

Python 3 – Django les bases



Objectifs du cours – Les bases de Django

Serveur de page, base de données, langage
L'architecture du Framework Django
Le modèle MVC et MVT
Lancement du serveur
Première application et page
Connection à une base de données
TP

Constituant techniques d'un site internet

- ↩ Un serveur Internet
 - ↩ On appelle une URL, on retourne un fichier
 - ↩ Les plus connus : Apache, nginx
- ↩ Un langage de programmation coté serveur
 - ↩ Permet de faire des boucles et des conditions ...
 - ↩ Les plus connus : PHP, ruby et python
- ↩ Une base de données
 - ↩ Permet de stocker des données (CRUD)
 - ↩ Les plus connues : mySQL, Oracle, SQLServer
- ↩ Un client Internet (Navigateur)
 - ↩ Permet d'interpréter et afficher une page HTML
 - ↩ Les plus connus : Chrome, Safari, Edge, Brave, ...

Les constituants Logique d'un site internet

- ↩ inventés par Tim Berners-Lee entre 1990 et 1994
- ↩ **HTML**
 - ↩ La structure des pages échangées
 - ↩ Des composants de mise en forme (css) et d'interaction (javascript)
- ↩ **HTTP**
 - ↩ HyperText Transfert Protocol
 - ↩ Le protocole réseau d'échange client serveur
 - ↩ Utilise par défaut le port 80
 - ↩ Une version Sécurisé (https)
- ↩ **L' URL**
 - ↩ Se base sur le DNS, c'est la convention de nommage des adresses
 - ↩ Se décompose entre avec une syntaxe particulière

Le Langage HTML

- « HyperText Markup Language » ou langage de balises pour l'hypertexte
 - Permet de structurer le contenu d'une page internet
 - Utilise des balises, des attributs pour définir les éléments d'une page
- Les balises, éléments de base du HTML
 - Titres `<H1> </H1>`
 - Paragraphes `<P></P>`
 - Tableau
 - Mise en forme
- Les attributs d'une balise
 - Permet de préciser les propriétés d'une balise
- Les liens hypertextes `<a href= ...`
 - Permet de simplifier la navigation entre les pages
- Les images
- Les formulaires pour saisir de la donnée

La syntaxe d'une URL

protocole://[nom:mot_de_passe@]adresse:port / chemin ? requête # fragment

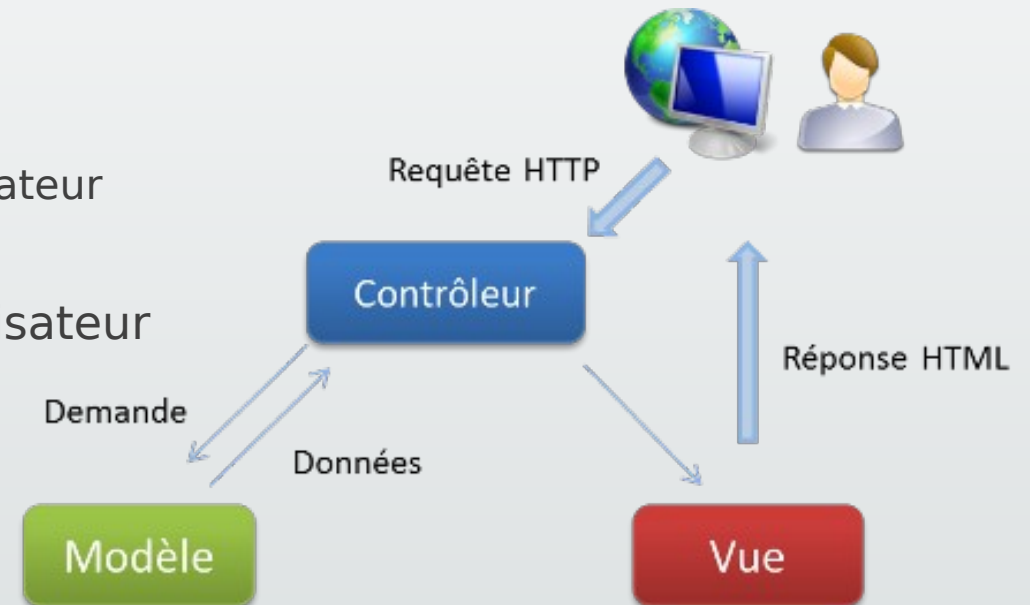
- ↩ **Protocole** : http, https, ftp, mailto, ...
- ↩ **Nom:mot_de_passe@** : identifiant directe (optionnel)
- ↩ **Adresse** : identifiant associé à une adresse IP pour accéder au site
- ↩ **Port** : par défaut 80
- ↩ **Chemin** : Répertoire et/ou fichier auquel on souhaite accéder
- ↩ **Requête** : paramètre transmis à l'application
- ↩ **Fragment** : positionnement dans la page

Présentation de Django

- Framework web : Ensemble d'outils et de composants logiciels formant une structure pour créer rapidement un site internet dynamique avec Python
- Permet de se reposer sur un ensemble de composant sans avoir à réinventer la roue et se concentrer sur son application
- Django se base sur une architecture MVC/T (Modèle, Vue, Contrôleur/Template)
- Django intègre :
 - son propre serveur Internet (PHP a besoin d'apache ou nginx pour fonctionner)
 - Un ORM : permet d'accéder aux bases de données sans « réelle » connaissance en SQL
 - Un gestionnaire de route (pour lier les url avec les programmes python)
 - Un gestionnaire d'utilisateur (pour gérer l'identification et la sécurité)
 - Un gestionnaire de template (pour la mise en forme des pages)

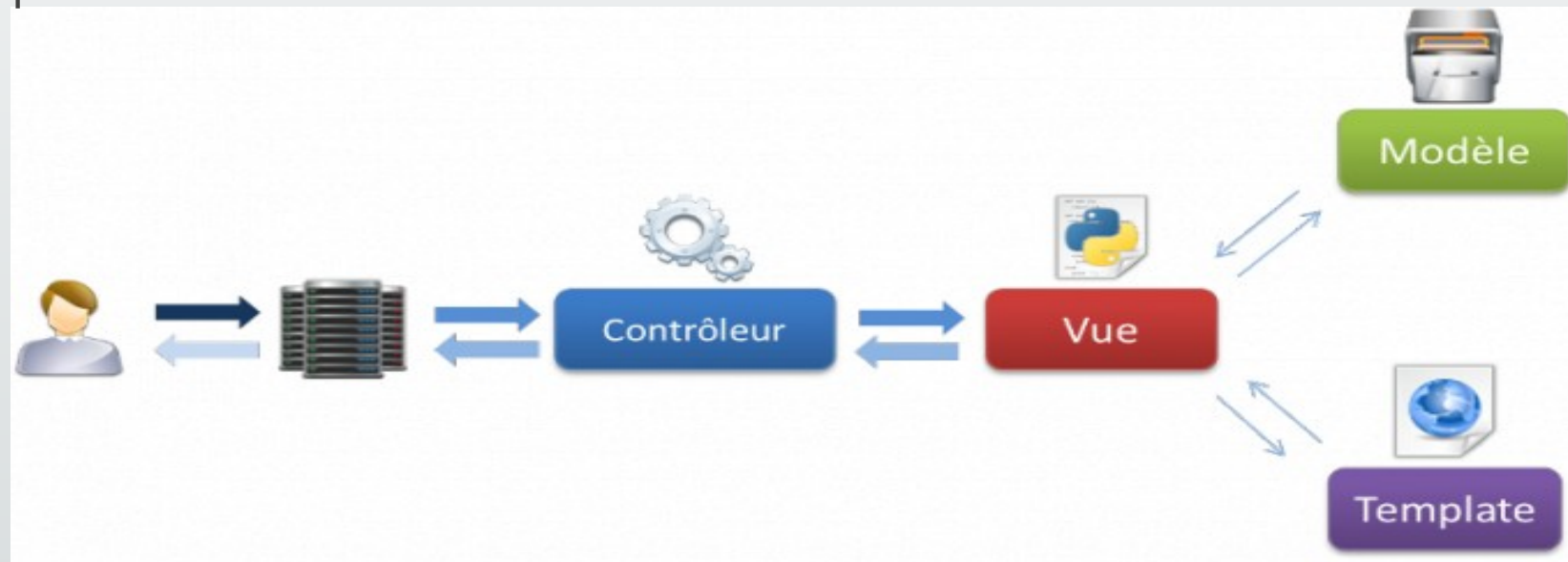
L'architecture Modèle-Vue-Contrôleur

- **Modèle** : espace de stockage et manipulation des données
 - Permet de stocker les données (le plus souvent dans une base de données...)
 - Permet d'accéder et manipuler les données (CRUD)
- **Vue** : page permettant de visualiser les données
 - Permet d'afficher les données
 - Permet aussi de récupérer les interactions avec l'utilisateur
- **Contrôleur** : prend en charge les événements de l'utilisateur
 - Appel d'une page
 - Validation d'un formulaire
 - Récupération des données dans le modèle



L'architecture Modèle-Vue-Template de Django

- Django gère lui-même la partie contrôleur (gestionnaire de route)
- Django introduit une notion de template, pour le rendu des page HTML
- On doit donc gérer 4 notions avec Django :
 - Le **Routage** des url vers la bonne application/ programme python
 - La gestion des données (CRUD) avec le **Modèle**
 - L'affichage des données au format HTML avec un **Template**
 - La **Vue** qui sert de chef d'orchestre à l'ensemble



Projet et Applications

- Un site Django est un projet décomposé en multiples applications
- Un projet et le répertoire racine du site
- Une application est un sous-répertoire du projet
- On peut récupérer des applications sous la forme de package
 - <https://djangopackages.org/>

Remarque : Un projet contient toujours un sous-répertoire (application) avec le même nom que le projet

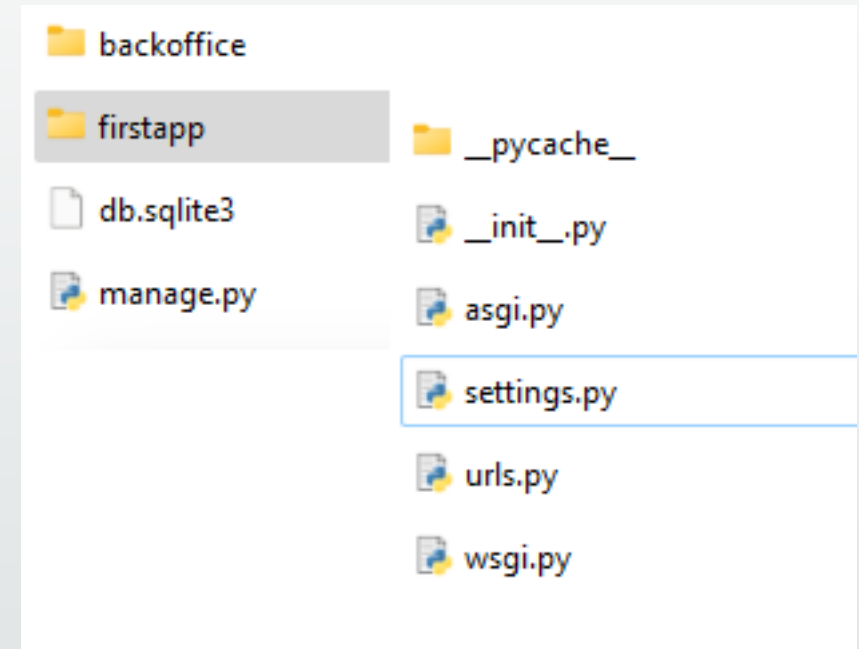
Installation et premier site en Django

- Depuis la console taper la commande
pip install django
OU pip3 install django
- Toujours depuis la console taper la commande
django-admin startproject firstapp
OU python -m django startproject firstapp

django va créer une structure « standard » avec un ensemble de fichier et de dossier

- On se positionne dans le premier dossier firstapp
- Pour lancer le site, taper la commande
python manage.py runserver

Le port par défaut est 8000, on accède au site par l'adresse suivante : <http://127.0.0.1:8000>



Premier lancement de Django

- Pour arrêter le site, il suffit de taper [Ctrl]+C depuis la console
- Pour utiliser un autre port, il faut taper la commande
➤ **python firstapp/manage.py runserver 4242**
- Pour autoriser d'autres à se connecter à son serveur local il faut utiliser
➤ **python firstapp/manage.py runserver 0.0.0.0:4242**
- Il est possible de lancer plusieurs serveur django sur le même poste (avec des ports différents)

Analyse des fichiers présent dans le dossier

- `manage.py` : utilitaire pour lancer le site et l'administrer (en ligne de commande)
- `db.sqlite3` : par défaut django crée une base de données au format sqllite

Dossier firstapp (répertoire du projet et répertoire du site)

- `__init__.py` : fichier vide indiquant à python qu'il s'agit d'un package, ce qui indique à python de compiler les fichier python
- `settings.py` : contient les paramètres du site (on y revient)
- `urls.py` : contient les chemins du site (liens entre les url et les applications/programmes python)
- `asgi.py` et `wsgi.py` : interfaces pour accéder au site depuis une application

Paramétrages Settings.py

- Applications/modules : les éléments chargés (idem les imports en python)
- Template / modèle des pages de django
- Base de données : paramètres d'accès à la base de données
- Sécurité
 - Définis les règles pour la définition des mots de passe
 - Ajouter MinimumLengthValidator 'OPTIONS': {'min_length': 5, } réduira à 5 le nombre de caractère pour un mot de passe
- Internalisation
 - LANGUAGE_CODE : pour passer en Français 'fr-fr'
 - TIMEZONE = « Europe/Paris » pour être sur le bon fuseau horaire
- Static : dossiers ou seront les élément static (css, images, ...)

La page d'administration

- ↩ Django possède nativement une d'administration
- ↩ Mais il faut créer un super utilisateur pour l'utiliser
- ↩ Lancer la commande pour créer les tables nécessaires à la gestion des utilisateurs
 - ↩ **python firstapp/manage.py migrate**
- ↩ Lancer la commande pour créer un superutilisateur
 - ↩ **Python firstapp/manage.py createsuperuser**
- ↩ On accède la page administration par l'URL
 - ↩ <http://127.0.0.1:8000/admin>
- ↩ Il est possible de créer de nouveau utilisateur ainsi que des groupes d'utilisateurs

Création d'une application

- Se positionner dans le dossier du projet firstapp et saisir la commande :
 - **python manage.py startapp blog**
- Le nouveau dossier créer contient les fichiers suivants
 - admin.py contient les éléments à gérer dans l'administration
 - models.py contient les modèles pour accéder aux données
 - tests.py contient les fonctions de tests unitaires de l'application
 - views.py contient les pages de l'application
- Pour ajouter l'application au projet il faut modifier le fichier settings.py
 - Ajouter dans INSTALLED_APPS l'application 'blog'

Première page, création de la vue

🔗 Ouvrir le fichier /firstapp/blog/views.py et saisir :

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

# Create your views here.
def home(request):
    # Exemple de page HTML, non valide pour que l'exemple soit concis
    text = "<h1>Bienvenue sur mon blog !</h1>"
    text = text + "<p>premier texte de présentation !</p>"
    return HttpResponse(text)
```

Première page, définition de la route vers la vue

🔗 Ouvrir le fichier firstapp/firstapp/urls.py et saisir :

```
from django.contrib import admin
from django.urls import path
from blog import views                                # importe la vue

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', views.home, name='home')           # définit la fonction à appeler
]
```

Première page, accès à la page

- Aller sur la page `http://127.0.0.1:8000/blog`
- Modifier le fichier `firstapp/firstapp/urls.py` en enlevant « `/blog` »
 - La nouvelle page est directement accessible à l'adresse <http://127.0.0.1:8000>
 - Il n'est pas nécessaire de relancer le serveur

Gestion des templates, création et paramétrage

- Ajouter un répertoire firstapp/templates
- Modifier la page firstapp/firstapp/settings.py au niveau du paramètre TEMPLATES
 - Ajouter import os
 - 'DIRS' : [os.path.join(BASE_DIR, 'templates')]
- Créer une page index.html dans le dossier templates et saisir :

```
<html>
  <head>
    <title>Mon template</title>
  </head>
  <body>
    <h1> Mon template </h1>
    <p> Bienvenue sur ma page de blog </p>
  </body>
</html>
```

Gestion des templates, utilisation

- ↩ Modifier la page firstapp/blog/views.py
 - ↩ Ajouter dans les déclarations

```
from django.template import loader
```
 - ↩ remplacer dans la fonction home

```
template = loader.get_template('index.html')  
return HttpResponse(template.render())
```
 - ↩ Le lien vers la page affiche le contenu de la page HTML
- ↩ On Définit une variable dans la fonction home et on modifie le render

```
data = {'prenom' : 'charlene'}  
return HttpResponse(template.render(data))
```
- ↩ On modifie le template avec la ligne suivante

```
<p>Hello {{prenom}} sur ma page de blog</p>
```

Gestion des templates, la balise « for »

➤ On modifie la `blog/views.py` pour ajouter une variable avec plusieurs valeurs

```
data = { 'prenom' : 'charlene',  
        'montres' : ['tissot', 'mondaine', 'seiko']  
}
```

➤ On ajoute dans le template `index.html` le code suivant

```
<h2> la liste de mes montres </h2>  
{% for montre in montres %}  
    <h3>{{montre}}</h3>  
{% endfor %}
```

Gestion des templates, la balise « if »

- On modifie la vue pour ajouter une variable avec plusieurs valeurs

```
data = { 'prenom' : 'charlene',  
        'montres' : ['tissot', 'mondaine', 'seiko'],  
        'age' : 17  
}
```

- On ajoute dans le template le code suivant

```
{% if age < 18 %}  
    <p> <b>PAS OPEN BAR {{age}}    </b></p>  
{% else %}  
    <p> OPEN BAR</p>  
{% endif %}
```

La gestion des variables de requêtes

- Il est possible de récupérer les paramètres transmis
- Dans le firstapp/views.py on ajoute dans la fonction home
`strAgePython = request.GET.get('ageurl',10) # si non renseigné, la valeur est à 10`
`'age' : int(strAgePython)`
- Ajouter à l'url de la page : « ?ageurl=25 »
 - `http://127.0.0.1/blog/?ageurl=25`

Les fichiers statiques

- Ajouter un répertoire firstapp/static Ajouter à la fin du fichier firstapp/firstapp/settings.py

```
STATICFILES_DIRS = [ os.path.join(BASE_DIR, 'static') ]
```
- Copier une image dans le dossier 'static'
- Ajouter dans templates/index.html la balise suivante

```
{% load static %}  

```

Accès aux bases de données

- ▮ Il est possible d'exécuter directement des requêtes SQL en python
- ▮ Mais django fournit un ORM pour accéder aux données
- ▮ Pourquoi utiliser un ORM?
 - ▮ Facilité l'échange avec la base de données (pas de requête SQL)
 - ▮ La structure du modèle devient celle des tables
 - ▮ Les tables sont mises à jour en même temps que la structure (migrate)
 - ▮ On ne réinvente pas la roue pour ce qui est du CRUD

Création d'un premier modèle de données

- ▮ Dans firstapp/settings.py, remplacer dans INSTALLED_APPS l'application 'blog' par '`blog.apps.BlogConfig`',
- ▮ Dans firstapp/blog/models.py on va définir une classe


```
class BlogPage(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    image = models.ImageField(upload_to="static/images/", blank=True)
    date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```
- ▮ Pour activer la gestion des images dans django il faut ajouter le module pillow
pip install pillow
- ▮ On lance ensuite la « migration » prise en compte de la classe
python manage.py makemigrations blog

Administration d'un modèle de données

- ❑ Modifier le fichier `firstapp/blog/admin.py`

```
from .models import BlogPage  
admin.site.register(BlogPage)
```
- ❑ On lance ensuite la « migration » prise en compte de la classe
`python manage.py migrate`
- ❑ Il est à présent possible d'aller sur la page d'administration et
 - ❑ d'accéder à la table
 - ❑ Ajouter des enregistrements
 - ❑ ... CRUD

Améliorer l'administration du modèle

- On ajoute une nouvelle classe (surcharge) dans le modèle `blog/models.py`

```
from django.contrib import admin

class BlogPagesAdmin(admin.ModelAdmin):
    list_display = ('title', 'date')
    list_filter = ('date',)
    search_fields = ('title', 'content')
```

- On modifie aussi le fichier `admin.py` pour prendre en compte cette nouvelle classe

```
from .models import BlogPage, BlogPagesAdmin
admin.site.register(BlogPage, BlogPagesAdmin)
```

Accéder aux données depuis le template

- On ajoute l'accès au modèle dans la vue blog/views.py

```
from .models import BlogPages
```
- On ajoute dans la fonction home la récupération des données

```
# on récupère les données de la base de données
lstBlogPages = BlogPages.objects.all()
```
- On ajoute les données à la data

```
'lstBlogPages': lstBlogPages
```
- On boucle et d'accéder aux champs de la table

```
{% for blogpage in lstBlogPages %}
    
    <p>{{blogpage.title}} - {{blogpage.content}}</p>
    <hr>
{% endfor %}
```

Liens entre modèles de donnée

- Il est possible de lier un modèle défini dans une autre application
 - Lien unique : 1 vers plusieurs
 - Lien multiple : plusieurs vers plusieurs
- On définit deux autres structures dans le modèle :
 - Catégorie avec un champ « titre »
 - Tag avec un champs « titre » aussi
- On fait le makemigrations et migrate pour prendre en compte les deux modèles
- On ajoute dans admins.py les deux classes crée dans le models
 - `admin.site.register(Categorie)`
 - `admin.site.register(Tag)`

Liens entre modèles de données : 1-N

- On ajoute la déclaration du modèle de catégorie dans le view et le modèle de **blog**

```
From .models import Categorie
```

- On peut définir un champ lié dans le modèle du post (avec un lien d'intégrité)

```
fk_category = models.ForeignKey(Categorie,  
on_delete=models.CASCADE, blank=True, null=True)
```

- Il est possible de récupérer le titre de la catégorie dans le template avec `blogpage.fk_category`

- Dans le models On peut récupérer les données liées au modèle

```
'categorie': blogPage[0].fk_category,  
'categorieId': blogPage[0].fk_category.id
```


Liens entre modèles de données : N-M

- On ajoute la déclaration du modèle de catégorie dans le view et le modèle de **blog**

```
From .models import Tag
```






- On définit la liaison dans le modèle du post

```
tagList = models.ManyToManyField(Tag)
```

- Il est possible de récupérer la liste de la catégorie dans le template avec `blogpage.tagList` et de boucler les informations

```
{% for tagelement in blogpage.taglist.all %}  
    {{tagelement}} #  
{%endfor%}
```

Tri et filtrage des données depuis une page

-  Il est possible de trier les données avec `.order_by()`
`blogPage.objects.all().order_by('title')`
-  Il est possible de limiter le nombre d'enregistrements récupéré avec `[:]`
`blogPage.objects.all().order_by('title')[:3]` # retourne les 3 premiers enreg.
-  Il est possible de filter sur une donnée précise avec `.filter()` à la place du `all`
`blogPage.objects.filter(title="suitedeluxe")` # sélectionne le titre exact
`blogPage.objects.filter(title__startswith="suite")` # le titre commence par
`blogPage.objects.filter(price__gte=150)` # quand le prix est supérieur à
-  L'élément étant une liste, on accède au premier enregistrement avec
 `blogPage[0]`