Les étapes Site/Appli Django

1. On créé un projet (toute l’architecture du dossier et sous-dossiers)

django-admin startproject monprojet

1. Lancer le serveur de développement

py manage.py runserver

1. Créer les chemins d’urls :

* Fichier urls.py
* Importer les vues *(les fonctions, les noms des pages web)*

1. Créer les vues : *(les fonctions, les rôles, les noms des pages web)*

* Fichier : views.py
* Importer render

1. Créer un dossier templates + sous-dossier contenant les fichiers html
2. Configurer les templates : *(pas besoins lorsqu’on créé les applications)*

* Ficher settings.py
* Importer os.path
* TEMPLATES = [

{

'DIRS': [os.path.join(BASE\_DIR,'monsite/templates'),

'APP\_DIRS': **True**,

},

]

1. Insérer les données dans un templates

* Dans un contexte (clé, valeur) depuis le fichier views.py
* Insérer ce contexte dans le templates, dans des filtres
* Voir documentation

1. Créer une application (plus tard…)
2. Créer les fichiers statiques

* un dossier nommé ‘static’ => un sous-dossier nommé => ‘css’ => un fichier css
* pareil pour les fichiers images, javscript et autres…
* ne pas oublier {% **load** static %}
* et src ou href="{% **static** 'chemin/d’accès' %}" ...
* Faire des réglages dans le fichier settings.py

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR,'monsite/templates'),

]

1. Mettre en forme le site *(après)*

Par exemple :

* Utiliser un contexte, avec des filtres pour insérer des données dans un template
* Dans le dossier css, on peut stocker directement les images du site

**Les modèles *(10)***

1. Créer des modèles (les tables de la base de données)

* Créer une application
* Créer une application : py manage.py startapp monappli
* Configurer l’application :
  + Fichier : settings.py
  + INSTALLED\_APPS = [

'Nom de l’application',

]

* Dans fichier : models.py
* Créer des tables *(des classes)*
* Ouvrir le logiciel TablePlus
  + Ouvrir TablePlus =>
  + Create new connexion (nouvelle connexion) =>
  + Choisir la base de données (ex : Sqlite)
  + Remplir les infos et =>
  + Select file : choisir le fichier db.sqlite3 qui correspond à la base de données
* Lancer la migration : (pour crééer tout ce qui a trait à la base de donnée

py manage.py migrate

* + Sur TablePlus,faire :
  + Connection => reload (pour rafraîchir la base de données et voir les migrations obtenues)

1. Créer des migrations *(les tables correspondantes à nos modèles)*

py manage.py makemigrations monappli

* + C’est une commande qui met à jour la base de données
  + un fichier est créé par défaut contenant les détails des tables
  + c’est un fichier qui contiendra toute l’historique des modifications qu’on fera sur le modèle
  + faire ces migrations chaque fois que vous faites une modification dans nos modèles du fichier models.py

1. Appliquer les migrations créées avec makemigrations

py manage.py migrate monappli

* + cette commande permet de répercuter les changements effectués par makemigrations
  + et permet, grâce à cet historique, de recréer intégralement les tables supprimées
  + récupère les fichiers de migrations précédemment créés et les appliques à la base de données existante.

1. Le paramètre blank dans un formulaire

* blank=true signifie que le champ est requis (obligatoire)

1. La différence entre blank dans une table (un modèle)

* blank=true signifie, dans un formulaire, qu’un champs ne peut être vide,
* null=true signifie, au niveau de la table, qu’un champ (une colonne) peut être vide. *NB : à ne pas utiliser dans les champs (colonnes) de types* CharField ().

**Les requêtes *(11)***

1. Faire des requêtes dans la base de données : des **QuerySets**

**Insérer des données (créer, ajouter) dans une table**

* Lancer un interpréteur python interactif

py manage.py shell

* Importer la table (le modèle, la classe)

**>>> from** **monappli.models** **import** Personne

* + Créer des lignes, les champs (des instances ou objets) de la table | Commande SQL **INSERT**

**>>>** p = Personne(numero\_inscription='01', nom='mario', prenom='bross', date\_naissance='01/02/2022', email='mario@mail.fr', telephone='0102030455')

* + Sauvegarder ces données |  Commande SQL **UPDATE**

**>>>** p.save()

* + Créer les migrations de cette sauvegarde

**>>>** py manage.py makemigrations monappli

* + Appliquer ces migrations

**>>>** py manage.py migrate monappli

**Insérer des données** (**2ème façon**)

* + **>>>** p = Personne.objects.create(numero\_inscription='01', nom='mario', prenom='bross', date\_naissance='01/02/2022', email='mario@mail.fr', telephone='0102030455'")

*NB : pas besoin ici de sauvegarder*

* + On peut insérer plusieurs données à un champ [**ManyToManyField**](https://docs.djangoproject.com/fr/4.0/ref/models/fields/#django.db.models.ManyToManyField) voir documentation

**>>>** john = Author.objects.create(name="John")

**>>>** paul = Author.objects.create(name="Paul")

**>>>** george = Author.objects.create(name="George")

**>>>** ringo = Author.objects.create(name="Ringo")

**>>>** entry.authors.add(john, paul, george, ringo)

**Extraire les données d’une table |** Commande SQL **SELECT**

Exemple :

**>>> from** **monappli.models** **import** Personne

* + Extraire (afficher) les données de toute la table (tous les objets, lignes, champs) de la table

**>>>** Personne.objects.all()

* + On peut Extraire les données en fonction d’une colonne :
    - la donnée précise d’une table : (le nom de la personne qui s’appelle mario)

**>>>** Personne.objects.get(nom='mario')

* + - Ou plusieurs données précises : la personne dont le nom est mario et le prénom bross

**>>>** Personne.objects.get(nom='mario', prenom='bross', ...)

* + On peut également Extraire les données en fonction de la clé primaire :

* + - la personne dont la clé primaire est 1 :
  + **>>>** Personne.objects.get(pk=1)

**Modifier les données dans une table |** Commande SQL **UPDATE**

* + Modifier, par exemple, le nom de la première personne d’une table

**>>>** p **=** Personne.objects.get(pk=1)

**>>>** p.nom='thomas'

**>>>** p.save()

**Supprimer les données dans une table |** Commande SQL **DELETE**

**>>> from** **monappli.models** **import** Personne

* + Supprimer, par exemple, toutes les données de la table

**>>>** Personne.objects.all().delete()

Ou

**>>>** p = Personne.objects.get(pk=1)

**>>>** p = delete()

* + On supprimer des éléments précis de la table

**>>>** p = Personne.objects.get(pk=1)

**>>>** p[0].delete() *# supprime le 1er élément*

**>>>** p[:5].delete() *# supprime les 5 1er éléments*

**Filtrer les données dans une table *(rechercher dans les champs)* |** Commande SQL **WHERE**

*(voir dans la documentation)*

*Les paramètres*  [**filter()**](https://docs.djangoproject.com/fr/4.0/ref/models/querysets/#django.db.models.query.QuerySet.filter), [**exclude()**](https://docs.djangoproject.com/fr/4.0/ref/models/querysets/#django.db.models.query.QuerySet.exclude) et [**get()**](https://docs.djangoproject.com/fr/4.0/ref/models/querysets/#django.db.models.query.QuerySet.get) de [**QuerySet**](https://docs.djangoproject.com/fr/4.0/ref/models/querysets/#django.db.models.query.QuerySet).

**>>> from** **monappli.models** **import** Personne

* + On veut par exemple afficher toutes les personnes nées en 2021

**>>>** Personne.objects.filter(date\_\_year='2021')

1. Ajouter des propriétés à une table (à un modèle)
   * Si on veut afficher sur un client des opérations effectués sur les données d’une table ou y afficher des informations supplémentaires qu’on y rajoute de manières temporaires on peut le faire en créant une méthode de la classe avec un décorateur @property

*(voir dans la documentation)*

@property

**def** output\_field(self):

***# algorihme ici***

**return** FloatField()

1. On peut surcharger certaines méthodes spéciales à l’intérieur d’un modèle (d’une classe)

* On créé la méthode save qui permet d’effectuer des opérations (dynamiquement) au moment où on sauvegarde les données. Ça évite de le faire à la main.

*(voir dans la documentation)*

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

***# code de surchage ici***

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

1. Créer des relations entre les tables

Les relations entre les tables (12)

1. Créer un super utilisateur (un administrateur)

**...\>** py manage.py createsuperuser

1. Créer des relations (des liaisons, clés étrangères) entre les tables

Dans le fichier models.py

* + Relation 1,n : plusieurs à 1 (plusieurs articles peuvent être publié par plusieurs 1 auteur)

**class** **Membership**(models.Model):

person = models.ForeignKey(Person, on\_delete=models.CASCADE)

* + - on\_delete=models.CASCADE : si on supprime le champ, toutes les données reliées à ces champs disparaissent.
    - on\_delete=models.SET\_DEFAULT, default= 1 ou 0
    - on\_delete=models.SET\_NULL, null=True
    - on\_delete=models.PROTECT
      * créer des migrations et les appliquer (makemigrations et migrate)

*Associer une clé étrangère à une table :*

* + - Lancer le terminal shell : py manage.py shell
    - Importer toutes les tables (les modèles) dont on a besoin

**>>> from** **django.contrib.auth.models** **import** Message ou User ?

**>>> from** **trombinoscoop.models** **import** Message,Personne

* + - Sélectionner l’entrée de la table et lui associer la clé étrangère

**>>>** m = Message.objects.get(pk=1)

**>>>** m.auteur = Personne

**>>>** m.save()

* + - On peut supprimer cette association

**>>>** m.auteur = None

**>>>** m.save()

* + Relation n,n : plusieurs à plusieurs

**class** **Personne**(models.Model):

ami = models.ManyToManyField(Person)

* + - * créer des migrations et les appliquer (makemigrations et migrate)

Associer plusieurs clés étrangères à une table :

* + - Lancer le terminal shell : py manage.py shell
    - Importer toutes les tables (les modèles) dont on a besoin

**>>> from** **trombinoscoop.models** **import** Personne

* + - Sélectionner l’entrée de la table et lui associer la clé étrangère

**>>>** campus = Employe.objects.get(pk=1)

**>>>** emploi = Employe.objects.get(pk=2)

**>>>** variable.emploi.set([campus, emploi])

**Voir la documentation**

* + Méthodes permettant de manipuler les objets associés (liés) :
    - Add, set, remove, clear

**Voir la documentation**

* + Accéder aux relations inverses : Voir documentation
  + Récapitulatif

**Les vues *(13)***

* + Voir comment retourner un objet de type HttpResponse :
  + Voir comment retourner un objet de type JsonResponse : un objet de type Json
  + Voir comment retourner une erreur 404 : si un objet n’existe pas
  + Comment retourner un fichier HTML avec Render
  + Voir comment rediriger une url avec Redirect : vers une autre vue.
    - * Il faut indiquer le nom de l’attribut name
  + Voir le raccourcis :[**get\_object\_or\_404()**](https://docs.djangoproject.com/fr/4.0/topics/http/shortcuts/#django.shortcuts.get_object_or_404)
    - * Permet le lever une exception

**from** **django.shortcuts** **import** get\_object\_or\_404, render

**from** **.models** **import** Question

*# ...*

**def** detail(request, question\_id):

nom = get\_object\_or\_404(Personne, pk=question\_id)

**return** render(request, 'polls/detail.html', {'question': question})

Ou mieux :

**from** **django.shortcuts** **import** get\_object\_or\_404

**def** my\_view(request):

obj = get\_object\_or\_404(MyModel, pk=1)

* + Comment restreindre l’accès aux utilisateurs connectés

Utiliser @login\_required pour empêcher aux utilisateurs de voir cette vue

**from** **django.contrib.auth.decorators** **import** login\_required

@login\_required(redirect\_field\_name='my\_redirect\_field')

**def** my\_view(request):

...

* + Comment restreindre l’accès avec des conditions :

**from** **django.contrib.auth.decorators** **import** user\_passes\_test

**def** email\_check(user):

**return** user.email.endswith('@example.com')

@user\_passes\_test(email\_check)

**def** my\_view(request):

...

**Les gabarits *(14)***

1. Les variables :
   * Passer des variables depuis la vue vers un template (un fichier html)

On donne un contexte à la vue **{'first\_name': 'John', 'last\_name': 'Doe'}**

Dans le html, ça fait {{fonction.methode}} ou {{variable.champ}}

My first name is John. My last name is Doe.

1. Les conditions : voir comment les utiliser dans le template (fichier html)

Définir une fonction dans la vue (views.py) et faire la condition dans le template : c’est comme dans python

{% **if** nom %} # si le nom existe dans views.py

<h1> Si la condition est remplie, afficher </h1>

{% **elif** athlete\_in\_locker\_room\_list %}

<h1> Si la condition est remplie, afficher </h1>

{% **endif** %}

1. Les boucles : voir comment les utiliser dans le template (le html)
2. Les URLs : voir comment les utiliser de façon dynamique
3. La balise  with : permet de créer un alias dans un template (html). Voir comment l’utiliser. Elle permet de réduire la place que prennent les variables dans le code html
4. Les filtres : quand doit-on utiliser des filtres ?

Il est préférable de les faire dans le fichier views.py pour alléger le template (html)

1. Comment faire pour n’afficher que les premier mots d’un contenu, d’un paragraphe : voir les filtre striptags et safe
2. L’échappement HTML automatique : voir le filtre safe. Mais en réalité, Django fait l’échappement automatiquement.
3. Étendre un template avec extends.

**Les formulaires *(17)***

1. Créer un formulaire :

* *Dans le dossier contenant les fichiers du site, créer un fichier de type python nommé* ***forms.py***

*Clic droit sur le dossier -> New -> Python file -> Name - > Python file*

**Dossier-Site/src/sous-dossier-site/forms.py**

* *Dans ce fichier* **forms.py**

1. *on importe depuis Django le module* **forms** *(la classe, le type d’objet) qui permet de créer les formulaires*

**from** **django** **import** **forms**

1. *on créé un type de formulaire (un type d’objet, une classe) qui hérite module* **forms**

**class** **Authentification**(forms.Form):

HOMME = 'H'

FEMME = 'F'

SEXES = [

(HOMME, 'Homme'),

(FEMME, 'Femme')

]

*3. on créé les champs du formulaire (des variables et attributs) : des instances*

**variable = forms.TypeDeChamp(parametre=valeur, ou, attribut=valeur...)**

*NB. pour une variable équivalente à la balise ‘select’ (choices), une liste déroulante des choix possibles, on définit les options et leurs valeurs à l’intérieur de la fonction, dans un tuple*

**sexe = forms.CharField(choices=SEXES, default=HOMME)**

*NB. pour une variable équivalente à une case à cocher‘checkbox’, exemple :*

**case = forms.BooleanField(initial=True ou False)**

1. ss
2. ggg