# Rapport - Projet informatique

Félix Bélanger-Robillard

April 12, 2017

# Contents

# 1 Introduction

Le présent projet avait pour but de créer un webservice servant les fonctionnalités de BiBler, puis de l'intégrer à ReLiS (Revue Littéraire Systématique) afin d'offrir une plateforme web permettant d'allier les fonctionnalités propres à ces deux logiciels, le tout sous la supervision de monsieur Eugène Syriani et avec la précieuse collaboration de Brice-Michel Bigendako qui s'occupe de ReLiS. Le projet intégrait des connaissances en PHP et Python. Le projet m'a aussi amené à travailler avec des serveurs de type Apache 2 et avec les systèmes de permissions Unix. Pour la rédaction du rapport, j'ai utilisé de nouveaux paquets LaTeX et au niveau de la documentation du projet, j'ai appris à utiliser Sphinx pour la générer.

-Contexte et but, mentionner nouvelles connaissances
-Vue d'ensemble architecture du logiciel
-Mentionner tout, montrer qu'il y avait des challenges non trivial
-Section: intro( définition de concepts, termes et outils ), core, évaluation, conclusion, remerciements, références bibliographiques (technologies), annexes,
conclusion: résumé, puis future sabrina et khady, A+

## 1.1 Terminologie

BiBler est un outil de gestion de référence bibliographique. BiBler se sert de BibTeX, une librairie LaTeX de gestion référence, et offre une interface graphique afin de faciliter l'importation, l'exportation et la modification de références. BiBler offre également un aperçu des références et facilite l'insertion au sein de textes. BiBler génère également des clés d'identification pour les documents insérés qui peuvent ensuite être utilisés dans un document LaTeX.

ReLiS, Revue Littéraire Systémique, est un outil web rédigé en PHP dans le framework CodeIgniter. ReLiS vise à aider les chercheurs au moment de consulter la littéraire d'un sujet précis. ReLiS permet à ces chercheurs de réduire l'ampleur de la tâche en automatisant en bonne partie la classification et avec des outils permettant un processus systématique de révision des articles.

# 2 Webservice

## 2.1 Analyse des besoins

Le besoin initial était de mettre BiBler en webservice d'une part et de l'intégrer à ReLiS d'une autre part. La première décision prise à ce niveau, a été de conserver le code de BiBler et d'utiliser un framework Python afin de ne pas avoir à réécrire le code du logiciel et dans le soucis de pouvoir maintenir le webservice plus aisément. Avec le webservice directement dépendant du code source originel, la maintenance à ce niveau, avec une nouvelle version de BiBler, sera de remplacer la libraire à l'emplacer physique de l'importation pour le webservice et de s'assurer ensuite que les tests unitaires sont toujours valides.

Afin de conserver au minimum le nombre de technologies employées, un framework qui n'utilise que Python était souhaité et, idéalement, avec un minimum de contraintes sur son utilisation. Il fallait également que l'application puisse être déployée sur un serveur Apache 2 à cause des contraintes des serveurs du département. C'est pour cette raison que Web.py [1] a été choisi comme framework pour le webservice. De cette façon, avec BiBler, tout comme pour son webservice, le seul langage utilisé demeure Python 3.

Au niveau de l'intégration avec ReLiS, les besoins étaient de pouvoir appeler le webservice, de formatter correctement les fichiers BibTeX, de générer les clés pour les articles et de pouvoir intégrer l'abstract au niveau de la référence bibtex. Le dernier besoin a entraîné une maintenance au niveau de BiBler, puisque ce n'était pas un champ qui y était présent. La génération de clés était également une exigence partiellement difficile à remplir, car la décision avait été prise de ne pas conserver de données utilisateurs du côté du webservice. Pour cette raison, il fallait également ajouter au sein de ReLiS une vérification au niveau des doublons pour modifier leur clé, puisque les clefs des publications d'un auteur au cours d'une année allait être les mêmes.

## 2.2 Conception

Plusieurs choix ont été effectués par rapport à l'implantation du webservice par rapport à l'approche desktop de BiBler. Le webservice ne conserve aucune donnée envoyée au serveur, il effectue l'opération et retourne le résultat désiré. Afin d'arriver à ce résultat, une nouvelle instance de l'application est instanciée à chaque appel de méthode. Sans cette particularité, l'exportation de résultats sur une instance commune aurait retourné l'ensemble des fiches importées et comme on ne veut pas avoir à gérer le contenu utilisateur, ç'aurait été un effet indésirable. Comme on ne peut avoir d'instance commune sur plusieurs requêtes, il y a tout de même possibilité d'envoyer un fichier et de le traiter avec un seul appel au webservice et d'exporter ensuite un fichier.

Pour l'intégration avec ReLiS, puisque ReLiS est construit sur une architecture Modèle-Vue-Controlleur (MVC), l'idée est d'ajouter un controlleur spécifiquement pour ReLiS avec une vue associée. Tout en sachant les méthodes dévoilées par le webservice, il a été également pensé de faire appel au patron de proxy pour encapsuler les méthodes PHP nécessaires aux appels vers le webservice. De cette façon, les travaux futurs sur ReLiS ou tout autre application php severraient grandement facilités.

## 2.3 Design

Le webservice dépend de BiBler et n'utilise que la classe BiBlerApp qui correspond à l'API de cette dernière. Ensuite, le webservice instancie un object BiBlerApp et encapsule chacune des méthodes demandées. Chaque méthode est invoquée par un URL différent. Chacun de ses URLs réfèrent à une classe dont la méthode POST renvoit à la méthode correspondante de BiBler qui est encapsulée par la classe BiBlerWrapper du webservice pour assurer les instanciations locales nécessaires et les appels de méthodes de l'API. Les méthodes de l'API qui n'ont pas été encapsulées sont celles qui effectuent des query dans une application déjà existante qui contient des entrées. Elles pourraient être encapsulées, mais vu le choix de ne conserver aucune donnée, le résultat serait toujours nul.

## 2.4 Implantation

## 2.5 Tests

## 2.6 Intégration et déploiement

## 2.7 Maintenance

Post versus GET, documenter Impact, maintenance Patron proxy performance
Les méthodes d'export ont pour attrait de pouvoir envoyer un fichier BibTeX et de le transformer en fichier BibTeX corrigé, en CSV, en HTML ou en SQL. Par contre, cela nécessite l'écriture dans un fichier.

L'utilisation attendue du webservice est via une requête http POST vers une méthode du webservice. Dans le cas de ReLiS ou d'un service basé sur PHP, la fonction suivante permet de faire une requête, attendu que $url contient l'url de la méthode désirée et que $data contient les données à transmettre sous forme de String:

```
function httpPost($url, $data)
{

   $ch = curl_init( $url );
   curl_setopt( $ch, CURLOPT_POST, 1);
   curl_setopt( $ch, CURLOPT_POSTFIELDS, urlencode($data));
```

```
   curl_setopt( $ch, CURLOPT_FOLLOWLOCATION, 1);
   curl_setopt( $ch, CURLOPT_HEADER, 0);
   curl_setopt( $ch, CURLOPT_RETURNTRANSFER, 1);

   $response = curl_exec( $ch );
    return $response;
}
```

Pour l'intégration avec ReLiS, un champ abstract a été ajouté au code originel de BiBler. Pour ce faire, abstract a été ajouté à l'énumération des champs possibles. Une classe Abstract héritant de Field a égalément été ajoutée et abstract a été ajouté aux champs possibles de divers éléments, comme Article.

# 3 Conclusion

## 3.1 Remerciements

Un merci infini à ceux qui m'ont aidé et accompagnés dans cette tâche:
Eugène Syriani, Ph.D., pour la supervision et les conseils précieux
Brice-Michel Bigendak, pour ReLiS et le débogage en PHP, L'équipe technique du DIRO, pour le support technique et le serveur

# References

[1] Web.py. http://webpy.org/. Accessed: 2017-04-12.