Créer des urls dynamiques pour des articles (qui changent en fonctions du fichier html) :

1. Définir les vues (les noms des pages) dans views.py et retourner leur fichier html

*def article\_01(request) :*

*return render(request, ‘monappli/article\_01.html’)*

*def article\_02(request) :*

*return render(request, ‘monappli/article\_02.html’)*

*def article\_03(request) :*

*return render(request, ‘monappli/article\_03.html’)*

1. Aller dans urls.py importer ces 3 vues (ces 3 pages) et créer des urls pour ces pages :

*from . views import index, article\_01, article\_02, article\_03*

*urlpatterns = [*

*path(‘’, index, name= ‘index’),*

*path(‘article\_01/’, page\_01, name= ‘article\_01’),*

*path(‘article\_02/’, page\_02, name= ‘article\_02’),*

*path(‘article\_03/’, page\_03, name= ‘article\_03’)*

*]*

1. Ce qui est difficile ici, c’est qu’il faut créer une fonction différente qui correspond à chaque vue.
2. À la place, il faut créer une seule url et indiquer dans cet url le type de donnée de l’article à afficher (de manière dynamique) comme ceci :
   1. Dans views.py, définir une seule fonction (une vue, une page)

*def article(request) :*

*return render(request, ‘monappli/article.html’)*

* 1. Dans urls.py, définir une seule url (sans compter l’index) et lui insérer un élément qui permet de d’afficher (récupérer) dynamiquement le type du nom de la page (int, str...) : *-<type de données>*
  2. Dons si le nom de la page est une chaine de caractère, on indique par exemple : -<str:numero\_article>

*from . views import index, page*

*urlpatterns = [*

*path(‘’, index, name= ‘index’),*

*path(‘article-<str:numero\_article>/’, page, name= ‘article’),*

*]*

1. Retourner dans views.py, insérer ces symboles dans les paramètres de la fonction après request, précédé d’un fstring(f’).

NB : le nom article reste static ; le f’ converti les caractères en string

*def article(request, numero\_article) :*

*return render(request, f‘monappli/article\_{numero\_article}.html’)*

1. Pour afficher ces articles, il suffit de préciser dans l’url soit \_01, \_02, \_03 etc... en fonction

Les requêtes (suite)

Filtrer les données dans la table : équivalent des conditions en Mysql

Exemple : on veut afficher tous les éléments qui ont une colonne dont la donnée est True

1. Dans l’interpréteur python,

from monappli.models import nomDelaClasse

nomDelaClasse.objects.filter(colonne=True)

NB : True affiche 1 dans la table et False affiche 0

Avec le filtre, on peut faire plusieurs choses :

Ne filtrer que quelqu’un ou quelques éléments (avec les crochets [])

Supprimer tous les éléments (avec .delete())

etc...

nomDelaClasse.objects.filter(colonne=True)[2]

nomDelaClasse.objects.filter(colonne=True).delete()

On peut filtrer les colonnes de type dates :

On peut faire des comparaisons

On peut même filtrer les données les une à la suite des autres (enchainer les filtres

etc...

Ajouter (des données) des propriétés à un modèle (à une table) : créer des fonctions à l’intérieur des classes :

Pour rajouter des données à l’intérieur d’une table, on n’est pas obligé de passer les champs (par les colonnes)

On peut créer une (nouvelle) fonction (une méthode) à l’intérieur de la classe qui va par exemple afficher l’information qu’on veut rajouter dans la colonne (le champ)

Exemple :

class nomDeClass(models.Model) :

colonne = models.CharField(max\_length=100)

*def methode(self) :*

*if(self.methode) :*

*return « l’article exite »*

*return « l’article n’existe pas »*

1. À l’intérieur de l’interpréteur interactif >>>,
2. Faire (créer une variable et stocker la donnée de la colonne visée) :

variable = nomDelaClasse.objects.get(pk=3)

1. Pour afficher le contenu (le résultat) de la fonction, on fait :

variable.colonne()

1. Pour éviter de mettre des parenthèses devant la méthode, et donc créer une propriété et non pas une méthode, il faut retourner dans la classe (la table) et, au-dessus de la méthode (le fonction) écrire : @property

*@property*

*def methode(self) :*

*if(self.methode) :*

*return « l’article exite »*

*return « l’article n’existe pas »*

NB : si les changements ne sont pas prise en compte, sortir de l’interpréteur interactif pour valider le changement, puis, vous y reconnecter.

1. Conclusion : cette propriété (@property) permet, en fonction des colonnes ou de n’importe quoi, de retourner des valeurs. À chaque fois qu’on veut avoir les valeurs associés à (contenues dans) nos modèles (tables), on est pas obligé de créer une nouvelle colonne (un nouveau champ). On créé les champs (uniquement) lorsqu’on veut stocker une information qui va durer dans la table (dans la base de données). Si l’on veut juste stocker une information qui va s’afficher à la volée dans une page (template) html et qui n’ont pas besoin d’être stockées dans la base de données, on n’a pas besoin de créer une colonne spécifique.

Surcharger les méthodes (les fonctions) à l’intérieur d’une classe (d’une table) :

Ça permet d’effectuer des opérations au moment où on sauvegarde les variables (au moment où on insère des données dans la table, où on fait la requête)

Par exemple actuellement j’insère la colonne slug à la main. Mais si je veux modifier automatiquement la colonne slug en fonction des autre colonne, avant la sauvegarde (ou au moment de la sauvegarde), d’abord, je ne luis donnée pas de valeur par défaut entre parenthèses ()

Ensuite, je vais automatiser cette opération à l’intérieur de la méthode .save()

NB : chercher à savoir ce qu’est un slug : il retire les accents et tout ce qui est inutile autour d’une chaine (un texte) et la sépare avec des tirets

1. On créé une méthode (fonction) save ()

*def save(self, \*args, \*\*kwargs) :*

*if not self.slug :*

*self.slug = slugify(self.titre\_colonne)*

*super().save(\*args, \*\*kwargs)*

Les relations entre les tables (entre les modèles) :

Créer un super utilisateur (un administrateur) :

1. Dans le terminal python, taper :

py manage.py createsuperuser

entrer le nom d’utilisateur (un Username)

entrer une email :

entrer un password :

créer une table (un modèle) pour les catégories (de livre) :