



Ministère de l'Education Nationale
Université de Montpellier II
Place Eugène Bataillon
34095 Montpellier Cedex 5



Projet GMIN103 Base de données avancée Thésaurus

RAPPORT (DÉCEMBRE 2011)

Travail réalisé par :

Baptiste LE BAIL
Thibaut MARMIN
Namrata PATEL
Clément SIPIETER
Steeve TUVÉE

Introduction

Un thésaurus est un type de langage documentaire qui constitue un vocabulaire normalisé. Il regroupe de manière organisée les termes d'un même domaine de connaissance. Cet outil linguistique permet de décrire des concepts et de lever les ambiguïtés induites par les relations de synonymie, d'homonymie et de polysémie présentes dans le langage naturel.

L'outil développé lors de ce projet sera composé de termes décrivant des concepts, reliés entre eux par des relations hiérarchiques, synonymiques et associatives. L'utilisateur aura la possibilité d'explorer la hiérarchie et de gérer (ajouter / modifier / supprimer) les termes et les concepts.

Ce travail, réalisé par une équipe de cinq étudiants¹, est présenté dans ce rapport selon trois phases distinctes : analyse, conception, implémentation.

¹Baptiste Le Bail, Thibaut Marmin, Namrata Patel, Clément Sipieter, Steeve Tuvée

Méthode de travail

Le projet à été réalisé en une durée d'environ un mois. Nous nous sommes réunis en moyenne deux fois par semaine pour mettre en commun nos travaux et réflexions.

Nous avons consacré la majeure partie du projet à la **phase d'analyse**. Celle-ci étant cruciale pour l'aboutissement du projet, tous les membres de l'équipe ont travaillé ensembles afin d'avoir plusieurs visions sur les différentes modélisations imaginées. Nous avons donc convergé vers une modélisation finale, que nous avons pris soin de justifier.

La **phase de conception** a engendré de nombreuses questions concernant l'utilisation du modèle objet-relationnel. Nous avons dû faire le choix d'un SGBD et des choix concernant le modèle objet-relationnel. Ceux-ci étant généraux au projet, nous avons mis ce travail de réflexion en commun. Lors de cette phase, Baptiste et Steeve ont mis en place les vues utilisateurs de l'interface web. Clément et Namrata ont travaillé sur la mise en place de deux schémas relationnels : un schéma uniquement relationnel et un schéma objet-relationnel.

Les choix et réflexions de la phase de conception nous ont amené, lors de la **phase d'implémentation**, à développer l'application à l'aide d'un framework comportant un ORM (Object Relational Mapping). Cette implémentation à été réalisée par Thibaut, en incluant les templates HTML créés par Baptiste et Steeve sur les modèles mis en place à la phase précédente. Le projet ayant été réalisé à l'aide d'un ORM (donc sans rédaction de requêtes), nous avons tenu tout de même à travailler le modèle objet-relationnel. Clément et Namrata ont donc rédigé les scripts SQL de création et de consultation d'une base de donnée Oracle, utilisant des aspects objets.

La rédaction du rapport à été réalisée par Clément, Namrata et Thibaut.

Table des matières

1	Analyse	1
1.1	Fonctionnalités	1
1.2	Modélisation	2
1.2.1	Une première piste...	2
1.2.2	Évolution	3
1.2.3	Décision finale	4
2	Conception	7
2.1	Une histoire de paradigmes	7
2.1.1	Le paradigme objet	7
2.1.2	Le paradigme objet-relationnel	7
2.1.3	Le paradigme relationnel pur	8
2.2	ORM	9
2.2.1	Qu'est qu'un ORM?	9
2.2.2	Pourquoi?	9
2.3	Décisions	9
2.3.1	Postgresql	9
2.3.2	Framework & ORM	9
2.4	Templates et formulaires	10
3	Implémentation	15
3.1	Le framework Symfony2	15
3.1.1	Présentation du framework	15
3.1.2	Structure	15
3.1.3	L'ORM Doctrine2	16

3.2	Structure de <i>Thésaurus Rex</i>	16
3.2.1	Entité <i>Terme</i>	16
3.2.2	Entité <i>Concept</i>	17
3.3	Schéma relationnel généré	19
3.3.1	Schéma simplifié	19
3.3.2	Schéma complet	20
3.3.3	Ajout d'un déclencheur	21
3.4	Templates finaux	23

Chapitre 1

Analyse

La phase d'analyse est un élément indispensable à la bonne réalisation du projet. Dans un premier temps, les fonctionnalités de l'application y sont décrites et caractérisées, notamment à l'aide d'un diagramme UML de cas d'utilisations. Une seconde partie présente les étapes successives qui nous ont permis d'arriver à la modélisation finale, sous la forme de diagrammes UML de classes et de « discussions ».

1.1 Fonctionnalités

L'application devra modéliser un thésaurus stocké dans une base de données, et devra permettre sa consultation au travers d'une interface web. L'utilisateur pourra donc consulter ou administrer les données. La consultation du thésaurus se fera par navigation dans une vue hiérarchique ou par nœud. Un outil de recherche facilitera l'accès aux données. Le module d'administration permettra quand à lui d'ajouter, de modifier et de supprimer les termes et les concepts, ainsi que les relations.

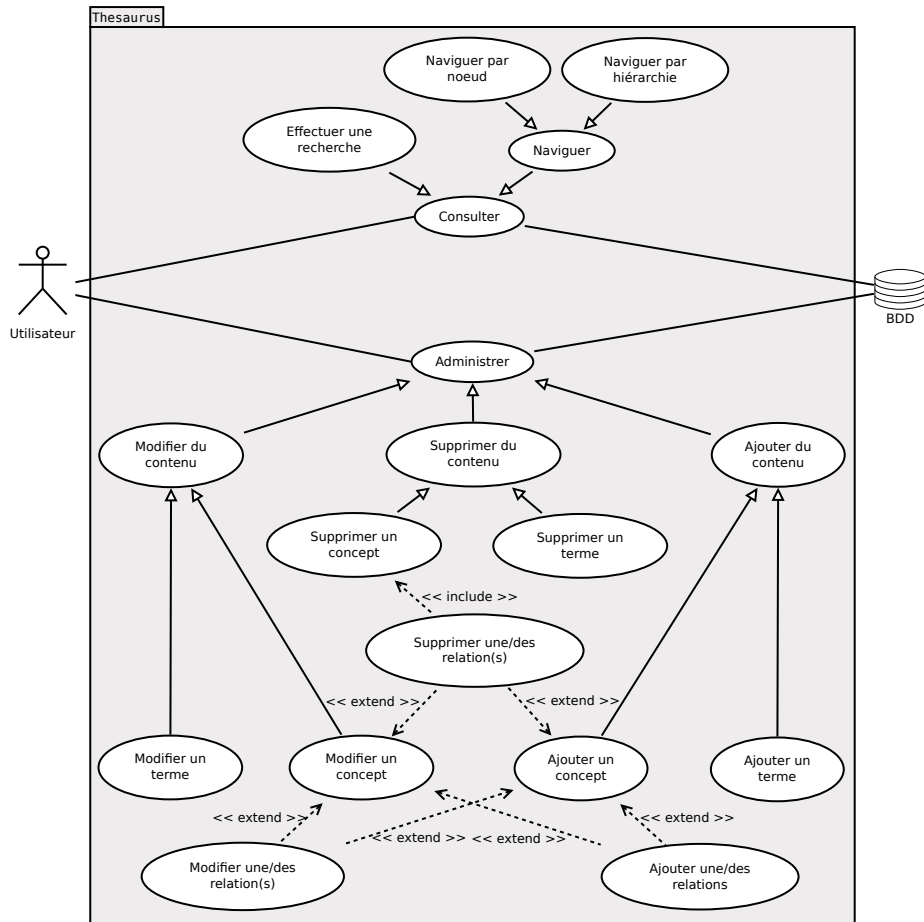


FIG. 1.1 – Diagramme de cas d'utilisations

1.2 Modélisation

Un thésaurus est composé de deux types d'entités : les Termes et les Concepts.

Concepts : ils sont décrits par un descripteur (terme vedette), et sont organisés hiérarchiquement par relations de concepts généraux (parents) et spécifiques (fils). Les concepts peuvent aussi être mis en relation par simple association.

Termes : ils sont définis par un libellé et sont en relation de synonymie entre eux.

1.2.1 Une première piste...

Pour notre première modélisation, nous avons fait le choix de respecter la définition stricte du thésaurus rappelée ci-dessus.

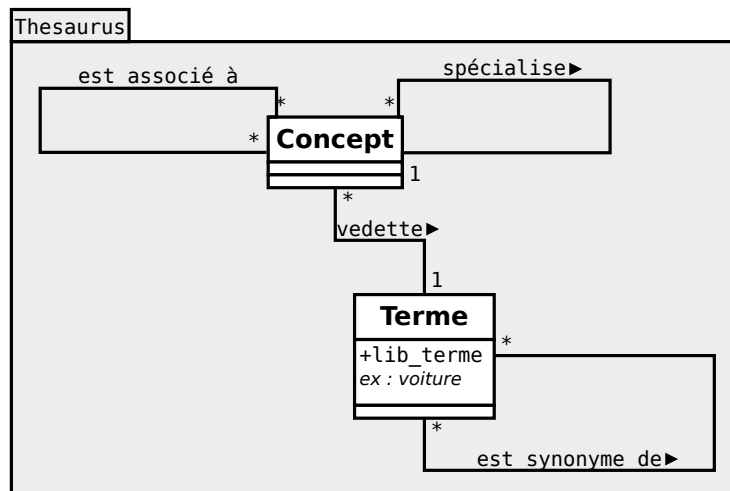


FIG. 1.2 – Première version du diagramme de classes.

Bien que la définition soit rigoureusement respectée, cette modélisation est limitée : il est impossible de lever le problème de polysémie sur les termes. En effet prenons le terme « avocat ». Celui-ci peut désigner un fruit ou un métier. La relation de synonymie est donc ambiguë bien que cette modélisation permet la création de deux concepts ayant comme le même terme vedette.

1.2.2 Évolution

Pour résoudre ce problème d'ambiguïté des synonymes, il est nécessaire de distinguer ce type de relation en ajoutant une dépendance à un concept.

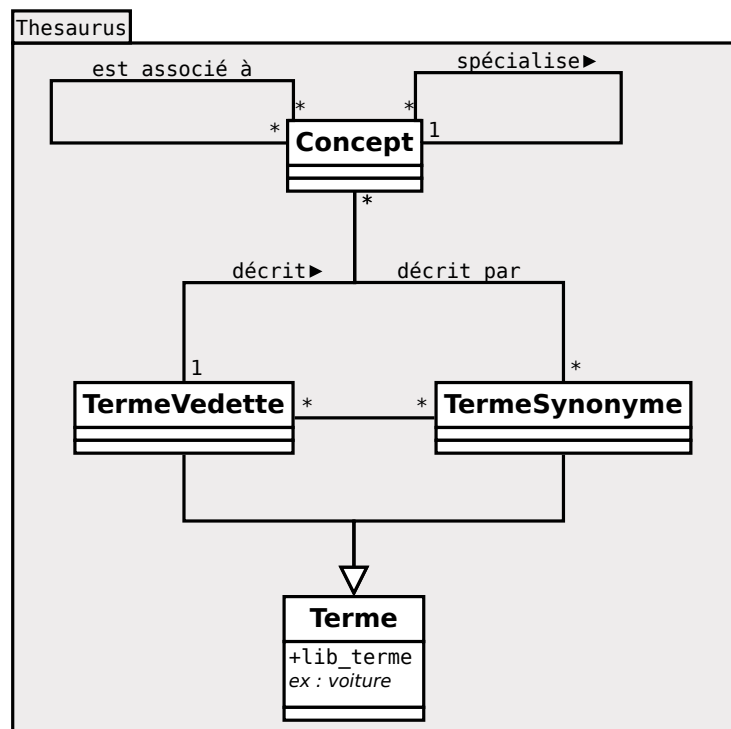


FIG. 1.3 – Évolution du diagramme de classes.

Ce modèle est légèrement éloigné de la définition d'un thésaurus car les termes ne sont plus en relation de synonymie entre eux, mais avec des concepts. Nous avons donc choisi de spécialiser la classe Terme en deux classes filles TermeVedette et TermeSynonyme. Un concept est maintenant décrit à la fois par plusieurs termes : un terme vedette de type TermeVedette et des synonymes de type TermeSynonyme.

1.2.3 Décision finale

Pour cette dernière étape de réflexion, nous avons choisi de simplifier la modélisation.

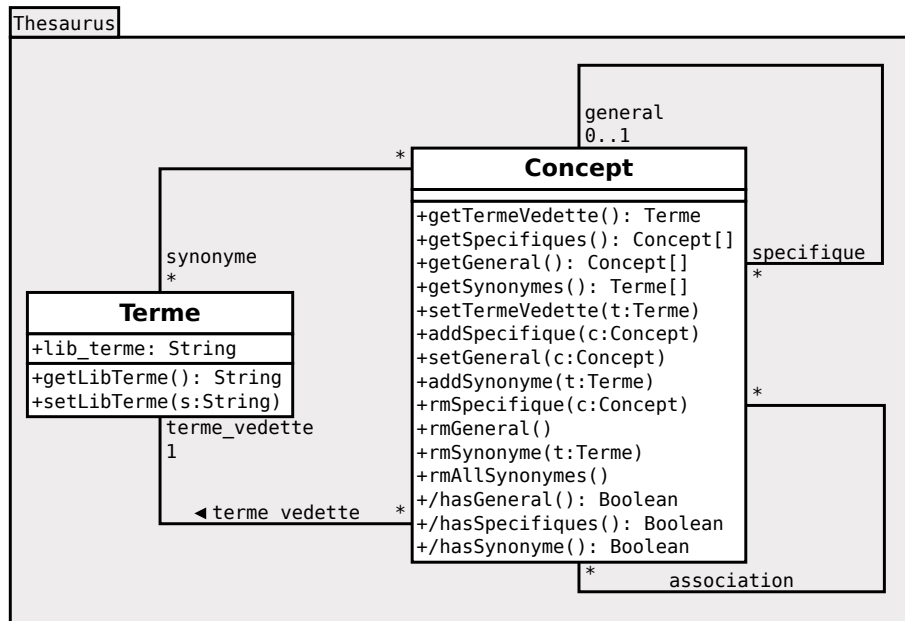


FIG. 1.4 – Version finale du diagramme de classes.

La suppression de l'héritage de terme nous rapproche du modèle initial, tout en conservant la résolution du problème lié à la polysémie. En effet, la distinction entre les termes vedette et les termes synonymes peut être ignorée.

Reprenons le problème de l'« avocat » : le terme sera stocké une fois en base, mais sera terme vedette de deux concepts : celui d'« avocat » le fruit, et celui d'« avocat » le métier. Le premier concept (fruit) pourra ne contenir aucun synonyme, tandis que le deuxième concept pourra avoir comme synonyme « défenseur ».

Chapitre 2

Conception

Objet ? Objet-relationnel ou relationnel pur ? C'est cette question qui est traitée en premier ici. Dans une seconde partie l'utilité des ORM est expliquée, puis les templates de l'application seront présentés.

2.1 Une histoire de paradigmes

2.1.1 Le paradigme objet

Le modèle objet pour le stockage des données présente les mêmes avantages que pour la programmation orienté objet, à savoir une grande capacité d'abstraction, de factorisation et de maintenance. Malheureusement ce paradigme n'est que peu implémenté par les SGBD (O2, db4o, ObjectStore) ce qui nuit grandement à la portabilité d'une application basé sur ce paradigme.

2.1.2 Le paradigme objet-relationnel

De nombreux SGBD tels que ORACLE offrent un paradigme hybride entre l'objet et le relationnel, l'objet-relationnel. Ce paradigme est standardisé par la norme SQL3 cependant celle-ci n'est jamais implémentée dans sa globalité ce qui pose à nouveau un problème de portabilité (un script SQL pour ORACLE ne s'exécutera pas sur PostgreSQL et vice-versa).

Schéma objet-relationnel

Voici le script ORACLE de création des types et tables pour notre application, utilisant des aspect objet-relationnel.

```
1 CREATE TYPE terme_t AS OBJECT (  
2     lib_terme VARCHAR2(32)  
3 );  
4  
5 CREATE TABLE termes OF terme_t;
```

```

6
7 CREATE TYPE ref_terme_t AS OBJECT (
8     ref_terme REF terme_t
9 );
10
11 CREATE TYPE nested_termes_t AS TABLE OF ref_terme_t;
12
13
14
15 CREATE TYPE concept_t;
16
17 CREATE TYPE ref_concept_t AS OBJECT (
18     concept REF concept_t
19 );
20
21
22 CREATE TYPE nested_concepts_t AS TABLE OF ref_concept_t;
23
24 CREATE TYPE concept_t AS OBJECT (
25     ref_terme_vedette REF terme_t,
26     ref_concept_general REF concept_t,
27     synonymes nested_termes_t,
28     concepts_associees nested_concepts_t
29 );
30
31 CREATE TABLE concepts OF concept_t,
32 NESTED TABLE synonymes STORE AS nested_termes,
33 NESTED TABLE concepts_associees STORE AS nested_concepts;
34
35
36 ALTER TABLE termes ADD CONSTRAINT pk_termes
37 PRIMARY KEY (lib_terme)
38
39 ALTER TABLE concepts ADD CONSTRAINT pk_concepts
40 PRIMARY KEY (ref_terme_vedette)

```

2.1.3 Le paradigme relationnel pur

La vision historique des bases de données est le relationnel pur, il offre des performances reconnus. En revanche, son utilisation peut poser problème lors d'une modélisation objet de l'application. Il faut alors créer un nouveau schéma pour la base de données, et développer des procédures de stockage pour gérer la persistance des objets.

Schéma relationnel

Le schéma relationnel pur présenté ci-dessous permet la persistance des objets nécessaire à notre application. D'après le diagramme UML de classe, nous avons ajouté une table des synonymes et une table d'associations.

TERME (lib_terme)

CONCEPT (terme_vedette[#], concept_general[#])

SYNONYME (terme[#], concept[#])

ASSOCIATION (concept1[#], concept2[#])

2.2 ORM

2.2.1 Qu'est qu'un ORM ?

Un ORM ou *mapping objet-relationnel* (en anglais *object-relational mapping*) est un outil informatique qui donne l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en servant d'interface entre celle-ci et le code de l'application.

2.2.2 Pourquoi ?

Nous avons vu que les systèmes de gestion de base de données orientées objet sont actuellement peu nombreuses, que la norme SQL3 n'est que partiellement implémentée, notamment sur le SGBD Postgresql que nous avons décidé d'utiliser.

En revanche, cette méthode d'abstraction présente le désavantage de générer un schéma différents de la vision objet avec laquelle est modélisée l'application. Il peut parfois être nécessaire d'intervenir directement dans la base de données (développement de TRIGGERS, dump SQL, ...), ce qui nécessitera un effort supplémentaire de d'analyse et de compréhension.

2.3 Décisions

2.3.1 Postgresql

Initialement, nous avons choisi de travailler avec Postgresql pour plusieurs raisons :

- découvrir l'objet relationnel sur un nouveau système (autre qu'ORACLE),
- travailler avec un logiciel libre,
- respectueux des standards SQL,
- et précurseur aux débuts de l'objet-relationnel (premier SQL à intégrer des aspects objet).

2.3.2 Framework & ORM

Nous avons choisi de travailler à l'aide du framework Symfony (version 2), intégrant l'ORM Doctrine (version 2). Ces outils seront présentés dans la partie suivante.

Les raisons qui nous ont poussé à faire le choix de développer l'application à l'aide d'un ORM sont les suivantes :

- nous avons pris conscience que l'application à développer ne nécessitait pas la vision objet-relationnel,
- nous tenions à travailler dans un environnement libre en utilisant Postgresql, mais aujourd'hui il ne fournit pas autant de fonctions orientées objet,

- nous souhaitons travailler avec des technologies matures,
- nous souhaitons découvrir la mise en place d’une application à l’aide d’un ORM.

2.4 Templates et formulaires

Accueil

La page d’accueil de l’application *Thésaurus Rex* est composée d’un court texte de bienvenue. La partie gauche est composée du logo, d’un champs de recherche, et de deux liens d’accès aux parties de l’application : gestion des concepts et gestion des termes. Ce panel est visible sur toutes les vues.

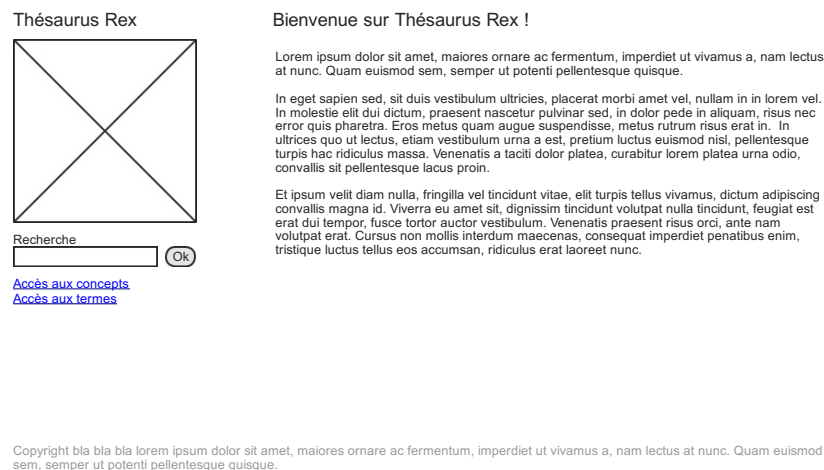
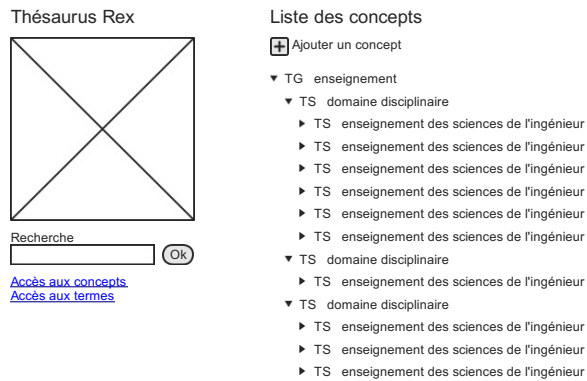


FIG. 2.1 – Template de la page d’accueil de Thésaurus Rex.

Vue hiérarchique des concepts

La vue hiérarchique permet de visualiser l’ensemble des concepts sous la forme d’arbre en affichant le terme vedette de chaque concept. *TG* signifie *Terme Général*, il s’agit des racines de la hiérarchie (ils ne possèdent pas de concept général). *TS* signifie *Terme spécifique*, il s’agit des concepts possédant un terme général.



Copyright bla bla bla lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque.

FIG. 2.2 – Template de la vue hiérarchique des concepts.

Visualisation d'un concept

Lors du clique sur un concept à partir de la vue hiérarchique, le concept est affiché avec son environnement sémantique direct :

- son concept générique,
- ses concepts spécifiques,
- ses concepts associés,
- ses termes synonymes.

Cette vue permet une navigation rapide dans la hiérarchie grâce à la multitude de liens.

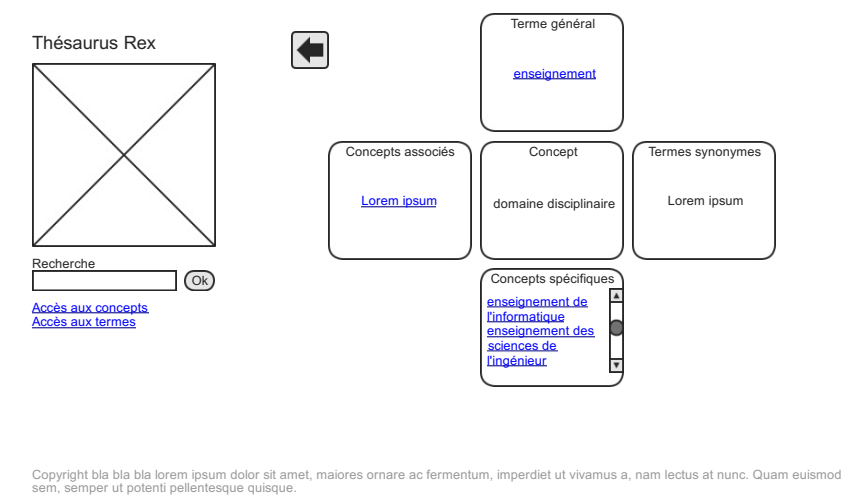


FIG. 2.3 – Template d’affichage d’un concept.

Ajout / Édition d’un concept

L’ajout et l’édition d’un concept se fait à l’aide des formulaires ci-après.

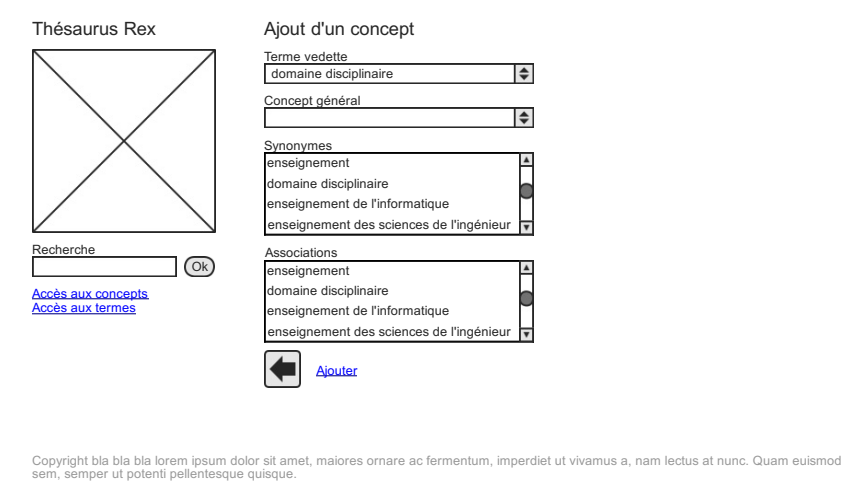
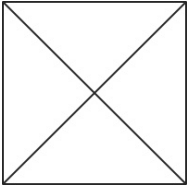


FIG. 2.4 – Template du formulaire d’ajout d’un concept.

Thésaurus Rex



Recherche

[Accès aux concepts](#)
[Accès aux termes](#)

Edition du concept "domaine disciplinaire"

Terme vedette

Concept général

Synonymes

Associations

[Editer](#)
[Supprimer](#)

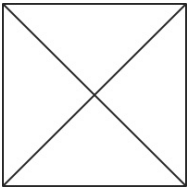
Copyright bla bla bla lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque.

FIG. 2.5 – Template du formulaire d'édition d'un concept.

Liste des termes

L'ensemble des termes peuvent être visualisés sous la forme d'une liste.

Thésaurus Rex



Recherche

[Accès aux concepts](#)
[Accès aux termes](#)

Liste des concepts

Ajouter un terme

enseignement
domaine disciplinaire
enseignement de l'informatique
enseignement des sciences de l'ingénieur
enseignement d'une langue ancienne
...

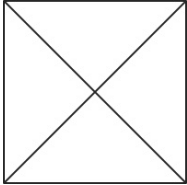
Copyright bla bla bla lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque.

FIG. 2.6 – Template de visualisation des termes.

Édition d'un terme

Lors du clique sur un terme dans la liste, le formulaire d'édition suivant apparaît.

Thésaurus Rex



Recherche

[Accès aux concepts](#)
[Accès aux termes](#)

Edition du terme "enseignement"

Terme vedette

[Editer](#)
[Supprimer](#)

Copyright bla bla lorem ipsum dolor sit amet, maiores ornare ac fermentum, imperdiet ut vivamus a, nam lectus at nunc. Quam euismod sem, semper ut potenti pellentesque quisque.

FIG. 2.7 – Template du formulaire d'ajout d'un concept.

Chapitre 3

Implémentation

Cette dernière partie présente Symfony2, le framework utilisé pour le développement de cette application. Elle détaille également la structure et les entités créées, ainsi que le schéma relationnel généré par l'ORM Doctrine2, intégré dans le framework. Enfin, les vues finales sont présentées.

3.1 Le framework Symfony2

3.1.1 Présentation du framework

Symfony est un framework PHP développé par l'équipe française SensioLabs dirigée par Fabien Potencier. Le but d'un tel outil est d'accélérer le développement et l'accélération d'applications web en se basant sur MVC¹, modèle classiquement utilisé dans les applications modernes.

Il s'agit d'un outil open source qui rassemble divers projets reconnus (Movaji pour le côté MVC, Doctrine pour la partie ORM, Twig pour le moteur de template).

Le projet, créé en 1998, a atteint aujourd'hui une grande maturité. Pour preuve l'utilisation de ce framework par Yahoo et Dailymotion.

Pour ce projet, nous avons utilisé la deuxième version de Symfony (Symfony2).

3.1.2 Structure

Bundle

Il s'agit d'un module ou d'un plugin :

- portable,
- facilement installable dans un projet Symfony2,

¹Modèle Vue Contrôleur

- qui possède une architecture MVC.

Un bundle ne possède pas de définition exacte. Il peut être vu comme un projet à part entière, une partie d'un projet, un plugin, etc. . .

Chaque Bundle possède des vues, des entités, des contrôleurs, etc. . .

Entity

Une entité (Entity) est une classe présente dans un Bundle. Elle peut être en relation avec l'ORM afin d'être persistante, posséder des formulaires, être associée à des actions ou à des vues.

3.1.3 L'ORM Doctrine2

Doctrine2 est un des ORM le plus utilisé à ce jour. Il est sous licence libre GNU LGPL. C'est l'ORM par défaut de Symfony depuis la version 1.3. Parfaitement intégré au framework, les entités créées dans les Bundles Symfony peuvent contenir des tags qui permettent de dialoguer avec Doctrine et de définir la clé primaire, les relations, les types de champs, etc. . .

3.2 Structure de *Thésaurus Rex*

Pour le développement de cette application, nous avons créé un Bundle nommé *ProjetBDDThesaurusBundle*, contenant lui même les deux entités déclarées dans notre diagramme de classe final : *Terme* et *Concept*.

3.2.1 Entité *Terme*

```

1  <?php
2
3  namespace ProjetBDD\Thesaurus\Bundle\Entity;
4
5  use Doctrine\ORM\Mapping as ORM;
6
7  /**
8   * ProjetBDD\Thesaurus\Bundle\Entity\Terme
9   *
10  * @ORM\Table()
11  * @ORM\Entity(repositoryClass="ProjetBDD\Thesaurus\Bundle\Entity\TermeRepository")
12  */
13  class Terme
14  {
15      /**
16       * @var integer $id
17       * @ORM\OneToOne(targetEntity="Concept",mappedBy="terme_vedette")
18       * @ORM\Column(name="id", type="string", length="255")
19       * @ORM\Id
20       */
21      private $id;
22
23
24      /**
25       * Get id
26       *
27       * @return integer

```



```

28      */
29      public function getId()
30      {
31          return $this->id;
32      }
33
34      public function setId($id)
35      {
36          $this->id = $id;
37      }
38
39      public function __toString() { return $this->id; }
40  }

```

3.2.2 Entité *Concept*

```

1 <?php
2
3 namespace ProjetBDD\Thesaurus\Bundle\Entity;
4
5 use Doctrine\ORM\Mapping as ORM;
6 use Symfony\Component\Validator\Constraints as Assert;
7
8
9 /**
10  * ProjetBDD\Thesaurus\Bundle\Entity\Concept
11  *
12  * @ORM\Table()
13  * @ORM\Entity(repositoryClass="ProjetBDD\Thesaurus\Bundle\Entity\ConceptRepository")
14  */
15 class Concept
16 {
17     /**
18      * @var integer $id
19      *
20      * @ORM\Column(name="id", type="integer")
21      * @ORM\Id
22      * @ORM\GeneratedValue(strategy="AUTO")
23      */
24     private $id;
25
26     /**
27      * @ORM\ManyToOne(targetEntity="Terme", inversedBy="id")
28      * @Assert\NotBlank()
29      */
30     private $terme_vedette;
31
32     /**
33      * @ORM\ManyToOne(targetEntity="Concept", inversedBy="concepts_specifiques")
34      */
35     private $concept_general;
36
37     /**
38      * @ORM\OneToMany(targetEntity="Concept", mappedBy="concept_general")
39      */
40     private $concepts_specifiques;
41
42     /**
43      * @ORM\ManyToMany(targetEntity="Terme")
44      */
45     private $synonymes;
46
47     /**
48      * @ORM\ManyToMany(targetEntity="Concept", mappedBy="associations")
49      */
50     private $associes_avec_moi;
51
52     /**
53      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associes_avec_moi")
54      * @ORM\JoinTable(name="concept_concept",
55      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
56      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
57      */
58     private $concepts_concepts;
59
60     /**
61      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="synonymes")
62      * @ORM\JoinTable(name="terme_synonyme",
63      * joinColumns={@ORM\JoinColumn(name="terme1_id", referencedColumnName="id")},
64      * inverseJoinColumns={@ORM\JoinColumn(name="terme2_id", referencedColumnName="id")})
65      */
66     private $synonymes_synonymes;
67
68     /**
69      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
70      * @ORM\JoinTable(name="terme_concept",
71      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
72      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
73      */
74     private $associations;
75
76     /**
77      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
78      * @ORM\JoinTable(name="concept_concept",
79      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
80      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
81      */
82     private $concepts_concepts;
83
84     /**
85      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
86      * @ORM\JoinTable(name="terme_concept",
87      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
88      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
89      */
90     private $associations;
91
92     /**
93      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
94      * @ORM\JoinTable(name="concept_concept",
95      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
96      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
97      */
98     private $concepts_concepts;
99
100     /**
101      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
102      * @ORM\JoinTable(name="terme_concept",
103      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
104      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
105      */
106     private $associations;
107
108     /**
109      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
110      * @ORM\JoinTable(name="concept_concept",
111      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
112      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
113      */
114     private $concepts_concepts;
115
116     /**
117      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
118      * @ORM\JoinTable(name="terme_concept",
119      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
120      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
121      */
122     private $associations;
123
124     /**
125      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
126      * @ORM\JoinTable(name="concept_concept",
127      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
128      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
129      */
130     private $concepts_concepts;
131
132     /**
133      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
134      * @ORM\JoinTable(name="terme_concept",
135      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
136      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
137      */
138     private $associations;
139
140     /**
141      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
142      * @ORM\JoinTable(name="concept_concept",
143      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
144      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
145      */
146     private $concepts_concepts;
147
148     /**
149      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
150      * @ORM\JoinTable(name="terme_concept",
151      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
152      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
153      */
154     private $associations;
155
156     /**
157      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
158      * @ORM\JoinTable(name="concept_concept",
159      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
160      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
161      */
162     private $concepts_concepts;
163
164     /**
165      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
166      * @ORM\JoinTable(name="terme_concept",
167      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
168      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
169      */
170     private $associations;
171
172     /**
173      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
174      * @ORM\JoinTable(name="concept_concept",
175      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
176      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
177      */
178     private $concepts_concepts;
179
180     /**
181      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
182      * @ORM\JoinTable(name="terme_concept",
183      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
184      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
185      */
186     private $associations;
187
188     /**
189      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
190      * @ORM\JoinTable(name="concept_concept",
191      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
192      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
193      */
194     private $concepts_concepts;
195
196     /**
197      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
198      * @ORM\JoinTable(name="terme_concept",
199      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
200      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
201      */
202     private $associations;
203
204     /**
205      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
206      * @ORM\JoinTable(name="concept_concept",
207      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
208      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
209      */
210     private $concepts_concepts;
211
212     /**
213      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
214      * @ORM\JoinTable(name="terme_concept",
215      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
216      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
217      */
218     private $associations;
219
220     /**
221      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
222      * @ORM\JoinTable(name="concept_concept",
223      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
224      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
225      */
226     private $concepts_concepts;
227
228     /**
229      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
230      * @ORM\JoinTable(name="terme_concept",
231      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
232      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
233      */
234     private $associations;
235
236     /**
237      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
238      * @ORM\JoinTable(name="concept_concept",
239      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
240      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
241      */
242     private $concepts_concepts;
243
244     /**
245      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
246      * @ORM\JoinTable(name="terme_concept",
247      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
248      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
249      */
250     private $associations;
251
252     /**
253      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
254      * @ORM\JoinTable(name="concept_concept",
255      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
256      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
257      */
258     private $concepts_concepts;
259
260     /**
261      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
262      * @ORM\JoinTable(name="terme_concept",
263      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
264      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
265      */
266     private $associations;
267
268     /**
269      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
270      * @ORM\JoinTable(name="concept_concept",
271      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
272      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
273      */
274     private $concepts_concepts;
275
276     /**
277      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
278      * @ORM\JoinTable(name="terme_concept",
279      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
280      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
281      */
282     private $associations;
283
284     /**
285      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
286      * @ORM\JoinTable(name="concept_concept",
287      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
288      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
289      */
290     private $concepts_concepts;
291
292     /**
293      * @ORM\ManyToMany(targetEntity="Terme", inversedBy="associations")
294      * @ORM\JoinTable(name="terme_concept",
295      * joinColumns={@ORM\JoinColumn(name="terme_id", referencedColumnName="id")},
296      * inverseJoinColumns={@ORM\JoinColumn(name="concept_id", referencedColumnName="id")})
297      */
298     private $associations;
299
300     /**
301      * @ORM\ManyToMany(targetEntity="Concept", inversedBy="associations")
302      * @ORM\JoinTable(name="concept_concept",
303      * joinColumns={@ORM\JoinColumn(name="concept1_id", referencedColumnName="id")},
304      * inverseJoinColumns={@ORM\JoinColumn(name="concept2_id", referencedColumnName="id")})
30
```

```
58     * )
59     */
60     private $associations;
61
62
63     /**
64      * Get id
65      *
66      * @return integer
67      */
68     public function getId()
69     {
70         return $this->id;
71     }
72
73     /**
74      * Set terme_vedette
75      *
76      * @param object $termeVedette
77      */
78     public function setTermeVedette($termeVedette)
79     {
80         $this->terme_vedette = $termeVedette;
81     }
82
83     /**
84      * Get terme_vedette
85      *
86      * @return object
87      */
88     public function getTermeVedette()
89     {
90         return $this->terme_vedette;
91     }
92
93     /**
94      * Set concept_general
95      *
96      * @param object $conceptGeneral
97      */
98     public function setConceptGeneral($conceptGeneral)
99     {
100         $this->concept_general = $conceptGeneral;
101     }
102
103     /**
104      * Get concept_general
105      *
106      * @return object
107      */
108     public function getConceptGeneral()
109     {
110         return $this->concept_general;
111     }
112
113     /**
114      * Set concepts_specifiques
115      *
116      * @param object $conceptsSpecifiques
117      */
118     public function setConceptsSpecifiques($conceptsSpecifiques)
119     {
120         $this->concepts_specifiques = $conceptsSpecifiques;
121     }
122
123     /**
124      * Get concepts_specifiques
125      *
126      * @return object
127      */
128     public function getConceptsSpecifiques()
129     {
130         return $this->concepts_specifiques;
131     }
```

```

132
133  /**
134   * Set synonymes
135   *
136   * @param object $synonymes
137   */
138  public function setSynonymes($synonymes)
139  {
140      $this->synonymes = $synonymes;
141  }
142
143  /**
144   * Get synonymes
145   *
146   * @return object
147   */
148  public function getSynonymes()
149  {
150      return $this->synonymes;
151  }
152
153  /**
154   * Set associations
155   *
156   * @param object $associations
157   */
158  public function setAssociations($associations)
159  {
160      $this->associations = $associations;
161  }
162
163  /**
164   * Get associations
165   *
166   * @return object
167   */
168  public function getAssociations()
169  {
170      return $this->associations;
171  }
172
173  /**
174   * Set associes_avec_moi
175   *
176   * @param object $associes_avec_moi
177   */
178  public function setAssociesAvecMoi($associes_avec_moi)
179  {
180      $this->associes_avec_moi = $associes_avec_moi;
181  }
182
183  /**
184   * Get associes_avec_moi
185   *
186   * @return object
187   */
188  public function getAssociesAvecMoi()
189  {
190      return $this->associes_avec_moi;
191  }
192
193  public function __toString() { return $this->terme.vedette->__toString(); }
194  }

```

3.3 Schéma relationnel généré

3.3.1 Schéma simplifié

terme (id)

concept (id, terme.vedette_id#, concept_general_id#) (Impossibilité de mettre en clé primaire une clé étrangère :s)

concept_terme (concept_id#, terme_id#)

concept_concept (concept1_id#, concept2_id#)

3.3.2 Schéma complet

thibaut=> \d+ terme

```

Table « public.termes »
  Colonne |          Type          | Modificateurs | Stockage
-----+-----+-----+-----
  id      | character varying(255) | non NULL      | extended

```

Index :

"terme_pkey" PRIMARY KEY, btree (id)

Référencé par :

TABLE "concept" CONSTRAINT "fk_28f759cce6a95e6d" FOREIGN KEY (terme_vedette_id) REFERENCES termes(id)

TABLE "concept_terme" CONSTRAINT "fk_3bf6677a26062764" FOREIGN KEY (terme_id) REFERENCES termes(id) ON DELETE CASCADE

Contient des OID: non

thibaut=> \d+ concept

```

Table « public.concept »
  Colonne |          Type          | Modificateurs | Stockage
-----+-----+-----+-----
  id      | integer                | non NULL      | plain
  terme_vedette_id | character varying(255) | Par défaut, NULL | extended
  concept_general_id | integer                |                | plain

```

Index :

"concept_pkey" PRIMARY KEY, btree (id)

"idx_28f759ccd859483c" btree (concept_general_id)

"idx_28f759cce6a95e6d" btree (terme_vedette_id)

Contraintes de clés étrangères :

"fk_28f759ccd859483c" FOREIGN KEY (concept_general_id) REFERENCES concept(id)

"fk_28f759cce6a95e6d" FOREIGN KEY (terme_vedette_id) REFERENCES termes(id)

Référencé par :

TABLE "concept" CONSTRAINT "fk_28f759ccd859483c" FOREIGN KEY (concept_general_id) REFERENCES concept(id)

TABLE "concept_terme" CONSTRAINT "fk_3bf6677af909284e" FOREIGN KEY (concept_id) REFERENCES concept(id) ON DELETE CASCADE

TABLE "concept_concept" CONSTRAINT "fk_79a50ccb47f1482c" FOREIGN KEY (concept1_id) REFERENCES concept(id)

TABLE "concept_concept" CONSTRAINT "fk_79a50ccb5544e7c2" FOREIGN KEY (concept2_id) REFERENCES concept(id)

Triggers :

tgr_verif_root AFTER DELETE OR UPDATE ON concept FOR EACH ROW E

```
XECUTE PROCEDURE fc_verif_root()
```

```
Contient des OID: non
```

```
thibaut=> \d+ concept_terme
```

```

          Table « public.concept_terme »
  Colonne |          Type          | Modificateurs | Stockage
-----+-----+-----+-----
concept_id | integer                | non NULL      | plain
terme_id   | character varying(255) | non NULL      | extended

```

```
Index :
```

```
    "concept_terme_pkey" PRIMARY KEY, btree (concept_id, terme_id)
```

```
    "idx_3bf6677a26062764" btree (terme_id)
```

```
    "idx_3bf6677af909284e" btree (concept_id)
```

```
Contraintes de clés étrangères :
```

```
    "fk_3bf6677a26062764" FOREIGN KEY (terme_id) REFERENCES terme(id) ON DELETE CASCADE
```

```
    "fk_3bf6677af909284e" FOREIGN KEY (concept_id) REFERENCES concept(id) ON DELETE CASCADE
```

```
Contient des OID: non
```

```
thibaut=> \d+ concept_concept
```

```

          Table « public.concept_concept »
  Colonne | Type | Modificateurs | Stockage
-----+-----+-----+-----
concept1_id | integer | non NULL      | plain
concept2_id | integer | non NULL      | plain

```

```
Index :
```

```
    "concept_concept_pkey" PRIMARY KEY, btree (concept1_id, concept2_id)
```

```
    "idx_79a50ccb47f1482c" btree (concept1_id)
```

```
    "idx_79a50ccb5544e7c2" btree (concept2_id)
```

```
Contraintes de clés étrangères :
```

```
    "fk_79a50ccb47f1482c" FOREIGN KEY (concept1_id) REFERENCES concept(id)
```

```
    "fk_79a50ccb5544e7c2" FOREIGN KEY (concept2_id) REFERENCES concept(id)
```

```
Triggers :
```

```
    tgr_gestion_relations AFTER INSERT OR DELETE OR UPDATE ON concept_concept FOR EACH ROW EXECUTE FUNCTION fc_gestion_relations()
```

```
Contient des OID: non
```

3.3.3 Ajout d'un déclencheur

Dans le but de gérer la symétrie des associations entre concepts (table `concept_concept`), nous avons développé un déclencheur qui :

- lors d'un `INSERT` ajoute l'association symétrique,
- lors d'un `UPDATE` modifie l'association symétrique correspondante,
- lors d'un `DELETE` supprime l'association symétrique correspondante.

Ce déclencheur assure l'intégrité des relations à tout moment.

Script de mise en place du déclencheur

```

1  /* Active PL/PGSQL */
2  CREATE LANGUAGE plpgsql;
3
4  /* Creation de la fonction associee au trigger de gestion de la reflexivite des associations */
5  CREATE OR REPLACE FUNCTION fc_gestion_relations() RETURNS trigger as $tgr_gestion_relations$
6  DECLARE
7  nb INTEGER;
8  BEGIN
9  IF (TG_OP = 'DELETE') THEN
10     SELECT count(*) INTO nb FROM concept_concept WHERE concept1_id = OLD.concept2_id AND
        concept2_id = OLD.concept1_id;
11     IF (nb > 0) THEN
12         DELETE FROM concept_concept WHERE (concept1_id=OLD.concept2_id AND
            concept2_id=OLD.concept1_id);
13     END IF;
14     RETURN OLD;
15  ELSEIF (TG_OP = 'UPDATE') THEN
16     SELECT count(*) INTO nb FROM concept_concept WHERE concept1_id = OLD.concept2_id AND
        concept2_id = OLD.concept1_id;
17     IF (nb > 0) THEN
18         UPDATE concept_concept SET concept1_id = NEW.concept2_id, concept2_id = NEW.concept1_id
            WHERE concept1_id=OLD.concept2_id AND concept2_id = OLD.concept1_id;
19     END IF;
20     RETURN NEW;
21  ELSEIF (TG_OP = 'INSERT') THEN
22     SELECT count(*) INTO nb FROM concept_concept WHERE concept1_id = NEW.concept2_id AND
        concept2_id = NEW.concept1_id;
23     IF (nb = 0) THEN
24         INSERT INTO concept_concept VALUES(NEW.concept2_id,NEW.concept1_id);
25     END IF;
26     RETURN NEW;
27  END IF;
28  RETURN NULL;
29  END;
30  $tgr_gestion_relations$ LANGUAGE plpgsql;
31
32  /* Creation de la fonction associee au trigger de verification des racines */
33  CREATE OR REPLACE FUNCTION fc_verif_root() RETURNS trigger as $tgr_verif_root$
34  DECLARE
35  nb INTEGER;
36  BEGIN
37  IF (OLD.concept_general_id IS NULL) THEN
38     SELECT count(*) INTO nb FROM concept WHERE concept_general_id IS NULL;
39     IF (TG_OP = 'DELETE' AND nb < 1) THEN
40         RAISE EXCEPTION 'Au_moins_une_racine_doit_etre_presente.';
41     ELSEIF (TG_OP = 'UPDATE' AND nb < 1 AND NEW.concept_general_id IS NOT NULL ) THEN
42         RAISE EXCEPTION 'Au_moins_une_racine_doit_etre_presente.';
43     END IF;
44  END IF;
45  RETURN NULL;
46  END;
47  $tgr_verif_root$ LANGUAGE plpgsql;
48
49
50  /* Creation du declencheur tgr_gestion_relations */
51  CREATE TRIGGER tgr_gestion_relations AFTER DELETE OR UPDATE OR INSERT
52  ON concept_concept FOR EACH ROW
53  EXECUTE PROCEDURE fc_gestion_relations();
54
55  /* Creation du declencheur tgr_verif_root */
56  CREATE TRIGGER tgr_verif_root AFTER DELETE OR UPDATE
57  ON concept FOR EACH ROW
58  EXECUTE PROCEDURE fc_verif_root();

```

3.4 Templates finaux

Les templates sont réalisés en HTML5 / CSS3. Nous avons tenu à rester fidèle aux vues réalisées lors de la conception.

Accueil

Thésaurus Rex



Recherche

Accéder aux concepts
Accéder aux termes

Projet BDD : réalisation d'un Thésaurus

L'application est développée dans le cadre de l'UE GMIN103 du Master Informatique de l'Université Montpellier 2.

Vous avez la possibilité de :


- vous rendre sur la page permettant de Visualiser l'ensemble les Concepts sous forme hiérarchique,
- Accéder à la vue d'un concept, détaillant ainsi ses voisins (concept général et concepts spécifiques), synonymes et relations,
- Ajouter, modifier, et supprimer des Concepts,
- Ajouter, modifier, et supprimer des Termes.

Application développée dans le cadre de l'UE Base de données avancées (GMIN103) Master Informatique. https://github.com/marminthibaut/bdd_projet

FIG. 3.1 – Aperçu de la page d'accueil.

Liste des concepts

Thésaurus Rex



Recherche

Accéder aux concepts
Accéder aux termes

Concept list

+ Ajouter un concept

- T enseignement
 - T domaine disciplinaire
 - T enseignement de l'informatique
 - T enseignement des sciences de l'ingénieur
 - T enseignement d'une langue ancienne
 - T enseignement d'une langue régionale
 - T enseignement d'une langue vivante
 - T enseignement de la danse
 - T enseignement de la littérature
 - T domaine disciplinaire agricole
 - T éducation civique
 - T enseignement de l'AII (automatisme informatique industrielle)
 - T enseignement de l'ESF (économie sociale et familiale)
 - T enseignement de l'histoire de l'art
 - T enseignement de l'histoire géographie
 - T enseignement de la construction mécanique
 - T enseignement de la musique
 - T enseignement de la philosophie
 - T enseignement de la productique
 - T enseignement de la technologie
 - T enseignement de la TSA (technologie des systèmes automatisés)
 - T enseignement des arts plastiques
 - T enseignement des langues et cultures d'origine
 - T enseignement des mathématiques
 - T enseignement des sciences économiques et sociales

FIG. 3.2 – Hiérarchie des concepts.

Affichage d'un concept

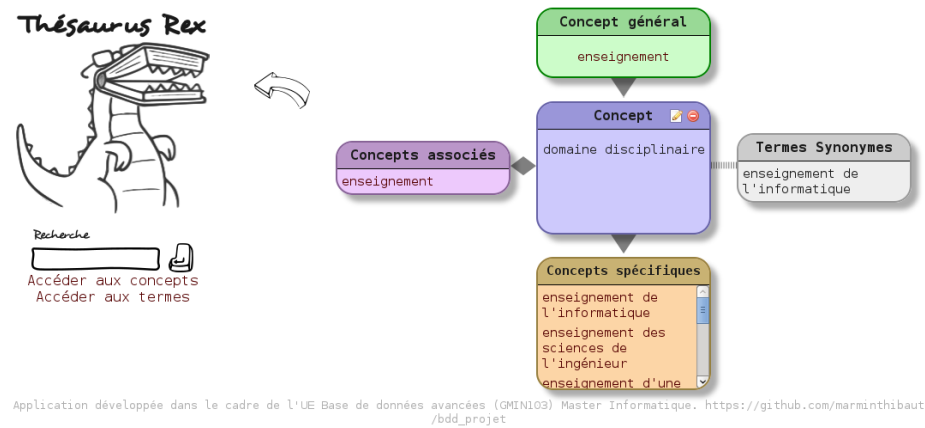


FIG. 3.3 – Aperçu de la page d'un concept.

Modification d'un concept

The screenshot shows the 'Thésaurus Rex' application interface for editing a concept. The title is 'Edition du concept "domaine disciplinaire"'. The form contains several fields: 'Terme vedette' (domaine disciplinaire), 'Concept général' (enseignement), 'Synonymes' (enseignement, domaine disciplinaire, enseignement de l'informatique, enseignement des sciences de l'ingénieur), and 'Associations' (enseignement, domaine disciplinaire, éducation à l'orientation, domaine transversal). Below the form are two buttons: 'Editer' and 'Supprimer'. On the left, there is a dinosaur logo and a search bar with the text 'Recherche'. Below the search bar are two links: 'Accéder aux concepts' and 'Accéder aux termes'. Below the main content area, there is a footer with the text: 'Application développée dans le cadre de l'UE Base de données avancées (GMIN103) Master Informatique. https://github.com/marminthibaut/bdd_projet'.

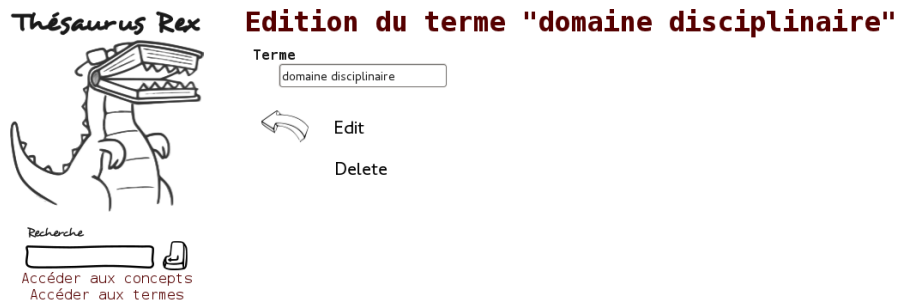
FIG. 3.4 – Formulaire de modification d'un concept.

Liste des termes



FIG. 3.5 – Aperçu de la page de liste des termes.

Modification d'un terme



Application développée dans le cadre de l'UE Base de données avancées (QM103) Master Informatique. https://github.com/marminthibaut/bdd_projet

FIG. 3.6 – Formulaire d'édition d'un terme.