

# Chapitre II : Le Modèle dans le pattern MVT

Enseignantes :

Hounaida Moalla & Naima Halouani



# Plan

---

- Définition
- Définition de la structure
- Les associations
- Configuration de la base de données
- Activation des modèles
- Les requêtes
- Interface d'administration

# Définition

---

Un modèle est le composant du pattern MVT qui donne une description unique et définitive des données manipulées par l'application Django. Il contient les champs et le comportement essentiels des données permanentes. Généralement, à chaque classe du modèle, Django crée une seule table dans la base de données.

Les caractéristiques de bases de Django sont:

- Chaque modèle est une classe Python qui hérite de [django.db.models.Model](#).
- Chaque attribut du modèle représente un champ de base de données.
- Une API d'accès à la base de données offerte par Django.
- Mise à jour du schéma de la base de données pour correspondre au modèle actuel.

# Définition de la structure (1)

- Chaque attribut d'une classe du modèle est représenté par une instance d'une sous classe **Field**.
- Chaque classe définit un type particulier pour l'attribut de la classe du modèle
- Le tableau suivant illustre les classes les plus utilisées:

Type de l'attribut	Description
<b>models.CharField(max_length=valeur)</b>	définir un champ texte avec un nombre limité de caractères (max_length).
<b>models.TextField()</b>	définir un champ text sans limite de caractères.
<b>models.DateTimeField(default=timezone.now)</b>	Définir un champ de type date ou heure.
<b>Models.DateField( default=date.today)</b>	Définir un champ de type date initialisé à la date du jour.
<b>Models.IntegerField(default=0)</b>	Définir un champ de type entier
<b>models.FloatField()</b>	Définir un champ de type réel.

# Définition de la structure (2)

Type de l'attribut	Description
<b>models.DecimalField(max_digits=N one, decimal_places=None, <u>**options</u>)</b>	C'est un champ réel en précisant (le nombre total de chiffres (max_digits) et le nombre de chiffre après la virgule (decimal_places).
<b>models. IPAddressField(...)</b>	une adresse ip textuelle type 192.168.0.1
<b>models.ImageField ( upload_to = None , height_field = None , width_field = None , max_length = 100 , <u>**options</u> )</b>	Définir un champ de type image. Réellement ce champ contiendra le nom du fichier image qui sera sauvegardée dans le répertoire MEDIA dans la structure du projet.

# Définition de la structure (3)

Type de l'attribut	Description
<code>models.BooleanField(default=False)</code>	Définir un champ de type booléen avec une valeur par défaut False.
<code>models.EmailField ( max_length = 254 ,.. )</code>	C'est un champ qui contiendra un email. Il permet de vérifier que la valeur est une adresse e-mail valide à l'aide de <a href="#">EmailValidator</a> .
<code>models.FileField(upload_to='répertoire_de_téléchargement')</code>	Définir un champ de téléchargement de fichier.

# Définition de la structure (4)

- Certaines classes **Field** possèdent des paramètres obligatoires. La classe **CharField**, par exemple, a besoin d'un attribut **max\_length**. Ce n'est pas seulement utilisé dans le schéma de base de la base de données, mais également pour valider les champs,
- Un champ **Field** peut aussi autoriser des paramètres facultatifs
- Le tableau suivant illustre certains paramètres (attributs) des classes Field.

Paramètre	Description
défaut	La valeur par défaut du champ. Cela peut être une valeur ou un objet exécutable. Dans ce dernier cas, l'objet est appelé lors de chaque création d'un nouvel objet.
help_text	Texte d'aide supplémentaire à afficher avec le composant de formulaire.
primary_key	Si la valeur est True, ce champ représentera la clé primaire du modèle.  Si vous n'indiquez aucun paramètre <b>primary_key=True</b> dans les champs d'un modèle, Django ajoute automatiquement un champ <b>IntegerField</b> pour constituer une clé primaire.
unique	Si la valeur est True, ce champ doit être unique dans toute la table.

# Définition de la structure (5)

Paramètre	Description
<b>null</b>	Si null=True, django stocke les valeurs vides avec null dans la base de données. La valeur par défaut est False.
<b>blank</b>	Si blank=True, le champ peut être vide. La valeur par défaut est False. Blank est différent de null. Null est purement lié à la base de données, alors que blank est lié à la validation. Quand un champ possède blank=True, la validation de formulaire permet la saisie de valeurs vides.
<b>choices</b>	<p>Une séquence composée d'itérables de tuple binaires ('abréviation', 'valeur_visible') représentant les choix possibles pour ce champ.</p> <p>LANG_CHOICES=[('fr', 'Français'), ('ar', 'Arabe'), ('ag', 'Anglais')]</p> <p>Langue=models.CharField(max_length=2, choices=LANG_CHOICES, default='fr')</p>



# Les associations (1)



Dans les bases de données relationnelles, les relations entre les tables représentent un concept très proche des liens d'associations entre les classes orientées objets.



Django a la caractéristique d'ajouter automatiquement et d'une manière transparente des **id** à chaque table : champs similaire à la notion de clé primaire en relationnel.



Le champs **id** n'a aucune signification sémantique, mais sert à une identification unique d'un objet.

Django dispose des moyens nécessaires pour mettre en place les relations d'attributs dans les modèles :

- 1- one-to-one
- 2- many-to-one
- 3- many-to-many
- 4- Héritage

# Les associations (2)

Les associations entre les classes du modèle sont exprimées par des types de champs particuliers.

Le tableau suivant illustre ses associations:

Type de l'attribut	Description
<code>models.ForeignKey(ClasseAssociée, on_delete=models.CASCADE)</code>	Définir un lien vers un autre modèle. Pour définir une association many-to-one on utilise <code>ForeignKey()</code>
<code>models.OneToOneField(ClasseAssociée, on_delete=models.CASCADE, primary_key=True, )</code>	Définir une association one-to-one vers un autre modèle.
<code>models.ManyToManyField(ClasseAssociée)</code>	Pour définir une relation plusieurs-à-plusieurs, utilisez un champ <code>ManyToManyField</code>

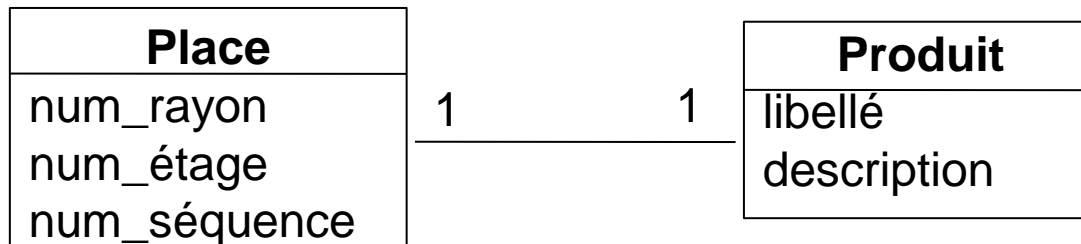
# Les associations (3)

## 1- One-to-one

Ce type de relation peut être précisé dans l'une ou l'autre des deux classes tout en pouvant fixer un choix sémantique.

Pour définir une association one-to-one dans Django, nous utilisons le champs **models.OneToOneField** du module **django.db**.

**Exemple :** Un produit ne peut avoir qu'une seule place dans les rayons d'un magasin, alors qu'il est préférable que toute place soit occupée, bien qu'elle ne peut porter qu'un seul produit.



# Les associations (4)

```
from django.db import models
class Place(models.Model) :
    # ...
class Produit(models.Model) :
    # ...
    place = models.OneToOneField ( Place,
    on_delete=models.CASCADE,
    primary_key=True,
    default=None,

    )
```

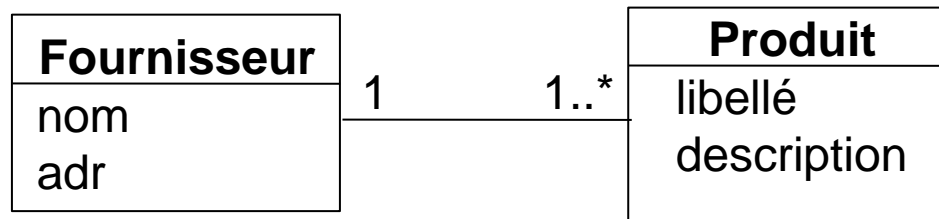
NB : d'une manière transparente, le champs **id** de la classe **Place** sera inséré dans la classe **Produit** sous le nom **id\_place**.

# Les associations (5)

## 2- Many-to-one

Pour définir une association many-to-one dans Django, nous utilisons le champs **models.ForeignKey** du module **django.db**.

**Exemple :** Un produit est fourni par un seul fournisseur



# Les associations (6)

---

```
from django.db import models

class Fournisseur(models.Model) :
    # ...

class Produit(models.Model) :
    # ...
    frs=models.ForeignKey(Fournisseur)
```

Le champs ForeignKey nécessite  
un argument de position : nom du  
modèle associé

# Les associations (6)

## 3- Many-to-Many

La relation n-n entre deux entités consiste à préciser les liens réciproques dans les deux entités.



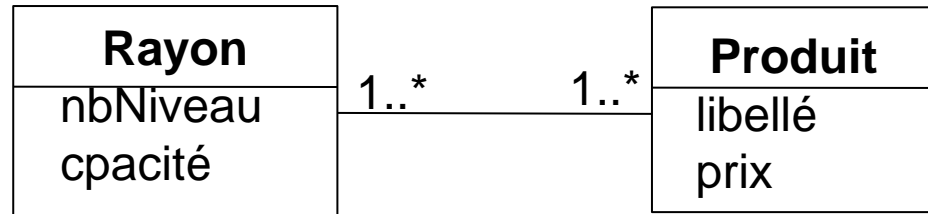
Ceci sera mis en place par l'addition d'une troisième entité de jonction qui s'attache aux entités initiales par deux relations 1-n.

Pour définir une association many-to-many dans Django, nous utilisons le champ `models.ManyToManyField`

```
from django.db import models
class Produit(models.Model) :
    # ...
class Rayon(models.Model) :
    # ...
    produits=models.ManyToManyField('Produit')
```

# Les associations (7)

**Exemple :** Un produit est placé dans plusieurs rayon et un rayon contient plusieurs produits.



Dans une association **Many-To-Many** une table associée de jonction doit être définie, et la table associée doit avoir une clé primaire indépendante, et les clés primaires des deux tables d'en-tête "plusieurs" doivent être introduites en tant que clés étrangères de la table associée .

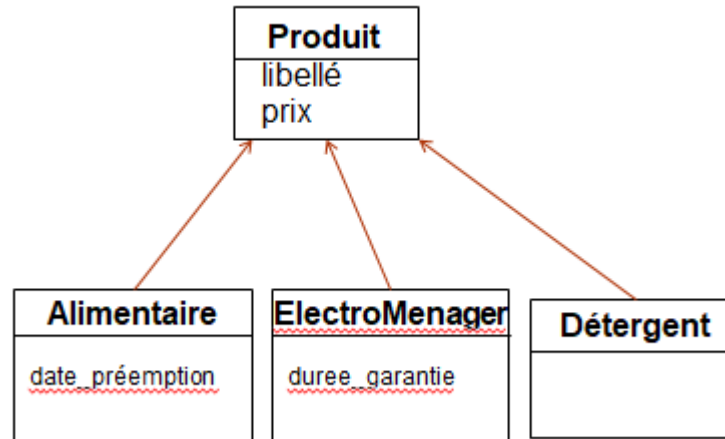
La table de jonction Rayon-Produit aura la structure suivante:

Id-Rayon-Produit	PK-Rayon	PK-Produit



# Les associations (8)

## 4- L'héritage



**Class ElectroMénager(Produit):**

`duree_garantie=Models.IntegerField(default=0)`



La création d'une instance de **ElectroMénager** créera une instance de **Produit** et une instance de **ElectroMénager** qui contiendra la clé primaire (**id**) du produit avec l'attribut **duree\_garantie**

# Configuration de la base de données (1)

---

Le moteur de base données utilisé par Django est SQLite3. Cette configuration est inscrite dans le fichier de configuration `setting.py` dans un dictionnaire `DATABASES`.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

Les clés et leurs valeurs respectives du sous dictionnaire relatif à la clé **'default'** sont décrits dans le tableau suivant:

# Configuration de la base de données (2)

Clé	valeur
'ENGINE'	Driver de l'SGBD utilisé : 'django.db.backends.sqlite3', 'django.db.backends.postgresql', 'django.db.backends.mysql' ou 'django.db.backends.oracle'
'NAME'	Nom et répertoire d'enregistrement de la base de données os.path.join(BASE_DIR, 'db.sqlite3') Ou bien 'BD_Commerce'
'USER'	Nom de l'utilisateur administrateur par exemple : 'root'
'PASSWORD'	Mot de passe de l'utilisateur administrateur.
'HOST'	Adresse de la base de données
'PORT'	Port de communication avec la base de données

# Configuration de la base de données (3)

---

Si on souhaite utiliser une autre base de données que SQLite, il faut installer le connecteur de la base de données approprié.

Il faut installer le driver de mysql pour python ainsi par exemple:

**pip install mysql-connector-python**

**pip install mysqlclient**

**pip3 install PyMySQL**

# Configuration de la base de données (4)

---

et changer les clés suivantes dans l'élément '**default**' de **DATABASES** pour indiquer les paramètres de connexion de votre base de données :

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'commerce',  
        'USER': 'root',  
        'PASSWORD': '',  
        'HOST': '127.0.0.1',  
        'PORT': '3306',  
    }  
}
```

# Activation des modèles

---

- Une fois les différentes classes des modèles ont été définies, il faut indiquer à Django que des changements ont été effectués aux modèles. Il faut lancer la commande suivante:

```
Python manage.py makemigrations
```

- Les changements dans le modèle devraient être stockés sous forme de *migration*.
- Une migration se charge de mettre à jour la base de données, sans perte de données, en exécutant la commande suivante:

```
Python manage.py migrate
```

# Les requêtes (1)

---

Python offre un shell interactif pour jouer avec l'API de manipulation de la Base de données. Pour lancer un shell Python, utilisez cette commande :

```
python manage.py shell
```

Pour pouvoir manipuler les tables de la base de données, il faut importer les classes correspondante à partir du modèle.

```
>>> from applicationNom.models import ClasseNom1, ..
```

# Les requêtes (2)

## Les commandes

- Afficher tous les enregistrements d'une table.

```
>>> Classe.objects.all()
>>> Classe.objects.all()[deb:fin:pas]
# limitation des QuerySet
```

Equivalente à une requête sql  
Select \* from classe

- Ajouter un enregistrement dans une table

```
>>> obj=Classe(attribut1=valeur1,attribut2=valeur2,...)
>>> obj.save()
```

Equivalente à une requête sql  
Insert into classe values .....

```
>>> obj=Classe.objects.create(attribut1=valeur1,attribut2=valeur2,...)
```



# Les requêtes (3)

## Les commandes

- Modifier la valeur d'un attribut.

```
>>> obj.attribut # afficher sa valeur  
>>> obj.attribut=nouvelleValeur  
>>> obj.save()
```

Equivalente à une requête sql  
Update classe set attribut=NouValeur ...

- Afficher les enregistrements correspondants à un attribut particulier (QuerySet)

```
>>> Classe.objects.filter(attribut=valeur)  
>>> Classe.objects.filter(attribut__startswith='...')
```

Equivalente à une requête sql  
Select \* from classe where attribut=valeur

# Les requêtes (4)

## Les commandes

### Recherches dans les champs

La recherche dans les champs qui constitue le cœur des clauses SQL WHERE.

La syntaxe s'exprime par des paramètres nommés dans les méthodes [filter\(\)](#), [exclude\(\)](#) et [get\(\)](#) de [QuerySet](#).

Les paramètres nommés de base de ces requêtes prennent la forme syntaxique suivante:

**Syntaxe : champ\_\_typerequete=valeur**

Champ__typerequete	Signification
champ__lt/lte	La valeur du champ est inférieur (inférieur ou égale) à une valeur.
champ__gt/gte	La valeur du champ est supérieur (supérieur ou égale) à une valeur.
champ__startswith champ__endswith	Un champ qui commence (se termine) par une chaîne
champ__contains champ__icontains	Un champ texte qui contient une certaine chaîne. C'est un test d'inclusion sensible (insensible) à la casse.
champ__iexact	Une recherche insensible à la casse

# Les requêtes (5)

## Les commandes

### Recherches dans les champs (Exemples)

```
>>>Produit.objects.filter(libelle__iexact='Table')
>>>Produit.objects.filter(description__icontains='diamètre')
>>>Produit.objects.filter(libelle__startswith='Table')
>>>Produit.objects.get(libelle__exact='Table')
>>>Produit.objects.exclude(prix__lte=60)
>>> Produit.objects.get(prix__gt=70,prix__lt=100)
```

# Les requêtes (6)

## Les commandes

- Afficher un seul enregistrement correspondant à une valeur unique.

```
>>> obj= Classe.objects.get(pk=valeur)
```

Equivalente à une requête sql  
Select avec une jointure

- Ajouter un ou plusieurs objets dans un champ de type ManyToMany

```
>>> Classe.AttributMTM.add(objet1,objet2,...)
```

# Les requêtes (7)

## Les commandes

- Supprimer un enregistrement.

```
>>> obj=Classe.objects.get(id=1)  
>>> obj.delete()
```

Equivalente à une requête sql  
Delete from classe where id=1

- Ajouter un ou plusieurs objets dans un champ 'attributMTM' de type ManyToMany

```
>>> Classe.attributMTM.add(objet1,objet2,...)
```

# L'interface d'administration (1)

---

Django met à la disposition des programmeurs ainsi qu'aux administrateurs du site web une interface offrant toutes les opérations CRUD pour tous les modèles définis dans les applications du projet Django.

- Création d'un utilisateur administrateur

```
python manage.py createsuperuser
```

Username: admin

Email address: admin@example.com

Password: \*\*\*\*

# L'interface d'administration (2)

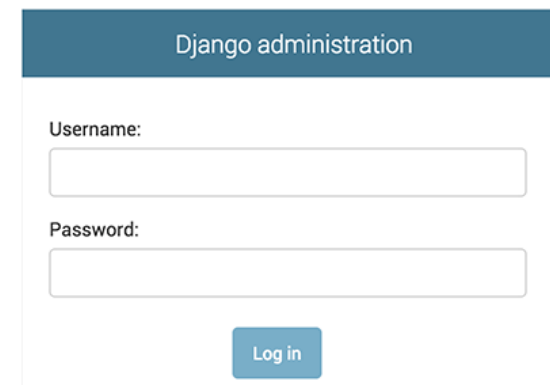
- Démarrage du serveur de développement

```
python manage.py runserver
```

- Lancement de l'interface d'administration

Dans un navigateur lancer : *http://127.0.0.1:8000/admin/*

- Apparition de l'interface de login



The screenshot shows the Django administration login page. It has a dark blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". At the bottom right, there is a blue button labeled "Log in".

# L'interface d'administration (3)

L'interface d'administration donne accès à la gestion des utilisateurs et groupes et aussi aux entités manipulées dans toutes les applications du site.

## Administration du site

Gestion des utilisateurs  
Et des groupes



### AUTHENTIFICATION ET AUTORISATION

Groupes

+ Ajouter    ✎ Modifier

Utilisateurs

+ Ajouter    ✎ Modifier

CRUD des modèles  
de l'application catalogue



### CATALOGUE

Personnes

+ Ajouter    ✎ Modifier

Produits

+ Ajouter    ✎ Modifier

CRUD des modèles  
de l'application eMag



### EMAG

Catégories

+ Ajouter    ✎ Modifier

Produits

+ Ajouter    ✎ Modifier



# L'interface d'administration (4)

## Gestion des groupes

Administration de Django BIENVENUE, DELL. VOIR LE

Accueil > Authentification et autorisation > Groupes > Ajouter groupe

Ajout de groupe

Nom :

Permissions :

permissions disponible(s) ?

Q Filtre

- admin | entrée d'historique | Can add log entry
- admin | entrée d'historique | Can change log entry
- admin | entrée d'historique | Can delete log entry
- admin | entrée d'historique | Can view log entry
- auth | groupe | Can add group
- auth | groupe | Can change group
- auth | groupe | Can delete group
- auth | groupe | Can view group
- auth | permission | Can add permission
- auth | permission | Can change permission
- auth | permission | Can delete permission
- auth | permission | Can view permission
- auth | utilisateur | Can add user
- auth | utilisateur | Can change user

Tout choisir

Choix des « permissions » ?

- magasin | produit | Can view produit
- magasin | produit | Can add produit
- magasin | categorie | Can view categorie
- magasin | categorie | Can add categorie
- magasin | categorie | Can change categorie

Tout enlever

Choix du nom  
Du groupe

Choix des permissions  
Accordées au groupe groupe

# L'interface d'administration (5)

## Gestion des utilisateur

Administration de Django

BIENVENUE **DELL** [VOIR LE SITE](#) / [MODIFIER LE MOT DE PASSE](#) / [DÉCONNEXION](#)

[Accueil](#) > [Authentification et autorisation](#) > [Utilisateurs](#) > [Ajouter utilisateur](#)

### Ajout de utilisateur

Saisissez tout d'abord un nom d'utilisateur et un mot de passe. Vous pourrez ensuite modifier plus d'options.

Nom d'utilisateur :

Requis, 150 caractères maximum. Uniquement des lettres, nombres et les caractères « @ », « . », « + », « - » et « \_ ».

Mot de passe :

Votre mot de passe ne peut pas trop ressembler à vos autres informations personnelles.

Votre mot de passe doit contenir au minimum 8 caractères.

Votre mot de passe ne peut pas être un mot de passe couramment utilisé.

Votre mot de passe ne peut pas être entièrement numérique.

Confirmation du mot de passe :

Saisissez le même mot de passe que précédemment, pour vérification.

Enregistrer et ajouter un nouveau

Enregistrer et continuer les modifications

ENREGISTRER

Choix du nom  
De l'utilisateur

Choix du mot  
de passe

# L'interface d'administration (6)

Informations personnelles  
De l'utilisateur

Choix des permissions

## Gestion des utilisateur

Accueil · Authentification et autorisation · Utilisateurs · gestionnaire

✔ L'objet utilisateur « gestionnaire » a été ajouté avec succès. Vous pouvez l'éditer à nouveau ci-dessous.

### Modification de utilisateur

Nom d'utilisateur :   
Requis. 150 caractères maximum. Uniquement des lettres, nombres et les caractères - @ \_ . , + = & \* ~

Mot de passe : **algorithme: pbkdf2\_sha256 itérations: 180000 salaire: UWv3Dx\*\*\*\*\* empreinte: T8yxvp\*\*\*\*\***  
Les mots de passe ne sont pas enregistrés en clair, ce qui ne permet pas d'afficher le mot de passe de cet utilisateur, mais il est possible de le changer en utilisant ce formulaire.

### Informations personnelles

Prénom :

Nom :

Adresse électronique :

### Permissions

☒ Actif  
Précise si l'utilisateur doit être considéré comme actif. Décochez ceci plutôt que de supprimer le compte.

☐ Statut équipe  
Précise si l'utilisateur peut se connecter à ce site d'administration.

☐ Statut super-utilisateur  
Précise que l'utilisateur possède toutes les permissions sans les assigner explicitement.

Groupes :

Choix des « groupes »

**NB :** assurez-vous que les attributs **is\_active** et **is\_staff** de votre compte sont définis à **True**. Le site d'administration permet uniquement l'accès aux utilisateurs qui ont ces deux champs définis à **True**.

# L'interface d'administration (7)

## Gestion des utilisateur

Permissions de l'utilisateur :

permissions de l'utilisateur disponible(s) ⓘ

🔍 Filtrer

- admin | entrée d'historique | Can add log entry
- admin | entrée d'historique | Can change log entry
- admin | entrée d'historique | Can delete log entry
- admin | entrée d'historique | Can view log entry
- auth | groupe | Can add group
- auth | groupe | Can change group
- auth | groupe | Can delete group
- auth | groupe | Can view group
- auth | permission | Can add permission
- auth | permission | Can change permission
- auth | permission | Can delete permission
- auth | permission | Can view permission
- auth | utilisateur | Can add user

Tout choisir ⓘ

Choix des « permissions de l'utilisateur » ⓘ

⌕ Tout enlever

Permissions spécifiques à cet utilisateur. Maintenez appuyé « Ctrl » ou « Commande (touche pomme) » sur un Mac, pour en sélectionner plusieurs.

Dates importantes

Dernière connexion :

Date :  Aujourd'hui 📅

Heure :  Maintenant ⌚

Note : l'heure du serveur précède votre heure de 1 heure.

Date d'inscription :

Date :  05/02/2021 Aujourd'hui 📅

Heure :  13:53:16 Maintenant ⌚

Note : l'heure du serveur précède votre heure de 1 heure.

Supprimer

# L'interface d'administration (8)

## Gestion des utilisateur

En cas d'oubli du mot de passe d'un utilisateur admin peut lui affecter un nouveau mot de passe soit graphiquement par l'interface Admin ou par commandes sur le shell python



The screenshot shows the Django Admin interface for changing the password of a user named 'DSI21Module1'. The breadcrumb trail is 'Administration de Django > Accueil > Authentification et autorisation > Utilisateurs > DSI21Module1 > Changer le mot de passe'. The page title is 'Modifier le mot de passe : DSI21Module1'. Below the title, it says 'Saisissez un nouveau mot de passe pour l'utilisateur DSI21Module1 .'. There are two password input fields. The first field is labeled 'Mot de passe:' and has a small icon of a key. Below it, there are four lines of feedback text: 'Votre mot de passe ne doit pas être trop similaire à vos autres informations personnelles.', 'Votre mot de passe doit contenir au moins 8 caractères.', 'Votre mot de passe ne peut pas être un mot de passe couramment utilisé.', and 'Votre mot de passe ne peut pas être entièrement numérique.'. The second field is labeled 'Mot de passe (encore):' and has a small icon of a key. Below it, it says 'Entrez le même mot de passe qu'avant, pour vérification.'.

```
# Lancement du shell
python manage.py shell
# Chargement du modèle User
from django.contrib.auth.models import User
# Récupération de l'objet correspondant à notre
utilisateur root
u = User.objects.get(username='root')
# Changement du mot de passe
u.set_password('mon_mot_de_passe_res_complicue')
# Sauvegarde
u.save()
# On quitte le shell avec exit()
```

# L'interface d'administration (9)

---

## Ajout d'un produit

L'opération de mise à jour (CRUD) d'un modèle dans l'interface d'administration doit être précédée par l'enregistrement de ce modèle dans le fichier **admin.py** de l'application.

```
from django.contrib import admin

# Register your models here.
from .models import Produit,...

admin.site.register(Produit)
admin.site.register(...)
```

# L'interface d'administration (10)

## Ajout d'un produit

L'opération d'ajout d'un produit est gérée par l'interface suivante:



Administration de Django BIENVENUE, DELL. [VOIR LE SITE](#) / [MODIFIER LE MOT DE PASSE](#) / [DÉCONNEXION](#)

[Accueil](#) > [Magasin](#) > [Produits](#) > [Ajouter produit](#)

### Ajout de produit

Libelle :

Description :

Cat :   

Img :  Aucun fichier choisi

# L'interface d'administration (11)

## Suppression d'un produit

Administration de Django

Accueil » Magasin » Produits

Sélectionnez l'objet produit à changer

Action :   1 sur 4 sélectionné

<input type="checkbox"/>	PROD	Supprimer les produits sélectionnés
<input type="checkbox"/>	serviette serviette de bain 70x50	
<input type="checkbox"/>	table meuble de cuisine	
<input type="checkbox"/>	yaourt délice mamzouj	
<input checked="" type="checkbox"/>	portable samsung y7 téléphone	

4 produits