Faculté
des Sciences
Département des Sciences Informatiques

UMONS
Université de Mons

Générateur Dynamique de Contenu Statique

Projet de 1^{ere} Master réalisé par Maxime DE WOLF

Année académique 2017–2018

Directeur: Decan Alexandre

Service: Sciences Informatiques

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Table des matières

1 Introduction

Dans le cadre du cours de lecture et rédactions scientifiques, il nous est demandé de réaliser un projet d'envergure relativement importante. Dans notre cas, il s'agit d'implémenter un générateur dynamique de contenu statique. Ce rapport a pour but de vous présenter l'avancement de ce projet.

Dans un premier temps, nous dresserons le cahier des charges de ce projet. Cela consistera à lister les fonctionnalités nécessaires à respecter afin que ce projet soit mené à bien. Nous illustrerons également quelques cas d'utilisations afin de clarifier le cahier des charges.

Ensuite, nous listerons rapidement quelques générateurs de contenu statique existants. Nous exhiberons leurs différences ainsi que leurs fonctionnalités.

Troisièmement, nous dresserons une liste de l'ensemble des fonctionnalités que devra posséder notre générateur de contenu statique. Nous en profiterons également pour le comparer aux autres générateurs présentés en Section 3. Cela nous permettra donc de justifier la nécessité de ce projet.

Enfin, nous parlerons des techniques mises en place pour répondre aux besoins présentés en Section 4. En d'autres termes, cette section détaillera l'approche qui sera utilisée lors du développement de notre générateur de contenu statique.

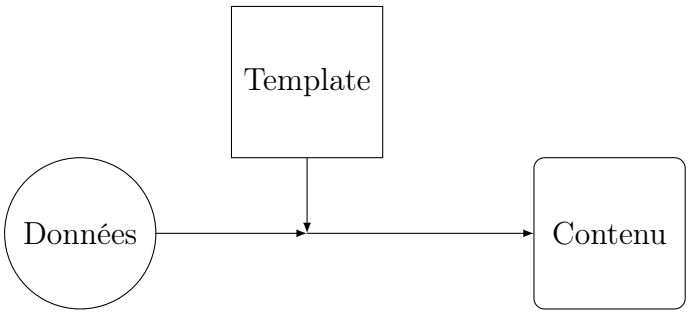

FIGURE 1 – Utilisation type d'un générateur de contenu statique

2 Cahier des charges

Cette section a pour rôle de présenter le cahier des charges de ce projet. Nous introduirons également le concept de générateur de contenu statique. La seconde partie de cette section listera quelques cas d'utilisation afin d'illustrer le fonctionnement souhaité par ce projet. Notez que les fonctionnalités en elles-même seront détaillées dans la Section 4.

Le cahier des charges de ce projet est volontairement resté assez flou. En effet, les seules consignes données sont de concevoir un générateur de contenu statique qui donne à l'utilisateur une grande souplesse d'utilisation. La simplicité d'utilisation du générateur est également un critère important. Afin de clarifier ce que veut dire "souplesse d'utilisation", quelques cas d'utilisations seront illustrés dans cette section.

2.1 Générateur de contenu statique

Un générateur de contenu statique sert -comme son nom l'indique- à générer du contenu. Il sera qualifié de "statique" si ce contenu ainsi généré est prêt à l'emploi et ne nécessite plus aucune transformation. A savoir que les générateurs de contenu statique sont très utilisés pour produire des sites web statiques. Celui-ci aura une vocation un peu plus générale car il permettra de générer à peu près n'importe quel type de contenu statique. Leur utilisation est souvent liée au concept de *templates* qui permettent de séparer les informations de la mise en page en elle-même. La Figure 1 montre une utilisation type de ce genre de générateur.

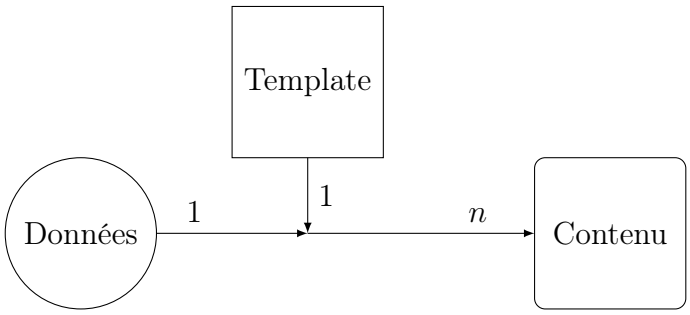

FIGURE 2 – Représentation du cas **OneToAll**.

2.2 Cas d'utilisations

Nous distinguons principalement trois cas d'utilisations propres aux générateurs de contenu statique. Chacun d'entre eux sera illustré à l'aide d'une figure. Nous appellerons ces trois cas comme suit : *OneToAll*, *AllToOne* et *AllToMany*.

OneToAll

La cas *OneToAll* désigne l'utilisation d'un générateur de contenu statique en vue de générer plusieurs fichiers en sortie à partir d'un seul fichier en entrée. Cela correspond, par exemple, à afficher un produit par page à partir d'un fichier contenant l'ensemble des produits disponibles. Cela se traduit par la Figure 2 en considérant que **n** est le nombre de produits contenus dans le fichier.

AllToOne

Le second cas, *AllToOne*, est le cas inverse de *OneToAll*. En effet, ce cas désigne l'utilisation d'un générateur de contenu statique qui génère un seul fichier à partir de plusieurs. En pratique, cela consiste par exemple à afficher l'intégralité des posts d'un blog sur une seule page, chaque page étant symbolisée par un fichier. Cela donne la Figure 3 où **n** est le nombre de fichiers contenant un post.

AllToMany

Le dernier cas est celui que nous appellerons *AllToMany*. Ce cas est en fait une généralisation des deux cas précédents. Effectivement, *AllToMany*

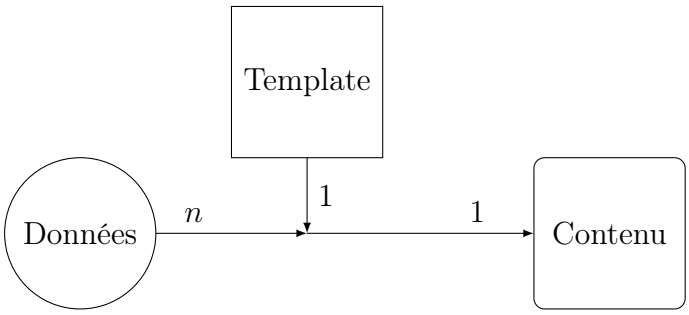

FIGURE 3 – Représentation du cas **AllToOne**.

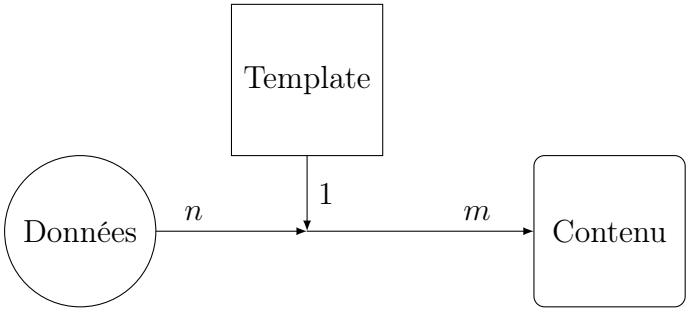

FIGURE 4 – Représentation du cas **AllToMany**.

désigne l'utilisation d'un générateur de contenu statique en vue de générer plusieurs fichiers en sortie à partir de plusieurs fichier d'entrée. Mis en contexte, il désigne par exemple la génération de fichiers contenant chacun plusieurs posts à partir de fichiers contenant chacun un post. La Figure 4 permet une vision plus claire de ce cas où **n** désigne le nombre de fichiers contenant chacun un post et **m** le nombre de pages (resp. fichiers) contenant un certain nombre de posts.

2.3 Cas non-couverts

Un lecteur attentif aura remarqué que nous ne discutons pas de la multiplicité au niveau de l'application des *templates*. Par soucis de complétude, nous les listerons dans cette section. Ils feront éventuellement partie de l'implémentation finale si le temps nous le permet. Nous appellerons ces cas *MultiTemplatesIn* et *MultiTemplatesOut*.

Notez que ces cas d'utilisations seraient réalisables sans implémentation

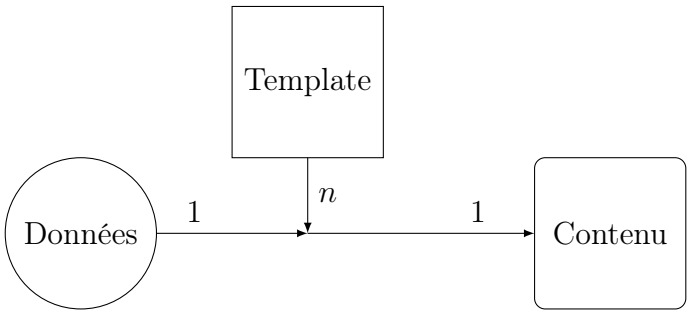

FIGURE 5 – Représentation du cas **MultiTemplatesIn**.

additionnelle. Malheureusement cela impliquerait quelques contraintes pour l'utilisateur ce qui nuirait à la souplesse d'utilisation de notre générateur de contenu statique. Nous discuterons de cela plus avant dans la Section 5.

MultiTemplatesIn

Ce cas d'utilisation implique d'utiliser un générateur de contenu statique dans le but de produire un fichier contenant plusieurs fois les mêmes données mais agencées par un *template* différent. Nous n'avons pas trouvé d'exemple pratique mais cela pourrait servir à comparer plusieurs *templates* entre eux. La Figure 5 illustre ce cas plus clairement, où **n** est le nombre de *templates* à appliquer sur le fichier de données.

MultiTemplatesOut

Ce cas d'utilisation est similaire au précédent, il implique également l'application de plusieurs *templates* sur un même fichier de données. La différence vient du fait que *MultiTemplatesOut* produit un fichier en sortie par *template* appliqué. En pratique, cela revient à dire que pour un fichier contenant les informations propres à un *CV*, on veut générer un fichier LaTeXet une page web. Ces deux contenus contiendraient les mêmes informations mais l'un pourra être imprimé (une fois le PDF produit) et l'autre sera mis à disposition sur Internet. Cela se traduit par la Figure 6 où **n** est le nombre de *templates* à appliquer sur le fichier.

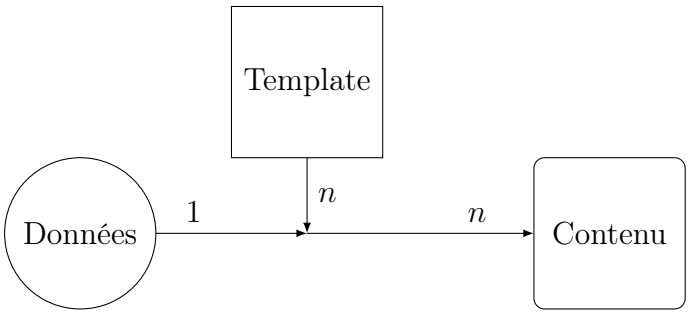

FIGURE 6 – Représentation du cas **MultiTemplatesOut**.

3 Etat de l'art

Cette section détaille, de manière non-exhaustive, l'existence d'autres générateurs de contenu statique similaires à ce projet. Leurs fonctionnalités principales seront présentées afin d'être comparées avec notre propre générateur.

Pelican

Pelican [1] est un générateur de site web statique dont les principales fonctionnalités sont :

— Le support de *pages* (e.g. "Contact", ...)
— Le support d'*articles* (e.g. posts d'un blog)
— La régénération rapide de fichiers grâce à un système de caches et d'écriture sélective.
— La gestion de thèmes créés à partir de *templates Jinja2*
— La publication d'articles dans plusieurs langues

Lektor

Lektor [3] est également un générateur de site web statique. Ses principales fonctionnalités sont les suivantes :

— Un système de construction intelligent qui ne reconstruit que le contenu qui a été modifié
— Un outil graphique qui permet la modification de pages sans toucher au code source
— Utilisation de système de *templates Jinja2* pour le rendu du contenu
— Un outil permettant la création relativement simple de sites web multilingues

On peut donc en conclure que c'est un outil assez similaire à Pelican sauf que Lektor propose un outil graphique pour l'édition de contenu.

Jekyll

Enfin, Jekyll [4] est le plus connu des générateurs de sites web statiques *open source*. Il est écrit en *Ruby* à la différence de Pelican et Lektor, écrits en *Python*. Voici ses principales fonctionnalités :

— Utilisation de *templates Liquid*
— Support de contenu de type *pages* (e.g "Contact", "Accueil", ...) et *articles* (e.g. posts d'un blog)

— Lancement d'un serveur local pour observer le rendu graphique des fichiers générés

Malgré leurs différences, chacun de ces générateurs de sites web statiques fonctionnent selon le cas d'utilisation *AllToOne* (voir Figure 3). Ce cas est très utilisé dans le cas de sites web de type blog d'où sa popularité.

4 Analyse fonctionnelle

Cette section est destinée à justifier les choix de conception de ce projet afin de remplir le cahier des charges. Certains choix seront inspirés directement de solutions existantes listées dans la section précédente.

La section suivante, sera directement liée à celle-ci car elle expliquera comment ces choix seront techniquement mis en œuvre lors de la phase d'implémentation.

4.1 Langage de programmation

Afin de garantir la facilité d'utilisation de notre générateur de contenu statique, notre choix de langage de programmation s'est tourné vers *Python 3*. En effet, ce langage possède une syntaxe suffisamment flexible pour nous permettre de proposer à notre tour une syntaxe intuitive pour l'utilisateur.

4.2 Programmation modulaire

Pour ce qui est de la souplesse d'utilisation, nous appliquerons les concepts de la programmation modulaire. Cela permettra à l'utilisateur de modifier ou de créer facilement de nouvelles fonctionnalités pour notre générateur de contenu statique. Nous fournirons tout de même les fonctionnalités de bases nécessaires au bon fonctionnement du projet.

4.3 Règles

Pour ce qui est de l'utilisation en elle-même de notre générateur de contenu statique, nous avons pensé à un système de règles relativement simple. Ces règles proposeront un système de variables pour permettre à l'utilisateur de réutiliser la même valeur dans plusieurs champs de la même règle. Ces règles seront contenues dans un fichier qui sera passé en paramètre à notre programme qui les exécutera une par une. Chacune de ces règles symbolisera un des trois cas d'utilisation exposé en Section 2 : *AllToOne, OneToAll* et *AlltoMany*. De plus amples informations sur la syntaxe de ces règles seront détaillées dans la section suivante.

4.4 Système de fichiers *logs*

Un système de fichiers *logs* permettrait de faciliter la correction de certaines règles lorsque les fichiers en sorties ne correspondent pas à ce que l'utilisateur attendait. Ces fichiers *logs* n'auront comme utilité que de lister

l'historique des règles exécutées pas à pas. Ce système viserait à atteindre l'objectif de simplicité d'utilisation -et d'aide à l'utilisation dans ce cas-ci- mentionné dans le cahier des charges en Section 2.

5 Analyse technique

Cette section est destinée à préciser les détails d'implémentation des principales fonctionnalités présentées dans la section précédente. Cette section n'est pas complète car l'implémentation de certaines fonctionnalités n' a pas encore été étudiée.

5.1 Règles

Les règles nécessaires au bon fonctionnement de notre générateur de contenu statique possèdent quatre champs obligatoires : *Target*, *Template*, *Data* et *Output*. Chacun de ces champs désigne un ou plusieurs fichiers grâce aux expressions régulières. La signification de chacun de ces quatre champs est expliquée ci-dessous.

Target

Les fichiers passés dans ce champs serviront de données pour le *template*. Chaque fichier produira un fichier en sortie. C'est donc ce champ qui permet de gérer la multiplicité des fichiers de sortie.

Template

Ce champ désigne le fichier *template* à utiliser sur les données chargées grâce aux champs *Target* et *Data*.

Concernant le cas *MultiTemplatesIn* mentionné dans la Section 3, nous pouvons solutionner ce problème en utilisant l'héritage de *template* de notre moteur de *template*. Par défaut nous utiliserons le moteur *Jinja2* [2] qui supporte cette fonctionnalité. Le problème qui en découle est donc que ce cas d'utilisation ne sera réalisable que si on utilise un moteur de *template* qui supporte une telle fonctionnalité.

L'autre cas non-couvert, également exposé en Section 3, pouvant être solutionné est *MultiTemplatesOut*. Dans l'état actuel des choses, il faudrait écrire la même règle **n** fois juste en changeant à chaque fois le champ *template* par le fichier correspondant où **n** est le nombre de *templates* à appliquer sur le fichier. Cette méthode deviendrait donc extrêmement lourde si le nombre de *templates* à appliquer est important.

Data

Les fichiers passés dans le champ *Data* serviront également de données pour le *template* comme *Target*. La différence est que ce champ n'influence

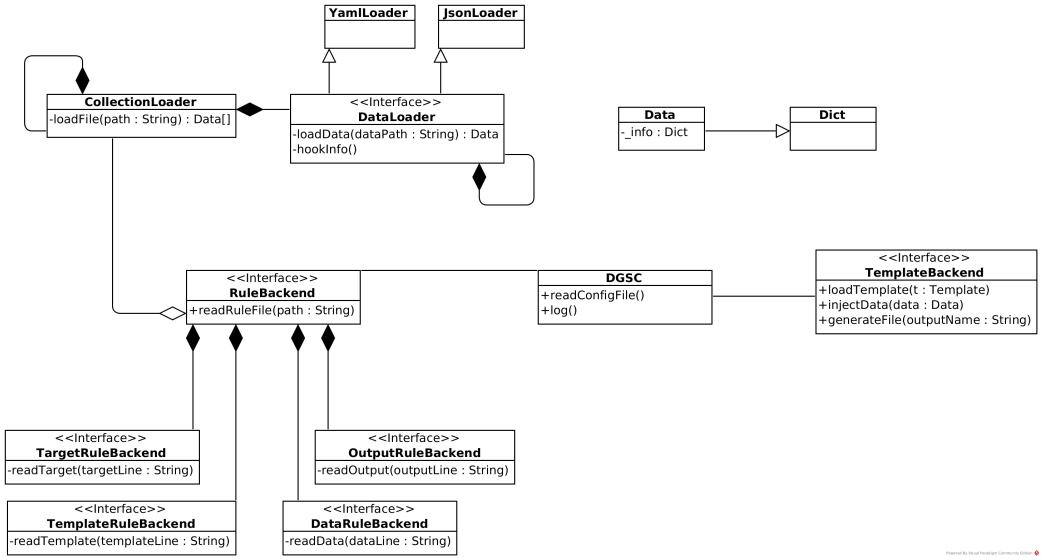

FIGURE 7 – Diagramme de classes de notre générateur de contenu statique

pas le nombre de fichiers de sortie. Les champs *Target* et *Data* permettent donc de gérer les multiplicités des fichiers d'entrée.

Output

Ce champ permet de spécifier le nom des fichiers de sortie. L'utilisation de variables de champs mentionnées dans la Section 4 permettra par exemple de nommer chaque fichier de sortie en fonction de son "homonyme" passé au champ *Target* ,et cela, peut importe la multiplicité en sortie.

5.2 Structure

Comme mentionné dans la Section 4, nous appliquons la programmation modulaire afin de permettre une grande flexibilité à notre générateur de contenu statique. Nous avons donc conçu l'architecture dans cette optique comme le montre la Figure 7. Un utilisateur pourra donc modifier le comportement de notre générateur en implémentant lui même l'une de ces interfaces. Il pourra, entre autres, modifier la syntaxe des règles mentionnées ci-dessus en implémentant l'interface *RuleBackend*.

Comme expliqué dans la section précédente, nous fournirons par défaut une implémentation de ces interfaces qui permettra à l'utilisateur de réaliser les cas d'utilisations listés en Section 2.

5.3 Configuration

Un fichier de configuration permettra à l'utilisateur de spécifier quelques options à notre générateur de contenu statique. Parmi ces options, nous retrouverons entre autres : le répertoire de travail en entrée, le répertoire de travail en sortie, une liste de classes implémentant les interfaces du générateur, l'emplacement du fichier de *logs* (le cas échéant), ... Chacune de ses options aura une valeur par défaut, l'utilisateur ne devra donc spécifier que les éléments qui diffèrent de la configuration par défaut.

Bibliographie

[1] Documentation de pelican. http://docs.getpelican.com/en/stable/. (consulté le 14/11/17).

[2] Page d'accueil de jinja2. http://jinja.pocoo.org/. (consulté le 31/12/17).

[3] Page d'accueil de lektor. https://www.getlektor.com/. (consulté le 14/11/17).

[4] Présentation de jekyll. https://jekyllrb.com/. (consulté le 14/11/17).