

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.

Question1 : Créer un nouveau projet Django ou utiliser un projet déjà créé

Question2: Ajouter dans ce projet une application nommé MyContact

Question3 : Vous devrez également créer un modèle que vous appellerez Contact avec les champs suivants :

- firstname
- lastname
- Email
- Message :

Question4: Appliquer les migrations pour créer la base de données

Méthode1 pour créer un formulaire

FirstProject > FirstApp > templates > <> myform1.html

□ Créer un fichier **myform1.html** dans le dossier templates de l'application de votre projet

```
<form action="" method="post">
{% csrf_token %}
<p>
<label >firstname: </label>
<input type="text" name="firstname" >
</p>


<p>
<label >lastname: </label>
<input type="text" name="lastname" >
</p>

<p>
<label >Email: </label>
<input type="email" name="email">
</p>

<p>
<label ">Message: </label>
<textarea type="text" name="message"></textarea>
</p>

<input type="submit" value="envoyer">
</form>
```

□ Ajouter la fonction **controleform1** dans le fichier views.py

FirstProject > FirstApp >  views.py

```
from django.http import HttpResponseRedirect

from django.shortcuts import render
from .models import Contact


# Create your views here.
def controleform1 (request):

    if request.method=='POST':
        f=request.POST['firstname']# firstname c'est name dans page html
        l=request.POST['lastname']
        e=request.POST['email']
        m=request.POST['message']
        ]
        contact=Contact.objects.create(firstname=f,lastname=l,Email=e,msg=m)
        #contact=Contact(firstname=f,lastname=l,Email=e,msg=m)
        #contact.save()
    return HttpResponseRedirect(' <h2> Data has been submitted </h2>')
    return render(request,"myform1.html")
```

□ Editer les fichiers urls pour affecter un path à la fonction controleform1

Méthode2 pour créer un formulaire

□ Dans le dossier du projet ajouter un fichier nommé forms.html

FirstProject > FirstApp >  forms.py >

```
from django import forms
class contactform2(forms.Form):

    firstname=forms.CharField(max_length=10)
    lastname=forms.CharField(max_length=10)
    Email=forms.EmailField()
    msg=forms.CharField(widget=forms.Textarea)
```

□ Ajouter la fonction **controleform2** dans le fichier views.py. N'oubliez pas d'importer la

```
def controleform2 (request):
    if request.method == 'POST': # S'il s'agit d'une requête POST
        form = contactform2(request.POST) # Nous reprenons les données

        if form.is_valid(): # Nous vérifions que les données envoyées sont
            valides

            # Ici nous pouvons traiter les données du formulaire f =
                form.cleaned_data['firstname']
            l = form.cleaned_data['lastname'] e =
                form.cleaned_data['Email']
            m = form.cleaned_data['msg']

            # Nous pourrions ici envoyer l'e-mail grâce aux données que nous
            venons de récupérer

            contact=Contact.objects.create(firstname=f,lastname=l,Email=e,msg=
            m)
        return HttpResponse(' <h2> Data has been submitted </h2>')

        else: # Si ce n'est pas du POST, c'est probablement une requête GET
            form = contactform2() # Nous créons un formulaire vide

        return render(request,"myform2.html",{ 'mycontactform2':form})
```

classe contactform2

□ Créer un fichier **myform2.html** dans le dossier templates de l'application de votre projet

```
<form action="" method="post">
{% csrf_token %}
{{mycontactform2.as_p}}
<input type="submit" value="envoyer">
</form>
```

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>



Lancer le serveur

□ Editer les fichiers urls pour affecter un path à la fonction controleform2

Méthode3 pour créer un formulaire

□ Dans le fichier forms.py ajouter la classe **contactForm3** qui hérite de 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 "

```
FirstProject > FirstApp > views.py > controleform3  
from .models import Contact  
from django.forms import ModelForm  
class contactform3(ModelForm):  
    class Meta:  
        model=Contact  
        fields=('firstname','lastname','Email','msg')
```

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. Pour cela, ajouter la fonction `controleform3` dans le fichier `views.py`

```
def controleform3 (request):  
    if request.method == 'POST': # S'il s'agit d'une requête POST  
        form = contactform3(request.POST) # Nous reprenons les données  
  
        if form.is_valid(): # Nous vérifions que les données envoyées sont  
            #valides  
            form.save()  
            return HttpResponse(' <h2> Data has been submitted </h2>')  
  
        else: # Si ce n'est pas du POST, c'est probablement une requête GET  
            form = contactform3() # Nous créons un formulaire vide  
  
    return render(request,"myform3.html",{ 'mycontactform3':form})
```

□ Créer un fichier **myform3.html** dans le dossier templates de l'application de votre projet

```
<form action="" method="post">  
{% csrf_token %}  
{{mycontactform3.as_p}}  
<input type="submit" value="envoyer">  
</form>
```

□ Editer les fichiers urls pour affecter un path à la fonction `controleform3`

➡ On obtient donc trois formulaires identiques créés de trois façons différentes !

❑ **crispy-forms** est un module python -créé pour Django- qui permet de construire une mise en page avec des composants pour contrôler le rendu HTML sans écrire de template HTML. Nous allons installer et mettre en place 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',  
    'FirstApp',  
    'crispy_forms',  
]  
CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

Dans le fichier HTML ajouter les propriétés de crispy pour améliorer l'affichage du formulaire

```
{%load crispy_forms_tags %}  
<form action="" method="post">  
    {% csrf_token %}  
    {{mycontactform3|crispy }}  
    <input type="submit" value="envoyer">  
</form>
```

❑ Lancer le serveur

- Créer une application sendemail
- Définir la classe ContactForm dans un fichier forms.py

```
from django import forms

class ContactForm(forms.Form):
    from_email = forms.EmailField(required=True)
    subject = forms.CharField(required=True)
    message = forms.CharField(widget=forms.Textarea, required=True)
```

- Editer le fichier views.py et créer la page email.html qui permet d'afficher le formulaire

sendemail >  views.py >

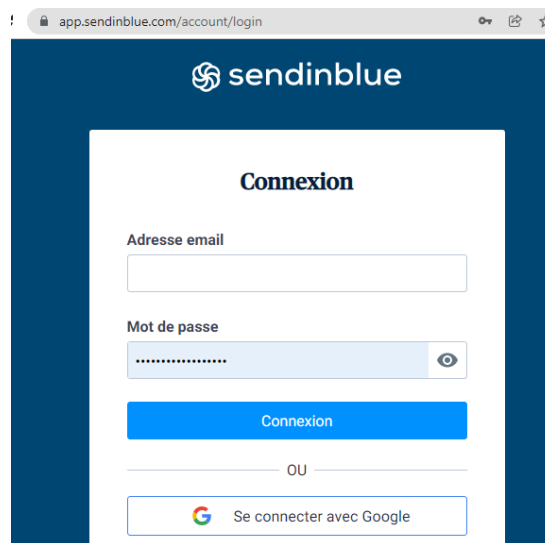
```
from django.core.mail import send_mail, BadHeaderError
from django.http import HttpResponse,
HttpResponseRedirect from django.shortcuts import
render, redirect
from .forms import
ContactForm def
contactView(request):
    if request.method ==
        'GET': form =
            ContactForm()
    else:
        form =
            ContactForm(request.POST) if
            form.is_valid():
                subject = form.cleaned_data['subject']
                from_email =
                    form.cleaned_data['from_email'] message =
                    form.cleaned_data['message']
                try:
                    send_mail(subject, message,
from_email, ['youradresmail@xxxx.com'])
                except BadHeaderError:
                    return HttpResponse('Invalid header
                        found.') return redirect('success')
            return render(request, "email.html", {'form':
form}) def successView(request):
    return HttpResponse('Success! Thank you for your message.')
```

- Ajouter le code nécessaire dans le fichier urls.py

Service de courrier électronique

Pour envoyer des e-mails réels, vous aurez besoin d'un service géré comme SendGrid , mailgun ou SES. Pour implémenter avec SendGrid, créez un compte gratuit et sélectionnez l'option SMTP, qui est plus facile à configurer. IL faut suivre les étapes nécessaires afin de créer un compte et avoir les paramètres SMTP

<https://app.sendinblue.com/account/login>



SMTP et API


SMTP Clés API

Vos paramètres SMTP

Serveur SMTP	smtp-relay.brevo.com
Port	587
Identifiant	7d6d63001@smtp-brevo.com

[🔄 Régénérer le nom d'utilisateur SMTP et le mot de passe principal](#)

Vos clés SMTP

Nom de la clé SMTP	Valeur de la clé SMTP	Status
<input type="checkbox"/> Mot de passe principal	***** 	Actif

Ajouter la configuration nécessaires dans le fichier setting

```
config > settings.py > ...  
132  
133 EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend' # new  
134 DEFAULT_FROM_EMAIL = 'youremail@xxxx.com'  
135 EMAIL_HOST = 'smtp-relay.sendinblue.com' # new  
136 EMAIL_HOST_USER = 'youremailn@xxxx.com' # new  
137 EMAIL_HOST_PASSWORD = 'xxxxxxxx' # new  
138 EMAIL_PORT = 587 # new  
139 EMAIL_USE_TLS = True
```