Encyclopédie en ligne de démonstrations formelles

Walid Moustaoui

Rapport de stage

Encyclopédie en ligne de démonstrations formelles

DEDUCTEAM - INRIA SACLAY - ENS PARIS-SACLAY

Walid MOUSTAOUI
2017-2018

Tuteur : M. Stefano GUERRINI
Chef d'équipe : M. Gilles DOWEK
Encadrant : M. François THIRÉ

Remerciement

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Je tiens tout d'abord à remercier mon professeur, Stefano Guerrini qui m'a beaucoup aidé dans ma recherche de stage et dans la rédaction du rapport de stage.

Je remercie tout particulièrement mon directeur de stage Gilles Dowek, pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien. Je tiens aussi à remercier François Thiré, de m'avoir encadrée et conseillée tout au long de ce stage.

Je remercie également toutes l'équipe Deducteam et plus spécialement les personnes avec lesquelles j'ai travaillé directement lors de ce stage : Mohamed Yacine El Haddad, Guillaume Genestier, Gaspard Férey, Rodolph Lepigre. Ils furent d'une aide précieuse dans les moments les plus délicats, je leurs en suis donc très reconnaissant.

Enfin, je voudrais aussi remercier ma famille et mes amis, pour leur soutien inconditionnel.

Résumé

Logipedia est une encyclopédie en ligne qui contient des démonstrations formelles exprimées en Dedukti ainsi que le traductions de ces démonstrations dans autres langages (tels Coq, Matita, PVS, Lean, OpenTheory). Une démonstration formelle est une séquence finie de propositions que chacune est déduite des prédédents par règle logique. Une librairie de démonstrations contient

— des Théorèmes et leurs démonstrations,
— des Axiomes,

mais aussi

— des Paramètres,
— et des Définitions.

Donc le but de Logipedia est de stocker toutes ces données, ainsi que leurs traductions vers d'autres langages. Et de les afficher à travers un site web. Pour répondre à ces attentes, j'ai tout d'abord importé les données dans une base de données MongoDB. Puis j'ai créé un site web dynamique effectuant plusieurs requêtes vers la base de données pour pouvoir afficher toutes les démonstrations.

Logipedia is an online encyclopedia of formal demonstrations in
Dedukti as well as demonstrations translated from Dedukti to other languages (such as Coq, Matita, PVS, Lean, OpenTheory). A formal demonstration is a finite sequence of propositions such as we have :

— Axioms
— Parameters
— Theorems
— And sometimes definitions in the context of concrete demonstrations

So the purpose of Logipedia is to store all this data, as well as their translation to other languages. And to post them through a website. To meet these expectations, I imported first of all the data in a database MongoDB. Then I created a dynamic website making several queries to the database to be able to display all the demonstrations.

Table des matières

Introduction **4**

I Organisation du travail et présentation des données **5**

1 Organisation du travail et choix des technologies **6**
1.1 Organisation du travail 6
1.2 Choix des technologies 7
1.3 Le cahier des charges 8

2 Présentation des données **10**
2.1 Démonstration formelle 10
2.2 Dedukti et les autres langages 11

II Une base de données de démonstrations formelles **13**

3 Une base de données relationnelle **14**

4 Une base de données non relationnelle **15**
4.1 Présentation 15
4.2 Insertion 16

III Une encyclopédie en ligne de démonstrations formelles **19**

5 Gestion du front-end **20**
5.1 Les différentes pages 20
5.2 Evolution du front 20

6 Gestion du back-end PHP **23**
6.1 Clôture transitive et construction des fichiers téléchargeables . 23
6.2 Recherche et affichage 24

Conclusion **25**

Bibliographie **26**

Introduction

Dans le cadre de ma formation d'ingénieur à Sup Galilée, j'ai réalisé un stage technicien de trois mois où j'ai pu intégrer l'équipe Deducteam. Deducteam est composé de 15 membres actifs dont le responsable est Gilles Dowek, mon tuteur de stage.

L'objectif de cette équipe est d'explorer les applications de la théorie de la démonstration à la conception de cadres logiques, à l'interopérabilité entre systèmes de preuves et à la construction de bibliothèques mathématiques universelles. Tout cela est possible grâce à la création d'un nouveau langage et vérificateur de preuve : Dedukti. L'objectif étant de faire de Dedukti, le langage de référence pour l'échange de démonstrations entre systèmes et ainsi d'amener les autres systèmes de preuves à adopter ce langage, comme langage d'échange.
Pour ce faire, il fut impératif de créer une encyclopédie en ligne de démonstrations formelles exprimées en Dedukti, c'est donc le sujet de mon stage et cette encyclopédie en ligne se nomme Logipedia.

Cela nous amène à nous poser trois questions. Tout d'abord, que souhaitons-nous afficher concrètement. Nous allons donc voir dans un premier temps, la représentation des données ainsi que mon organisation du travail. De plus, nous pouvons nous demander comment stocker toutes ces données. C'est donc pour cela, que nous allons voir dans un second temps la gestion des données au niveau du back-end. Enfin, nous pouvons aussi nous demander, pourquoi choisir un site web pour les représenter. Et c'est donc pour cela, que nous terminerons par voir l'évolution du front-end ainsi que la gestion des données côté serveur.

Première partie

Organisation du travail et présentation des données

Chapitre 1

Organisation du travail et choix des technologies

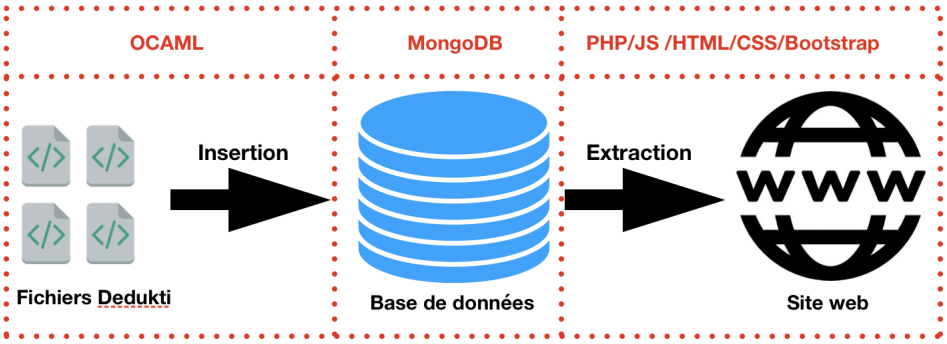
1.1 Organisation du travail

Je fus très régulièrement en contact avec Gilles Dowek et François Thiré, mes deux encadrants de stage.
Pour le cahier des charges, ce fut dans un premier temps assez simple, car ils savaient à peu près ce qu'ils souhaitaient avoir. C'est-à-dire, de pouvoir entreposer un nombre important de démonstrations formelles. Et de pouvoir les afficher sur un site web.
Pour ce faire, j'ai eu la chance de travailler dans le même bureau que François Thiré ainsi que trois autres doctorants qui m'ont apporté à plusieurs reprise leur aide. Quant à Gilles Dowek, il fut très disponible et nous avons eu recours assez souvent à des réunions pour faire le point et remettre à jour le cahier des charges.
J'ai aussi eu la chance de pouvoir présenter à plusieurs reprise le projet à l'équipe Deducteam pour pouvoir récolter les différents avis et conseils. J'ai aussi pu en apprendre d'avantage sur les données que je manipule grâce aux différentes invitations aux conférences, présentations, soutenance de thèse,etc..
Pour ce qui est du moyen de contact, nous nous réunissions assez souvent avec Gilles Dowek et François Thiré. Ainsi, que par mail, où les échanges se faisaient assez rapidement. De plus, étant donné, que je travaille en parallèle avec François Thiré sur le même projet, nous avons un dépôt Git qui nous permet aisément d'accéder aux modifications et mise à jour faite sur le projet.
Dépôt : https ://github.com/Deducteam/Logipedia

J'ai donc eu la chance d'avoir un entourage assez disponible, nous allons donc voir maintenant les attentes quant aux choix technologiques.

1.2 Choix des technologies

Pour ce qui est des choix technologiques, il était attendu une seule chose : que le projet doit être récupérable par la suite et donc utiliser un minimum de langage différent. C'est donc pour cela que le choix du langage pour l'importation des données se fait en OCAML. Car toute l'équipe Deducteam utilise et maîtrise ce langage. Cependant, pour le reste, je fus entièrement libre.
Étant donnée, que nous souhaitons avoir beaucoup de données et de pouvoir regrouper ces données selon un certain critère, nous avons donc choisi de stocker ces données dans une base de données. Où nous pourrons accéder facilement aux données via des requêtes et rapidement grâce à des tables d'indexage.
Étant donnée que j'ai eu la chance de pouvoir apprendre à créer, manipuler et optimiser une base de données SQL, je me suis donc dans un premier temps penché vers celle-ci. Mais nous verrons par la suite que nous avons finalement choisi une base de données NoSQL, plus précisément une base de données MongoDB.
Donc actuellement nous avons donc l'importation des données en OCAML, une base de données, il manque plus que le site web. Il fallait donc une base qui structure le site et son style, cela est fait en HTML/CSS. Étant donné que ce sera un site dynamique qui fera plusieurs requêtes sur la base de données, il fallait donc utiliser du PHP. Il fallait que le site soit adaptable à tout format, plus précisément à toute taille d'écran, aux smartphones et tablettes. Il fallait donc qu'il soit « responsive », et pour cela, j'ai utilisé le framework Bootstrap. Ce choix a aussi été motivé par le fait que certains membres de l'équipe ont une compétence sur ce framework.
Voici un schéma représentatif des différentes étape et technologie du projet :



Nous avons vu les différents langages adoptés, nous allons voir sur quel support j'ai pu programmer.
J'ai mis en place une machine virtuelle où j'ai installé Ubuntu étant donné que la majorité de l'équipe code sur Linux et que leur librairie à été développé sous ce système. Il m'est aussi arrivé, à de nombreuses reprises, de coder sous MacOS.
Nous avons vu quels étaient les choix des technologies adoptés, nous allons voir dès à présent, en détails, le cahier des charges.

1.3 Le cahier des charges

Tout d'abord, comme nous l'avons vue précédemment, les fonctionnalités attendues se sont enrichies au fur et à mesure du projet. Voici la liste finale des fonctionnalités attendues :

- Création d'un dépôt contenant toutes les démonstrations
- Accès à toutes les démonstrations de manière intuitive
- Possibilité d'avoir les démonstrations traduites en différents langages
- Possibilité de basculer entre les démonstrations le plus rapidement possible et simplement possible.
- Possibilité d'afficher toutes les dépendances d'une démonstration et de pouvoir les consulter
- Possibilité de télécharger un fichier contenant la démonstration proof-checkable(« vérifiable »).

Nous allons voir plus en détails toutes ces fonctionnalités par la suite.
Voici un diagramme de cas d'utilisation représentatif :

Nous avons vu qu'elles étaient les choix technologiques ainsi que les détails de l'organisation du travail. Nous allons donc passer à la présentation des données utilisées.

Chapitre 2

Présentation des données

2.1 Démonstration formelle

Le but de Logipedia est d'afficher à travers un site web une encyclopédie de démonstrations formelles. Nous allons donc dans un premier définir ce qu'est une démonstration formelle.
Une démonstration formelle est une séquence finie de propositions que chacune est déduite des prédédents par règle logique. Une librairie de démonstrations contient

— des Théorèmes et leurs démonstrations,
— des Axiomes,

mais aussi

— des Paramètres,
— et des Définitions.

Nous aurons donc des théorèmes composé d'axiomes et pouvant être composé de paramètres, définitions ou/et encore de théorèmes.
Nous allons voir dans un premier temps un exemple assez simpliste représentatif de ce qu'est une démonstration formelle. Puis nous allons apercevoir la complexité réelle des démonstrations stocké dans Logipedia.
Exemple :

Objectif : Démontrer que $1 + 1 = 2$

$$\underbrace{1 = S(0)}_{\text{Définition 1}}$$

$$\underbrace{2 = S(1)}_{\text{Définition 2}}$$

$$\underbrace{\text{"}\mathit{Pour\ tout}\ x\ (x + 1 = S(x))\text{"}}_{\text{Axiome}}$$

$$\underbrace{"nat, 0, S, +, ="}_{\text{Paramètres}}$$

Preuve : En utilisant l'axiome nous avons :
$$1 + 1 = \mathrm{S}\ (\ 1\ )$$
Et en utilisant la seconde définition nous avons :
$$1 + 1 = \mathrm{S}\ (\ 1\ ) = 2$$
On a donc $1 + 1 = 2$.

Nous avons donc ici une preuve qui utilise deux axiomes et un paramètre. Cependant, en réalité, les démonstrations ne sont pas aussi simpliste que celle-ci. Voici un exemple de démonstration que nous pouvons trouver dans Logipedia (nous n'effectuerons pas la preuve), le petit théorème de fermat : "Si p est un nombre premier et si a est un entier non divisible par p, alors a^{p-1} est un multiple de p". C'est-à-dire :

$$a^{p-1} \equiv 1 \bmod \mathrm{p}.$$

La preuve pour ce théorème en comptant la preuve de chaque théorème utilisé pour le prouver, fait un peux moins de mille lignes avec des lignes pouvant atteindre 36500 caractères.
Nous manipulons donc des données de grande taille.

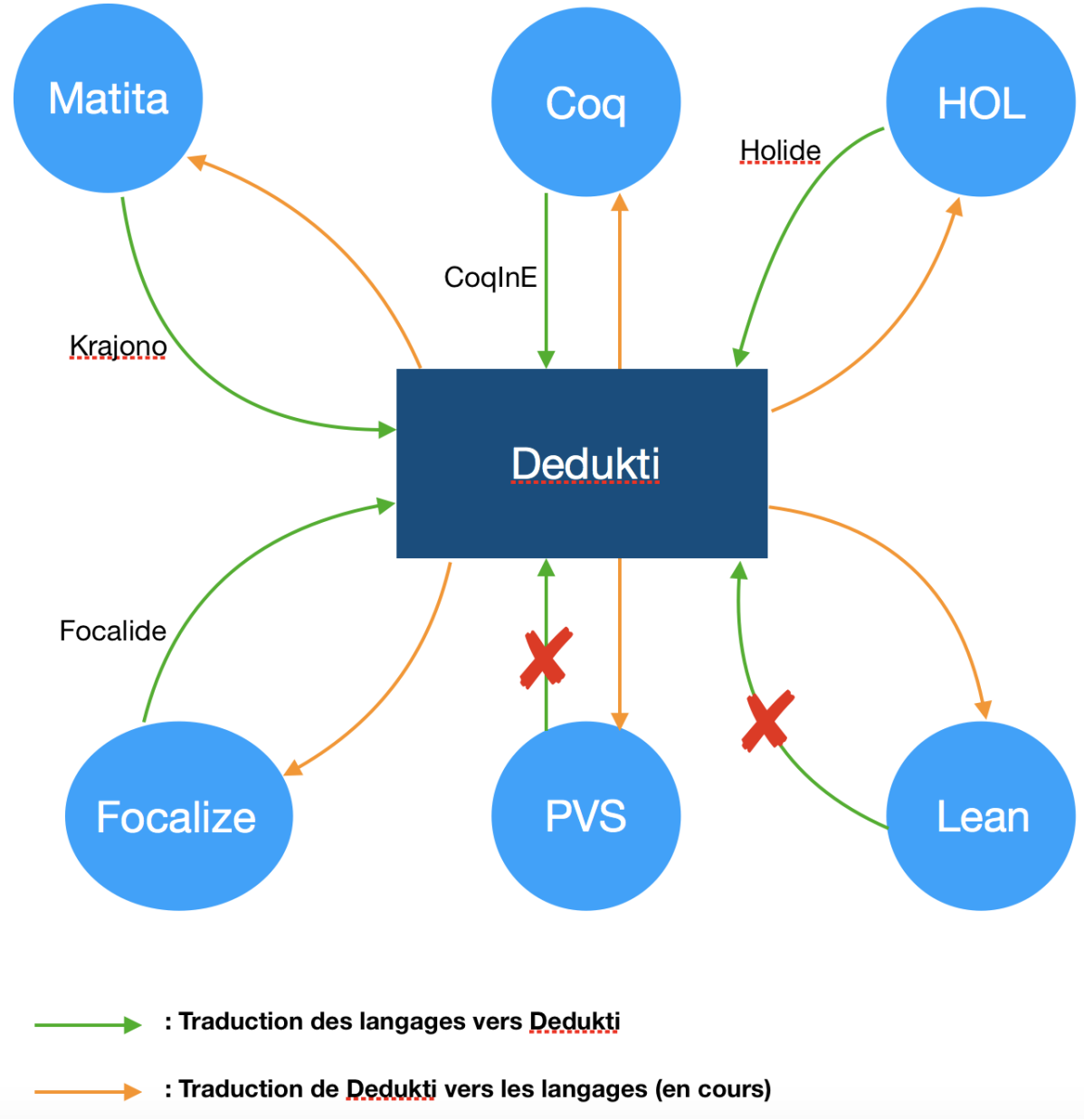
2.2 Dedukti et les autres langages

Tout d'abord, comme nous l'avons vu dans l'introduction, Logipedia est en premier lieu une bibliothèque de démonstrations écrites en Dedukti. Dedukti est un vérificateur de preuve, c'est-à-dire un type-checkeur. Il vérifie si la démonstration est bien typée et retourne donc si la démonstration est valide ou non. Puis nous avons d'autres vérificateurs de preuve qui effectue ce même travail d'une manière différente. Parmi ces langages, nous avons :

— Coq
— Matita
— Lean
— PVS
— OpenTheory

À l'heure actuelle nous pouvons traduire vers ces langages des démonstrations exprimées en Dedukti. Et c'est donc le but de Logipedia, c'est-à-dire, d'avoir comme base de référence la démonstration en Dedukti, puis la traduction de Dedukti vers ces langages. Voici un schéma représentatif des différents

langages possible de traduire à partir de Dedukti et inversement. Ceux qui sont présents dans Logipedia sont seulement ceux cité précédemment.



Krajono, Focalide, CoqInE, Holide étant les traducteurs de ces langages vers Dedukti. Focalize n'est actuellement pas encore dans Logipedia.
Nous avons vu en détail l'organisation du travail et présenté les données, nous allons maintenant inspecter la base de données.

Deuxième partie

Une base de données de démonstrations formelles

Chapitre 3

Une base de données relationnelle

Comme vue précédemment, nous nous sommes lancés dans un premier temps, sur l'utilisation d'une base de données SQL (ici mySQL). Car je maîtrise assez le SQL pour pouvoir répondre aux attentes du client.
Ayant déjà utilisé mySQL, je n'avais pas besoin d'effectuer d'installation mis à part la librairie pour pouvoir effectué des insertions en OCAML. Après multiple recherche, la librairie ocaml-mysql répondait parfaitement à nos critères. Donc j'ai pu effectuer mes requêtes en OCAML et ainsi créer mes tables.
Cependant, lors de l'insertion des tuples, nous avions une utilisation énorme de ressource pour l'effectuer (ceci était dû à la librairie). Et donc, pour les petites démonstrations cela prenais énormément de temps et pour les grandes démonstrations, cela nous générait une erreur. Il nous fallait donc choisir une autre librairie.
Cependant, à ce moment du projet, une question ressortait. Comment pourrait-on définir un standard d'entrée à la base de données. À la suite d'une réunion, nous décidemment donc d'utiliser une base de données MongoDB, car elle peut utiliser des fichiers JSON pour l'insertion ou l'exportation de la base de données. Les fichiers JSON ont une structure précise et permettre de définir un standard (comme le XML). Nous verrons cela en détails dans le prochain chapitre. C'est donc pour ces raisons que nous avons choisie une base de données MongoDB.
Nous allons maintenant passer à la définition de notre base de données MongoDB.

Chapitre 4

Une base de données non relationnelle

4.1 Présentation

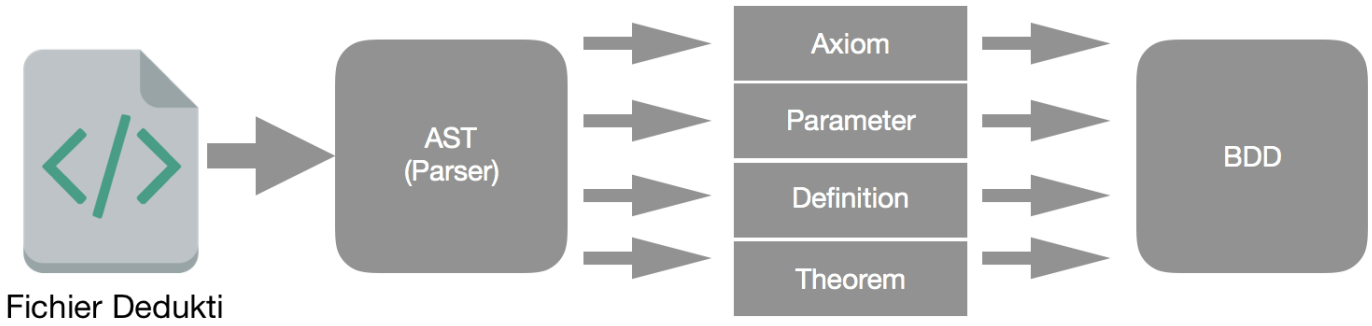
MongoDB est une base de données non-relationnelle. MongoDB a des champs de taille variable. Chaque tuple est un document (JSON). Nous pouvons modifier la structure sur un tuple précis, ce qui n'est pas possible en SQL. C'est-à-dire, que nous pouvons avoir un ou plusieurs tuple (document) qui a des champs différents des autres. Tandis qu'en SQL, nous définissions la table au préalable et chaque tuple doit respecter les champs de cette table. Dans notre cas, nous avons une petite exception où nous avons modifié la structure, ce fut donc un bon choix que d'utiliser une base de données MongoDB.
À travers les fichiers JSON, nous pouvons définir une structure. Tout d'abord, JSON est un format léger d'échange de données. Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou générable par des machines. Voici un exemple de fichier JSON :

```
[
{
  "langID":1,
  "type":"Type of the parameter in Dedukti",
  "nameID":"Name of the parameter",
  "md":"Module of the parameter"
},
{
  "langID":2,
  "type":"Type of the parameter in Matita",
  "nameID":"Name of the parameter",
  "md":"Module of the parameter"
}
]
```

C'est un exemple pour montrer que l'on peut aisément définir une structure, il suffit d'un crochet et d'un ensemble d'accolades. Nous verrons plus en détails ce que composent ces accolades plus tard. Nous allons donc passer à l'insertion dans la base de données.

4.2 Insertion

Tout d'abord, il faut savoir que nous n'avons pas besoin d'avoir un fichier qui créer la base de données et les différentes tables. Nous le faisons directement lors de l'insertion des tuples en OCAML. Étant données que MongoDB fonctionne avec des documents, si la base de données n'existe pas il l'a créé automatiquement sinon il ajoute ces documents à cette base. Les tables se nomme collection en MongoDB. Donc lors de l'insertion, si la collection n'existe pas le système de gestion de bases de données la crée sinon il insère le document(tuple) dans cette collection. Donc tout est géré directement lors de l'insertion.
Comme dit précédemment l'insertion se fait en OCAML. Dans un premier temps, nous déclarons toutes nos collections associées à la base de données voulues. Puis nous créons un fichier JSON vide. Enfin, nous déclarons notre fonction d'insertion qui prend les paramètres et les associer aux champs voulus. Nous avons par la suite une fonction print_bdd qui fait appel a un parser et qui itère les insertions selon le retour du parser. C'est-à-dire, que dans un premier temps nous avons un parser qui va analyser les démonstrations et nous retourner de quel type elle est (Axioms, Parameter, Definitions ou Theorems). Puis, nous, selon le retour nous appelons la bonne fonction d'insertion, car ces quatre collections n'ont pas les mêmes champs.
Voici un schéma représentatif des étapes d'insertions :



Voici tous les champs présent dans la base de données ainsi que leurs répar-

tition selon la collection :

Format des champs :
- langID : Référence au langage vers lequel il est traduit → Integer.
  - Dedukti : 1
  - Matita : 2
  - Coq : 3
  - Lean : 4
  - PVS : 5
  - OpenTheory (HOL) : 6
- nameID : Nom de la démonstration → String.
- md : Module de la démonstration → String.
- statement : String.
- type : String.
- proof : String.

Répartition :

Definitions :
- langID.
- nameID.
- md.
- statement.
- type.

Theorems :
- langID.
- nameID.
- md.
- statement.
- proof.

Parameters :
- langID.
- nameID.
- md.
- type.

Axioms :
- langID.
- nameID.
- md.
- statement.

Donc, à ce stade, nous avons toutes les démonstrations dans différents langages dans la base de données. Cependant, il nous reste une fonctionnalité qui nécessite de nouvelles collections : la possibilité d'afficher les dépendances d'un théorème.
Tout d'abord, il faut savoir que les dépendances d'une démonstration sont les mêmes peut importe le langage. Nous avons donc besoin de les insérer une seule fois. Nous les insérons donc lors de la construction du terme en Dedukti, c'est-à-dire lorsque nous construisons les différents champs statement,type,etc.. La méthode est la même que pour les autres collections. Il y a deux types de dépendances : la dépendance entre démonstration et celle entre module. Un module est un ensemble de démonstrations liées par une certaine logique. Par exemple, nous avons le module fermat qui regroupe tous les différents théorèmes de fermat.
Donc, lorsqu'une démonstration1 utilise une autre démonstration2 , nous

l'ajoutons en dépendance à la première démonstration1.
Voici les champs ajoutés à la base de données ainsi que leur répartition selon la collection :
Format des champs :

- idDep : Nom de la démonstration dépendante → String.
- mdDep : Module de la démonstration dépendante → String.

Répartition :

Dependances :

- nameID
- md
- idDep
- mdDep

DependancesMod :

- md
- mdDep

Donc, à cette étape, nous avons une base de données complète et prête à être utilisé pour le site web. Nous allons donc passer à la création du site web.

Troisième partie

Une encyclopédie en ligne de démonstrations formelles

Chapitre 5

Gestion du front-end

5.1 Les différentes pages

Actuellement, le site web se structure en trois types de pages :

— La page d'index : Cette page est la page d'accueil et permet d'effectuer une recherche et d'afficher 10 démonstrations aléatoirement.

— La page de théorèmes : Cette page permet d'afficher chaque démonstration, leurs dépendances, de pouvoir accéder aux demonstrations dépendantes, de pouvoir télécharger le fichier type-checkable et enfin de pouvoir effectuer une recherche.

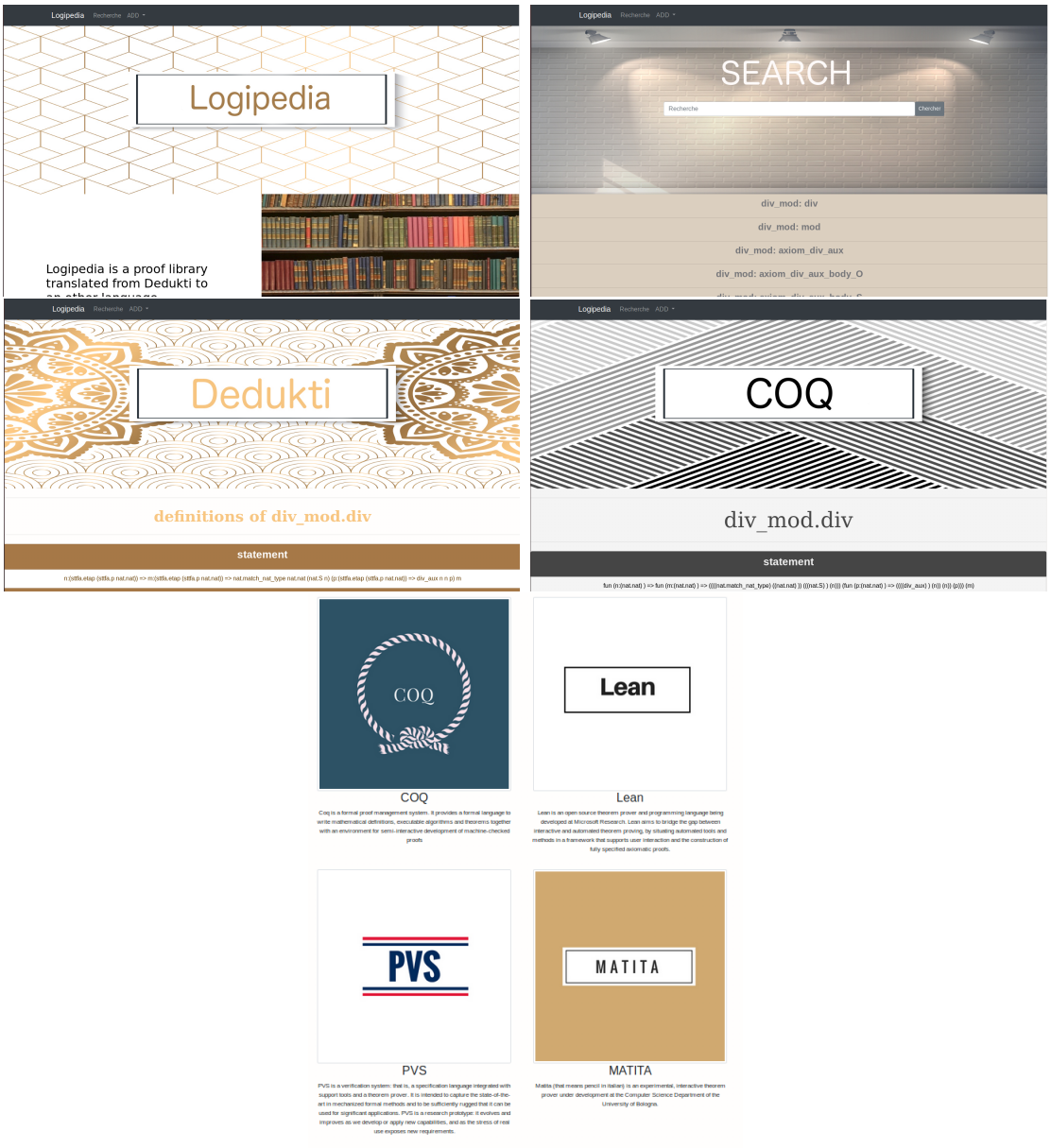
— La page about : Cette page définie ce qu'est Logipedia et leurs contributeurs. Et permet d'envoyer un mail à l'administrateur.

Il faut savoir que nous avons une page par théorème, aux quelles s'ajoutent donc les deux pages particulières : la page d'accueil et la page about.
Pour faciliter l'expérience utilisateur nous avons choisi d'avoir une seule page, scrollable, par théorème, qui présente les traductions de ce théorème dans les différents langages. Avec pour faciliter la vue de chaque langage, des marqueurs pour chaque langage (géré en CSS) qui ont pour lien la class ayant pour nom le langage souhaité.

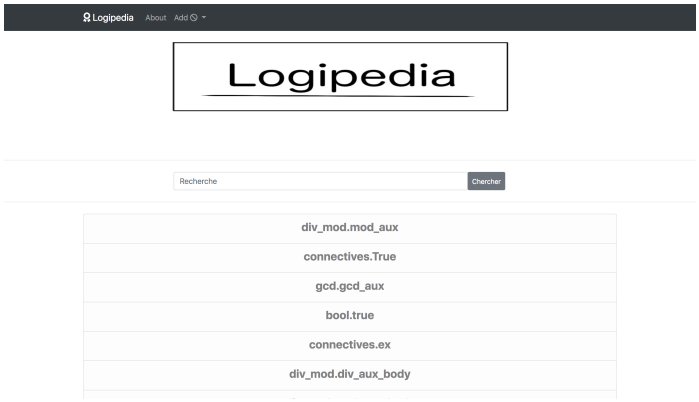
5.2 Evolution du front

Dans cette partie, nous allons nous baser sur une seule et même page. Au départ, je suis partie sur un style start-up avec beaucoup de JS et donc d'interaction avec l'utilisateur. Cependant, après une première réunion, nous avons décidé d'avoir un site plus simple, avec le moins de lien possible pour faciliter l'expérience de l'utilisateur. Par la suite, une réunion design à été prévu et j'ai donc décidé de proposer plusieurs styles différents pour le site,

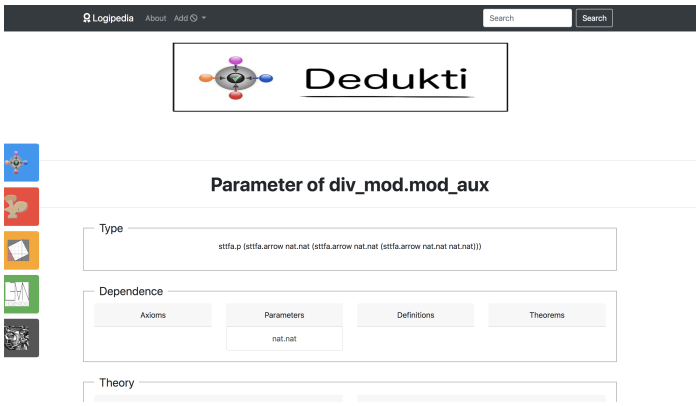
tel que :



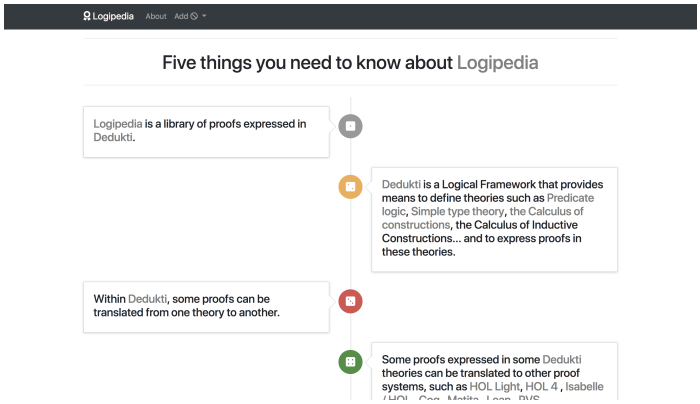
Ici nous pouvons voir la page d'index (qui est la page Logipedia), de recherche (que nous avons par la suite choisi de déplacer dans la page d'index), la page Dedukti, celle de Coq, et aussi dans la page Dedukti, les différents langages qui était proposé à ce moment là du projet. Je ne pouvais afficher toutes les pages par choix d'ergonomie, cependant, nous pouvons voir que je voulais proposer plusieurs possibilités. Sois un thème assez tape à l'œil (tel que la page Dedukti), sois quelque chose d'un peu plus sobre (tel que la page Coq). Puis à la suite d'une réunion avec toute l'équipe, nous avons décidé d'avoir un site plus sobre tel que :



Page d'index, qui permet aussi de faire une recherche et d'afficher dix démonstrations aléatoirement.



Page qui permet d'afficher et de télécharger la démonstration sous différents langages. Nous pouvons apercevoir cinq marqueurs sur le côté qui permettent de se déplacer directement au langage souhaité. Ainsi que les dépendances de la démonstration, qui sont toutes les définitions, axiomes et paramètres qui apparaissent dans la preuve. Et enfin les théories, qui sont la clôture transitive de tout les paramètres et axiomes apparaissant dans la preuve.



Page about, qui permet de décrire ce qu'est Logipedia et de contacter un administrateur.

Vous pouvez consulter toutes les pages du site à l'url : http ://logipedia.inria.fr/

Chapitre 6

Gestion du back-end PHP

Pour ce qui est du back-end en PHP, nous avons principalement les différents appels à la base de données, ainsi que tous les passage de données à travers la variable de session et les variables globales ainsi que tout les calculs fais coté serveur. Nous allons donc commencer par voir le calcul de la clôture transitive ainsi que la construction des fichiers téléchargeable. Puis nous allons voir les différents formulaires de recherche ainsi que comment sont fait les différents affichages.

6.1 Clôture transitive et construction des fichiers téléchargeables

Nos dépendances forment un arbre, et la clôture transitive d'une démonstration dans notre cas, permet de récupérer l'arbre de dépendances (c'est-à-dire chaque nœud du graph, ainsi que les feuilles). Pour ce faire, nous récupérons chaque dépendances récursivements jusqu'à atteindre les feuilles de l'arbre.
Ce traitement est actuellement réalisé au niveau du serveur, en PHP. Cependant, à terme, étant donné que l'arbre de dépendance est rarement modifié, nous allons stocker cet arbre dans la base de données. Puis à chaque modification de la base de données, recalculer au niveau du back-end la clôture transitive. Cela nous permettra d'avoir un gain considérable au niveau du chargement des pages pour les démonstrations ayant beaucoup de dépendances.
Donc, actuellement, nous avons défini une fonction récursive qui prend en entrée un nom de démonstration, ainsi qu'un tableau qui cumulera les différentes démonstrations dépendantes. Donc, après avoir appelé cette fonction, nous avons un tableau avec toutes les dépendances de la démonstration. C'est donc à partir de cela que nous allons construire notre fichier.

Pour ce faire, nous avons tout d'abord défini une fonction d'écriture sur fichier. Puis nous bouclons sur le tableau de dépendances. Et donc nous écrivons sur le fichier dans l'ordre des feuilles vers la racine de l'arbre. Puis, selon le langage souhaité, il faut respecter certains critères tels qu'une syntaxe précise, où le faite de rassembler les démonstrations par module dépendant. Et dans cette condition, nous calculons la clôture transitive des modules dépendants, de la même manière que l'on a calculé celle des dépendances entre démonstrations.
Une fois notre fichier créé, nous envoyons le chemin du fichier au script download, qui effectue plusieurs vérifications (tel que vérifier si le fichier est réellement présent, car il se peut qu'il y ait eu une erreur lors de la création de celui-ci), puis qui rend disponible le fichier à travers un bouton download.

6.2 Recherche et affichage

Nous avons deux moyens d'effectuer une recherche. L'un à travers la page d'index, l'autre lorsque nous consultons une démonstration au niveau de la navbar (la barre d'onglet présente en tête de page). Cependant, l'affichage du résultat de la recherche se fait dans la même page, la page d'index. Cela est plus pratique d'un point de vue ergonomique.
Pour la recherche, nous avons un formulaire, puis nous récupérons dans une chaîne de caractère la valeur de "l'input", donc la saisie de l'utilisateur. Puis, nous la découpons pour nous permettre de récupérer chaque mot. À l'heure actuelle, nous avons deux possibilités : soit l'utilisateur a entré un mot, soit deux. S'il y en à plus, nous récupérons seulement les deux premiers. Car nous partons du principe que la recherche correspond soit au module, soit au nom de la démonstration, soit aux deux. Puis, lorsque nous avons récuperé chaque mot, nous vérifions à travers une requête à la base de données, si ce mot correspond à un module ou à un nom de démonstration. Si c'est un module, nous affichons toutes les démonstrations de ce module, à travers des liens qui redirige vers la page de la démonstration en Dedukti et vers les autres langages. Si c'est un nom de démonstration, nous affichons un lien correspondant à cette démonstration. Pour ce faire, nous envisageons toutes les combinaisons possibles lorsqu'il y a deux mots saisis. Soit le nom en premier puis le module, soit l'inverse.

Conclusion

Ce stage avait pour ambition de créer une plateforme qui permettra de réunir différentes communautés de développeurs et d'utilisateurs de démonstrations formelles. Comme vous avez pu le voir en lisant ce rapport, nous avons fait un grand pas dans cette direction. À travers, la création d'un dépôt contenant toute les démonstrations dans différents langages. Mais aussi, à travers la création d'un site web, permettant de partager ces connaissances à travers le monde en entier, et ce gratuitement. Cela permettra à la fois de réunir la communauté des développeurs de démonstrations formelles, mais d'attirer différents autres utilisateurs, tel que des professeurs qui souhaiterait connaitre une démonstration mathématique précise.
À travers ce stage, j'ai pu en apprendre d'avantage sur les démonstrations formelles. Mais aussi consolider mes connaissances en programmation fonctionnelle (OCAML). De plus, j'ai pu perfectionner mes connaissances dans le développement web (PHP, Javascript, HTML/CSS, Bootstrap). J'ai aussi eu la chance de découvrir et me former dans la création et l'utilisation d'une base de données non-relationnelle (MongoDB). Enfin, j'ai pu améliorer mes méthodes de travail, tel que l'utilisation d'un dépôt Github.
Ce stage m'a permis d'entrevoir le monde de la recherche et ces nombreuses facettes. De part, les différentes conférences et réunion faite au sein du laboratoire. Mais aussi à travers une méthodologie de travail qui prône l'échange et le partage de connaissances, notamment par les nombreuses contributions des personnes faisant partie du laboratoire.
Le projet est encore en pleine expansion, il reste encore quelques rajouts à faire pour pouvoir atteindre l'objectif final, tel que l'ajout d'un langage compréhensible par un être humain. C'est donc pour cela que le projet est open source et contribuable à travers le dépôt Github : https ://github.com/Deducteam/Logipedia.
Vous pouvez d'ors et déjà consulter le site web à travers ces liens :

— http ://logipedia.science/
— http ://logipedia.inria.fr/
— http ://logipedia.saclay.inria.fr/

Bibliographie

— http ://www.lipn.univ-paris13.fr/ guerrini/stages/info2/Consignes/ConsignesStagesINFO2.html
— https ://www.scribbr.fr/memoire/exemple-introduction-de-memoire/
— https ://www.inria.fr/equipes/deducteam
— https ://fr.wikipedia.org/wiki/D
— https ://deducteam.github.io/
— https ://coq.inria.fr/
— http ://pvs.csl.sri.com/
— http ://www.gilith.com/opentheory/
— https ://leanprover.github.io/