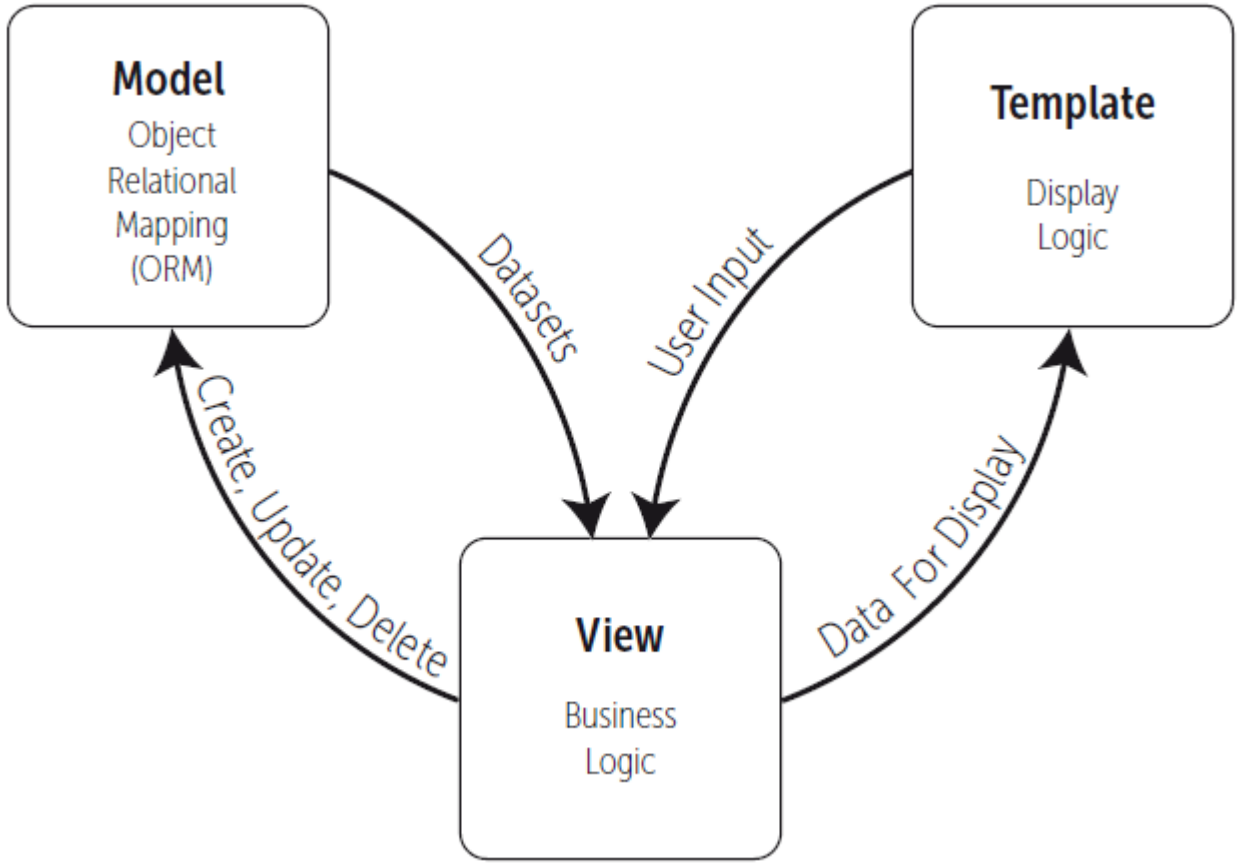
python™ django

RURAL
WEB
FACTORY (16)

by SIMPLON

RWF Saison 3
JL Renaudon 2019

Model – View - Template



Nous allons créer notre première vue:

Une vue permet d'afficher des éléments de la base de données, des pages statiques, mais aussi des formulaires dans un template.

Nous allons créer une vue qui nous permette d'afficher tous les bonzaïs présents dans notre base de données.

Notre vue doit donc créer une liste de tous les bonzaïs qui sera ensuite affichée par notre template.

Il existe de nombreuses méthodes différentes pour créer cette vue

Vue crée comme une fonction:

#views.py

```
1   from django.shortcuts import render
2
3   from .models import Bonzai
4
5   def index(request):
6       context = {'bonzai_list' : Bonzai.objects.all(),}
7       return render(request, 'bonzai/bonzai.html', context)
8
```

Quelques explications:

```
def index(request):
```

On définit une fonction nommée **index** à laquelle on passe l'argument **request**

```
context = {'bonzai_list' : Bonzai.objects.all(),}
```

On affecte un dictionnaire qui contient la liste de tous les bonzaïs à la variable **context**

```
return render(request, 'bonzai/bonzai.html', context)
```

Enfin avec la méthode **render** on passe le **context** au template bonzai.html

Cette méthode d'écriture fait partie de la famille des Function Based Views

La même vue écrite à partir d'une vue générique de django:

```python
from django.shortcuts import render
from django.views.generic import ListView

from .models import Bonzai

class BonzaiListView(ListView):
    template_name = 'bonzai/bonzai.html'
    def get_queryset(self):
        return Bonzai.objects.all()
```

Quelques explications:

Ici la vue est définie comme une classe et non plus comme une fonction:

```
6   class BonzaiListView(ListView):
```

Elle hérite de la classe **ListView** importée en ligne 2

```
7       template_name = 'bonzai/bonzai.html'
```

On lui indique le template auquel on veut envoyer la vue, **par défaut** django cherchera un template **bonzai_list.html**

```
8       def get_queryset(self):
9           return Bonzai.objects.all()
```

Ici on vient surcharger la méthode **get_queryset** pour qu'elle renvoie la liste de tous les bonzaïs, dans ce cas (renvoi de tous les objets) ce type de vue n'est pas très utile, en revanche nous verrons plus loin comment la modifier pour qu'elle serve dans différents cas. Elle renvoie une liste nommée « **nomdumodel »_list** par défaut.

Cette méthode fait partie de la famille des Class Based Views

Nous venons de voir 2 façons différentes de générer la même vue, il est à noter que dans les deux cas le template reste le même.

En revanche, l'appel de la vue dans le fichier urls.py de l'application ne se fait pas de la même façon dans les deux cas:

Cas de la fonction:

```
1   from django.urls import path
2   from . import views
3
4   urlpatterns = [
5       path('', views.index, name="accueil"),
6       ]
```

Quelques explications:

La fonction **path()** reçoit quatre paramètres, dont deux sont obligatoires: **route** et **view**, et deux autres facultatifs **kwargs** et **name**.

Dans notre cas:

route = ‘’ ce qui signifie que cette vue sera appelée quand l’url sera:

http//<chemindusite>/APP/ /APP/ car c’est ce que nous avons défini dans urls.py du projet

view = views.index l’ordre qui est donné à django est le suivant:

utilises la fonction **index** définie dans le fichiers **views**

kwargs n’est pas utilisé

name est le nom qui est donné à cette url et permet de l’appeler avec la méthode **reverse**

Cas de la classe:

```
1   from django.urls import path
2   from . import views
3
4   urlpatterns = [
5       path('', views.BonzaiListView.as_view(), name= 'bonzai_par_type')
6       ]
7
```

Quelques explications:

Dans ce second cas seule la façon d'appeler la vue change:

views = views.BonzaiListView.as_view()

On appelle la méthode as_view() de la classe BonzaiListView qui en a hérité de la classe ListView

C'est ici que réside un des nombreux avantages de la programmation objet, on utilise une méthode héritée sans s'occuper de la façon dont elle est construite et sans avoir à la redéfinir.

Nous avons défini notre vue de l'ensemble de nos bonzaïs. Essayons maintenant d'afficher une vue de détail pour nos bonzaïs:

Comme pour la liste nous allons écrire les 2 différents types de vues:

Sous forme d'une fonction:

```python
from django.shortcuts import render, get_object_or_404
from django.views.generic import ListView
from django.views.generic.detail import DetailView

from .models import Bonzai

def BonzaiDetail(request, bonzai_id):
    bonzai = get_object_or_404(Bonzai, pk=bonzai_id)
    return render(request, 'bonzai/detail_bonzai.html', {'object':bonzai})
```

Quelques explications:

Nous importons 1 nouvelle méthode:

get_object_or_404: méthode qui renvoie une page 404 si la requête **get** échoue

```
7   def BonzaiDetail(request, bonzai_id):
```

Cette fois-ci nous ajoutons le paramètre **bonzai_id** à la fonction car nous voulons afficher le détail d'un bonzaï en particulier

```
8       bonzai = get_object_or_404(Bonzai, pk=bonzai_id)
```

Nous affectons à la variable **bonzai** le résultat de la requête **get(Bonzai, pk=bonzai_id).**

Dans la table **Bonzai** nous récupérons l'objet bonzai dont la clé primaire(**pk**) est égale à **bonzai_id.**

```
9       return render(request, 'bonzai/detail_bonzai.html', {'object':bonzai})
```

Puis nous renvoyons au template le contexte {xxx}

Sous forme d'une classe:

```python
from django.shortcuts import render, get_object_or_404
from django.views.generic import ListView
from django.views.generic.detail import DetailView

from .models import Bonzai

class BonzaiDetailView(DetailView):
    model = Bonzai
    template_name ='bonzai/detail_bonzai.html'

```

Quelques explications:

Nous importons une nouvelle classe **DetailView** qui permet d'afficher le détail d'un objet de la base de données

```
7   class BonzaiDetailView(DetailView):
```

Nous créons donc une classe qui hérite de la classe **DetailView**

```
8       model = Bonzai
9       template_name ='bonzai/detail_bonzai.html'
```

Ensuite simplement nous indiquons le modèle qui doit être utilisé dans la vue ainsi que le template qui affichera la vue. Par défaut la vue renvoie un objet nommé **object** qui comporte toutes les caratéristiques de l'objet bonzai.

Comme pour l'affichage de la liste des bonzaïs, il nous faut définir l'url correspondante:

```
1   from django.urls import path
2   from . import views
3
4   urlpatterns = [
5           path('', views.index, name="accueil"),
6           path('bonzai/<int:bonzai_id>', views.BonzaiDetail, name = 'detail_bonzai'),
7       ]
```

Dans ce cas nous utilisons la vue définie à l'aide d'une fonction

```
1   from django.urls import path
2   from . import views
3
4   urlpatterns = [
5           path('', views.index, name="accueil"),
6           path('bonzai/<int:bonzai_id>', views.BonzaiDetailView.as_view(), name = 'detail_bonzai'),
7       ]
8
```

Dans le second la vue définie à l'aide d'une classe

Les templates qui sont appelés par les vues se situent dans le dossier bonzai du dossier templates de l'application:

Chemin du template xxx.html:

APP/templates/bonzai/xxx.html

Pour faciliter l'expérience utilisateur UX, il est préférable que toutes les pages utilisent le même canevas. Nous allons donc créer un template de base, que nous viendrons compléter par des contenus spécifiques en fonction des vues appelées.

Ce template de base sera nommé base.html, il peut être commun à tout le site ou à une seule application, pour cela nous allons l'enregistrer à la racine du dossier templates de APP. Il contiendra le header et le footer de notre site.

Nous allons reprendre le fichier html vide que vous trouverez sur le drive

https://drive.google.com/open?id=1E2JEjVz9jtW8NDqEWMMID_A-S0y1PHCn

```
{% load static %}
{% load bootstrap4 %}
{# Load CSS and JavaScript #}
{% bootstrap_css %}
{% bootstrap_javascript jquery='full' %}
{# Display django.contrib.messages as Bootstrap alerts #}
{% bootstrap_messages %}
```

{% load static %} permet d'utiliser des raccourcis d'url pour les fichiers statiques

{% load bootstrap %} charge la bibliothèque bootstrap

Dans la balise <head> nous allons ajouter les lignes ci-dessous:

```
12      <link href="{% static 'main.css' %}" rel="stylesheet" type="text/css">
13      <link rel="shortcut icon" href="{% static 'favicon.png' %}">
```

href=« {% static ‘nomdefichier.ext’ %} utilise les paramètres définit par STATIC_URL et STATIC_ROOT définit dans settings.py

Plus simplement va chercher les fichiers dans le dossier static de l’application.

Puis ajoutons:

```
21      <title>
22          {% block title %}
23          {% endblock title %}
24      </title>
```

La balise {% block nomdublock %} <contenu par défaut> {% endblock nomdublock %}

Puis ajoutons:

```
21      <title>
22          {% block title %}
23          {% endblock title %}
24      </title>
```

La balise {% block nomdublock %}

<contenu par défaut peut être vide>

{% endblock nomdublock %}

Le bloc ainsi défini est chargé dans la page qui vient étendre base.html, si son contenu n'est pas redéfini dans la nouvelle page c'est le contenu par défaut qui est chargé.

Attention tout block ouvert doit être fermé!

Compléter le reste du fichier comme ci-dessous:

```
<header class="text-center">
    <div class="jumbotron">
        <div>
            {% block title_header %}
            {% endblock title_header %}
        </div>
    </div>
    <div class="main_menu">
        {% include '_menu.html' %}
    </div>
</header>
<body>
    <div class="container ">
        {% block container_1 %}
        {% endblock container_1 %}

        {% block container_2 %}
        {% endblock container_2 %}
    </div>
</body>

<footer>
    <div class="jumbotron text-center"> Les bonzais d'Armel</div>
</footer>
```

on ajoute différents blocs au header et au body définis comme précédemment.

le bloc {% include xxx.html %} permet de venir inclure du code html à l'intérieur de la page de base.

c'est dans _menu.html que nous définirons notre menu hamburger qui s'ajoutera à toute nos pages

bonzai.html

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

{% block title %}
    Les bonzaïs d'Armel
{% endblock title %}

{% block title_header %}
    Les bonzaïs d'Armel
{% endblock title_header %}

{% block container_1 %}
<h4 class="titre-section"> Première présentation </h4>
    <div class="card-columns">
        {% for b in bonzai_list %}
            <div class="card">
                <img class="card-img-top" src="{{ b.img_arbre.url }}" alt="{{ b.nom }}" >
                <div class="card-body ">
                    <h4 class="card-title text-center">{{ b.nom }} </h4>
                    <p class="card-text text-center">{{ b.age}} ans {{ b.taille }}m </p>
                    <div class="text-center">
                    <a href="{% url 'detail_bonzai' b.id %}" class="btn btn-success">Voir la description</a>
                    </div>
                </div>
            </div>
        {% endfor %}
    </div>
{% endblock container_1 %}
```

Quelques explications:

{% extends ‘base.html %} indique que le code va compléter la page base.html

Différents blocks peuvent voir leur contenu compléter ou créé.

Rappelez-vous la vue envoie une liste de bonzaïs à ce template nommé ‘bonzai_list’, pour afficher ces éléments il faut donc aller chercher chaque élément l’un après l’autre.

Pour cela on utilise une boucle for:

```
1   {% for b in bonzai_list %}
2           <img  src="{{ b.img_arbre.url }}" alt="{{ b.nom }}" >
3           <h4> {{ b.nom }} </h4>
4           <p>{{ b.age}} ans {{ b.taille }}m </p>
5           <a href="{% url 'detail_bonzai' b.id %}">Voir la description</a>
6   {% endfor %}
```

On crée la boucle {% for b in bonzai_list %} pour chaque élément de la liste bonzai_list , que l'on appelera b, on éxécute le code qui suit

<img src="{{ b.img_arbre.url }}" alt="{{ b.nom }}"> la balise img récupère l'url de l'image de b et son nom.

N'y a-t-il rien qui vous choque dans ce qui est écrit au-dessus: b.img_arbre.url ?

L'important ici est de bien comprendre que b est une instance de la classe Bonzai (un enregistrement dans la table bonzai, il comprend donc tous les attributs de Bonzai) d'où l'appel b.nom pour le nom.

Les valeurs sont toujours appelées entre {{ }}.

Voir la description

ici on définit le lien vers la page de détail pour le bonzaï on utilise **{% url 'nomdelurl' kwarg %}** on appelle l'url à partir de son attribut **name**, **kwarg** correspond **<int:pk>**
<int:pk> indique au resolver d'url que le nombre entier trouvé correspond au pk de l'objet

path('bonzai/<int:pk>', views.BonzaiDetailView.as_view(), name = 'detail_bonzai'),

detail_bonzai.html

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

{% block title %}
    Les bonzais d'Armel  {{ object.nom }}
{% endblock title %}

{% block title_header %}
    {{ object.nom }} {{ object.type_arbre }}
{% endblock title_header %}

{% block container_1 %}
    <div class="detail-b">
        <h4 class="title-detail">{{ object.nom }}</h4>
        <img src="{{ object.img_arbre.url }}" alt="{{ object.nom }}" class="pictdetail"></img>
        <h5> Description:</h5>
        <p class="text-center">{% lorem 4 b random %}</p>
    </div>
{% endblock container_1 %}
```