Créer un projet Django (un site) :

django-admin startproject nom\_du\_projet

Lancer le serveur de développement

py manage.py runserver

***NB :*** *si le serveur ne démarre pas, aller dans manage.py et supprimer le commentaire : # !/usr/bin/env python*

Créer (configurer) l’url (du site)

1. Aller dans urls.py
2. Importer la vue (la fonction python : le nom de la page html) correspondante à l’url

*from . views import nom\_de\_la\_page*

1. Dans urlpatterns, créer un chemin : *path(‘url/’, vue, name= ‘’)*

*from django.contrib import admin*

*from django.urls import path, include*

*from . views import index*

*urlpatterns = [*

*path(‘’, nom\_de\_la\_page, name=‘nom\_de\_la\_page’),*

*path(‘’, index, name=‘index’),*

*on inclus (après) la liste des urls de l’application*

*path(‘appli/’, include(‘appli.urls’)),*

*path(‘admin/’, admin.site.urls)*

*]*

Créer une vue (une fonction python) correspondante à l’url : le nom de la page HTML

1. Aller dans views.py
2. Importer (le module) la fonction à renvoyer

*from django.shortcuts import render*

1. Créer une fonction nom\_de\_la\_page avec en paramètre (request)

*def nom\_de\_la\_page(request) :*

*return render(request, ‘nom\_du\_template.html’)*

Créer un template (du site) : une page (fichier) html

1. Clic droit sur le dossier contenant le dossier des fichiers du projet (du site)
2. Créer nouveau dossier nommé : templates
3. Clic droit sur le dossier templates et créer un nouveau fichier html
4. Copier le chemin absolu de ce templates : clic droit -> copie path...
5. Aller dans le fichier settings.py -> TEMPLATES = [os.path.join(BASE\_DIR, ‘chemin\_d’accès\_au\_templates’) c-à-d : nomDuDossier/templates]
6. Importer os.path

Comment insérer des données dans un template (un fichier html) :

1. Aller dans la vue (fichier views.py)
2. Après le nom du fichier (template) .html, rajouter un paramètre context={‘clé’ : valeur}
3. Aller dans le template (le fichier html correspondant), aux endroits souhaités du code html, insérer la clé du context entre (les double accolades {{ }} pour afficher la valeur (le contenu) de cette clé.
4. Voir la documentation sur les Gabarits pour savoir comment s’utilisent cette syntaxe.

Créer une applications (du site) : un mini-projet séparé du projet principal (du site)

1. Cliquer sur le dossier contenant les fichiers sources du projet (le dossier parent ayant le même nom que le dossier enfant)
2. Lancer le terminal et taper :

py manage.py startapp nom\_de\_l’application

1. Allers dans settings.py -> INSTALLED\_APPS : rajouter le nom de du dossier de l’appli à la fin (ou nom de l’appli)

INSTALLED\_APPS = [

‘nom\_de\_l’appli’,

]

NB :

* chaque application gère ses propres Modèles, ses propres vues et ses propres chemins d’urls.
* Avantage : on peut réutiliser une application sur un ou plusieurs projets

Créer les urls de l’application et les insérer dans le (site) projet

1. Clic droit sur le dossier contenant les fichiers de l’appli
2. Créer un fichier nommé : urls.py
3. À l’intérieur de ce fichier, créer des urls de l’appli : une liste d’urls
4. Inclure ces urls de l’application dans le fichier urls.py du site (projet)

*from django.urls import path*

*from . views import nom\_de\_la\_page\_d’appli*

*urlpatterns = [*

*path(‘’, index, name=‘index’)*

*path(‘appli/’, nom\_de\_la\_page\_d’appli, name=‘nom\_de\_la\_page\_appli’)*

*# dans le fichier urls.py du site*

*path(‘appli/’, include(‘appli.urls’)),*

*]*

Créer les vues de l’application

1. Clic droit sur le dossier contenant les fichiers de l’appli
2. Créer un fichier nommé : views.py
3. À l’intérieur de ce fichier, créer des vues de l’appli

*from django.shortcuts import render*

*def nom\_de\_la\_page(request) :*

*return render(request, ‘nom\_du\_template.html’)*

Créer les templates dans l’application

1. Clic droit sur le dossier de l’application
2. Créer nouveau dossier nommé : templates
3. Clic droit sur le dossier templates et créer un nouveau fichier html
4. *( On a donc deux dossiers templates, deux fichiers views.py et deux fichiers urls.py )*.
5. Pour marquer la différence (ce n’est pas obligé), il faut :

* À l’intérieur du dossier template, créer un sous-dossier ayant le même nom que l’application si c’est l’appli et un même nom du projet (site) si c’est le template est dans le site.
* Dans ce sous-dossier, glisser le fichier html créé.
* Dans les vues (fichiers views.py), aller rajouter appli/page.html ou site/page.html selon les cas.

Créer des urls dynamiques

Les bases de la gestion du fichier static :

Créer un fichier static (fichier CSS) directement dans une application :

1. Clic droit sur (le dossier de) l’application
2. Créer un nouveau dossier nommé : static
3. À l’intérieur de ce dossier static, créer un sous dossier nommé css
4. À l’intérieur du sous-dossier css, créer un fichier css : style.css
5. Sur ce fichier styles.css, écrire le code css
6. Aller dans la balise ‘head’ du fichier html (du template) faire la liaison avec ce fichier css - indiquer son chemin d’accès - : <link rel="stylesheet" type="text/css" href="{% static ‘css/style.css’ %}">
7. Au dessus de <!DOCTYPE html>, indiquer : {% load static %}
8. Redémarrer le serveur.

Créer un fichier static (fichier CSS) en dehors de l’application (dans le site) : C’est mieux (cela prend en compte le site et l’appli)

1. Déplacer (glisser) tout le dossier ‘static’ créé dans l’application à l’intérieur du (dossier du) site et valider
2. Aller dans le fichier settings.py
3. Tout en bas du fichier, en dessous de STATIC\_URL, créer la liste : STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, ‘nomDuSite/static’)

# on peut mettre ici autant de dossiers (ou fichiers) qu’on veut

]

1. Dans la balise ‘head’ du fichier html (template) : <link rel="stylesheet" type="text/css" href="{% static ‘css/style.css’ %}">, on peut laisser dans href = ... uniquement ‘css/style.css’.

Mettre en forme le site

Bases de données relationnelles

Télécharger : Tableplus pour la base de données

...

Les modèles (les tables de la base de données) :

1. Dans le fichier models.py
2. On créé une classe python (un objet, le nom de la table)
3. On créé des variables (attributs) de la classe (des objets, des noms de colonnes ou champs)
4. On attribut des valeurs aux variables (

Les champs de modèle (les colonnes) :

1. Les types de colonnes (champs) :

str = chaîne de caractère (CharField / TextField)

bool = booléen (BooleanField)

int = entier (IntegerField)

id = clé primaire (par défaut)

etc...

Créer un modèle (une table) :

1. Créer une application
2. L’ajouter dans le fichier settings.py (INSTALLED\_APPS [])
3. Aller dans models.py pour créer la table (une classe qui hérite du module modèle)
4. Syntaxe :

Class NomDeLaClasse(models.Model) :

nomDulaColonne = models.typeDeChamp(paramètres)

Class NomDeLaClasse(models.Model) :

variable = models.CoLonne(paramètres)

Créer des migrations (les fichiers python qui créent des tables correspondantes aux modèles du fichiers models.py) :

1. Après avoir créé une table dans models.py
2. On créé des migrations (un fichier de migrations) : py manage.py makemigrations nomDelApplication
3. Un fichier 000\_\_init\_\_ est créé (numéro de la migration)
4. Dans ce fichier, on retrouve la table créée par Django
5. Cette migration se fait à chaque fois qu’on fait des modifications dans le fichier models.py (dans la table)
6. Elle fait le lien entre
7. Cette migration laisse ou montre (dresse) l’historique des changements qu’il y a eu dans la base de données. Ainsi, on peut revenir à un état antérieur de la table.

Appliquer les migrations (insérer (ajouter) les tables créées dans models.py dans la base de données :

1. py manage.py migrate nomDelApplication
2. *(NB : py manage.py sqlmigrate nomDelApplication : permet d’afficher sous forme de sql ce qui va être exécuté lorsqu’on applique la migration : voir l’apperçu du code sql, à quoi il ressemble)*.
3. Lancer py manage.py migrate nomDelApplication une deuxième fois pour être sûr que toutes les migrations ont été appliquées
4. Lancer cette commande sans spécifier le nom de l’application va appliquer les migrations à la fois sur le site et sur l’application : (partout)
5. NB : c’est dans le fichier db.sqlite3 que sont stockées les données des tables créées.
6. Ces fichiers de migrations sont à inclure dans le dossier Git lorsqu’on travaille à plusieurs sur un projet.

Le paramètre blank des tables :

1. blank = false (ce champ ne peut pas être vide – dans le formulaire -) : c’est un paramètre par défaut.
2. blank = true (ce champ peut être vide) :

Le paramètre null :

1. blank=true est différent de null=true
2. blank : c’est niveau des formulaires et null : est au niveau de la base de données
3. Dans un champ CharField, ne pas mettre null=true
4. On peut définir une valeur par défaut : default=0

Faire des requêtes dans la base de données :

1. Faire des requêtes c’est manipuler des objets dans la base de données
2. Se (dé)placer dans le dossier contenant les fichiers sources du projet
3. Entrer dans l’interpréteur interactif python shell :

*py manage.py shell*

1. Y importer les classes du fichier models.py :

*from monappli.models import nomDelaClasse - (la table) -*

Insérer (Créer) des données (des objets) à l’intérieur de ces classes (de ces tables) :

*L’équivalent de : INSERT INTO table VALUES (? ? ?)*

*variable = nomDelaClasse(attribut=‘valeur’, sexe=‘test’, age=‘test’)*

*variable.save() (on sauvegarde)*

1. On peut aussi créer ces objets d’une autre façon (la meilleure):

nomDelaClasse.objects.create(*attribut=‘valeur’, sexe=‘test’, age=‘test’*)

Avantage ici : l’objet est automatiquement créé dans la base de données

Sélectionner (afficher les modèles) les objets des tables (les données des tables) :

*L’équivalent de SELECT \* FROM table*

1. Dans l’interpréteur python >>>,
2. Importer le modèle (la classe, l’objet, la table) :

*from monappli.models import nomDelaClasse*

1. On affiche tous les objets avec *.all()* :

*nomDelaClasse.objects.all()*

1. Lorsqu’on valide, un QuerySet s’affiche avec les id des objets existants dans la table

Récupérer et afficher un modèle précis (un objet précis, une donnée précise) : une seule entrée

*L’équivalent de SELECT nomDelaColonne FROM table WHERE conditions*

1. On affiche une donnée spécifique avec .get(), suivi d’un ou plusieurs paramètres
2. Dans l’interpréteur python >>>,
3. Importer le modèle (la classe, l’objet, la table) :

*from monappli.models import nomDelaClasse*

1. On affiche l’objet spécifique avec *.get()* :

*nomDelaClasse.objects.get()*

*nomDelaClasse.objects.get(nomDelacolonne=‘SaValeur’, ...)*

on peut spécifier id=numéro (l’index de l’objet) pour afficher son contenu

ou pk qui signifie primary key

nomDelaClasse.objects.get(id=2)

nomDelaClasse.objects.get(pk=2)

la différence :

id c’est vraiment le nom de la colonne

pk c’est vraiment la clé primaire

conseil : mieux vaut utiliser pk

Modifier les données de la table :

*L’équivalent de UPDATE nomDelaColonne FROM table*

1. Mettre la valeur (le contenu) de l’objet à modifier dans une variable

variable = NomDelaClasse.objects.get(pk=1)

1. Modifier cet objet (cette classe, cette donnée)

variable.nomDelaColonne = ‘nouvelleValeur’

variable.save()

Supprimer les données de la table :

1. On importe la classe (la table) depuis le fichier models.py

*from monappli.models import nomDelaClasse*

1. La suppression se fait avec *.delete()*

*Soit on supprimer toutes les données directement :*

*nomDelaClasse.objects.all().delete()*

Soit On stocke toutes les données de la table dans une variable puis,

on supprime une ligne précise de la table :

variable = *nomDelaClasse.objects.all()*

*variable[0].delete()*

ou supprimer un ensemble de ligne :

variable = *nomDelaClasse.objects.all()*

*variable[ :5].delete()*