# Développement web côté serveur

# Framework python-django

**Travaux pratiques N°6**

**(Projet mysite : suite TP N°5)**

**Navigation et Accès à la base de données**

**Objectifs du TP :**

* Passer des données à partir du Model vers le Template
* Créer un menu dynamique

Lors du dernier TP, nous avons créé notre Template de site et un Template de l'application page qui hérite du Template de site. Notre projet commence enfin à ressembler à un vrai site Web, mais il reste statique : Le Template de l'application page n'affiche pas le contenu du champ "bodytext" du Model Page stocké dans la base de données du site et nous n'avons toujours pas de navigation sur le site pour pouvoir charger les différentes pages du site.

Dans ce TP, nous allons modifier le Template de page pour afficher le contenu de la page sélectionnée de manière dynamique et transformer le texte de substitution dans le menu de gauche en liens de navigation vers les pages de notre site.

Pour atteindre ce résultat, nous allons accomplir quatre tâches:

1. Modifier le fichier "urls.py" de l’application pages pour capturer un lien de page;

2. Réécrire notre View "index\_pages" pour sélectionner la page correcte et renvoyer le contenu au Template;

3. Modifier le Template page.html de l'application pages pour afficher le contenu dynamique envoyé par la View "index\_pages" ;

4. Transformez la liste des espaces réservés pour le menu dans la barre latérale gauche en un menu de navigation réel.

**1-Modification du fichier "urls.py" de l'application pages**

Dans le TP précédent, nous avons configuré notre fichier "urls.py" de l'application pages pour charger la View "index\_pages" lorsque nous accédons à l'URL racine (http://127.0.0.1:8000/pages/).

Si nous voulons afficher une autre page, nous devons donner davantage d'informations à Django.

La pratique courante consiste à ajouter un lien dans l'URL vers le contenu que nous souhaitons afficher, par exemple, /services/ est un lien vers notre page "Services" stockée dans la base de données. Dans le TP3, lors de la création du modèle Page, nous avons ajouté un champ appelé permalink qui contient une chaîne de texte que notre application Django utilisera pour faire correspondre notre URL à la page correcte.

Pour que Django sache quelle page nous demandons, nous devons pouvoir extraire le lien de la page à partir de l'URL demandée. Nous faisons cela en modifiant notre fichier urls.py :

|  |
| --- |
| # pages\urls.py  **1** from django.urls import path  **2**  **3** from . import views  **4**  **5** urlpatterns = [  **6** # path('', views.index\_pages, name='index\_pages'),  **7 path('<str:pagename>', views.index, name='index\_pages'),**  **8 path('', views.index\_pages, {'pagename': ''}, name='index\_pages'),**  **9** ] |

|  |
| --- |
| E:\mydjango\mysite\pages\urls.py |
| *# pages\urls.py* **from** django.urls **import** path  **from** . **import** views  urlpatterns = [  path(**'<str:pagename>'**, views.index\_pages, name=**'index\_pages'**),  *path('', views.index\_pages,* {**'pagename'**: **''**}, *name='index\_pages'),* ] |

Nous avons commenté le pattern de l'URL d'origine (**ligne 6**) afin que vous puissiez voir ce qui a changé. Dans la **ligne 8**, nous utilisons un ***groupe de capture***. Tout ce qui se trouve entre les chevrons sera capturé et envoyé à la vue en tant que paramètre (pagename). **<str:pagename>** signifie "capturez tout ce qui se trouve après **<nomdomaine>/pages/** et envoyez-le à la vue comme paramètre de type string sous le nom pagename". Par exemple, si l'URL pages/services / est passé à cette fonction **path**, la View **index\_pages** est appelée et la chaîne «services» est transmise à la vue en tant que paramètre **pagename**.

**str:** est un convertisseur de chemin qui convertira les données capturées en string. Django fournit les convertisseurs de chemin suivants qui sont disponibles par défaut :

* **str** - correspond à n’importe quelle chaîne non vide, à l’exception du séparateur de chemin, **'/'**. C’est ce qui est utilisé par défaut si aucun convertisseur n’est indiqué dans expression.
* **int** - correspond à zéro ou un autre nombre entier positif. Renvoie le type *int*.
* **slug** - correspond à toute chaîne composée de lettres ou chiffres ASCII, du trait d’union ou du caractère soulignement. Par exemple, **construire-votre-1er-site-django**.
* **uuid** - correspond à un identifiant UUID. Pour empêcher plusieurs URL de correspondre à une même page, les tirets doivent être inclus et les lettres doivent être en minuscules. Par exemple, **075194d3-6885-417e-a8a8-6c931e272f00**. Renvoie une instance [**UUID**](https://docs.python.org/3/library/uuid.html#uuid.UUID).
* **path** - correspond à n’importe quelle chaîne non vide, y compris le séparateur de chemin, **'/'**. Cela permet de correspondre à un chemin d’URL complet au lieu d’un seul segment de chemin d’URL comme avec **str**.

La valeur par défaut est une chaîne, donc **str:** n’est pas nécessaire du point de vue technique, mais il est préférable d’être explicite.

Puisque la fonction path () ne peut pas capturer une chaîne vide, nous devons créer un cas spécial pour la page d'accueil, c'est exactement ce que nous faisons dans la ligne 8. Lorsque l'utilisateur navigue jusqu'au **<nomdomaine/pages/** du site, la ligne 8 définit la valeur du paramètre **pagename** à la chaîne vide ('').

**2-Réecriture de la View index\_pages**

Une fois que nous avons modifié notre fichier **urls.py** répartiteur d'URL pour pouvoir capturer des informations à partir de l'URL, nous modifions ensuite la View **index\_pages**.

Avant de commencer à réécrire la View, voyons comment Django utilise les modèles pour extraire des informations de la base de données. Pour cet exercice, nous allons utiliser le shell interactif Django.

En activant votre environnement virtuel, exécutez la commande suivante:

|  |
| --- |
| **(venv2019) C:\Users\hassouni\Django2019\mysite2019>python manage.py shell**  **Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32**  **Type "help", "copyright", "credits" or "license" for more information.**  **(InteractiveConsole)**  **>>>** |

Depuis le shell interactif de Django, entrez le code suivant:

|  |
| --- |
| 1. >>> **from pages.models import Page** 2. >>> **pg = Page.objects.get(permalink='/')** 3. >>> **pg.title** 4. 'LP GLAASRI' 5. >>> **pg.update\_date** 6. datetime.datetime(2014, 1, 9, 8, 0, tzinfo=<UTC>) 7. >>> **pg.bodytext** 8. '<h1>Licence Professionnelle en Génie Logiciel et Administration Avancée des Systèmes et Réseaux Informatiques</h1>\r\n<h2>Objectifs de la formation</h2>\r   \n<h2>Durée de la formation</h2>\r\n<h2>Public cible</h2>\r\n<h2>Contenu de la formation</h2>' |

Examinons ce code de plus près:

* **Ligne 1**. Nous importons le Model **Page** à partir de notre application **pages**.
* **Ligne 2**. Nous récupérons une seule page de la base de données et la stockons dans l'objet **pg**. Dans cet exemple, nous récupérons la page d'accueil.
* Maintenant que nous avons un objet instance du Model Page , nous pouvons accéder à ses attributs:
  + **Ligne 3**. Affichage de l'attribut **title**
  + **Line 5**. Affichage de l'attribut **update\_date**
  + **Ligne 7**. Affichage de l'attribut **content**

Nous pouvons maintenant utiliser ce que nous avons appris dans le shell interactif pour créer une nouvelle vue:

|  |
| --- |
| # pages\views.py  **1** from django.shortcuts import render  **2**  **3 from . models import Page**  **4**  **5 def index\_pages(request, pagename):**  **6 pagename = '/' + pagename**  **7 pg = Page.objects.get(permalink=pagename)**  **8 context = {**  **9 'title': pg.title,**  **10 'content': pg.bodytext,**  **11 'last\_updated': pg.update\_date,**  **12 }**  **13 # assert False**  **14 return render(request, 'pages/page.html', context)** |

À l’exception de la première ligne, il s’agit d’un tout nouveau code, examinons-le en détail:

* **Line 3**. Importe le Model **Page** dans le fichier views.py.
* **Line 5**. Ajoute le paramètre **pagename** à la définition de la View **index\_pages**. La chaîne capturée par le répartiteur d'URL sera affectée au paramètre **pagename** lors du chargement de la View.
* **Line 6**. Django supprime la barre oblique située au début des URL. Nous devons donc ajouter une barre oblique au nom de la page, sinon les liens URL de notre Model seront relatifs à la page en cours et non à la racine.
* **Line 7**. C'est le même code que nous avons utilisé pour charger une page lorsque nous explorions le modèle dans le shell interactif. L'objet **pg** contiendra la page demandée par l'URL.
* **Lines 8 to 12**. Nous utilisons notre objet **pg** pour renseigner un dictionnaire à transmettre au Template. Dans Django, ce dictionnaire s'appelle le contexte. Les variables de contexte (clés du dictionnaire) seront utilisées par le Template pour rendre le contenu dynamique au navigateur.
* **Line 13**. Utilisé pour tester la vue. Nous donnerons plus de détail ultérieurement.
* **Line 14**. ***render ()*** requiert un objet request, le nom du Template et un paramètre *context* (dictionnaire) à transmettre au Template. Django compilera ensuite la page Web à partir des informations fournies (Template + context) et renverra une page HTML entière au navigateur (HttpResponse).

|  |
| --- |
| E:\mydjango\mysite\pages\views.py |
| *# pages\views.py* **from** django.shortcuts **import** render  **from** . models **import** Page  **def** index\_pages(request, pagename):  pagename = **'/'** + pagename  pg = Page.objects.get(permalink=pagename)  context = {  **'title'**: pg.title,  **'content'**: pg.bodytext,  **'last\_updated'**: pg.update\_date,  }  *# assert False* **return** render(request, **'pages/page.html'**, context) |

**Test de la View**

Un moyen simple mais très puissant de tester pour vous assurer que la View transmet les informations correctes au Template consiste à utiliser la page d'erreur de Django pour examiner le résultat de la View. La page d’erreur de Django peut être déclenchée en insérant ***assert False*** dans votre code. Décommentez la ligne 13 dans votre code, exécutez le serveur de développement et accédez à la racine de votre site Web.

Lorsque la page d’erreur de Django s’affiche, faites défiler jusqu’à la fin des informations de la trace.

A screenshot of a computer

Description automatically generated

Si vous cliquez sur ***Local vars au derssous de assert False***, le cadre se développe pour afficher les informations que vous transmettez au Template. Dans la figure ci-dessous, le dictionnaire de contexte contient les variables de page que vous transmettez à la View.

A screenshot of a computer

Description automatically generated

Cette capacité à regarder à l'intérieur de vos Views est extrêmement utile. Si vous prenez l’habitude d’utiliser la View d’erreur de Django de cette manière, vous pouvez réduire considérablement votre temps de débogage lors du développement de sites Web plus avancés.

Maintenant que vous savez comment utiliser la page d’erreur, il est temps de modifier les Templates (n'oubliez pas de commenter ou de supprimer la ligne 13 de votre view "index\_pages" avant de poursuivre!).

**3-Modification des Templates**

Nous allons procéder étape par étape afin que vous puissiez voir le processus divisé en des tâches plus simples. Tout d'abord, nous modifions le Template "page.html" de l'application pages pour afficher le contenu de la page (valeur de l'attribut content) stocké dans la base de données:

|  |
| --- |
| # pages\templates\pages\page.html  **1** {% extends "base.html" %}  **2**  **3** {% block content %}  **4 {{ content }}**  **5** {% endblock content %} |

|  |
| --- |
| E:\mydjango\mysite\pages\templates\pages\page.html |
| {% **extends "base.html"** %}  {% **block content** %}  {{ **content** }} {% **endblock content** %} |

Nous n'avons modifié qu'une ligne dans ce Template. Nous avons remplacé le texte de substitution de la ligne 4 par une variable de Template ***({{contenu}}).*** Cette variable contiendra le contenu de la page extrait de la base de données au moment de l'exécution.

Enregistrez le Template, puis lancez le serveur de développement. Lorsque vous chargez la page d'accueil, vous remarquerez que quelque chose ne va pas: votre code HTML bien formaté s'affiche dans un grand bloc laid! Les balise HTML n'ont pas été interprétées.(Voir figure ci-dessous).

A screenshot of a computer

Description automatically generated

En effet, par défaut, Django ***échappe automatiquement à tout code HTML avant qu'il ne soit rendu dans un Template***. C’est un exemple de ce que Django fait automatiquement pour protéger votre site Web contre les dommages malveillants de tout attaquant qui tente d’injecter du code exécutable sur votre site Web.

Pour empêcher Django d'échapper automatiquement au code HTML, utilisez la balise ***autoescape***:

|  |
| --- |
| # pages\templates\pages\page.html  **1** {% extends "base.html" %}  **2**  **3** {% block content %}  **4 {% autoescape off %}**  **5** {{ content }}  **6 {% endautoescape %}**  **7** {% endblock content %} |

Aux lignes 4 et 6, nous avons entouré la variable de contenu avec un bloc ***autoescape*** pour que Django n’échappe pas au code HTML qui se trouve dans la page. Si vous rechargez la page, il devrait maintenant avoir une meilleure allure! (Figure ci-dessous).

A screenshot of a computer

Description automatically generated

Votre vue devrait maintenant fonctionner et pouvoir afficher les pages de votre site. Testez-le maintenant en essayant de naviguer vers ***http://127.0.0.1:8000/pages/services*** et ***http://127.0.0.1:8000/pages/about***.

|  |
| --- |
| **http://127.0.0.1:8000/pages/services** |
|  |
| [**http://127.0.0.1:8000/pages/about**](http://127.0.0.1:8000/pages/about) |
|  |

**Amélioration des Templates**

Notre site Web a une belle apparence, mais il reste quelques choses à faire pour le site et les Templates de page avant de créer la navigation:

1. Définissez le titre de la page. et

2. Ajoutez une date de dernière mise à jour (last updated) au bas du contenu de la page.

**Title de la page**

Pour mettre à jour le titre de la page pour chaque page, nous devons d’abord ajouter un ensemble de balises de bloc au Template de site base.html:

|  |
| --- |
| # mysite2019\templates\base.html  **1** <!doctype html>  **2** <html>  **3** <head>  **4** <meta charset="utf-8">  **5** <title>  **6 {% block title %}**  **7** Untitled Document  **8 {% endblock title %}**  **9** </title> |

|  |
| --- |
| E:\mydjango\mysite\mysite\templates\base.html |
| {% **load static** %} <!Doctype **html**> <**html**>   <**head**>  <**meta charset="utf-8"**>  <**title**>  {% **block title** %}  Untitled Document  {% **endblock title** %}  </**title**>  <**link href="**{% **static 'main.css'** %}**" rel="stylesheet" type="text/css"**>  </**head**>  ………. |

Cela est simple: A la ligne 6 nous avons introduit une balise d'ouverture de bloc **{% block title %} nommée title** et à la ligne 8, une balise de fermeture du même bloc {% **endblock title** %}.

Nous devons maintenant remplacer la nouvelle balise de bloc dans notre Template de page.

|  |
| --- |
| # pages\templates\pages\page.html  **1** {% extends "base.html" %}  **2**  **3 {% block title %}{{ title }}{% endblock title %}**  **4**  **5** {% block content %}  **6**  **7** {% autoescape off %}  **8** {{ content }}  **9** {% endautoescape %}  **10**  **11 <p>**  **12 Page last updated: {{ last\_updated|date:'D d F Y' }}**  **13 </p>**  **14**  **15** {% endblock content %} |

|  |
| --- |
| E:\mydjango\mysite\pages\templates\pages\page.html |
| {% **extends "base.html"** %}  {% **block title** %}  {{ **title** }} {% **endblock title** %}  {% **block content** %}  {% **autoescape off** %}  {{ **content** }}  {% **endautoescape** %}   <**p**>  Page last updated: {{ **last\_updated**|**date**:**'D d F Y'** }}  </**p**>  {% **endblock content** %} |

La substitution se produit à la ligne 3 - nous remplaçons le texte de substitution dans le Template de site (Untitled Document ) par la variable {{title}} qui, à l'exécution, contiendra le titre de notre page.

Nous avons également ajouté trois nouvelles lignes à ce fichier (lignes 11 à 13). Vous devriez reconnaître cela comme une variable avec du code supplémentaire. Ceci est un exemple d'application d'un filtre à une balise de Template. Dans cet exemple, nous utilisons le filtre de date de Django pour formater la date **last\_updated** obtenue à partir de notre base de données.

Le langage de Templates de Django comporte de nombreux filtres qui vont du formatage des Strings à la réalisation d’opérations logiques et mathématiques mineures.

Nous utilisons une chaîne de format qui produit une date longue, par exemple. “Vendredi 4 Mars 2022”. Il existe de nombreuses chaînes de format différentes pour les dates; une référence pratique est la documentation officielle de Django.

Enregistrez les fichiers, lancer le serveur de développement et votre navigateur devrait afficher la page d’accueil complétée, avec la valeur du champ title dans la barre du titre du Navigateur et la valeur de champ **last\_updated** bien présentées au bas de la page (Figure ci-dessous).

|  |
| --- |
| ***127.0.0.1:8000/pages/*** |
|  |
| ***http://127.0.0.1:8000/pages/services*** |
|  |
| ***http://127.0.0.1:8000/pages/about*** |
|  |

**Création d'un Menu**

Maintenant que nous pouvons afficher le contenu de notre page de manière dynamique, nous avons besoin d’un moyen de naviguer sur notre site Web.

Les menus peuvent être complexes dans un site Web moderne (animation et présentations souples en fonction de la taille de l'écran). Cependant, à la base, il s’agit simplement d’une liste de liens vers les différentes pages du site.

Nous allons implémenter un menu dans la barre latérale gauche de notre site, avec la valeur du champ title lié à la valeur du champ content (via le lien qui est la valeur du champ permalink). Nous voulons une entrée de menu pour chaque page. Nous allons donc créer une liste HTML des pages de notre site, avec la valeur du champ title comme texte d'ancrage et la valeur du champ permalink comme adresse URL.

Avant de faire cela, utilisons le shell interactif pour voir comment nous récupérons un ensemble d’objets (QuerySet) à partir de la base de données.

Quittez le serveur de développement et exécutez le shell interactif Django

|  |
| --- |
| (venv) E:\mydjango\mysite>python manage.py shell  Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32  Type "help", "copyright", "credits" or "license" for more information.  (InteractiveConsole) |

Entrer le code ci-dessous et examinons la sortie.

|  |
| --- |
| **1-**>>> from pages.models import Page  **2-**>>> page\_list = Page.objects.all()  **3-**>>> for page in page\_list:  **4-**... print(page.permalink, page.title)  **5-**...  **6-**/ LP GLAASRI  **7-**/about About us  **8-**/services Services |

* **Ligne 1**. Importe le modèle Page à partir du module models de l'application pages afin de travailler avec les objets instances de la classe Page qui correspondent aux lignes de la table pages\_page de la base de données du site.
* **Ligne 2**. L'instruction **Page.objects.all()** permet d'obtenir toutes les lignes de la table qui correspond au Model Page (pages\_page) et les transforme en objets instances de la classe Page pour les stocker dans la variable page\_list.
* **Lignes 3 to 5**. Nous utilisons une boucle Python ***for*** pour parcourir la liste des objets page\_list et imprimer les valeurs des attributs permalink et title de chaque objet dans la liste.
* **Lignes 6 à 8**. La sortie de l'exécution du code qui correspond aux contenus des champs permalink et title de chaque ligne dans la table associée au Model Page.

Revenons maintenant à notre projet, et utilisons ce que nous avons appris pour ajouter page\_list à notre dictionnaire context que la View transmet au template:

|  |
| --- |
| # \pages\views.py  # ...  **1** context = {  **2** 'title': pg.title,  **3** 'content': pg.bodytext,  **4** 'last\_updated': pg.update\_date,  **5 'page\_list': Page.objects.all(),**  **6** }  # ... |

Notre View index devient donc:

|  |
| --- |
| E:\mydjango\mysite\pages\views.py |
| *# pages\views.py* **from** django.shortcuts **import** render  **from** . models **import** Page  **def** index\_page(request, pagename):  pagename = **'/'** + pagename  pg = Page.objects.get(permalink=pagename)  context = {  **'title'**: pg.title,  **'content'**: pg.bodytext,  **'last\_updated'**: pg.update\_date,  **'page\_list'**: Page.objects.all(),  }  *#assert False* **return** render(request, **'pages/page.html'**, context) |

Nous devons maintenant modifier les Templates pour afficher le nouveau menu.

D'abord le Template de base:

|  |
| --- |
| E:\mydjango\mysite\mysite\templates\base.html |
| # ...  **1** <nav id="nav">  **2** <ul>  **3 {% block sidenav %}**  **4** <li>Menu 1</li><li>Menu 2</li><li>Menu 3</li>  **5 {% endblock sidenav %}**  **6** </ul>  **7** </nav>  # ... |

Ci-dessus une partie du fichier Template **base.html**. Aux lignes 3 et 5, nous avons ajouté une balise d'ouverture et fermeture de bloc que nous avons nommée «**sidenav**».

Maintenant, nous devons remplacer la nouvelle balise de bloc **sidenav** dans notre Template de page:

|  |
| --- |
| E:\mydjango\mysite\pages\templates\pages\page.html |
| # ... {% endblock title %}  **1 {% block sidenav %}**  **2 {% for page in page\_list %}**  **3 <li>**  **4 <a href="/pages{{ page.permalink }}">{{ page.title }}</a>**  **5 </li>**  **6 {% endfor %}**  **7 {% endblock sidenav %}**  {% block content %}  # ... |

Là encore, il ne s’agit que d’une vue partielle du Template page.html. Notez que l’emplacement du bloc dans le fichier importe peu, tant que vous ne le placez pas dans un autre bloc. Nous l'avons placé entre les blocs <<title>> et <<content>>, mais vous pouvez tout aussi le placer à la fin du fichier et cela fonctionne toujours.

Voyons ce que fait ce nouveau code:

* **Ligne 2**. C'est la balise template de Django pour une boucle for. Bien que la syntaxe soit différente, elle fonctionne exactement de la même manière que Python.
* **Ligne 4**. Le code pour notre élément de menu. Il s’agit d’une ancre HTML standard avec comme valeur de l'attribut href la valeur du champ **permalink** de l'objet Page et comme texte de l’ancre la valeur du champ **title**.
* **Lines 3 and 5**. Formate l'ancre en tant qu'élément de liste HTML. N'oubliez pas que les balises <ul> </ ul> sont fournies par le Template base.html. Il n'est donc pas nécessaire de les ajouter ici.
* **Line 6**. Toutes les balises de Template Django nécessitent une balise de fermeture. La balise for ne fait pas exception.

Et c’est tout. Si tout s’est passé comme prévu, lorsque vous redémarrez le serveur de développement, vous devriez avoir un site Web entièrement fonctionnel avec vos trois pages affichées correctement et un menu de gauche navigable (Figure ci-dessous).

|  |
| --- |
| http://127.0.0.1:8000/pages/ |
|  |
| Après clic sur le lien about (http://127.0.0.1:8000/pages/about) |
|  |
| Après clic sur le lien services (http://127.0.0.1:8000/pages/services) |
|  |

**Autres balises et filtres de Template**

Nous n'avons couvert qu'un très petit sous-ensemble des balises et filtres de Templates Django les plus courants dans les deux derniers TPs. Pour une référence complète, ainsi que des cas d'utilisation, voir le site Web de Django.

**Page Not Found! Ajout d'un 404**

Maintenant que les pages de notre site, la navigation et les Templates sont terminés, il reste encore une chose à faire: traiter le cas d'une page qui n’existe pas.

Si vous naviguez vers, par exemple, http://127.0.0.1:8000/pages/notapage, vous obtiendrez la page d'erreur de Django avec une erreur DoesNotExist. Cela n’est pas considéré comme une bonne chose avec les sites Web professionnels. Nous devons donc donner à l’utilisateur qui a entré une URL incorrecte un message d’erreur plus utile.

A screenshot of a computer

Description automatically generated

Comme pour la plupart des tâches de développement Web courantes, Django facilite l’élevage d’une 404 avec le raccourci get\_object\_or\_404.

La bonne façon de gérer une URL incorrecte consiste à utiliser un message d'erreur 404 (page introuvable). Comme pour la plupart des tâches de développement Web courantes, Django facilite l’élevage d’une 404 avec le raccourci **get\_object\_or\_404**. Modifions légèrement notre views.py pour mettre en œuvre ce raccourci (modifications en gras):

|  |
| --- |
| # pages\views.py  **1** from django.shortcuts import render, **get\_object\_or\_404**  **2**  **3** # ...  **4**  **5** def index(request, pagename):  **6** pagename = '/' + pagename  **7 pg = get\_object\_or\_404(Page, permalink=pagename)**  **8** context = {  **9** 'title': pg.title,  **10** 'content': pg.bodytext,  **11** 'last\_updated': pg.update\_date,  **12** 'page\_list': Page.objects.all(),  **13** }  **14** return render(request, 'pages/page.html', context) |

|  |
| --- |
|  |
| *# pages\views.py* **from** django.shortcuts **import** render, get\_object\_or\_404 **from** . models **import** Page  **def** index(request, pagename):  pagename = **'/'** + pagename  pg = get\_object\_or\_404(Page, permalink=pagename)  *#pg = Page.objects.get(permalink=pagename)* context = {  **'title'**: pg.title,  **'content'**: pg.bodytext,  **'last\_updated'**: pg.update\_date,  **'page\_list'**: Page.objects.all(),  }  *#assert False* **return** render(request, **'pages/page.html'**, context) |

Le secret est dans la ligne 7 - get\_object\_or\_404 exécutera **Page.objects.get (permalink = nom de la page)** et si aucun objet n'est renvoyé, une erreur **HTTP 404** sera renvoyée.

Si vous essayez à nouveau l'URL non valide, vous constaterez qu'au lieu d'une erreur **Does NotExist**, vous obtenez maintenant **Does NotExist**. Tout va bien jusqu'à présent, nous avons au moins le bon message d'erreur, mais pourquoi obtenons-nous toujours la page d'erreur Django?

A screenshot of a computer

Description automatically generated

En effet, Django n’affiche pas les messages d’erreur 404 lorsque DEBUG est à True. Faisons quelques modifications dans notre fichier settings.py:

|  |
| --- |
| # mysite\settings.py  DEBUG = **False**  ALLOWED\_HOSTS = [**'127.0.0.1'**] |



En plus de définir DEBUG sur False, nous devons également ajouter l’adresse de l’hôte local à ALLOWED\_HOSTS, sinon la sécurité de Django ne nous permettra pas d’accéder à la page. Si vous essayez maintenant d'accéder à l'URL non valide, vous obtiendrez la page d'erreur 404 par défaut (Figure ci-dessous).

A screenshot of a computer

Description automatically generated

Comme vous pouvez le constater, la page d'erreur 404 par défaut ne convient pas à un site Web professionnel: il n'y a pas de modèle de site, pas de navigation et seulement un message très élémentaire à l'intention de l'utilisateur. Pour restituer correctement la page d'erreur 404, nous devons créer un Template et enregistrer le Template dans notre dossier de Templates racine.

Pour que Django puisse le trouver, ce fichier doit s'appeler 404.html:

|  |
| --- |
| E:\mydjango\mysite\mysite\templates\404.html |
| {% extends "base.html" %}  {% block title %}Page Not Found{% endblock title %}  {% block sidenav %}  <li><a href="/">Home</a></li>  {% endblock sidenav %}  {% block content %}  <h1>oops!</h1>  <p>Apparemment vous savez pas où aller, voulez-vous réessayer </p>  <p>Thanks.</p>  {% endblock content %} |

Il n’ya rien de nouveau ici. Nous avons juste hérité du Template base.html et ajouter quelques éléments de navigation de base et un message à l’utilisateur.

Lorsque DEBUG est défini sur False, Django ne restitue plus vos médias statiques, les images de Template et le CSS ne seront donc pas affichés.

Pour tester le modèle 404, nous devons redémarrer le serveur de développement avec l'option --insecure. L'option --insecure indique à Django que vous êtes toujours en mode de développement et que vous devez vous servir des fichiers statiques.

Quittez le serveur de développement et relancez-le avec:

**python manage.py runserver --insecure**

Maintenant, lorsque vous essayez d'accéder à l'URL non valide, vous devriez obtenir un joli message 404, restitué avec le template 404.html de site (Figure ci-dessous).

A screenshot of a computer

Description automatically generated

La création de pages d'erreur personnalisées pour d'autres erreurs HTTP (par exemple, 500 Server Error) suit le même processus.

|  |
| --- |
| **N’oubliez pas de remettre DEBUG à True avant de passer au TP suivant.** |