

Les bases de Django

Pour réaliser ou reprendre un projet de site web

Intérêt de Django

- Framework web codé en Python
- Facilite la gestion des tâches liées au web (routage d'URL, accès à une base de données)
- Très complet
- Pour les « perfectionnistes ayant des deadlines »
- Peut-être utilisé sur des projets de toutes tailles
 - Le backend d'Instagram est codé en Django
 - Le backup de la startup du coin (ExcellencePriority par exemple) est en Django également

L'organisation d'un projet Django

Plus d'infos :

<https://sametmax.com/organisation-dune-application-django/>

<https://docs.djangoproject.com/fr/3.1/intro/tutorial01/#creating-a-project>

La structure d'un projet django est très codifiée (on parle de « customization vs configuration »)

<https://mitratech.com/resource-hub/blog/configuration-vs-customization-whats-difference-matter/>

Les grands dossiers / fichiers :

- Le dossier du projet (configuration du projet)
- Le fichier manage.py (permet d'exécuter le projet)
- Les dossiers d'applications
 - models.py (relation avec la base de données)
 - urls.py (lien entre une URL et le code à appeler)
 - views.py (les fonctions appelées)
 - admin.py (gestion de l'admin Django)
 - forms.py (gestion des formulaires)
 - Dossier static (contient les fichiers de ressources)
 - Dossier templates/nom_de_lapp (contient les templates utilisés par les vues)

L'organisation d'un projet Django

```
1  .
2  |— manage.py
3  |— mon_app
4  |   |— admin.py
5  |   |— forms.py
6  |   |— __init__.py
7  |   |— models.py
8  |   |— static
9  |   |   |— css
10 |   |   |   |— style.css
11 |   |   |   |— js
12 |   |   |   |   |— behavior.js
13 |   |— templates
14 |   |   |— mon_app
15 |   |   |   |— index.html
16 |   |— tests.py
17 |   |— urls.py
18 |   |— views.py
19 |— project
20 |   |— settings.py
21 |   |— urls.py
22 |   |— wsgi.py
```

Les apps django

Une app django est sensée résoudre une problématique précise

En pratique, une app est un dossier / module

Elle peut aussi être un module python que l'on installe (pip install ...)

Pour qu'une app soit utilisée, elle doit être ajoutée aux "INSTALLED_APPS" dans le settings.py


Les apps django

Extrait du fichier settings.py

```
1 INSTALLED_APPS = [  
2     'django.contrib.admin',  
3     'django.contrib.auth',  
4     'django.contrib.contenttypes',  
5     'django.contrib.sessions',  
6     'django.contrib.messages',  
7     'django.contrib.staticfiles',  
8  
9     # The added apps are written right here  
10    'search.apps.SearchConfig',  
11  
12    # The Addition of the filter app  
13    'django_filters',  
14    'bootstrapform',  
15  
16    # CSV import export featuring  
17    'import_export',  
18  
19    # For pdf conversion  
20    # 'easy-pdf'  
21 ]  
22  
23
```

Les modèles

```
1 from django.db import models
2
3 class Musician(models.Model):
4     first_name = models.CharField(max_length=50, blank=True, null=True)
5     last_name = models.CharField(max_length=50)
6     instrument = models.CharField(max_length=100)
7
8 class Album(models.Model):
9     artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
10    name = models.CharField(max_length=100)
11    release_date = models.DateField()
12    num_stars = models.IntegerField()
```



Les modèles

Définit des champs qui seront modélisés

Un modèle doit hériter de « `django.db.models.Model` »

En gros, une classe de modèle = une table sur la base de données
et un champ = une colonne

L'ORM se chargera de faire la traduction entre l'objet python et la base de données (et vice versa)

Les modèles sont composés de champs (variables de classes) et de méthodes / attributs classiques pour faire ce que vous voulez

Il est possible de faire des liens entre différents modèles
(relations *1 to 1*, *1 to many* et *many to 1*)

Gérés dans le fichier `models.py`

L'ORM

Object Relation Mapper

Fait le lien entre des objets python et la base de données

Permet d'effectuer les 4 types de requêtes CRUD (create, read, update, delete)

L'ORM

Object Relation Mapper

Fait le lien entre des objets python et la base de données

Permet d'effectuer les 4 types de requêtes CRUD (create, read, update, delete)

```

1 from blog.models import Post
2 from django.contrib.auth.models import User
3
4 ## CREATE
5 me = User.objects.get(username='ola')
6 Post.objects.create(author=me, title='Sample title', text='Test')
7
8 ## READ
9
10 ### all
11 Post.objects.all()
12 # <QuerySet [<Post: my post title>, <Post: another post title>, <Post: Sample title>]>
13
14 ### filter
15 Post.objects.filter(title__contains='title')
16 # <QuerySet [<Post: Sample title>, <Post: 4th title of post>]>
17
18 ### order
19 Post.objects.order_by('-created_date')
20 # <QuerySet [<Post: 4th title of post>, <Post: My 3rd post!>, <Post: Post number 2>, <Post: Sample title>]>
21
22 ## chainage
23 Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
24 # <QuerySet [<Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>, <Post: Sample title>]>
25
26 ### get one (error if multiple returned)
27 post = Post.objects.get(title="Sample title")
28
29 ## UPDATE
30 post.text = "blablabla"
31 post.save() # /\ effectue la transaction et met à jour la base
32
33 ## DELETE
34 post.delete()
35 Post.objects.all().delete()

```

L'ORM

Les migrations

Changer un modèle (ajout / suppression / modification de champs) va modifier la structure de la base de données

Cela est effectué dans une migration

Il faut faire attention aux migrations car tout le code qui touche aux données doit être mis à jour

Il faut gérer les valeurs par défaut quand on rajoute un champ

Les URLs

Le principe de base est de relier un motif de l'adresse à une fonction python (appelée « vue »)

Les urls sont gérées à 2 endroits

Un système de préfixe pour chaque « app » (dans urls.py de l'application de base)

Un système interne à chaque « app » (dans urls.py de l'appli)

L'url peut gérer des motifs pour faciliter l'écriture

Les URLs

```
1 from django.urls import include, path
2
3 urlpatterns = [
4     path('index/', views.index, name='main-view'),
5     path('bio/<username>', views.bio, name='bio'),
6     path('articles/<slug:title>', views.article, name='article-detail'),
7     path('articles/<slug:title>/<int:section>', views.section, name='article-section'),
8     path('weblog/', include('blog.urls')),
9     ...
10 ]
```

Les vues

La vue s'occupe de gérer le rendu de la page HTML.

Une vue est une fonction python classique (il y a aussi un mécanisme à base de classes, mais on n'en parlera pas)

Une vue a accès aux paramètres passés dans l'URL et de la requête (utilisateur, paramètres GET, POST et fichiers)

Une vue doit renvoyer une réponse

Les vues

```
1 from django.http import HttpResponse
2 import datetime
3
4 def current_datetime(request):
5     now = datetime.datetime.now()
6     html = "<html><body>It is now %s.</body></html>".format(now)
7     return HttpResponse(html)
```


Les templates ou gabarits)

Les templates sont souvent des pages HTML qui possède des variables qui seront remplacées dynamiquement par la vue

Il y a un moteur de template avec une syntaxe spécifique qui permet de manipuler les données (boucles, conditions, inclusions d'autres templates, ...)

Attention : le HTML n'est qu'une petite partie de ce qui va être utilisé pour afficher la page (il y a le CSS pour le style et le Javascript pour l'interactivité)

```
1 {% extends "base_generic.html" %}
2
3 {% block title %}{{ section.title }}{% endblock %}
4
5 {% block content %}
6 <h1>{{ section.title }}</h1>
7
8 {% for story in story_list %}
9 <h2>
10   <a href="{{ story.get_absolute_url }}">
11     {{ story.headline|upper }}
12   </a>
13 </h2>
14 <p>{{ story.tease|truncatewords:"100" }}</p>
15 {% endfor %}
16 {% endblock %}
```

Les
templates
ou gabarits

La gestion des utilisateurs

Django intègre un mécanisme d'autorisation (droits) et l'authentification

Il y a également un mécanisme d'utilisateurs et de super utilisateur (pour accéder à l'administration)

Le sujet est trop complexe pour être abordé simplement, je vous conseille la lecture de la documentation :
<https://docs.djangoproject.com/fr/3.1/topics/auth/default/>

La gestion des utilisateurs

Code de création d'un utilisateur

```
1 >>> from django.contrib.auth.models import User
2 >>> user = User.objects.create_user('john', 'lennon@thebeatles.com', 'johnpassword')
3
4 # At this point, user is a User object that has already been saved
5 # to the database. You can continue to change its attributes
6 # if you want to change other fields.
7 >>> user.last_name = 'Lennon'
8 >>> user.save()
9
```

La gestion des utilisateurs

Code d'authentification d'un utilisateur

```
1 def my_view(request):
2     username = request.POST['username']
3     password = request.POST['password']
4     user = authenticate(request, username=username, password=password)
5     if user is not None:
6         login(request, user)
7         # Redirect to a success page.
8         ...
9     else:
10        # Return an 'invalid login' error message.
11        ...
```

La gestion des utilisateurs

Code permettant de refuser l'accès à une vue si l'utilisateur n'est pas connecté

```
1 from django.contrib.auth.decorators import login_required
2
3 @login_required(login_url='/accounts/login/')
4 def my_view(request):
5     ...
```

L'admin Django

L'admin est une des fonctionnalités les plus avancées et pratiques de django

Il permet de créer très rapidement un « back office » complet et personnalisable permettant de gérer les données

Sa puissance reside dans l'utilisation des méta données du modèle pour permettre la modification des champs

Je vous recommande de lire la documentation :
<https://docs.djangoproject.com/fr/3.1/ref/contrib/admin/>

L'admin Django

Pour qu'un modèle puisse être manipulé depuis l'admin, il doit être enregistré.

Voilà le code permettant d'enregistrer un modèle dans l'admin

```
1 from django.contrib import admin
2 from myproject.myapp.models import Author
3
4 admin.site.register(Author)
```


L'admin Django

Code permettant de gérer un modèle depuis l'administration tout en personnalisant l'affichage (tri selon le champ "date de publication")

```
1 from django.contrib import admin
2 from myproject.myapp.models import Author
3
4 class AuthorAdmin(admin.ModelAdmin):
5     date_hierarchy = 'pub_date'
6
7 admin.site.register(Author, AuthorAdmin)
```