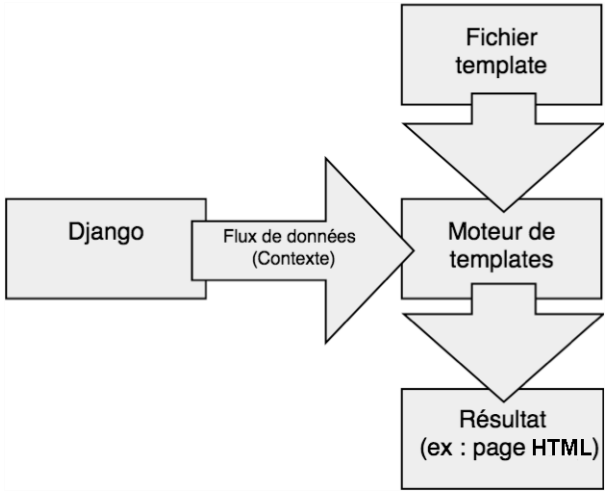
Chapitre 7
Les templates

1. Principes et généralités

Nous utilisons le mot anglais de template, car il est communément utilisé, répandu, et compris dans ce contexte par les professionnels. De plus, en cas de recherche sur le Web, il est plus facile de trouver la bonne information avec template qu'avec une traduction française. Pour mémoire, template peut être traduit ainsi : modèle/patron/gabarit.

Le principe général de fonctionnement d'un moteur de templates est le suivant : un fichier décrivant la structure de ce que l'on veut produire ou afficher va servir d'ossature ou de modèle. Le moteur (ou processeur) de templates va venir « habiller » ce squelette en fonction des données produites par le programme, le résultat sera une page dynamiquement générée. Concrètement, pour Django, c'est un fichier texte qui est rempli par le moteur de templates avec des variables générées par le framework afin de produire un document au format requis, le plus souvent HTML.

Schémas de principe :



Le principe mis en œuvre dans Django pour gérer des templates est donc, pour ceux qui connaissent déjà le principe, similaire à celui que l'on peut rencontrer dans d'autres langages/systèmes de templates, tels que Jinja2, HAML, OpenDoc, Mako, boiler plate, etc. Il y a cependant des spécificités propres aux templates Django, en particulier une excellente intégration avec l'ensemble du framework.

1.1 Intérêt des templates

Les principaux avantages apportés par ce type d'outil sont :

- La séparation des préoccupations (respect du principe SoC), en particulier pour assurer la séparation entre le webdesign et la programmation.
- Une grande flexibilité d'évolution de l'aspect du site, de sa présentation.
- La facilité de « localisation » (adaptation à une langue) de l'interface du site.
- La possibilité de faire travailler séparément et en même temps des personnes différentes sur la conception graphique et sur le code.
- La normalisation et la standardisation des contenus et des affichages.
- La réutilisation et la généricité du code produit.

© Editions ENI - All rights reserved

1.1.1 La séparation des préoccupations ou séparation des problèmes

Un principe important de la programmation moderne est le SoC (*Separation of Concerns*), ou la séparation des préoccupations. Ici le but est de séparer les aspects de design web, bien mieux maîtrisés par une partie de l'équipe (webdesigners, programmeurs JavaScript, etc.), des aspects de pure programmation du framework, mieux maîtrisés par les programmeurs. En bref, de séparer le développement de la mise en page. Un objectif chez tous les développeurs web est de développer des applications souples et faciles à maintenir. Pour cela il est fondamental de séparer la logique métier de la logique de présentation. Les systèmes de templates web permettent de maintenir cette séparation.

De ce point de vue, les templates Django répondent au besoin, car contrairement à ce que l'on peut voir dans d'autres systèmes, les templates Django ne sont absolument pas du code Python inséré dans du HTML, mais bien un langage permettant aux équipes de prendre totalement en charge cette partie du travail et de se consacrer au design.

– Pour les concepteurs du site : lorsque chaque page web est générée par un moteur de templates, elle peut être conçue de manière modulaire et structurée avec des composants pouvant être modifiés indépendamment les uns des autres. Ces composants peuvent être par exemple : un en-tête, un pied de page, une barre de navigation globale, une barre de navigation locale, un fil d'Ariane (breadcrumb) et du contenu tel que des articles, images, vidéos, etc.
– Pour les programmeurs : le langage de template offre des capacités logiques plus restreintes qu'un langage de développement classique, car ces capacités doivent servir uniquement à réaliser des adaptations de présentation et à prendre des décisions simples. Les templates ne sont absolument pas faits pour mettre en œuvre des algorithmes complexes.

De cette séparation entre la présentation et le code découlent également la plupart des autres avantages ci-dessous.

1.1.2 La flexibilité : facilité à modifier la présentation, l'aspect ou le design du site

L'une des raisons derrière la séparation est la nécessité d'une évolutivité maximale dans le code et les ressources consacrées à la partie de présentation. En effet, sous la pression des nouvelles demandes des utilisateurs, de l'évolution des préférences ou des goûts des utilisateurs, de la mode, et du désir de présenter de manière nouvelle des contenus préexistants, il est parfois nécessaire de modifier radicalement l'aspect public du site web tout en perturbant le moins possible l'infrastructure technique sous-jacente.

Un exemple d'actualité assez pertinent étant la nécessité d'adapter les contenus existants des sites Internet au surf sur mobile ou tablette. Grâce aux templates, cette adaptation, bien que conséquente, est assez simple à réaliser sans toucher au code de l'application.

Dans les templates Django, les modifications ne dépendent pas du contenu. Cela signifie que les concepteurs de sites web peuvent mettre à jour la présentation sans avoir à s'occuper des considérations techniques.

1.1.3 La facilité de « localisation » (adaptation à une langue) de l'interface du site

Les menus et d'autres éléments de l'affichage sont facilement uniformisés pour les utilisateurs du site dans une langue donnée, en créant les templates appropriés.

1.1.4 Webdesign et codage en parallèle

Comme le code et l'affichage sont bien séparés, il est aisé de comprendre qu'au-delà de la coordination nécessaire, deux équipes ou deux personnes différentes pourront travailler en même temps sur le codage et la présentation du site.

1.1.5 La normalisation et la standardisation des contenus et des affichages

Comme souvent, certains affichages sont similaires, par exemple : l'affichage d'une liste de produits, d'un tableau de statistiques, etc. Nous allons pouvoir utiliser la même template pour tous les éléments similaires, et ainsi rendre l'aspect du site plus cohérent et plus uniforme. L'utilisateur va comprendre la logique qui est à l'œuvre et mieux se repérer dans la navigation. De plus, certains langages de templates, dont Django, implémentent une fonctionnalité intéressante, qui est l'héritage entre templates. Ainsi, comme on va le voir dans ce chapitre au paragraphe consacré à l'héritage, on pourra définir des modèles structurés d'affichage, par exemple pour le site : par catégorie, par sous-catégorie et ainsi de suite. Ainsi on peut assurer une grande cohérence graphique de l'aspect du site tout en minimisant le travail nécessaire pour réaliser une telle opération.

1.1.6 Réutilisation et généricité

Une agence web va pouvoir capitaliser certaines parties répétitives de conception de page web. Les utilisateurs non spécialistes, qui n'ont pas la possibilité ou la capacité d'embaucher des développeurs pour concevoir un système cousu main, pourront utiliser tes templates disponibles sur Internet et les adapter, par exemple « Twitter bootstrap ».

© Editions ENI - All rights reserved

1.2 Django par rapport aux autres systèmes

Si l'on souhaite comparer en matière de fonctionnalités les templates de Django aux autres langages de templates, on remarque que le langage de templates fourni avec Django n'a rien à envier aux autres langages (voir http://en.wikipedia.org/wiki/Comparison_of_web_template_engines en anglais). En effet, les templates Django gèrent un langage structuré de programmation, avec des variables, des fonctions, l'inclusion de templates, des instructions de contrôle (boucles, tests...), l'évaluation et l'affectation, la gestion des erreurs et des exceptions ainsi que l'héritage. Enfin, les possibilités des templates Django sont largement extensibles par l'utilisateur.

Un reproche que l'on pourrait faire à Django est que les templates Django, bien que pouvant être directement édités dans un logiciel de création web (par exemple Komposer ou Dreamweaver), ne représentent pas toujours directement ce qui va être affiché au final (absence de notion de templates naturels). Cependant, on peut avec un peu de pratique contourner ce problème.

Un point très positif, les templates Django sont bien adaptés au framework Django dont ils collent bien à la logique et sont proches de la logique Python. Cela permet donc d'avoir à la fois une approche SoC et une absence de couture (seamless) entre les deux parties du développement.

Un autre point très positif est qu'on peut également remarquer que, bien qu'étant doté d'une syntaxe cohérente, comme on va le voir juste après, le langage de templates Django peut être utilisé avec n'importe quel format de fichier texte. En effet, les templates Django ne nécessitent pas de structuration particulière du fichier dans lequel ils sont utilisés. C'est un point fort par rapport aux langages de templates qui s'appuient par exemple sur une syntaxe spécifique, telle qu'un langage de balises comme XML ou HTML (par exemple Zope TAL). Ainsi, le langage de templates Django est utilisable non seulement pour produire du XML ou du HTML, mais également pour produire des fichiers JavaScript ou CSS et bien évidemment du texte brut. En fait les templates Django permettent de produire pratiquement tout type de fichier dont le format est basé sur du texte, par exemple du CSV (*Comma-Separated Values*).

Enfin, pour les développeurs qui seraient réfractaires à ce système ou qui auraient déjà une pratique et une habitude d'un autre langage de templates, Django permet dans une certaine mesure de compléter le système intégré de templates Django par un autre, comme cela sera expliqué plus tard. Les systèmes alternatifs que l'on peut installer sont Jinja2, HAML, OpenDoc et Mako. Cette possibilité permet donc aux développeurs de garder leurs habitudes de travail et de capitaliser sur des librairies de templates existantes, que l'on va pouvoir réutiliser avec quelques adaptations à la marge.

2. Le principe et la syntaxe du langage de templates de Django

Un fichier de template Django est, comme vu précédemment, tout simplement un fichier texte dans lequel sont installées des balises spécifiques, permettant de générer à peu près n'importe quel format basé sur du texte (HTML, XML, CSV, etc.).

Un fichier de template Django contient des commentaires, des **variables** (variables qui seront remplacées à la volée par la valeur de la variable lors de l'évaluation du template) et des **tags** (instructions, mots-clefs ou étiquettes) qui contrôlent la logique du flux d'affichage du fichier template.

Les instructions, variables et commentaires sont délimités par trois balises : **{{ .. }}** pour les variables, **{% ... %}** pour les tags ou instructions et enfin **{# ... #}** pour les commentaires sur une même ligne.

Par la suite nous continuerons à utiliser le mot **tag** pour rester en cohérence avec la documentation Django. Rappelons qu'il s'agit d'une instruction, d'un mot-clef. Les **tags** proposés par le langage de templates Django fonctionnent de manière similaire aux instructions d'un langage de programmation. Ainsi, le tag **if** permet de traiter des expressions booléennes, le tag **for** permet de construire des boucles. Attention, ces différents tags ne sont pas exécutés de manière totalement similaire aux expressions Python correspondantes. De base, seuls les tags et les filtres décrits ci-après sont supportés. L'utilisateur pourra en ajouter, soit en installant des add-ons ou des applications complémentaires, soit en développant ses propres tags et filtres.

Du point de vue technique, les variables à destination du template sont stockées dans un dictionnaire Python appelé `context` (contexte), ce dictionnaire est passé en paramètre des fonctions de rendu des templates par le programmeur.

– Les clefs du dictionnaire correspondent aux noms des variables qui seront accessibles lors de l'exécution du rendu du template.
– La valeur associée à la clef est l'objet Python correspondant. Le contexte global de l'évaluation d'un template est composé du dictionnaire passé en paramètre par l'utilisateur lors de l'appel au moteur de rendu.

Ce contexte utilisateur est complété par des variables globales si et seulement si on fait appel à `RequestContext(request,context)`, ce qui est le cas par défaut pour la fonction `render`. Chaque « template context processor » configuré dans `settings.py` ajoute alors ses propres variables « globales ».

Exemple :

```
mon_contexte = {
'liste_des_salles' :Salles.objects.all(),'ma_variable ' :2}
 result  = render('mon_template.html',mon_contexte,)
```

© Editions ENI - All rights reserved