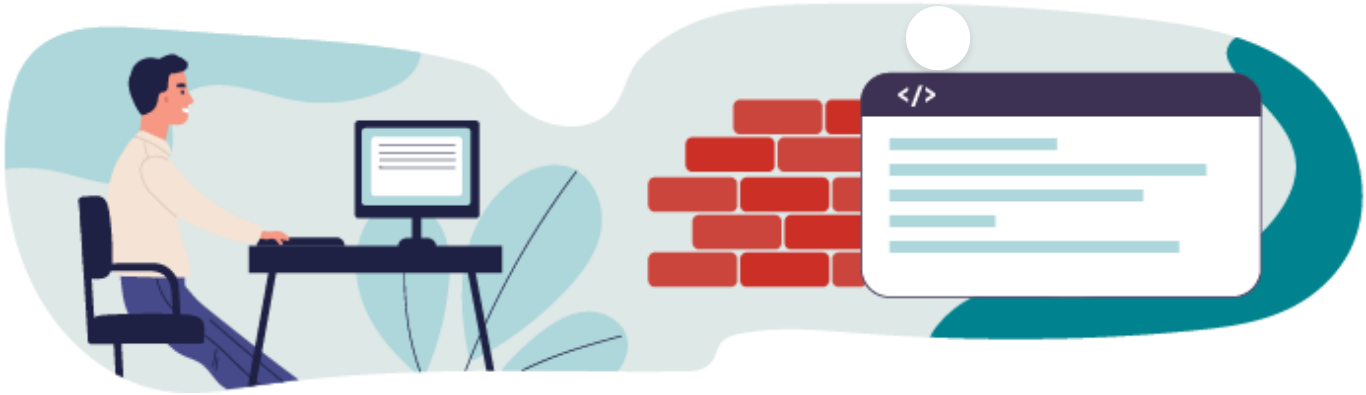
Débutez avec le framework Django

12 heures Moyenne

Mis à jour le 28/06/2023

Ajoutez structure et style à votre site grâce à un gabarit de base, du CSS et des fichiers statiques



Séparez le HTML valable pour l'ensemble du site du HTML spécifique à la page

Avez-vous remarqué à la fin du dernier chapitre que nous avons maintenant du code répété dans nos fichiers de gabarits ? Cela deviendra ingérable à mesure que nous ajouterons des gabarits à notre site : si nous voulons modifier la balise head, nous devrons la modifier dans chaque gabarit.

Voyons comment nous pouvons appliquer le principe DRY : Don't Repeat Yourself (Ne vous répétez pas), et éviter la répétition du code dans nos gabarits en utilisant un modèle de base.

Pour commencer, nous devons examiner nos gabarits existants et nous poser deux questions :

- Quelles sont les différences entre ces gabarits ?
- Qu'est-ce qui ne change pas ?

Si nous regardons tous les gabarits côte à côte, nous pouvons voir que les éléments communs aux trois sont :

- les balises <html>
- les balises <head>
- les balises <title>
- les balises <body>

Comme ces balises sont communes à toutes les pages, nous allons les définir à un seul endroit : notre gabarit de base.

Plaçons ce code HTML pour l'ensemble du site dans un nouveau fichier gabarit à l'adresse « listings/templates/listings/base.html ». Notez le code que nous ajoutons à la ligne X :

```html
# listings/templates/listings/base.html

<html>
    <head><title>Merchex</title></head>
    <body>

        {% block content %}{% endblock %}

    </body>
</html>
```

Dans notre HTML, nous avons ajouté une balise de gabarit Django : la balise `block` . Et nous l'avons fermé avec une balise `endblock` .

Nous avons également donné un nom à ce bloc : nous l'avons appelé `content` .

Nous pouvons considérer la balise `block` comme un espace réservé, dans lequel nous pouvons injecter du contenu ; en ce sens, il est similaire aux variables de gabarits.

Ce que nous voulons faire ensuite, c'est injecter notre HTML spécifique à la page dans ce bloc. Revenons à nos gabarits de page pour voir comment on fait.

Enrichissez un gabarit de base avec un modèle de page

Ouvrons notre gabarit « hello.html ». Pour commencer, supprimons tout le HTML que nous définissons maintenant dans notre gabarit de base. (Je supprime également une partie du code d'exemple que j'ai ajouté dans le dernier chapitre.) On devrait aboutir à quelque chose comme ça :

```html
# listings/templates/listings/hello.html

<h1>Hello Django !</h1>

<p>Mes groupes préférés sont :</p>

<ul>
```

```
    {% for band in bands %}
        <li>
            {{ band.name }}
        </li>
    {% endfor %}
</ul>
```

Super, maintenant notre gabarit de page ne contient que le contenu spécifique à cette page.

Ensuite, ajoutons ces balises de gabarits supplémentaires en haut et en bas du gabarit :

html

```
{% extends 'listings/base.html' %}

{% block content %}

<h1>Hello Django !</h1>
...
</ul>

{% endblock %}
```

La balise de gabarits `extends` en haut indique à Django que nous voulons que ce gabarit **hérite** de notre gabarit de base.

Autour de notre contenu, nous avons une balise d'ouverture et de fermeture `block` et nous avons donné à la balise d'ouverture le même nom que dans notre gabarit de base : `content` .

Comme vous l'avez peut-être deviné, Django va prendre tout ce qui se trouve dans le bloc nommé `content` dans notre gabarit de page et l'injecter dans le bloc du même nom dans notre gabarit de base. Voyons maintenant comment cela fonctionne dans notre navigateur :

La page « hello ».

Votre page devrait apparaître exactement comme elle était avant. Nous n'avons pas modifié l'apparence, mais nous avons changé la façon dont les choses fonctionnent sous le capot. Cela rendra notre application plus facile à gérer au fur et à mesure de son développement.

> Nous avons vu deux nouvelles balises de gabarits dans ce chapitre : `extends` et `block` , qui nécessitent qu'on leur passe un argument.
>
> C'est un peu comme lorsque nous passons un argument à une fonction Python, comme `print('Hello World!')` . Les arguments des balises de gabarits n'ont pas besoin d'être mis entre parenthèses, ils doivent simplement être séparés par des espaces.
>
> Parfois, l'argument doit être une chaîne de caractères placée entre guillemets (par exemple : `{% extends "bands/base.html" %}`), et parfois non (par exemple : `{% block content %}`), veillez donc à utiliser la syntaxe appropriée pour la balise en question.
>
> Vous trouverez des exemples pour chaque balise dans la **documentation sur les balises de gabarits intégrés**.

Bien entendu, maintenant que nous avons défini notre gabarit de base, nous pouvons l'utiliser pour toutes les pages existantes (et futures) de notre site. À la fin du chapitre, vous appliquerez votre gabarit de base à toutes vos pages.

Visualisons ce que nous venons de faire avec les gabarits de base :

00:42

Si vous avez besoin de vérifier l'une de ces étapes, regardez le screencast pour un récapitulatif.

01:13

Donnez du style à votre site grâce aux CSS

La balise `<head>` de notre site étant définie dans notre gabarit de base, c'est le bon moment pour ajouter une feuille de style.

Les fichiers tels que les feuilles de style CSS sont appelés fichiers **statiques** dans Django, car une fois l'application en cours d'exécution, ils ne changent pas.

Nous plaçons les fichiers statiques à un endroit spécifique de notre application. Créez un dossier dans listings/static/listings/ et à l'intérieur, créez un nouveau fichier appelé `styles.css` .

> Remarquez la structure de notre dossier : nous faisons la même chose qu'avec les gabarits, c'est-à-dire que nous attribuons un nom à nos fichiers CSS. Lisez l'encadré "Espace de noms des fichiers statiques" dans la **documentation sur les fichiers statiques**.

text

```
* { background: red }
```

Il ne s'agit pas de la feuille de style finale que nous utiliserons réellement dans notre application, c'est plutôt le « Hello World ! » des feuilles de style. Cela va donner à tous les éléments de notre page un fond rouge. C'est juste pour tester que notre feuille de style se charge correctement !

Ouvrons « listings/templates/listings/base.html » et ajoutons une balise `<link>` , sous le titre, pour charger notre feuille de style :

html

```
<html>
    <head>
        <title>Merchex</title>
```

```
4         <link rel="stylesheet" href="{% static 'listings/styles.css' %}" />
```

Que remarquez-vous à propos de l'attribut `href` de notre balise `<link>` ?

Il s'agit en fait d'une autre balise de gabarits ! Cette balise `static` indique à Django qu'il doit regarder dans le répertoire spécial `static` que nous avons créé précédemment et essayer de trouver le fichier que nous avons créé : « bands/styles.css ».

Maintenant, pour que la balise `static` fonctionne, nous devons d'abord la « charger » dans ce modèle. Nous le faisons en ajoutant une balise `load` au tout début du fichier, comme ceci :

html

```
1 {% load static %}
2
3 <html>
4
5 ...
```

Jetons un coup d'œil au résultat dans le navigateur :

La page « hello ».

Vous devriez voir une page rouge disgracieuse. Cela signifie que tout fonctionne !

> Si vous ne voyez pas de fond rouge, essayez d'arrêter et de redémarrer le serveur de développement. Cela donnera à Django une chance de trouver votre nouveau répertoire statique. Vous pouvez aussi essayer de faire un **rafraîchissement forcé** dans votre navigateur.

Une fois que vous êtes sûr que votre fichier CSS se charge correctement, vous pouvez le modifier et commencer à donner à votre site le style que vous souhaitez. Ce cours ne couvre pas le CSS, mais vous

pouvez soit suivre le cours Apprenez à créer votre site web avec HTML5 et CSS3, soit utiliser vos compétences existantes pour donner du style à votre site.

Mais si vous le souhaitez, donner du style à votre site n'est pas une obligation. En effet, votre navigateur appliquera un style par défaut à la plupart des éléments HTML que nous utiliserons dans ce cours. Si cela vous convient, vous pouvez simplement utiliser un fichier CSS vierge. C'est ce que je vais faire pour le reste de ce cours.

Vous pouvez suivre à nouveau les concepts de cette partie dans ce screencast :

01:07

C'est à vous ! Faites en sorte que tous vos gabarits de page enrichissent votre gabarit de base

Bien entendu, le gabarit de base que nous avons créé ici peut également être utilisé pour nos autres pages, et c'est ce que je voudrais que vous fassiez maintenant.

Mettez à jour chacun de vos gabarits de page, afin qu'ils héritent tous de votre gabarit de base.

Si vous êtes bloqué, rappelez-vous que tous vos gabarits de page doivent posséder :

- une balise `extends` ;
- une balise `block` ;
- une balise `endblock` .

Récapitulons l’architecture MVT

Nous avons abordé les trois principaux composants de l'architecture MVT : le M, le V et le T, et nous avons vu comment la séparation de notre application en ces trois domaines nous aide à assurer la « séparation des préoccupations » en découplant les différentes parties de notre application et en appliquant le principe de responsabilité unique.

Chaque élément de MVT a sa place : les modèles, les vues, les gabarits et même les modèles d'URL sont tous séparés dans leurs propres fichiers.

L'un des avantages d'avoir des gabarits dans leurs propres fichiers est qu'un développeur front-end peut travailler sur les fichiers de gabarits, tandis qu'un développeur back-end peut travailler simultanément sur les fichiers de modèles et de vues. Cela réduit (mais n'élimine pas complètement) la probabilité que les modifications apportées par les deux développeurs entrent en conflit les unes avec les autres.

> MVT/Django, est-ce toujours le bon choix ?

Lorsque j'ai découvert Django, j'ai pensé que je n'aurais plus jamais besoin d’apprendre un autre framework ! Il est tentant de considérer Django comme une « solution miracle » qui résout tous vos problèmes. Et cela résout effectivement beaucoup de problèmes ! Mais ce n'est pas toujours le meilleur choix.

Tout d'abord, Django est un framework monolithique. Il fournit des moteurs pour toutes les parties de l'application : le moteur de gabarits pour la présentation, l'ORM pour la persistance, etc. Et si vous vouliez changer une de ces parties ? Il est peut-être préférable de construire votre architecture MVC/T à partir des éléments de votre choix, comme Flask, SQLAlchemy, etc.

Deuxièmement, MVC/T n'est qu'un style d'architecture parmi d’autres. Il est très adapté aux applications CRUD simples. Mais pour les solutions d'entreprise, vous pouvez étudier des alternatives comme Clean Architecture.

Pour l'instant, comprenez simplement que MVC a ses limites, et vous pourriez un jour constater que vous atteignez l'une de ces limites !

Comprenez le rendu côté serveur et côté client

Le rendu côté serveur est la manière « ancienne » de générer le contenu HTML d'une page web, où chaque chargement de page implique un aller-retour relativement lent vers le serveur. Aujourd'hui, il existe de nombreux frameworks permettant de créer des applications front-end riches dans le navigateur, où le HTML est généré côté client et où seule une quantité minimale de données est envoyée entre le navigateur et le serveur, ce qui donne des applications rapides comme l'éclair.

Mais cela ne signifie pas que vous devez ignorer le rendu côté serveur ! C'est un bon point de départ. Et pour de nombreuses applications, le rendu côté serveur est une solution adéquate. Elle est simple à comprendre. Elle réduit la complexité de l'application car elle ne nécessite pas de construire une API REST. Et votre application peut tenir dans un seul repository, car il n'est pas nécessaire d'avoir une application frontale JavaScript distincte. C'est pourquoi le rendu côté serveur avec Django est idéal pour les applications de démonstration et les applications à l'interface simple.

Si les besoins de votre application évoluent par la suite, vous pouvez convertir votre projet Django et son rendu côté serveur en une API REST à l'aide du cadre Django REST.

En résumé

- Un gabarit de base nous permet de définir le code HTML de l'ensemble du site une seule fois, à un seul endroit, afin de conserver notre code HTML DRY.
- Nos gabarits de page ne contiennent alors plus que le code HTML spécifique à la page.
- Nous définissons des balises de gabarits `block` dans notre gabarit de base. Elles servent d'espaces réservés dans lesquels nous `injectons` le contenu de nos gabarits de page.
- Notre gabarit de base est l'endroit idéal pour charger un fichier CSS, en utilisant la balise de gabarits `static` .

Nous avons couvert tous les éléments fondamentaux de l'architecture MVT ! Maintenant vous allez répondre à un quiz pour vérifier votre compréhension. Je vous retrouve dans la prochaine partie du cours !

Indiquer que ce chapitre n'est pas terminé

Séparez la logique de l'application de la présentation avec un gabarit Django

Quiz : Posez les bases d'une application Django à l'aide de modèles, de vues et de gabarits

Les professeurs

Patrick Wampé

Développeur full stack et Data Scientist. Formateur dans plusieurs écoles d'informatique, il a également écrit un livre sur l'IA.

Patrick Heneghan

Software engineer in the UK, coding mostly in Python on backend systems.

Rafiq Hilali

British Software Engineer and Django expert with Lambert Labs. Currently based in BC, Canada.


Télécharger dans
l'App Store