## Objectifs de la feuille

- La console d'administration DJANGO
- Rendre modifiable son application
- Les actions du CRUD
- Personnalisation de sa console d'administration
- DJANGO Toolbar

## Introduction au site d'administration de DJANGO

La génération de sites d'administration pour votre équipe ou vos clients pour ajouter, modifier et supprimer du contenu est un travail pénible qui ne requiert pas beaucoup de créativité. C'est pour cette raison que DJANGO automatise entièrement la création des interfaces d'administration pour les modèles. DJANGO a été écrit dans un environnement éditorial, avec une très nette séparation entre les « éditeurs de contenu » et le site « public ». Les gestionnaires du site utilisent le système pour ajouter des nouvelles, des histoires, des événements, des résultats sportifs, etc., et ce contenu est affiché sur le site public. DJANGO résout le problème de création d'une interface uniforme pour les administrateurs du site qui éditent le contenu.
L'interface d'administration n'est pas destinée à être utilisée par les visiteurs du site ; elle est conçue pour les gestionnaires du site.

Nous avons d'abord besoin de créer un utilisateur qui peut se connecter au site d'administration. Lancez la commande suivante : `manage.py createsuperuser`

Saisissez le nom d'utilisateur souhaité et appuyez sur retour. On vous demande alors de saisir l'adresse de courriel souhaitée. L'étape finale est de saisir le mot de passe. On vous demande de le saisir deux fois, la seconde fois étant une confirmation de la première.

Le site d'administration de DJANGO est activé par défaut. Lançons le serveur de développement et explorons-le. À présent, ouvrez un navigateur Web et allez à l'URL « /admin/ » de votre domaine local – par exemple, *http://127.0.0.1:8000/admin/*. Vous devriez voir l'écran de connexion à l'interface d'administration .

Essayez maintenant de vous connecter avec le compte administrateur que vous avez créé à l'étape précédente. Vous devriez voir apparaître la page d'accueil du site d'administration de DAJNGO. Vous devriez voir quelques types de contenu éditable : groupes et utilisateurs. Ils sont fournis par `django.contrib.auth`, le système d'authentification livré avec DJANGO.

## Rendre l'application modifiable via l'interface d'admin

Mais où est notre application monapp ? Elle n'est pas affichée sur la page d'index de l'interface d'administration. Plus qu'une chose à faire : il faut indiquer à l'admin que les objets Product ont une interface d'administration. Pour ceci, ouvrez le fichier monapp/admin.py et éditez-le de la manière suivante :

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

Actualisez votre page et c'est magique !
Pour créer une entrée avec l'interface d'administration Cliquons sur Products. On remarque la présence d'un bouton Ajouter Product, ce qui permet d'ajouter une entrée dans la base pour ce modèle. Cliquons dessus. Remplissons les champs que nous avons définis dans notre modèle puis cliquons sur Enregistrer. Voila nous avons crée un produit en quelques clics sans créer aucune interface.

Vous pouvez bien évidemment modifier l'item que vous voulez. Cliquons sur une entrée au hasard et modifiez le code du produit par exemple, puis validez la sauvegarde en restant sur la fiche produit via le bouton Enregistrer et continuer les modifications. Ensuite, cliquons sur le bouton Historique. On y voit un historique complet sur les modifications de l'entrée, ainsi que le nom de l'utilisateur qui a fait les changements.

À noter ici :

- Le formulaire est généré automatiquement à partir du modèle Product.
- Les différents types de champs du modèle (DateTimeField, CharField) correspondent au composant graphique d'entrée HTML approprié. Chaque type de champ sait comment s'afficher dans l'interface d'administration de DJANGO.
- Chaque DateTimeField reçoit automatiquement des raccourcis Javascript. Les dates obtiennent un raccourci « Aujourd'hui » et un calendrier en popup, et les heures obtiennent un raccourci « Maintenant » et une popup pratique qui liste les heures couramment saisies.

La partie inférieure de la page vous propose une série d'opérations :

- Enregistrer – Enregistre les modifications et retourne à la page liste pour modification de ce type d'objet.
- Enregistrer et continuer les modifications – Enregistre les modifications et recharge la page d'administration de cet objet.
- Enregistrer et ajouter un nouveau – Enregistre les modifications et charge un nouveau formulaire vierge pour ce type d'objet.
- Supprimer – Affiche une page de confirmation de la suppression.

## Qu'est-ce que le CRUD ?

L′ interface d′administration DJANGO est CRUD (Create Read Update et Delete), c′est à dire qu′elle est capable de faire les opérations de bases :

```
/admin/monapp/                    => liste les modèles de l'application
/admin/monapp/product             => liste les objets du modèle
/admin/monapp/product/add         => crée un nouveau objet
/admin/monapp/product/<id>/       => consulter un objet en particulier
/admin/monapp/product/<id>/delete => supprime un objet en particulier
/admin/monapp/product/<id>/change => modifie un objet en particulier
```

C′est cool, mais quel est le but du site d′administration ?

Le site d′administration est un endroit où les différents modèles d′un projet DJANGO peuvent être gérés par, eh bien, des administrateurs ! Mais qui entendons-nous par là ? Pour commencer, les administrateurs ce sera vous et les autres développeurs du projet. Mais vous pouvez éventuellement confier le projet à un client : peut-être le propriétaire d′un magasin ou l′auteur d′un blog, des personnes qui ne sont pas des programmeurs et qui ne peuvent donc pas utiliser le shell DJANGO pour créer de nouveaux objets. Les non-programmeurs seraient perdus sans une interface permettant d′effectuer des opérations CRUD. Et vous devrez passer un temps considérable à construire et à tester cette interface pour eux. C′est pourquoi vous devriez vous réjouir de cette fonctionnalité, car DJANGO l′a conçue pour vous !

Cela signifie-t-il que vous n′aurez jamais à construire vos propres formulaires ? Pas tout à fait ! Si le site d′administration est très utile, il ne faut pas oublier que son public principal est constitué d′administrateurs et non d′utilisateurs finaux. Il s′agit d′une interface « back-end ». Il offre des fonctionnalités bien supérieures à celles auxquelles les utilisateurs finaux devraient avoir accès. Il est aussi très simple d′aspect.

Vos utilisateurs finaux s′attendront à utiliser des formulaires dont le style est identique à celui du reste de votre site. Vous pouvez également personnaliser la position d′un formulaire dans une page, exclure certains champs d′un formulaire et effectuer d′autres personnalisations de l′interface utilisateur. Pour ces raisons, vous devrez encore apprendre à intégrer des formulaires dans le front-end de votre application web dans les chapitres suivants.

## Personnaliser son interface d'administration

Il existe une batterie d'options qui vous permettront de mettre en avant les informations qui vous intéresse. Si nous voulons la liste des Product via deux colonnes, name et code, éditez monapp/admin.py de la manière suivante :

```
from django.contrib import admin
from .models import Product

class ProductAdmin(admin.ModelAdmin):
   list_display = ('name', 'code')

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

Les options de ModelAdmin : https://docs.djangoproject.com/fr/5.1/ref/contrib/admin/#modeladmin-options

Chargez le nouveau modèle disponible sur Célène. N'oubliez pas les étapes de migrations. Nous avons tous le même modèle dorénavant. Faites en sorte que toutes les entités du modèle s'affichent dans la console d'administration en rajoutant dans le fichier monapp/admin.py les lignes suivantes :

```
admin.site.register(ProductItem)
admin.site.register(ProductAttribute)
admin.site.register(ProductAttributeValue)
```

A l'aide de la console d'administration, créez deux attributs de produits «couleur» et «taille écran».
Pour chacun d'entre eux, créez des valeurs (bleu, blanc et rouge pour la couleur, 5 et 8 pouces pour la taille écran par exemple).

Nous voulons pouvoir créer des déclinaisons de produits directement sur la fiche produit. Le fichier monapp/admin.py est donc modifié ainsi :

```
from .models import Product, ProductAttribute, ProductAttributeValue, ProductItem

class ProductItemAdmin(admin.TabularInline):
        model = ProductItem

class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('name', 'code')
        inlines = [ProductItemAdmin,]

admin.site.register(Product, ProductAdmin)
admin.site.register(ProductItem)
admin.site.register(ProductAttribute)
admin.site.register(ProductAttributeValue)
```

Voila notre fiche produit avec possibilité de créer des déclinaisons très simplement. La première chose que l'on remarque c'est la facilité du code pour créer une interface complète avec ajout / modification / suppression / contrôles de données. Il nous aura fallu peu de lignes de code, sans compter les modèles. La seconde chose que l'on remarque c'est le select multiple pas du tout user-friendly pour le champ many-to-many «attributes» dans le cas où les attributs dépassent la centaine d'items. Il est possible de changer ce widget avec l'option filter_vertical :

```
class ProductItemAdmin(admin.TabularInline):
        model = ProductItem
        filter_vertical = ("attributes",)
```

Alors évidemment aucun widget n'est meilleur que l'autre mais l'un peut être meilleur que l'autre dans des situations différentes.
Éditez maintenant les colonnes « price_ht » et « price_ttc » directement dans le listing. Il faut également modifier le champ « list_display » en conséquence :

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        inlines = [ProductItemAdmin,]
        list_display = ["id", "name", "price_ht", "price_ttc", "code"]
        list_editable = ["name", "price_ht", "price_ttc"]
```

Renseignez les prix suivants et enregistrer:

- 200€ HT et 210€ TTC pour l'ipad
- 1000€ HT et 1200€ TTC pour l'iphone
- 80€ HT et 86€ TTC pour l'ipod

Vous pouvez remplacer un select par des boutons radio comme ceci:

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        inlines = [ProductItemAdmin,]
        list_display = ["id", "name", "price_ht", "price_ttc", "code"]
        list_editable = ["name", "price_ht", "price_ttc"]
        radio_fields = {"status": admin.VERTICAL}
```

Il est possible d'ajouter une searchbox qui effectue des recherches sur plusieurs champs:

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('id', 'name', 'date_creation', 'status')
        search_fields = ('name', 'status')
```

Notre exemple de recherche précédent ressemble assez à du bricolage. Comment l'utilisateur lambda peu-il savoir que le "online" est définit par la valeur 1 ? Il existe une autre option qui vous permet de filtrer les données plus proprement pour ce genre de cas:

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('id', 'name', 'date_creation', 'status')
        list_filter = ('status', 'date_creation')
```

Vous pouvez affiner votre filtre en utilisant la classe admin.SimpleListFilter:

```
class ProductFilter(admin.SimpleListFilter):

        title = 'filtre produit'
        parameter_name = 'custom_status'

        def lookups(self, request, model_admin) :
           return (
              ('online', 'En ligne'),
              ('offline', 'Hors ligne'),
        )

         def queryset(self, request, queryset):
                if self.value() == 'online':
                return queryset.filter(status=1)
                if self.value() == 'offline':
                return queryset.filter(status=0)

class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('id', 'name', 'date_creation', 'status')
        list_filter = (ProductFilter,)
```

Vous pouvez hiérarchiser par date :

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('id', 'name', 'date_creation', 'status')
        date_hierarchy = 'date_creation'
```

Vous remarquerez la présence d'un lien au dessus des actions qui vous permet de naviguer / filtrer par des intervalles de date. Vous pouvez trier par défaut en utilisant le signe - pour indiquer un ordre décroissant :

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ('id', 'name', 'date_creation', 'status')
        ordering = ('-date_creation',)
```

Vous pouvez ajouter une action dans la liste des actions très simplement avec le module admin. Par exemple nous voulons changer le statut des produits sélectionnés en « online » « offline»:

```
def set_product_online(modeladmin, request, queryset):
        queryset.update(status=1)
set_product_online.short_description = "Mettre en ligne"

def set_product_offline(modeladmin, request, queryset):
        queryset.update(status=0)
set_product_offline.short_description = "Mettre hors ligne"

class ProductAdmin(admin.ModelAdmin):
  model = Product
  actions = [set_product_online, set_product_offline]
```

Je sélectionne l'action puis je clique sur envoyer. Pour les produits sélectionnés, le statut aura

- la valeur 1 comme il est définit dans la fonction set_product_online que j'ai crée.
- la valeur 0 comme il est définit dans la fonction set_product_offline que j'ai crée.

Vous pouvez créer une colonne personnalisée et y mettre les informations que vous voulez :

```
class ProductAdmin(admin.ModelAdmin):
        model = Product
        list_display = ["id", "name", "price_ht", "price_ttc","tax"]
        list_editable = ["name", "price_ht", "price_ttc"]

        def tax(self, instance):
                return (instance.price_ttc / instance.price_ht)-1
        tax.short_description = "Taxes (%)"
```

A noter que la colonne n'est pas triable : `tax.admin_order_field = "price_ht"`

## Installation de DJANGO DEBUG TOOLBAR

`Commande :` pip install django-debug-toolbar

Dans le fichier monprojet/settings.py:

```
INSTALLED_APPS = [
...,
'debug_toolbar',
]

MIDDLEWARE = [
# ...
"debug_toolbar.middleware.DebugToolbarMiddleware",
# ...
]

INTERNAL_IPS = [
# ...
"127.0.0.1",
# ...
]
```

Dans le fichier monprojet/urls.py du projet:

```
urlpatterns = [
path("monapp/", include("monapp.urls")),
path('admin/', admin.site.urls),
path("__debug__/", include("debug_toolbar.urls")),
]
```

DJANGO TOOLBAR est maintenant visible.

Nous allons pouvoir maintenant parler optimisation. Reprenons le fichier monapp/admin.py et nos deux classes ProductItemAdmin et ProductAdmin :

```
class ProductItemAdmin(admin.TabularInline):
        model = ProductItem
        filter_vertical = ("attributes",)

class ProductAdmin(admin.ModelAdmin):
        model = Product
        inlines = [ProductItemAdmin,]
        list_filter = (ProductFilter,)
        date_hierarchy = 'date_creation'
        actions = [set_product_online, set_product_offline]
        list_display = ["code", "name", "price_ht", "price_ttc","tax"]
        list_editable = ["name", "price_ht", "price_ttc"]

        def tax(self, instance):
                return ((instance.price_ttc / instance.price_ht)-1)*100

        tax.short_description = "Taxes (%)"
        tax.admin_order_field = "price_ht"
```

Rendons-nous sur l'url http://127.0.0.1:8000/admin/monapp/product/3/change/.
Dans la DJANGO TOOLBAR, il faut 37 requêtes SQL pour afficher la page (9.52 ms)
Rendons-nous sur l'url http://127.0.0.1:8000/admin/monapp/product/2/change/.
Dans la DJANGO TOOLBAR, il faut 36 requêtes SQL pour afficher la page (3.72 ms)

Changeons la manière d'afficher les attributs par leur id, et non plus par leur couple attributs/valeur avec la ligne suivante :

```
class ProductItemAdmin(admin.TabularInline):
        model = ProductItem
        filter_vertical = ("attributes",)
        raw_id_fields = ["attributes"]
```

Dans certain cas il peut être intéressant de ne pas afficher un select mais plutôt un input simple . Dans cet input il faudra indiquer la clé de l'item que l'on désire associer au lieu de le sélectionner manuellement dans une liste. Le cas le plus évident sera les cas où les items seraient si nombreux que le chargement de la page serait trop lent. Cette option permet donc de répondre à ce genre de besoin. Vérifier le gain de temps et le nombre de requêtes…