ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Révision

Créez vous un projet en Django nommé Revision, dedans ce projet créez vous une application nommée lesTaches puis ajoutez cette application dans le fichier settings.py de votre projet avant de commençer la suit de cet atelier.

La création de formulaires avec Django est assez simple et pleine de fonctionnalités. Vous pouvez générer des formulaires à partir de vos modèles ou bien les créer directement depuis des classes. On peut également utiliser des formulaires avec un style plus élaboré comme Crispy Forms.

Exercice 1. Création d'une nouvelle app dans notre projet

Passons tout de suite à la pratique, et mettons cela en place. Pour plus de propreté, nous allons créer une nouvelle app dans notre projet, que nous appellerons 'myform'.

Reprenez le projet GestionTaches après avoir initialisé votre environnement virtuel et créé l'application myform

```
./manage.py startapp myform
```

Comme vous l'avez fait avec lesTaches ajouter cette application au fichier `settings.py` :

```
INSTALLED_APPS = (
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.sites',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'django.contrib.admin',
  'django.contrib.admindocs',
  'lesTaches',
  'myform',
  )
```

Exercice 2. Modèle de myform

On va à présent créer le modèle de notre application, on ouvre pour cela le fichier `models.py` contenu dans le dossier de notre application nommée 'myform' et on l'édite comme ceci :

```python
from django.db import models


class Contact(models.Model):
  name = models.CharField(max_length=200)
  firstname = models.CharField(max_length=200)
  email = models.EmailField(max_length=200)
  message = models.CharField(max_length=1000)
```

On crée donc une classe 'Contact' qui contient un nom, un prénom, un email et un message, puis :

```
# Générons la en base de données:
./manage.py makemigrations myform
# vérifions le contenu de la table avec
./manage.py sqlmigrate myform 0001
# et réalisons la migration
./manage.py migrate
```

Et voilà, notre modèle est présent en base de données, maintenant passons à la création de notre contrôleur afin de générer notre formulaire. On ouvre pour cela le fichier `views.py` présent dans notre application, et on l'édite :

```python
from django.shortcuts import render
from django.forms import ModelForm
from .models import Contact

class ContactForm(ModelForm):
  class Meta:
      model = Contact
      fields = ('name', 'firstname', 'email', 'message')

def contact(request):

  contact_form = ContactForm()
  return render(request,'contact.html', {'contact_form' :
     contact_form})
```

On importe notre modèle ainsi qu'un 'helper', la méthode render(), ainsi que la classe 'ModelForm'. On définit alors notre classe qui va permettre de générer notre formulaire en la faisant dériver de ModelForm et en lui spécifiant le modèle à inclure "Contact". Il ne nous reste plus qu'à déclarer une nouvelle variable qui sera une instance de la nouvelle classe créée et à la passer en tant que donnée à notre template.

Exercice 3. Routes pour la nouvelle app myform

Maintenant il va falloir créer les urls correspondantes :

Tout d'abord dans GestionTaches ajoutons dans `urls.py` la ligne suivante :

```
1 path('contacts/',include('myform.urls')),
```

et créons le fichier `urls.py` dans le répertoire `myform` :

```
1 from django.urls import path
2 from . import  views
3 urlpatterns=[
4   path('',views.contact,name='contact'),
5 ]
```

Créons maintenant notre template qui se nommera `contact.html` dans `myform/templates/` que l'on remplit comme ceci :

```
1 {{ contact_form.as_p }}
```

Voilà ! Pas plus !

Nous affichons ici directement notre objet contact_form et on lui associe une option d'affichage 'as_p' qui signifie que le formulaire sera affiché en utilisant la balise '<p>'

Exercice 4. Test du premier formulaire

Nous pouvons à présent apprécier le résultat ! Lancez votre serveur :

```
   ./manage.py runserver
```

Et dirigez-vous à l'adresse : `http://localhost:8000/contacts/` Vous devez obtenir un formulaire (moche) avec les libellés de la classe Contact.

Notre formulaire html utilisant des balises '<p>' a été automatiquement généré ! Pour l'instant il reste encore assez simple, mais nous allons voir que nous allons pouvoir très vite tout personnaliser et avoir un bien meilleur rendu ! D'autres options d'affichage telles que 'as_ul' et 'as_table auraient pu être utilisées.

Pour plus d'informations sur les forms, voir la doc officielle : `https://docs.djangoproject.com/fr/3.1/topics/forms/`

Exercice 5. Création d'un formulaire à la main (sans Model)

Mais avant cela, nous allons nous intéresser à la création de formulaires sans utiliser de modèle, directement avec le module forms de Django ! Pour ceux qui souhaitent approfondir la création de formulaires à partir de modèles, voir la documentation officielle. Créons maintenant notre formulaire `ContactForm2` sans utiliser de modèle, en modifiant comme suit le `views.py` de l'application 'myform' :

```
1  from django.shortcuts import render
2  from django.forms import ModelForm
3  from models import Contact
4
5  # Avec ModelForm
6  class ContactForm(ModelForm):
7    class Meta:
8        model = Contact
9        fields = ('name', 'firstname', 'email', 'message')
10
11 from django import forms
12
13 # Sans ModelForm
14 class ContactForm2(forms.Form):
15   name = forms.CharField(max_length=200)
16   firstname = forms.CharField(max_length=200)
17   email = forms.EmailField(max_length=200)
18   message = forms.CharField(max_length=1000)
19
20 def contact(request):
21   contact_form = ContactForm()
22   contact_form2 = ContactForm2()
23   return render(request,'contact.html', {'myform/
       contact_form' : contact_form, 'contact_form2' :
       contact_form2})
```

On inclut directement le module 'forms' de django et on crée une classe 'ContactForm2' dérivant de Form et non de ModelForm cette fois-ci. On crée ainsi notre classe, ce qui ressemble sensiblement à la déclaration de notre modèle. Puis on crée un objet contact_form2 dans notre fonction et on le passe en donnée de notre template. Maintenant éditez votre fichier `contact.html` de cette façon :

```
1 {{contact_form.as_p  }}
2 <br/>
3 {{ contact_form2.as_p }}
```

On obtient donc deux formulaires identiques créés de deux façons différentes !

Exercice 6. Personnalisation des formulaires et vérifications

Intéressons-nous maintenant à la personnalisation de ces formulaires avant de comprendre comment les utiliser et gérer leur validation. Les formulaires de type **ModelForm** ainsi que ceux de type **Form** peuvent contenir de nombreuses options de personnalisation. On peut tout de suite remarquer certaines options telles que : required, label, initial, widget, help_text, error_messages, validator, localize `required` va spécifier si le champ du formulaire peut être ignoré ou non, par défaut tous les champs sont requis. Cela implique que si l'utilisateur valide le formulaire avec des champs à vide ("" ou None), une erreur sera levée. Pour illustrer cela, voici ce que donnent les tests suivants après avoir ouvert une console python en mode django et avoir importé le module forms :

Ouvrons un shell et testons :

```
>>>from django inport forms
>>> f=forms.CharField()
>>> f.clean('foo')
'foo'
>>> f.clean('')
''
ValidationError: ['This field is required.'] ...
>>> f = forms.CharField(required=False)
>>> f.clean(None)
''
ValidationError: ['This field is required.'] ...
```

Nous simulons ici la vérification de formulaire avec la méthode `clean()`. Comme vous pouvez le constater, une erreur est bien levée si nous passons une chaîne vide ou `None` à notre champ. `label` va permettre de réécrire la façon dont le label du champ est affiché, par défaut, cela prend le nom de la variable et lui ajoute une majuscule à la première lettre et remplace tous les underscores par des espaces. Exemple :

```
>>>class CommentForm(forms.Form):
...     name = forms.CharField(label='Your name')
...     url = forms.URLField(label='Your Web site', required
    =False)
...     comment = forms.CharField()
...
>>> f = CommentForm()
>>> print(f)
<tr><th>
<label for="id_name">Your name:</label>
</th>
<td><input type="text" name="name" id="id\_name" />
```

```
12 </td></tr>
13 <tr><th>
14 <label for="id_url">Your Web site:</label>
15 </th><td>
16 <input type="text" name="url" id="id_url" /></td></tr>
17 <tr><th><label for="id_comment">Comment:</label>
18 </th><td>
19 <input type="text" name="comment" id=">>>
20 f = forms.CharField() /></td></tr>
```

Vous voyez également qu'ici on affiche notre 'form' sans utiliser d'option d'affichage, il est donc affiché par défaut en utilisant une table. Pour plus de commodité, on aurait pu rajouter l'option 'auto_id' à 'False', ce qui aurait pour effet de désactiver la création d'une balise label, et génèrerait ceci :

```
1  >>>f = CommentForm(auto_id=False)
2  >>> print(f)
3  <tr><th>Your name:</th>
4  <td><input type="text" name="name" /></td></tr>
5  <tr><th>Your Web site:</th><td>
6  <input type="text" name="url" />
7  </td></tr>
8  <tr><th>Comment:</th><td>
9  <input type="text" name="comment" />
10 </td></tr>
```

initial va permettre de donner des valeurs par défaut à vos champs, exemple :

```
1   >>>class CommentForm(forms.Form):
2  ...      name = forms.CharField(initial='Your name')
3  ...      url = forms.URLField(initial='http://')
4  ...      comment = forms.CharField()
5  >>> f = CommentForm(auto_id=False)
6  >>> print f
7  <tr><th>Name:</th><td>
8  <input type="text" name="name" value="Your name" />
9  </td></tr>
10 <tr><th>Url:</th><td>
11 <input type="text" name="url" value="http://" /></td></tr>
12 <tr><th>Comment:</th><td>
13 <input type="text" name="comment" />
14 </td></tr>
```

Nous nous arrêterons à ces trois options pour le moment et verrons les autres dans des cas plus poussés. Par contre il est très intéressant de se pencher sur l'option **'widget'** car elle va vous permettre de réaliser très facilement des champs spécifiques.

```
1 django import forms
2 class CommentForm(forms.Form):
3   name = forms.CharField()
4   url = forms.URLField()
5   comment = forms.CharField(widget=forms.Textarea)
```

Ou bien :

```
1 from django.forms.fields import DateField, ChoiceField,
     MultipleChoiceField
2 from django.forms.widgets import RadioSelect,
     CheckboxSelectMultiple
3 from django.forms.extras.widgets import SelectDateWidget
4
5 BIRTH_YEAR_CHOICES = ('1999', '2000', '2001')
6 GENDER_CHOICES = (('m', 'Male'), ('f', 'Female'))
7 FAVORITE_COLORS_CHOICES = (('blue', 'Blue'),
8                           ('green', 'Green'),
9                           ('black', 'Black'))
10
11 class SimpleForm(forms.Form):
12   birth_year = DateField(widget=SelectDateWidget(years=
       BIRTH_YEAR_CHOICES))
13   gender = ChoiceField(widget=RadioSelect, choices=
       GENDER_CHOICES)
14   favorite_colors = forms.MultipleChoiceField(required=False
       ,
15       widget=CheckboxSelectMultiple, choices=
           FAVORITE_COLORS_CHOICES)
```

Essayez-les !
Un petit dernier ...

```
1 class CommentForm(forms.Form):
2   name = forms.CharField(
3               widget=forms.TextInput(attrs={'class':'special
                   '}))
4   url = forms.URLField()
5   comment = forms.CharField(
6               widget=forms.TextInput(attrs={'size':'40'}))
```

Ce qui donnera :

```
1    f = CommentForm(auto_id=False)
2 >>> f.as_table()
3 <tr><th>Name:</th><td>
4 <input type="text" name="name" class="special"/></td></tr>
5 <tr><th>Url:</th><td>
6 <input type="text" name="url"/></td></tr>
7 <tr><th>Comment:</th><td>
8 <input type="text" name="comment" size="40"/></td></tr>
```

N'hésitez pas à parcourir les différents types de widgets dans la doc de django et à les essayer pour découvrir ce qu'ils produisent.

Exercice 7. ModelForms

A noter que vous pouvez également utiliser les widgets avec des Form venant de Modèles : Exemple :

```
1  from django.forms import ModelForm, Textarea
2
3 class AuthorForm(ModelForm):
4   class Meta:
5       model = Author
6       fields = ('name', 'title', 'birth_date')
7       widgets = {
8           'name': Textarea(attrs={'cols': 80, 'rows': 20}),
9       }
```

Modifier les formulaires de l'application myform en utilisant les informations ci-dessus. Tester par exemple en modifiant la vue de myform comme ci-dessous :

```
1 from django.shortcuts import render
2 from django.forms import ModelForm, Textarea
3 from       .models import Contact
4 from django import forms
5
6 class ContactForm(ModelForm):
7
8   class Meta:
```

```
 9    model = Contact
10        fields =('name','firstaname','email','message')
11      widgets = {
12
13            'message': Textarea(attrs={'cols':60,'rows':10}),
14
15        }
16
17    class ContactForm2(forms.Form):
18      name = forms.CharField(max_length=200,initial="votre nom
           ",label="nom")
19      firstname = forms.CharField(max_length=200,initial="
           votre prenom",label="prenom")
20      email = forms.EmailField(max_length=200,label="mel")
21      message = forms.CharField(widget=forms.Textarea(attrs={'
           cols':60,'rows':10}))
22
23
24  def contact(request):
25
26    contact_form = ContactForm()
27    #return render(request,'myform/conctact.html',{'
         contact_form':contact_form})
28
29    contact_form2 = ContactForm2()
30    return render(request,'myform/conctact.html',{'
         contact_form':contact_form,'contact_form2':
         contact_form2})
```

Essayez d'autres possibilités.
Enfin, nous allons nous intéresser à la validation des formulaires, comment gérer des messages pour la page suivante et comment travailler avec ceux-ci. Nous allons reprendre le fichier views.py de notre application 'myform' et remplacer son contenu par celui-ci :

```
1  from django.shortcuts import render, redirect
2  from django.forms import ModelForm, Textarea
3  from       .models import Contact
4  from django import forms
5  from django.urls import reverse
6  from django.http import HttpResponse
7  from django.contrib import messages
8
9  class ContactForm(ModelForm):
```

```
10    def __init__(self, *args, **kwargs):
11      super(ContactForm, self).__init__(*args, **kwargs)
12      self.fields['name'].label = "Nom "
13      self.fields['firstname'].label = "Prenom"
14      self.fields['email'].label = "mél"
15    class Meta:
16      model = Contact
17      fields = ('name', 'firstname', 'email','message')
18      widgets = {'message': Textarea(attrs={'cols': 60, 'rows'
           : 10}),}
19
20      def contact(request):
21      # on instancie un formulaire
22      form = ContactForm()
23      # on teste si on est bien en validation de formulaire (
           POST)
24      if request.method == "POST":
25          # Si oui on récupère les données postées
26          form = ContactForm(request.POST)
27          # on vérifie la validité du formulaire
28          if form.is_valid():
29              new_contact = form.save()
30              # on prépare un nouveau message
31              messages.success(request,'Nouveau contact '+
                   new_contact.name+' '+new_contact.email)
32              #return redirect(reverse('detail', args=[
                   new_contact.pk] ))
33              context = {'pers': new_contact}
34              return render(request,'detail.html', context)
35      # Si méthode GET, on présente le formulaire
36      context = {'form': form}
37      return render(request,'contact.html', context)
```

Avec une nouvelle vue `detail` :

```
1    def detail(request, cid):
2    contact = Contact.objects.get(pk=cid)
3    return HttpResponse('Nouveau contact '+contact.name+' '+
        contact.email)
```

et un template detail.html :

```
1  {% extends "base.html" %}
2      {% load static %}
```

```
3       {% block title %}
4       Contacts
5       {% endblock %}
6       {% block header %}
7       {% if messages %}
8           {% for message in messages %}
9               <h3>{{message}}</h3>
10          {% endfor %}
11      {% endif %}
12      {% endblock %}
13
14      {% block content %}
15      <ul>
16          <li>{{pers.firstname}}</li>
17          <li>{{pers.email}}</li>
18          <li>{{pers.message}}</li>
19      </ul>
20      {% endblock %}
```

Et nous allons définir une nouvelle url pour la page "detail".

Pour ce faire on modifiera les fichiers **urls.py** de GestionTaches et myform. Pour tester, il nous faudra également modifier le template contact.html :

```
1  {% extends "base.html" %}
2       {% load static %}
3       {% block title %}
4       Nouveau Contact
5       {% endblock %}
6
7       {% block navtitle %}
8           Nouveau Contact
9       {% endblock %}
10
11      {% block content%}
12          <form action="/contacts/" method="POST">
13          {% csrf_token %}
14          {{ form }}
15          <button type="submit">
16              Sauvegarder
17          </button>
18          </form>
19      {% endblock %}
```

Exercice 8. CrispyForms

Pour bénéficier de formulaires plus jolis, nous allons installer et mettre en marche le module *crispy-forms*. Pour cela ajoutez le à votre venv :

```
pip install django-crispy-forms
```

puis ajoutez crispy-forms aux apps installées dans settings.py :

```
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'lesTaches',
  'myform',
  'crispy_forms',
]
```

Puis lisez la documentation des crispy-forms pour adapter vos formulaires et les rendre plus attrayants ! Pour contact.html par exemple :

```
{% extends "base.html" %}
    {% load static %}
    {% load crispy_forms_tags %}
    {% block title %}
    Nouveau Contact
    {% endblock %}

    {% block navtitle %}
        Nouveau Contact
    {% endblock %}

    {% block content%}
        <form action="/contacts/" method="POST">
        {% csrf_token %}
        {{ form | crispy }}
        <button type="submit" class="btn btn-success">
            Sauvegarder
        </button>
        </form>
    {% endblock %}
```