app.moqups.com/GuillaumeNed33/Tr6FjBELx1/view. Accessed on 2020-01-30. 2019.

[6] Gilles Boyé Olivier Bonami. French Pronominal Clitics and the Design of Paradigm Function Morp http://www.lilec.it/mmm/wp/wp-content/uploads/2012/09/291-322-Bonami-Boye.pdf. Accessed on 2020-02-03.

[7] Benoit Sagot. "Informatiser le lexique". In : Memoire (2019).

[8] Benoît Sagot. Lefff. http://alpage.inria.fr/~sagot/lefff. Accessed on 2020-01-31. 2019.