OUTILS DE DÉVELOPPEMENT – FRAMEWORK PYTHON

Les structures de contrôles, les collections et les classes python dans les Framework python, l'accès aux données, la gestion des vues, la création des Template et des formulaires.

1) Elément du langage python
2) L'accès aux données et les ORM
3) Gestion des vues et les Template
4) Gestion des formulaires

PR. ABOUABID HAMZA

AGENDA

- Introduction: Rappel sur les notions de base
- Partie 1: Le Framework Django
- **Partie 2: Les URLs et les vues**
- Partie 3: Gestion des vues
- Partie 4: L’accès aux données et les ORM
- Partie 5: Template
- Partie 6: Les formulaires

PARTIE 2: LES URLS ET LES VUES

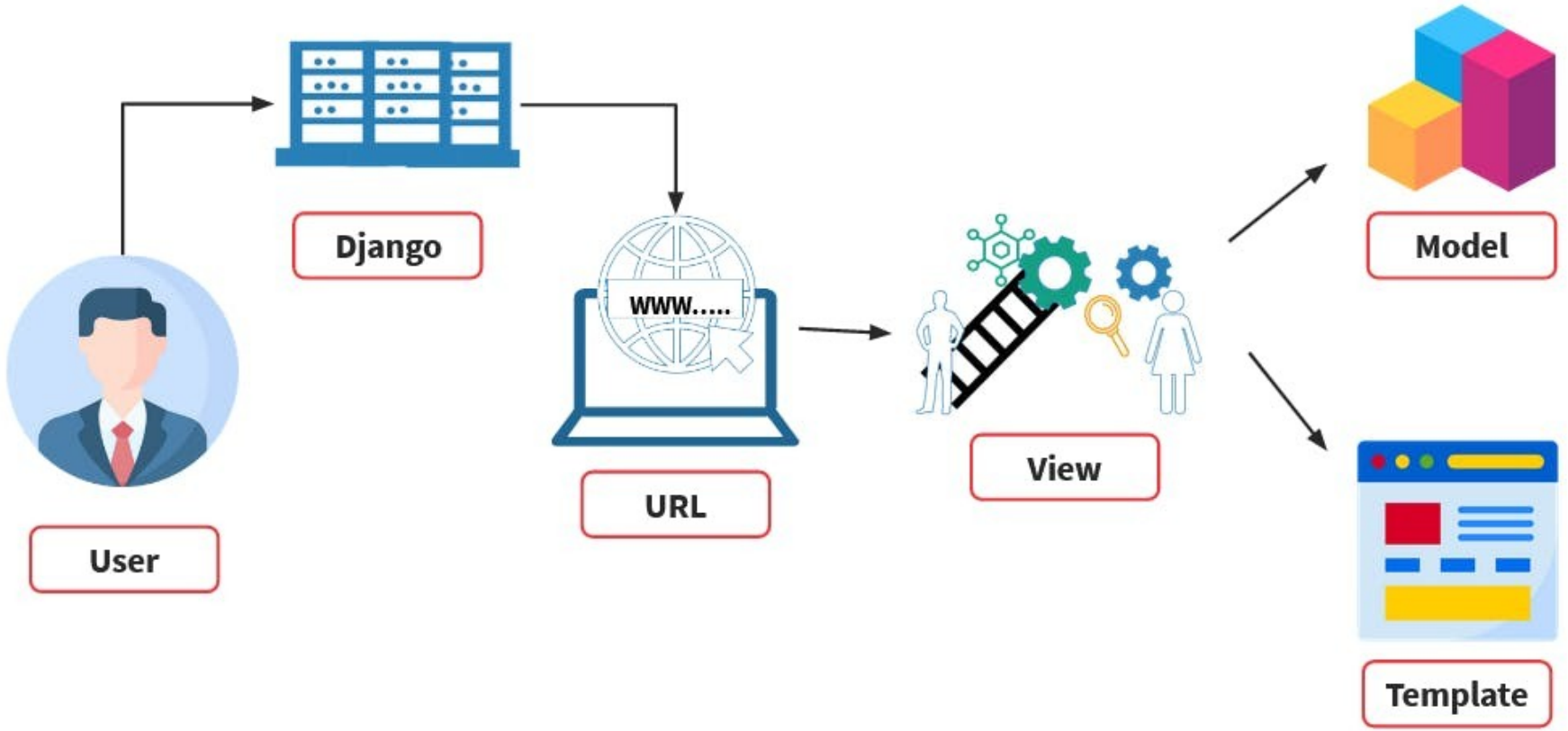
SÉANCE 4

1) **Rappel de l’architecture MVT**
2) Les fichiers url.py et view.py
3) Les url "dynamiques"

1) RAPPEL DE L'ARCHITECTURE MVT

Le premier rôle des **vues Django** est de lister toutes les pages possibles (toutes **URL possibles**).

↳ Les **vues** au sens de Django vont réquisitionner deux fichier python: **`url.py`** et **`view.py`**.



Source: https://blog.aiprobably.com/introduction-to-django

PARTIE 2: LES URLS ET LES VUES

SÉANCE 4

1) Rappel de l’architecture MVT
2) **Les fichiers url.py et view.py**
3) Les url "dynamiques"

2) LES FICHIERS URL.PY ET VIEW.PY

2.1. les URLs: (1/3)

- Pour faire fonctionner une application web Django, il faut **configurer au moins une *route* (URL)** qui va déclencher **un comportement**.
- La définition des **URL** possibles se fait dans un fichier nommée « **`urls.py`** ».
  - Il contient essentiellement une liste de correspondances entre des **URL** et des **fonctions Python**.
- Les routes se configurent dans les fichiers « **`urls.py`** »:
  - Du projet principale,
  - Les fichiers ***`urls.py`*** des autres applications.
- « **`urlpatterns`** » est une liste Python qui contient toutes les correspondances URL/fonction appeler.

2) LES FICHIERS URL.PY ET VIEW.PY

2.1. les URLs: (2/3)

- **Ajouter l'url de l'application dans les urls du projet :**

  - Importer la fonction include.
  - Ajouter l'url pattern de l'application dans la liste urlpatterns

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    …
     path('app1/',include('app1.app1urls')),
    …
]
```

```
|projectName
|projectName/
|    |  __init__.py
|    |  asgi.py
|    |  settings.py
|    |  urls.py
|    |  wsgi.py
|
|manage.py
```

2) LES FICHIERS URL.PY ET VIEW.PY

2.1. les URLs: (3/3)

- **Le fichier urls des application :**
  - Associer chaque url avec une méthode de la **vue**.

```
from django.urls import path
from . import views

urlpatterns=[
    path('url1/',views.hello ,name="view"),

path('template1/',views.index ,name="template")
]
```

- L'attribut `name=""` est utiliser pour effectuer une **inversion d'URL**.
  - Si vous appelez un modèle d'URL et qu'une autre application fait la même chose, l'URL trouvée par `reverse()` dépend du dernier modèle dans la liste des

```
|projectName
|   projectName/
|
|   applicationName/
|       migrations/
|           __init__.py
|       __init__.py
|       admin.py
|       apps.py
|       models.py
|       tests.py
|       views.py
|       urls.py
|
|   manage.py
```

2) LES FICHIERS URL.PY ET VIEW.PY

2.1. les views:

- Les **Views** ou Vues correspondent aux « **contrôleurs** » d'une application MVC.
  - Elles vont faire généralement le lien entre **le modèle** et les **Templates d'affichage**
  - Dans ces fichiers on définit toutes les fonctions python déclarées dans les tuples appelés **vues**.
  - Chaque **vues** reçoit en entrée un **objet représentant la requête HTTP** et dont les informations transmises pourront être exploitée.

Exemple:

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def hello(request):
    return HttpResponse("Hello World")

def index(request):
    return render(request,'index.html')
```

PARTIE 2: LES URLS ET LES VUES

SÉANCE 5

1) Rappel de l’architecture MVT
2) Les fichiers url.py et view.py
3) **Les url "dynamiques"**

3) LES URL DYNAMIQUE

- Ce type des urls sont utiliser pour pouvoir router des urls du type:
  - @IP/NomProjet/NomApp/variable

Exemple: http://localhost:8000/RHManagement/Management/userID=1

Exemple de la méthode:

- Modifions le fichier « urls.py » de l'application de la façon suivante :

```
path('Acceuil/<numId>',views.hello,name='Accueil')
```

- Modifiez maintenant la fonction `hello()` avec le profil suivant :

```
def hello(request, numId):
    return HttpResponse("Bonjour " + numId)
```

AGENDA

- Introduction: Rappel sur les notions de base
- Partie 1: Le Framework Django
- Partie 2: Les URLs et les vues
- **Partie 3: Gestion des vues**
- Partie 4: L'accès aux données et les ORM
- Partie 5: Template
- Partie 6: Les formulaires
- **Soumission du mini projet : 25% de la note finale**
- **Examen final : 75% de la note finale**

PARTIE 3: GETSION DES VUES

SÉANCE 5

1) **Fonctionnement des vues et L'objet `request`**
2) Les objets **`Request`** et **`Response`**
3) Application web Django MVT

1) FONCTIONNEMENT DES VUES ET REQUEST

1.1. Objectif d'une vues:

- Les **vues** créent un **contexte:**
  - Intègre un Template (ou d'autre type de retour) ,
  - Puis retournent un objet de type **`HttpResponse`** .

1.2. Fonctionnement d'une vue: (1/2)

1) Le serveur web reçoit une requête qui contient:
   - une adresse,
   - **un entête HTTP** ,
   - Les données venant des méthodes **GET** , **POST** ,
   - ...
2) L'adresse est traité par la table de routage (**url.py**) puis *Django* exécute la fonction associé a la route:
   - Cette fonction est la **vue** , puisqu'elle reçoit un contexte et renvoi un objet HttpResponse

1) FONCTIONNEMENT DES VUES ET REQUEST

1.2. Fonctionnement d'une vue: (2/2)

Exemple:

app2/url.py

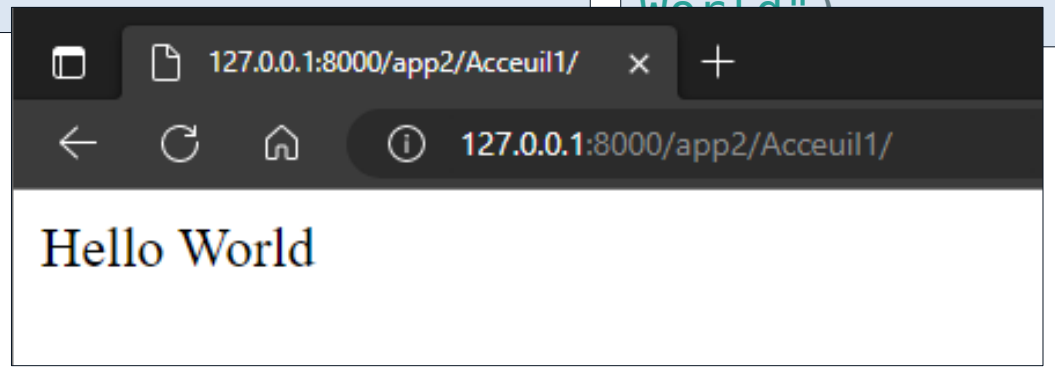
```
from django.urls import path
from . import views

urlpatterns=[
    path('Acceuil1/',
views.hello,name="view")
]
```

app2/views.py

```
from django.http import
HttpResponse

def hello(request):
    return HttpResponse("Hello
World")
```



1) FONCTIONNEMENT DES VUES ET REQUEST

1.3. L'objet `Request`:

- **`Request`** est une demande qui contient des **informations envoyées par un client**. Et le client attend **une réponse** en fonction des informations qu'il a envoyé.

  - **Utilisation**: Le contenu de l'objet `Request`

- Les attributs envoyée avec `Request`:

```
def hello(request):
    print(dir(request))
    return HttpResponse("Bonjour
monde!")
```

→

```
['COOKIES', 'FILES', 'GET', 'META',
'POST',
...,
'scheme', 'session', 'upload_handlers',
'user']
```

- Les données associée aux attributs envoyée avec `Request`:

```
def hello(request):

print(request.__dict__.get('COOKIES'))
    return HttpResponse("Bonjour
monde!")
```

→

```
{'csrftoken':
'akBU9YdOhgawhGQpZHbevT4Pg7fPK
1YC'}
```

PARTIE 3: GETSION DES VUES

SÉANCE 5

1) Fonctionnement des vues et L'objet request
2) **Les objets Request et Response**
3) Application web Django MVT

2) LES OBJETS REQUEST ET RESPONSE

2.1. Présentation de Request et Response:

- Django utilise des objets de **Request** et de **Response** pour transmettre l'état à travers le projet:
  - Django crée un objet **HttpRequest** qui contient des métadonnées sur la requête.
  - Django charge la vue appropriée, en passant le **HttpRequest** comme premier argument à la fonction de vue.
  - Chaque vue est responsable du retour d'un objet **HttpResponse**

Exemple:

```
def home(request):
    name= request.GET['name']
    return HttpResponse(f"Bonjour {name}!")
```

Les **métadonnées** de la requête passé en premier argument

Le retour d'un objet **HttpResponse**

2) LES OBJETS REQUEST ET RESPONSE

2.2. Les attributs `HttpRequest`:

- **`HttpRequest.method:`** La méthode HTTP utilisée dans la requête.

```
if request.method == 'GET':
        # code
elif request.method == 'POST':
        # code
```

- **`HttpRequest.body`:** Le corps brut de la requête HTTP (octets). Utiliser pour traiter les données de différentes manières que les formulaires HTML : images binaires, charge utile XML, etc.
- **`HttpRequest.GET`**: Un objet de type dictionnaire contenant tous les paramètres HTTP GET donnés.
- **`HttpRequest.POST`**: Un objet de type dictionnaire contenant tous les paramètres HTTP POST des donnés de formulaire.
- **`HttpRequest.COOKIES`**, **`HttpRequest.FILES`**, **`HttpRequest.META`**, **`HttpRequest.headers`**, **`HttpRequest.encoding`**, **`HttpRequest.scheme`**, …

2) LES OBJETS REQUEST ET RESPONSE

2.3. Les réponses HttpResponse:

- Une réponse où aucune erreur n'est à signaler possède le **code 200.**
- Il est possible de retourner d'autres types de réponse, comme la fameuse **404** indiquant que le document est introuvable.

Exemple:

```
from django.http import *

def Found(request):
    return HttpResponse (« Succès")

def NotFound(request):
    return
HttpResponseNotFound("Erreur")
```

- Les autre **réponses HTTP**:

```
from django.http import *
-------------------------------
--
HttpResponse                    →
200
HttpResponseRedirect            →
302
HttpResponsePermanentRedirect →
301
HttpResponseNotModified         →
304
HttpResponseBadRequest          →
400
HttpResponseNotFound            →
```

2) LES OBJETS REQUEST ET RESPONSE

2.4. Récupérer les argument passé avec un URL :

- L'url déclarer à l'aide de la fonction **path** peut contenir un **id** ou un **argument additionnel**:

Syntaxe :

```
path(route, view, kwargs=None, name=None)
```

- **Utilisation des ID dans un URL:** url dynamique (1/2)

- La chaîne « **route** » peut contenir des crochets angulaires (**<arg>**) pour capturer une partie de l'URL et l'envoyer comme argument de mot-clé à la vue.

Exemple :

```
path('utilisateur/<username>/', views.user, name='bio-Name')
```

- Les crochets angulaires peuvent inclure une spécification de convertisseur (**<type:arg>**) qui limite les caractères correspondants et peut également modifier le type de la variable transmise à la vue

Exemple :

```
path('utilisateur/<int:userId>/', views.userId, name='bio-Id')
```

2) LES OBJETS REQUEST ET RESPONSE

2.4. Récupérer les argument passé avec un URL :

- **Utilisation des ID dans un URL:** url dynamique (2/2)

Exemple :

url.py

```
urlpatterns = [
    path('index/', views.index, name='index'),
    path('utilisateur/<userName>/', views.user, name='bio-
Name'),
    path('utilisateur/<int:userId>/', views.userId,
name='bio-Id'),
    path('article/', include('article.urls')),
```

view.py

```
def user(request,userName):
    return HttpResponse(f"Nom: {userName}")

def userId(request,userId):
    return HttpResponse(f"Id: {userId}")
```

- La variable transmise à la vue doit être déclarer comme paramètres dans la vue:

2) LES OBJETS REQUEST ET RESPONSE

2.4. Récupérer les argument passé avec un URL :

- **Arguments passé dans un URL:**

- La fonction **path()** peut prendre un troisième argument facultatif qui doit être **un dictionnaire d'arguments** de mots clés supplémentaires à transmettre à la fonction d'affichage.

Exemple :

url.py

```
urlpatterns = [
    path('utilisateur/<int:Id>/', views.user, {'name': 'moham'},name='user-Id',)
]
```

- Dans cet exemple, pour une requête vers « /utilisateur/30/ », Django appellera

```
views.user(request, id=30, name='mohammed')
```

2) LES OBJETS REQUEST ET RESPONSE

2.4. Récupérer les argument passé avec un URL :

- **Les expressions réguliers dans un URL:**

- La fonction **re-path()** permet l'utilisation des expressions réguliers pour appeler un URL:

Syntaxe :

```
Re-path(route, view, kwargs=None, name=None)
```

- La syntaxe des groupes d'expressions régulières nommés est : `?P<arg>modèle`
  - où `arg` est le nom du groupe et `modèle` est un modèle à faire correspondre.

Exemple :

```
urlpatterns = [
    path('articles/2020/', views.article2020),
    re_path(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$',
views.month_archive)
]
```

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

Un retour JSON:

- **JsonResponse** est une sous-classe de HttpResponse qui aide à créer une réponse encodée en JSON.

Syntaxe :

```
JsonResponse(data, encoder=DjangoJSONEncoder,
safe=True,                  json_dumps_params=None,
**kwargs)
```

- ✓ **data**, est une instance dict. Si le paramètre **safe** est défini **False,** il peut s'agir de n'importe quel objet sérialisable JSON.
- ✓ **encoder** est utilisé pour sérialiser les données (par défaut : `DjangoJSONEncoder`)
- ✓ **json_dumps_params** un dictionnaire d'arguments de mots clés à transmettre à l'appel **json.dumps()** utilisé pour générer la réponse.

Exemple :

```
from django.http import JsonResponse

def nomUtilisateur(request,name):
    return JsonResponse({'userName': name})
```

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

- **Les Templates:**

- **TemplateResponse** est une sous-classe de **SimpleTemplateResponse** qui connaît le **HttpRequest** actuel.

Syntaxe :

```
TemplateResponse(request, template, context=None,
content_type=None,     status=None, charset=None, using=None,
headers=None)
```

Exemple :

views.py

```
from django.template.response import TemplateResponse

def home(request):
    return TemplateResponse(request, 'index.html', {'name': 'Jhon
Doe'})
```

Index.html

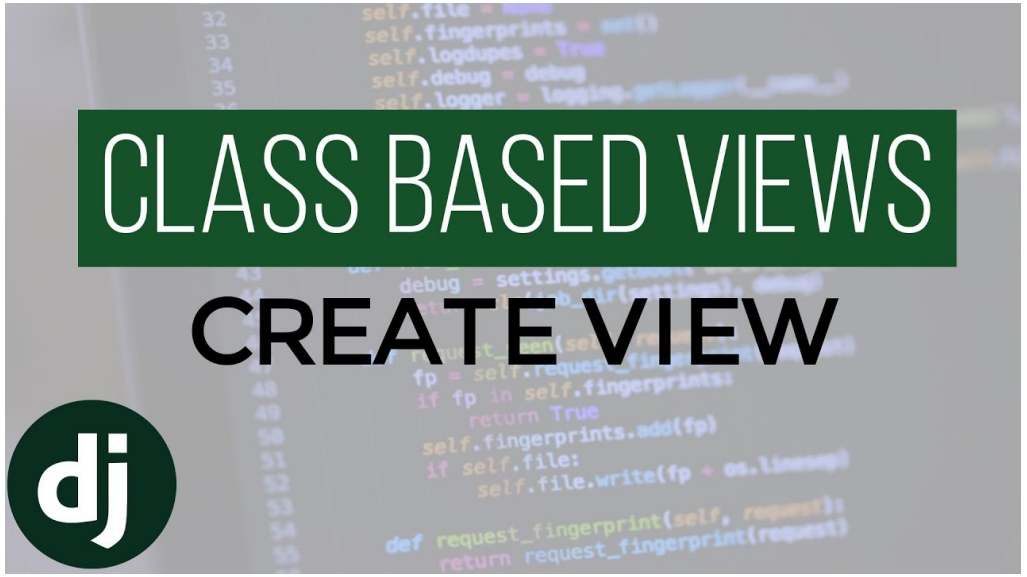
```
<div>
    <h1>Bonjour {{name}}</h1>
</div>
```

- **NB: Render()** est un raccourci pour **HttpResponse** , il fournit un moyen plus efficace de modifier des modèles et de charger des données dynamiquement.

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

- **Les vues génériques :**
  - `TemplateView`
  - `ListeView`
  - `CreateView`
  - `DetailView`
  - `UpdateView`
  - `DeleteView`



2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

Les vues génériques : TemplateView (1/2)

- **TemplateView** est une vue utiliser pour renvoyer un fichier type « **static** ».
  - C'est une vue générique basée sur des **classes** qui aide à créer une **vue** pour un **modèle spécifique** sans **réinventer la roue.**

- Les méthodes de la classe **TemplateView:**
  - **as_view**(cls, **initkwargs)
  - **dispatch**(self, request, *args, **kwargs): La méthode qui accepte un argument de requête plus des arguments et renvoie une réponse HTTP.
  - **get_context_data**(**kwargs):Renvoie un dictionnaire représentant le contexte du modèle. Les arguments de mots clés fournis constitueront le contexte renvoyé.
  - …

- Les attributs de la classe **TemplateView:**
  - **content_type** = None
  - **http_method_names** = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']
  - **response_class** = <class 'django.template.response.TemplateResponse'>
  - **template_name** = None

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

- **Les vues génériques : TemplateView** (2/2)

Exemple :

views.py

```
from django.shortcuts import render
from django.views.generic import TemplateView

class IndexView(TemplateView):
    template_name = "templates/index.html"

    def post(self, request, **kwargs):
        return render(request, self.template_name)
```

urls.py

```
urlpatterns = [
    path('Index1/',TemplateView.as_view(template_name="home.html")),
    path('Index2/',IndexView.as_view()),
    ]
```

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

- **Les vues génériques : ListeView, DetailView**

- **ListeView** est une page représentant une liste d'objets
- **DetailView** est identique à **ListView** sauf qu'il s'agit que d'un seul item

Exemple :

views.py

```
class ProductsView(ListView):
    model = Product
    template_name = "template/product_list.html"
    def get_context_data(self, **kwargs):
        context = super(ProductsView,
self).get_context_data(**kwargs)
        context['name'] = "Jhon Doe"
        return context
```

urls.py

```
urlpatterns = [
   path('Index2/',ProductsView.as_view()),
   ]
```

2) LES OBJETS REQUEST ET RESPONSE

2.5. Les différent type de retour d'une vue:

- **Les vues génériques : CreateView, UpdateView, DeleteView**

- **CreateView** est une vue générique pour créer une page pour enregistrer un item.
- **UpdateView** est une vue qui permet de modifier un item
- **DeleteView** est vue qui permet de supprimer un item

Exemple :

```
from django.urls import  path
from django.views.generic import *
from apps.models import Product

urlpatterns = [
    path('Product/Create',CreateView.as_view(model=Product)),
    path('Product/Update',UpdateView.as_view(model=Product)),
    path('Product/Delete',DeleteView.as_view(model=Product, success_url =
"/products/"))
]
```

PARTIE 3: GESTION DES VUES

SÉANCE 5

1) Fonctionnement des vues et l'objet **`request`**
2) Les objets Request et Response
3) **Application web Django MVT**
   - ↳ **Démonstration 2:**
     - Création de plusieurs application dans un projet
     - Création des vues HTTP et Template
     - Association des vues et url

PARTIE 3: GESTION DES VUES

SÉANCE 4: DÉMONSTRATION

DÉMO 2: Les URLs et les vues

- **Tache 1 :** Ouvrir un projet et Vérifier l'état du versionnage
- **Tache 2 :** Création d'une nouvelle application
- **Tache 3 :** Création des vues (différent type de retour)
- **Tache 4 :** Configurations des urls
- **Tache 5 :** Création du premier Template
- **Tache 6 :** Mis à jour du dépo « local git repo »

PARTIE 3: GESTION DES VUES

SÉANCE 5: TRAVAUX PRATIQUE

TP 5: Les URLs et les vues