

[supDeCode] ®



Mai 2021

Objectifs

- Découverte rapide du framework.

Pré-requis

- Programmation Python
- [Conception DB et langage SQL](#)
- [MySQL + phpMyAdmin](#)
- Design pattern MVC :
 - Routage
 - Entités
 - Contrôleurs
 - Vues
- ORM

Contenu (1/2)

- Historique
- Installation
- Créer un projet
- Créer une application
- Créer un contrôleur
- Créer une route
- Configurer une DB
- Créer des entités
- Effectuer la migration DB

Contenu (2/2)

- Utiliser l'interface d'administration auto-générée
- Utiliser l'ORM
- Créer une vue statique
- Gérer les fichiers statiques
- Gérer l'upload des images
- Utiliser le langage de template

Outils

- Interpréteur Python et ses accessoires.
[//www.python.org](http://www.python.org)
- Django
[//www.djangoproject.com](http://www.djangoproject.com)
- VS Code avec l'extension Python de Microsoft.
[//code.visualstudio.com](http://code.visualstudio.com)

Historique

- Créé en 2005 pour les besoins d'un journal local au Kansas.
Clin d'oeil aux guitariste de jazz Django Reinhardt...
- Actuellement en version 3.2.

Caractéristiques

- Framework MVC open-source et gratuit.
- Migration automatisée du modèle vers la DB.
- Interface auto-générée d'administration des entités.
- Serveur web de développement intégré.
- ORM et validation de formulaires.
- Quelques faiblesses :
 - Vocabulaire ambigu.
 - Langage de template limité.
 - ORM sans véritable hydratation.
 - Upload des fichiers pénible lors du passage en production.

Domaines d'utilisation

- Applications web de toutes tailles.

Documentation

- Site officiel.
<http://www.djangoproject.com>
- Documentation officielle.
<http://docs.djangoproject.com>

Installation sous Windows

- En shell via **pip** (*Package Installer for Python*, installé avec Python).

- Afficher la version

```
> py -m pip --version
```

-m (*me*) pour l'interpréteur associé à l'utilisateur courant.

- Upgrader (indispensable)

```
> py -m pip install --upgrade pip
```

- Installer/Désinstaller un package

```
> py -m pip install package
```

```
> py -m pip uninstall package
```

- Upgrader un package

```
> py -m pip install --upgrade package
```

Travailler avec un environnement virtuel

- Evite les conflits de versions de packages entre projets.
 - Définir un répertoire `.env` des environnements virtuels et s'y positionner.
 - Créer un environnement `acme-env` pour le projet ACME :

```
> py -m venv acme-env
```
 - Activer l'environnement :

```
> acme-env\Scripts\activate
```
 - Désactiver si besoin :

```
> acme-env\Scripts\deactivate
```

Installer Django dans l'environnement

- D'abord, upgrader pip :

```
(acme-env)> py -m pip install --upgrade pip
```

- Installer Django :

```
(acme-env)> py -m pip install django
```

- Lister les packages installés :

```
(acme-env)> py -m pip list
```

- Afficher la version de Django :

```
(acme-env)> py -m django --version
```

Créer un projet

- Créer le projet ACME :

```
(acme-env) > django-admin startproject acme
```

- Un projet peut contenir plusieurs applications.
Une même application peut être utilisée par plusieurs projets.
- Ce projet sera un catalogue de produits.
Il contiendra dans une unique application :
 - Une vue d'ensemble des produits ventilés par catégories.
 - Une vue de détail d'un produit.
 - Une vue d'erreur (produit inexistant).
 - Une interface d'administration auto-générée.

Squelette du projet

acme

Racine du serveur

+-- acme

Racine du projet

+-- __init__.py *Vide, définit le package.*

+-- asgi.py *Configuration serveur web ASGI*

+-- settings.py *Configuration du projet*

+-- urls.py *Table de routage du projet*

+-- wsgi.py *Configuration serveur web WSGI*

+-- manage.py

Utilitaire

Démarrer le serveur web

- Le serveur web de développement utilise l'interface WSGI (*Web Server Gateway Interface*) par défaut.
- L'interface ASGI (*Asynchronous Server Gateway Interface*) peut lui être substituée (en cours de mise au point).
- Démarrer le serveur :

```
(acme-env) > py manage.py runserver
```

Le shell est maintenant dédiée au serveur jusqu'à son arrêt (Ctrl+C).

Les warnings concernent la DB et seront réglés plus tard.

- Le serveur redémarre automatiquement après chaque modification le nécessitant SAUF en cas d'ajout ou de suppression de fichiers. Il faut alors le redémarrer manuellement.

Accéder au projet

- Le serveur écoute par défaut sur le port 8000.

Vérifiez le résultat dans votre navigateur :

[//localhost:8000](http://localhost:8000)

- Pour utiliser un autre port :

```
(acme-env) > py manage.py runserver port
```

Créer une application

- Créez l'application *products* :

```
(acme-env) > py manage.py startapp products
```

- Cette application contiendra les routes, les contrôleurs et les vues concernant les produits.

Squelette de l'application

`products`

Racine de l'application

`+-- __init__.py`

Vide, définit le package.

`+-- admin.py`

Inscription des entités

`+-- apps.py`

Configuration de l'application

`+-- migrations`

Répertoire des migrations DB

`+-- __init__.py`

Vide, définit le package.

`+-- models.py`

Entités (classes)

`+-- tests.py`

Test unitaires (fonctions)

`+-- views.py`

Contrôleurs (fonctions)

Lier l'application au projet

- Dans `products/apps.py`, une classe `ProductConfig` a été générée pour stocker les propriétés propres à l'application.
- Dans `acme/settings.py`, vous devez la déclarer dans la liste des applications installées : `INSTALLED_APPS` :

```
INSTALLED_APPS = [  
    'products.apps.ProductsConfig',  
    ...  
]
```

- Cette liste contient initialement les applications de base notamment pour la gestion des sessions et de l'authentification. Vous pourriez supprimer celles qui ne sont pas utiles à votre projet.

Un peu de vocabulaire MVC

- Attention aux confusions si vous venez de Symfony...

Django	Symfony	Remarque
Model	Entity	<i>Models</i> (le M de MVC) ou <i>Entities</i> , les deux termes sont synonymes.
View	Controller	Il est juste d'appeler <i>Controllers</i> (le C de MVC) les méthodes qui s'appuient sur le <i>Model</i> pour traiter les requêtes et retourner une réponse.
Template	View	Il est juste d'appeler <i>Views</i> (le V de MVC) les vues HTML, qu'elles utilisent ou non un langage de <i>templating</i> .

TD (1/2)

Créer un premier contrôleur

- A l'aide de la documentation :

<https://docs.djangoproject.com/en/3.2/topics/http/views>

créez le contrôleur `test` qui affichera le texte :

Test

en réponse à la route :

[/products/test](#)

- Patience, après la seconde partie de ce TD, vous pourrez voir le résultat dans votre navigateur.

TD (2/2)

Créer une première route

- A l'aide de la documentation :

<https://docs.djangoproject.com/en/3.2/topics/http/urls>

créez le fichier `products/urls.py` puis ajoutez-y la route `/test` en la reliant au contrôleur `test`.

- Dans `acme/urls.py`, utilisez `include()` * pour inclure les routes de l'application `products` aux routes du projet.
* *Notez que seule la route `admin` n'utilise pas `include()`.*
- Testez le résultat dans votre navigateur.

TD

Créer le routage de la future vue d'ensemble

- Créez maintenant le contrôleur `listProducts` qui affiche le texte :
Liste des produits
en réponse à la route :
[/products](#)
- Créez la route `/products` en la reliant au contrôleur `listProducts`.
- Testez le résultat dans votre navigateur.

TD

Créer le routage de la future vue de détail

- Créez le contrôleur `showProduct()` qui récupère l'*ID* d'un produit et affiche le texte dynamique :

Produit *ID*

en réponse à la route :

`/products/ID`

TD

Configurer la DB

- Via *phpMyAdmin*, créez la DB vide : **acmedjango**
- A l'aide de la documentation :

<https://docs.djangoproject.com/en/3.2/ref/databases>

installez le client *mySQL* recommandé.

- Sans utiliser de fichier séparé, placez les paramètres nécessaires dans **acme/settings.py** pour utiliser *mySQL* au lieu de *SQLite*.
- Vous testerez diapo suivante en lançant une migration...

TD

Migrer les entités des applications de base

- Dès maintenant, aidez vous de la documentation :

<https://docs.djangoproject.com/en/3.2/topics/migrations>

pour migrer les entités nécessaires aux applications de base installées et déclarées dans la liste `INSTALLED_APPS` de `acme/settings.py`.

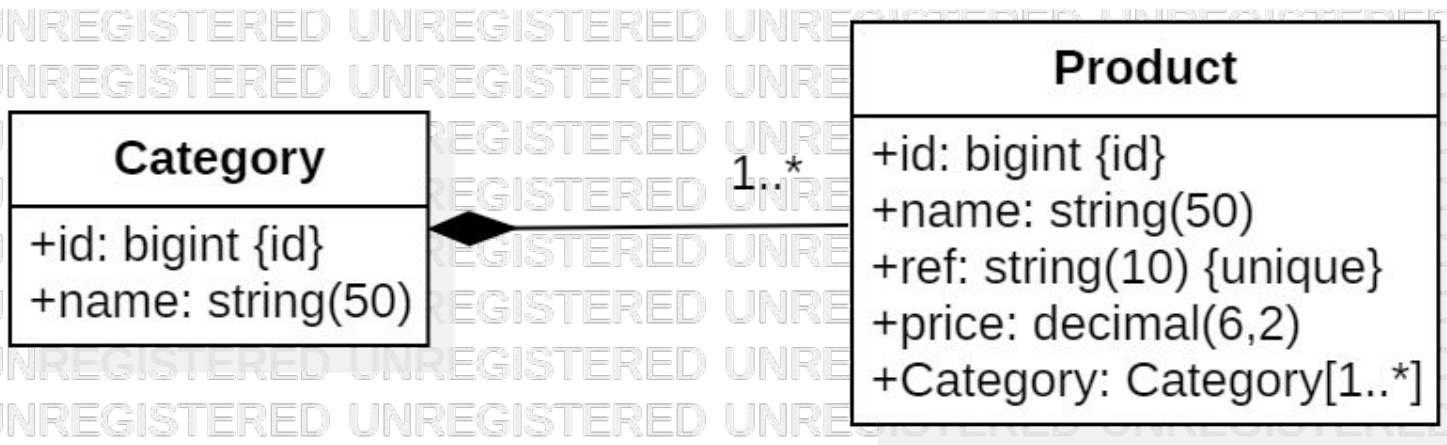
TD

Créez les entités

- A l'aide de la documentation :

[//docs.djangoproject.com/en/3.2/topics/db/models](https://docs.djangoproject.com/en/3.2/topics/db/models)

et du diagramme de classe :



créez les entités **Category** et **Product** sans vous soucier des images.

TD

Effectuer la migration des entités

- A l'aide des commandes :

- **makemigrations**

Journalisation

- **sqlmigrate**

Simulation SQL

- **migrate**

Exécution SQL

migrer les entités.

- Vérifiez le résultat dans la DB.

TD

Accéder à l'interface d'administration auto-générée

- Créez un administrateur :

```
(acme-env) > py manage.py createsuperuser
```

- Loguez vous à l'interface en administrateur :

[//localhost:8000/admin](http://localhost:8000/admin)

TD

Inscrire les entités administrables

- Dans `products/admin.py`, inscrivez les entités que vous souhaitez rendre administrables via l'interface d'administration :

```
admin.site.register(Category)
```

```
admin.site.register(Product)
```

TD

Utiliser l'interface d'administration (1/3)

- Ajoutez 3 catégories :
 - **Alpinisme**
 - **Randonnée**
 - **Escalade**
- Trouvez une méthode simple pour améliorer leur affichage.

TD

Utiliser l'interface d'administration (2/3)

- Ajoutez 10 produits :

- Alpinisme

- **Alpina** (210317 729) 1059.90 €

- **Oural** (210050 609) 240.95 €

- **Etna** (210029 309) 297.41 €

- Randonnée

- **Stefi** (310945 426) 132.90 €

- **Pronto** (310551 974) 119.96 €

- **Dorte** (310914 967) 127.46 €...

TD

Utiliser l'interface d'administration (3/3)

- Escalade

■	Anna	(320960 034)	105.00 €
■	Gunte	(430100 972)	116.96 €
■	Elfie	(430117 990)	125.96 €
■	Carla	(430112 303)	180.00 €

- Comme pour les produits, améliorer leur affichage (lors d'un ajout).
- Vous traiterez la gestion des images plus loin...

Utiliser l'ORM (1/9)

- Récupérer tous les enregistrements d'une table

```
Entity.objects.all()
```

- `objects` est le manager.
- `all()` retourne une instance de `QuerySet`.
- Un `QuerySet` est itérable.

- Exemple :

```
categories = Category.objects.all()
for category in categories:
    print(category) #Alpinisme Randonnée Escalade
```

Utiliser l'ORM (2/9)

- Filtrer un jeu d'enregistrements

Entity.objects.filter(field__ope=val)

Entity.objects.exclude(field__ope=val)

- `all()` est implicite.
- L'opérateur `__ope` est facultatif.
- `exclude()` est la négation de `filter()`.
- Ces deux méthodes sont chaînables.

- Exemple diapo suivante...

Utiliser l'ORM (3/9)

- Exemple :

```
categories = Category.objects
    .filter(id__lte=2)
    .exclude(name='Alpinisme')
for category in categories:
    print(category) #Randonnée
```

- Référence de QuerySet :

<https://docs.djangoproject.com/en/3.1/ref/models/queriesets>

Utiliser l'ORM (4/9)

- Trier un jeu d'enregistrements

```
Entity.objects.order_by(fields)  
    .asc()  
    .desc()
```

- Exemple :

```
products = Product.objects  
    .filter(category__name='Alpinisme') #One to many!  
    .order_by('name', 'price')  
for product in products:  
    print(product) #Alpina Etna Oural
```

Utiliser l'ORM (5/9)

- Limiter un jeu d'enregistrements

Entity.objects...() [a:b]

- Exemple :

```
products = Product.objects
    .all()
    .order_by('price')
    .desc()[:3]
for product in products:
    print(product) #Alpina Etna Oural
```

Utiliser l'ORM (6/9)

- Récupérer un enregistrement

```
Entity.objects.get(field__ope=val)
```

- Exemples :

```
print(Product.objects.get(id=5)) #Pronto
```

```
print(Product.objects.get(pk=5)) #Pronto
```

pk est un alias de la clé primaire.

- Exception si aucun enregistrement retrouvé.
- ★ L'ORM ne permet pas d'hydrater un objet déjà créé.

Utiliser l'ORM (7/9)

- Récupérer les enregistrements en relation One to Many

objEntityOne.EntityManyInLowerCase__set...

- Exemple :

```
category = Category.objects.filter(name='Alpinisme')
products = category.product__set
    .all()
    .order_by('name')
for product in products:
    print(product) #Alpina Etna Oural
```

Utiliser l'ORM (8/9)

- **Persister un objet (INSERT ou UPDATE)**

```
objEntity.save()
```

- Exemple :

```
product = Product()
product.name = 'Mountain'
product.ref = 'MNTN'
product.price = 123.45
product.category = Category.objects.get(pk=1)
product.save() #INSERT
product.price = 345.67
product.save() #UPDATE
```

Utiliser l'ORM (9/9)

- Supprimer un objet

```
objEntity.delete()
```

- Exemple :

```
product = Product()
product.name = 'Mountain'
product.ref = 'MNTN'
product.price = 123.45
product.category = Category.objects.get(pk=1)
product.save()      #INSERT
print(product.id)   #12
product.delete      #DELETE
print(product.id)   #None
```

TD

Gérer les *ID* inexistants

- Créez un contrôleur **noProduct** qui affiche simplement :

Produit inexistant

pour la route :

[/products/noProduct](#)

- A l'aide de la documentation :

[//docs.djangoproject.com/fr/3.2/topics/http/shortcuts/#redirect](https://docs.djangoproject.com/fr/3.2/topics/http/shortcuts/#redirect)

modifiez le contrôleur **showProduct** pour qu'il redirige vers cette route si l'*ID* est inexistant.

- Testez dans votre navigateur.

TD

Créer une vue statique

- Placez le template fourni `noProduct.html` dans un répertoire `products/templates` et aidez vous de la documentation :

[//docs.djangoproject.com/en/3.2/topics/templates](https://docs.djangoproject.com/en/3.2/topics/templates)

et de la méthode `render()` pour que le contrôleur `noProduct` rende ce template au lieu d'afficher un simple texte.

- Vérifiez le résultat dans votre navigateur.

Vous traiterez les CSS et l'image dans le TD suivant.

TD

Référencer les fichiers statiques

- Copiez le répertoire **assets** fourni à la racine du projet. Ces fichiers concernent la totalité du projet.
- Pour utiliser les fichiers statiques (CSS, JS, images...), déclarez le répertoire **assets** dans **acme/settings.py** :

```
STATICFILES_DIRS = [  
    BASE_DIR / 'assets',  
]  
STATIC_URL = '/assets/'
```

- Vérifiez le résultat dans votre navigateur.

TD

Upload des images (1/4)

- A la racine du projet, créez le répertoire `uploads` et son sous-répertoire `img` qui contiendra les images des produits uploadés.
- Dans `acme/settings.py`, déclarez ce répertoire comme celui des médias uploadés :

```
STATICFILES_DIRS = [  
    BASE_DIR / 'assets',  
]  
STATIC_URL = '/assets/'  
  
# Must be different from the STATIC target  
MEDIA_ROOT = BASE_DIR / 'uploads'  
MEDIA_URL = '/uploads/'
```

TD

Upload des images (2/4)

- Dans `acme/urls.py`, ajoutez :

```
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('products/', include('products.urls')),
    path('admin/', admin.site.urls),
] + static(settings.MEDIA_URL,
           document_root=settings.MEDIA_ROOT)
```

pour ajouter aux routes du projet la route `/uploads/...`

TD

Upload des images (3/4)

- A l'aide de la documentation :

<https://docs.djangoproject.com/fr/3.2/ref/models/fields>

ajoutez à l'entité **Product** une propriété **image** pour représenter une image uploadée. Précisez que l'image par défaut sera celle fournie :

img/noImage.jpg

et copiez la de suite dans **uploads/img**.

- Migrer à nouveau pour ajouter le champ **image** à la table **product**. Il contiendra le chemin relatif (depuis **uploads**) des images uploadées.
- Vérifier le résultat dans la DB.

TD

Upload des images (4/4)

- Utilisez l'interface d'administration pour ajouter les 10 images fournies aux produits correspondants.
- Vérifiez que, si vous ajoutez un produit sans image, c'est bien le chemin de l'image par défaut qui apparaît dans la DB.

TD

Utiliser le langage de template

- Apportez les modifications nécessaires pour que la vue de détail fonctionne pleinement y compris le lien vers la vue d'ensemble.
- A l'aide de la documentation du DTL (*Django Template Language*) :

<https://docs.djangoproject.com/en/3.2/ref/templates/language>

apportez les modifications nécessaires pour que la vue d'ensemble affiche les produits ventilés par catégories.

Le lien vers la vue de détail doit fonctionner.

- Testez l'ensemble du site dans votre navigateur.

TD

Utiliser l'héritage dans les vues

- Utilisez l'héritage pour simplifier l'ensemble des vues.