

## TP N°6 (TP N°5 Suite)

### FrameWork Python-Django

### Formulaires-Fondements

#### Objectifs du TP :

- Création d'un simple Formulaire

Les formulaires HTML sont une composante essentielle des sites Web modernes. Du simple champ de recherche de Google aux grands formulaires de plusieurs pages, les formulaires HTML sont le principal moyen de collecter des informations auprès des visiteurs et des utilisateurs de sites Web.

Le code pour un formulaire HTML de base est assez simple, par exemple:

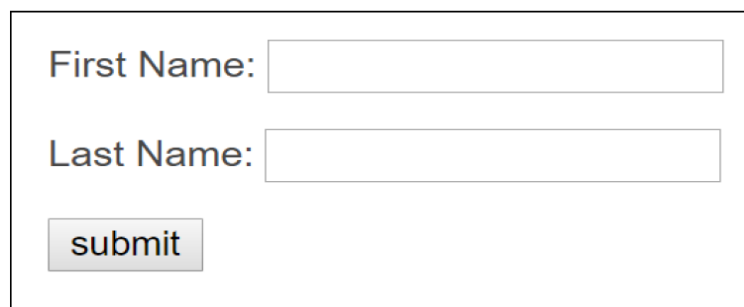
```
<form >
  <p>First Name: <input type="text" name="firstname"></p>
  <p>Last Name: <input type="text" name="lastname"></p>
  <p><input type="submit" value="submit"></p>
</form>
```

Les balises HTML `<form>` `</form>` définissent l'élément formulaire et chacun des champs de formulaire est contenu dans l'élément de formulaire. Dans ce formulaire, nous avons défini deux champs de texte (`<input type="text">`) et un bouton d'envoi (`<input type="submit">`). En HTML5, il existe de nombreux autres types d'élément de champ, notamment les champs d'email, les champs de date et d'heure, les cases à cocher, les boutons radio, etc.

Dans cet exemple, Nous avons mis les éléments de formulaire sous forme de paragraphes.

Il est également très courant de mettre éléments d'un formulaire sous forme de liste ordonnée ou non, ou sous forme de tableau avec les champs remplissant les lignes du tableau.

Le résultat d'affichage de ce formulaire dans une page Web est fourni ci-dessous.



Bien que la création d'un formulaire de base soit simple, les choses deviennent beaucoup plus compliquées lorsque vous devez utiliser le formulaire dans une situation réelle. Dans un site Web réel, vous devez valider les données soumises avec le formulaire. Si le champ est requis, vous devez vérifier qu'il n'est pas vide. Si le champ n'est pas vide, vous devez ensuite vérifier que les données soumises sont du type de données valide. Par exemple, si vous demandez une adresse électronique, vous devez vérifier qu'une adresse électronique valide est entrée.

Vous devez également vous assurer que votre formulaire traite les données saisies de manière sûre. Une méthode courante utilisée par les pirates pour cibler un site Web consiste à envoyer du code de programme malveillant via des formulaires afin d'essayer de pirater le site.

Pour compliquer davantage les choses, les utilisateurs du site Web attendent un retour quand ils ne remplissent pas correctement le formulaire. Par conséquent, vous devez également disposer d'un moyen d'afficher les erreurs sur le formulaire que l'utilisateur peut corriger avant de lui permettre de soumettre le formulaire.

Créer des formulaires, valider des données et fournir des commentaires est un processus fastidieux si vous codez tout à la main. Django est très flexible dans son approche de la création et de la gestion de formulaires. Si vous voulez vraiment concevoir vos formulaires à partir de rien, Django ne fait pas beaucoup pour vous gêner.

Cependant, cette façon de faire n'est pas recommandée. Si vous n'avez pas à faire une application très spéciale, Django dispose de nombreux outils et bibliothèques qui simplifient beaucoup la création de formulaires. En particulier, la classe de formulaires (**forms.Form**) de Django offre un ensemble très pratique de méthodes de classe qui se chargeront de la plupart du traitement et de la validation des formulaires.

Avec une classe Form, vous créez une classe spéciale qui ressemble beaucoup à un modèle Django. Les champs de la classe **Form** ont une méthode de validation intégrée, en fonction du type de champ, ainsi qu'un widget HTML associé.

Explorons un peu la classe Form dans le shell interactif de Django. Dans votre environnement virtuel, exécutez la commande suivante:

(venv2019) C:\Users\hassouni\Django2019\mysite2019> <b>python manage.py shell</b>
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license" for more information. (InteractiveConsole)
>>>
Une fois le shell en cours d'exécution, créez votre classe SimpleForm:
<b>1</b> >>> from django import forms <b>2</b> >>> class SimpleForm(forms.Form): <b>3</b> ...     firstname = forms.CharField(max_length=100) <b>4</b> ...     lastname = forms.CharField(max_length=100) <b>5</b> ...

Voyons ce que nous avons fait ici:

**Ligne 1** : Pour utiliser la classe **Form**, nous devons importer le module **forms** à partir de Django.

**Ligne 2** : Nous créons notre classe **SimpleForm**, qui hérite de la classe **forms.Form** de Django.

**Les lignes 3 et 4** définissent les champs `firstname` et `lastname` de notre Formulaire HTML de l'exemple précédent. Notez que les déclarations de champs d'un Form sont presque identiques aux déclarations de champs d'un Model de Django.

C'est le premier atout majeur de la classe Form de Django - vous n'avez pas besoin de vous rappeler une nouvelle syntaxe pour déclarer des champs de formulaire.

Continuons dans le shell de Django

```

>>> f = SimpleForm()
>>> print(f.as_p())

<p><label for="id_firstname">Firstname:</label> <input type="text" name="firstname"
maxlength="100" required id="id_firstname"></p>

<p><label for="id_lastname">Lastname:</label> <input type="text" name="lastname"
maxlength="100" required id="id_lastname"></p>

>>>

>>> print(f.as_table())

<tr><th><label for="id_firstname">Firstname:</label></th><td><input type="text"
name="firstname" maxlength="100" required id="id_firstname"></td></tr>

<tr><th><label for="id_lastname">Lastname:</label></th><td><input type="text"
name="lastname" maxlength="100" required id="id_lastname"></td></tr>

>>>

>>> print(f.as_ul())

<li><label for="id_firstname">Firstname:</label> <input type="text" name="firstname"
maxlength="100" required id="id_firstname"></li>

<li><label for="id_lastname">Lastname:</label> <input type="text" name="lastname"
maxlength="100" required id="id_lastname"></li>

```

Voyons ce qui se passe ici:

```

1 >>> f = SimpleForm()
2 >>> print(f.as_p())
3 >>> print(f.as_table())
4 >>> print(f.as_ul())

```

**Ligne 1 :** Dans cette ligne, nous avons simplement créé une instance de la classe SimpleForm et avons nommé l'instance f.

**Ligne 2 :** montre la spécificité de Django. **as\_p ()** est une méthode de classe qui formate le formulaire en tant que paragraphes. Vous pouvez voir à la sortie que Django a créé vos éléments de formulaire sans que vous ayez à écrire une seule balise HTML!

**Ligne 3 :** la méthode. **as\_table()** est une méthode de classe qui formate le formulaire en tant que éléments de tableau.

**Ligne 4 :** la méthode. **as\_ul()** est une méthode de classe qui formate le formulaire en tant que éléments d'une liste non ordonnée.

Vous remarquerez que Django ne génère pas pour vous l'élément `<form> </form>`, pas plus qu'il ne génère les éléments `<ul> </ul>` ou `<table> </table>` ni le bouton d'envoi. En effet, ils constituent des éléments structuraux sur votre page et doivent donc rester dans le Template.

La classe Form de Django gère également la validation pour vous. Revenons au shell pour essayer ceci:

```

1 >>> f = SimpleForm({})
2 >>> f.is_valid()
3 False
4 >>> f.errors
{'firstname': ['This field is required.'], 'lastname': ['This field is required.']}
>>>

```

Passons en revue ce que nous avons fait cette fois-ci:

**Ligne 1.** Nous avons créé une nouvelle instance de la classe SimpleForm et avons passé un dictionnaire vide ({} ) au formulaire.

**Ligne 2.** Lorsque Django a créé la classe Form, il a ajouté l'attribut "required" aux éléments <input > pour les champs firstname et lastname. Ainsi, lorsque nous exécutons la méthode **is\_valid ()** sur le formulaire vide, elle renvoie **False**.

**Ligne 4.** Enfin, si la validation du formulaire échoue, Django créera un dictionnaire des messages d'erreur. Nous pouvons accéder à ce dictionnaire via l'attribut **errors** de la classe Form.

Une autre fonctionnalité permettant d'économiser du temps dans la classe Form est que, lorsqu'un formulaire n'est pas valide, Django le restitue avec les messages d'erreur ajoutés automatiquement.

```

>>> print(f.as_p())
<ul class="errorlist"><li>This field is required.</li></ul>
<p><label for="id_firstname">Firstname:</label> <input type="text" name="firstname"
maxlength="100" required id="id_firstname"></p>
<ul class="errorlist"><li>This field is required.</li></ul>
<p><label for="id_lastname">Lastname:</label> <input type="text" name="lastname"
maxlength="100" required id="id_lastname"></p>
>>>

```

Vous pouvez voir que les erreurs ont été ajoutées au formulaire pour vous en tant que listes non ordonnées. Si vous deviez rendre ce formulaire dans votre navigateur, cela ressemblerait ci-dessous.

Maintenant que nous avons bien regardé le fonctionnement de la classe Form de Django, créons notre premier formulaire pour notre site Web. Nous allons commencer par un formulaire simple, commun à la plupart des sites Web: un formulaire de contact.

## Création d'un formulaire de Contact

Pour créer notre classe **ContactForm**, nous créons d'abord un nouveau fichier appelé `forms.py`. Vous pouvez créer ce fichier dans le dossier du site du projet, mais comme le formulaire de contact est une page du site, il est plus logique de le créer dans le dossier de l'application Pages:

C:\Users\hassouni\Django2019\mysite2019\pages\forms.py

```
1 from django import forms
2
3 class ContactForm(forms.Form):
4     yourname = forms.CharField(max_length=100, label='Your Name')
5     email = forms.EmailField(required=False, label='Your - emailaddress')
6     subject = forms.CharField(max_length=100)
7     message = forms.CharField(widget=forms.Textarea)
```

Cela ressemble à la classe `SimpleForm` que nous avons créée dans le shell, avec quelques différences:

**Ligne 4.** Si vous ne spécifiez pas l'attribut `label`, Django utilise le nom du champ pour le libellé. Nous voulons que le libellé du champ "yourname" soit plus lisible. Nous avons donc défini l'attribut `label` sur «**Your Name**».

**Ligne 5.** Nous ne souhaitons pas que l'email soit un champ obligatoire. Nous avons donc défini l'attribut "**required**" sur `False` pour que la personne qui soumet le formulaire puisse laisser ce champ vide. Nous modifions également le libellé par défaut du champ de courrier électronique en "Your e-mail address".

**Ligne 7.** Le champ de message doit permettre à la personne qui soumet le formulaire de saisir un message détaillé. Nous définissons donc le widget de champ sur une zone "**Textarea**", en remplacement du widget **TextInput** par défaut.

Maintenant que nous avons créé notre classe `ContactForm`, nous avons quelques tâches à accomplir pour pouvoir l'afficher sur notre site Web:

1. Ajoutez notre formulaire à la liste des URL dans notre application Pages;
2. Ajouter un lien de navigation à notre template de site;
3. Créer un template pour le formulaire de contact; et
4. Créez une nouvelle vue pour gérer le formulaire de contact.

### Ajouter une URL à l'application Pages

Pour afficher notre formulaire de contact, nous créons d'abord une URL pour celui-ci. Pour ce faire, nous devons modifier le fichier `urls.py` de notre application (modifications en gras):

pages/urls.py

```
1 from django.urls import path
2
3 from . import views
4
5 urlpatterns = [
6     path('', views.index, {'pagename': ''}, name='home'),
7     path('contact', views.contact, name='contact'),
8     path('<str:pagename>', views.index, name='index'),
9 ]
```

À la ligne 7, nous avons ajouté une URLconf qui dirigera l'URL se terminant par «contact» vers la nouvelle vue de contact que nous écrirons sous peu. Assurez-vous que la nouvelle URLconf est avant la vue d'index dans la liste **urlpatterns**. Si vous le placez après l'URLconf de la view index, Django lève une exception car <str: pagename> correspond à l'URL du contact et charge la view "index" à la place de la view "contact".

### Ajouter un lien de navigation au template du site

L'emplacement le plus courant pour un lien vers un formulaire de contact d'un site Web se trouve dans le menu. C'est pourquoi nous allons ajouter un lien pour notre formulaire de contact (modifications en gras):

```
1 <aside id="leftsidebar">
2   <nav id="nav">
3     <ul>
4       {% block sidenav %}
5         <li>Menu 1</li>
6         <li>Menu 2</li>
7         <li>Menu 3</li>
8       {% endblock sidenav %}
9       <li><a href="/contact">Contact Us</a></li>
10    </ul>
11  </nav>
12 </aside>
```

Nous avons apporté une modification au modèle **base.html**. À la ligne 9, nous avons inséré un élément de liste supplémentaire qui sera restitué à la fin des autres éléments de menu.

## Création du Template du formulaire de Contact

Pour afficher notre formulaire, nous devons créer un Template. Dans le répertoire `templates/pages` créer le fichier HTML `contact.html` contenant le code ci-dessous.

```
# pages/templates/pages/contact.html

1 {% extends "pages/page.html" %}
2
3 {% block title %}Contact Us{% endblock title %}
4
5 {% block content %}
6 <h1>Contact us</h1>
7
8 {% if submitted %}
9   <p class="success">
10     Your message was submitted successfully. Thank you.
11   </p>
12
13 {% else %}
14   <form action="" method="post" novalidate>
15     <table>
16       {{ form.as_table }}
17     <tr>
18       <td>&nbsp;</td>
19       <td><input type="submit" value="Submit"></td>
20     </tr>
21   </table>
```

```

22         {% csrf_token %}
23     </form>
24 {% endif %}
25 {% endblock content %}

```

Vous remarquerez que nous étendons le Template de page cette fois et remplaçons les blocs de titre et de contenu par un nouveau contenu pour notre formulaire de contact.

Ce qu'il faut noter:

**Ligne 1** : Nous étendons le Template de page cette fois et remplaçons les blocs de titre et de contenu par un nouveau contenu pour notre formulaire de contact.

**Ligne 8** : Nous utilisons la balise de modèle {% if% } pour la première fois. "submitted" est une valeur booléenne transmise depuis la view.

Les balises {% if% } / {% else% } / {% endif% } (lignes 8, 13 et 24) créent une branche logique indiquant «Si le formulaire a été soumis, affichez le message de remerciement, sinon, affichez le formulaire vierge.»

**Ligne 14** : C'est le début de notre formulaire POST. C'est du HTML standard. Notez l'attribut **novalidate** dans la balise <form>. Lors de l'utilisation de HTML5 dans certains des navigateurs les plus récents (notamment Chrome), les champs de formulaire seront automatiquement validés par le navigateur. Comme nous voulons que Django gère la validation du formulaire, l'attribut **novalidate** indique au navigateur de ne pas valider le formulaire.

**Ligne 16** : C'est la ligne qui rend les champs de formulaire. La méthode **as\_table** rendra les champs de formulaire sous forme de lignes de table. Django ne restitue pas les balises du tableau <table></table> ni le bouton d'envoi <input type = "submit">, nous les ajoutons donc à la ligne 15 et aux lignes 17 à 21.

**Ligne 22** : Tous les formulaires POST destinés à des URL internes doivent utiliser la balise de modèle {% csrf\_token% }. Cela protège contre les attaques de type "Cross Site Request Forgeries (CSRF)"

## Création de la view pour le formulaire Contact

Notre dernière étape consiste à créer la nouvelle view "contact" pour notre formulaire. Ouvrez votre fichier views.py et ajoutez le code ci-dessous.

```

C:\Users\hassouni\Django2019\mysite2019\pages\views.py
# pages\views.py
1 from django.shortcuts import render, get_object_or_404
2 from django.http import HttpResponseRedirect
3
4 from .models import Page
5 from .forms import ContactForm
6
7 # view index definition
8
9 def contact(request):
10     submitted = False
11     if request.method == 'POST':
12         form = ContactForm(request.POST)
13         if form.is_valid():
14             cd = form.cleaned_data
15             # assert False
16             return HttpResponseRedirect('/contact?submitted=True')

```

```

17     else:
18         form = ContactForm()
19         if 'submitted' in request.GET:
20             submitted = True
21
22     return render(request, 'pages/contact.html', {'form': form,
'page_list': Page.objects.all(), 'submitted': submitted})

```

Passons en revue les éléments importants de ce code:

**Lignes 2 et 5 :** Nous importons la classe `HttpResponseRedirect` de `django.http` et la classe `ContactForm` de `forms.py`.

**Ligne 11 :** Vérifiez si le formulaire a été POSTé. Sinon, passez à la ligne 18 et créez un formulaire vierge.

**Ligne 13 :** Vérifiez si le formulaire contient des données valides. Notez qu'il n'y a pas de mot clé pour traiter des données de formulaire non valides. C'est ce qui est vraiment génial avec la classe `Form`. Si le formulaire n'est pas valide, la view doit simplement passer à la ligne 22 et réafficher le formulaire, car Django ajoutera automatiquement les messages d'erreur pertinents à votre formulaire.

**Ligne 14 :** Si le formulaire est valide, Django normalisera les données et les enregistrera dans un dictionnaire accessible via l'attribut **`cleaned_data`** de la classe `Form`. Dans ce contexte, normaliser signifie le changer en un format cohérent. Par exemple, quel que soit le format de saisie utilisé, Django convertira toujours une chaîne de date en objet Python `datetime.date`.

**Ligne 15.** Nous ne faisons rien avec le formulaire soumis pour le moment. Nous avons donc introduit une erreur d'assertion afin de pouvoir tester la soumission du formulaire avec la page d'erreur de Django.

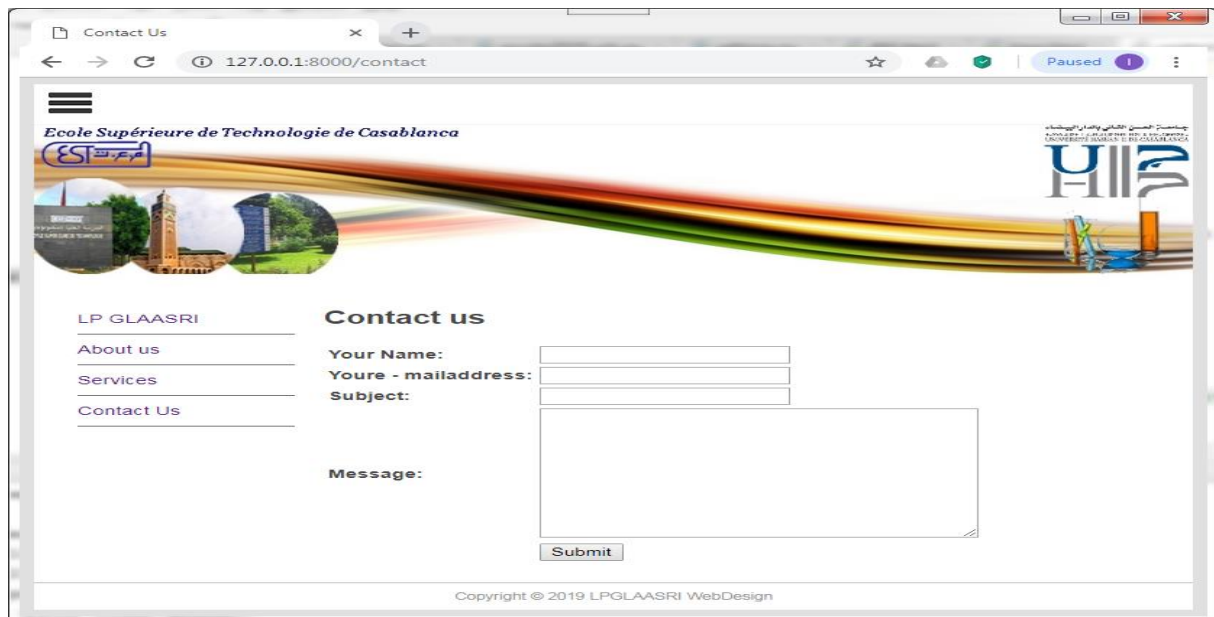
**Ligne 16 :** Une fois que le formulaire a été soumis avec succès, nous utilisons la classe **`HttpResponseRedirect`** de Django pour effectuer une redirection vers la view **`contact`**. Nous avons défini la variable **`submitted`** à `True`. Ainsi, au lieu de restituer le formulaire, la vue affichera le message de remerciement.

**Ligne 22 :** Rendez le modèle et les données à la vue. Notez l'ajout du QuerySet **`page_list`**. Si vous vous souvenez du TP précédent, le template de page a besoin de **`page_list`** pour pouvoir restituer les éléments de menu.

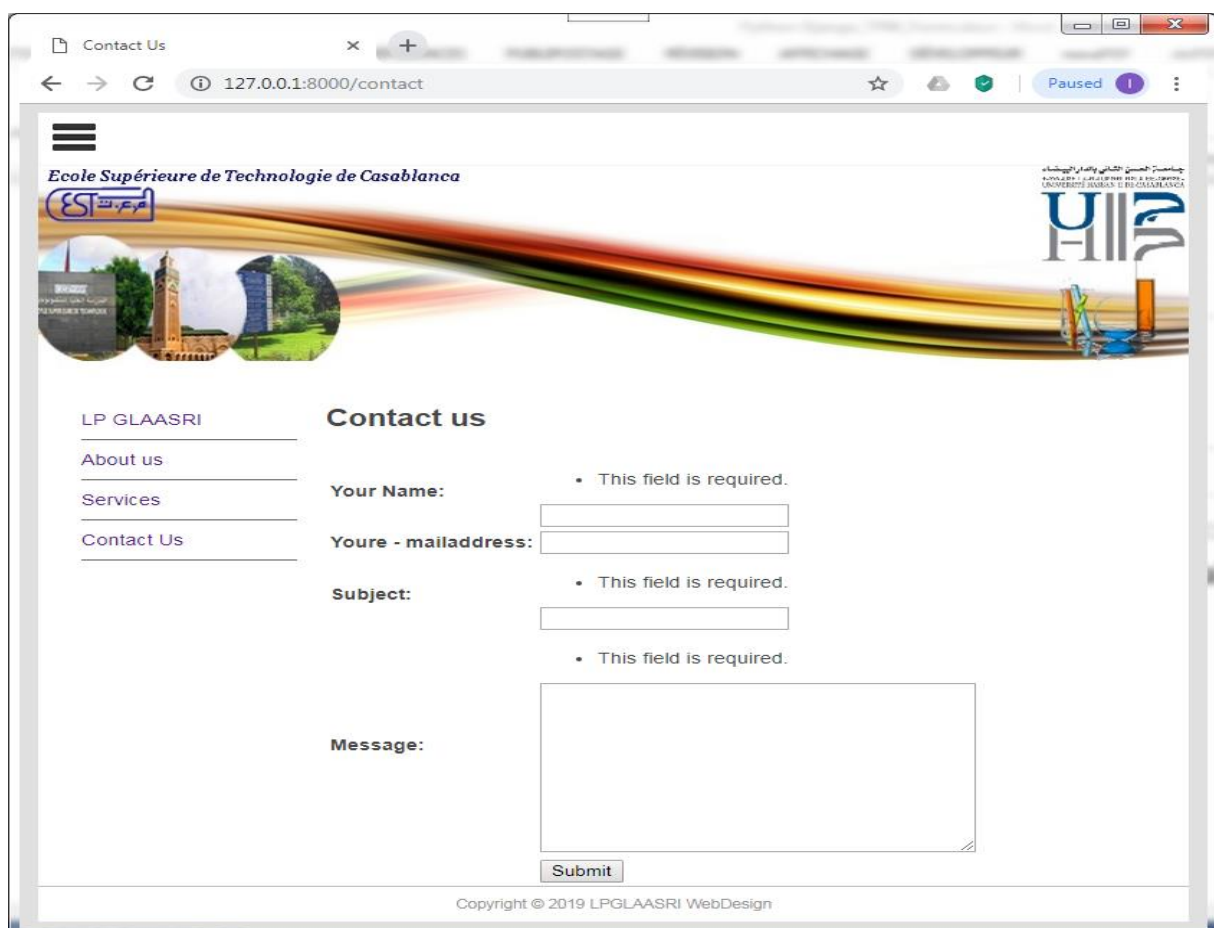
Nous ne faisons rien avec les données de formulaire soumises pour le moment. Nous avons plutôt ajouté **`"assert False"`** à la ligne 15 afin que nous puissions vérifier que le formulaire fonctionne correctement.

Continuez et supprimez la mise en commentaire de la ligne 15, enregistrez le fichier **`views.py`** puis accédez à **<http://127.0.0.1:8000/contact>** pour afficher votre nouveau formulaire de contact. Tout d'abord, notez qu'il existe un lien vers le formulaire de contact dans le menu de gauche.





Ensuite, soumettez le formulaire vide pour vous assurer que la validation du formulaire fonctionne. Les messages d'erreur devraient s'afficher.



Maintenant, remplissez le formulaire avec des données valides et soumettez-le à nouveau. Vous devriez obtenir une erreur d'assertion, déclenchée par l'instruction **"assert False"** dans la vue (ligne 15). Lorsque nous avons écrit notre view du formulaire de contact, nous avons dit à Django de placer le contenu de l'attribut **cleaned\_data** dans la variable **cd** (ligne 14).

127.0.0.1:8000/contact

Ecole Supérieure de Technologie de Casablanca

LP GLAASRI

About us

Services

Contact Us

### Contact us

Your Name:  This field is required.

Your email address:

Subject:  This field is required.

Message:

Copyright © 2019 LPGLAASRI WebDesign

Avec l'instruction **assert False** active dans notre view, nous pouvons vérifier le contenu de **cd** avec la page d'erreur Django. Faites défiler jusqu'à l'erreur "assert False" et ouvrez le panneau **Local vars**. Vous devriez voir la variable **cd** contenant un dictionnaire résultat de la soumission complète du formulaire.

**Traceback** [Switch to copy-and-paste view](#)

```
C:\Users\hassouni\ Django2019\venv2019\lib\site-packages\django\core\handlers\exception.py in inner
34.         response = get_response(request) ...

▶ Local vars

C:\Users\hassouni\ Django2019\venv2019\lib\site-packages\django\core\handlers\base.py in _get_response
126.         response = self.process_exception_by_middleware(e, request) ...

▶ Local vars

C:\Users\hassouni\ Django2019\venv2019\lib\site-packages\django\core\handlers\base.py in _get_response
124.         response = wrapped_callback(request, *callback_args, **callback_kwargs) ...

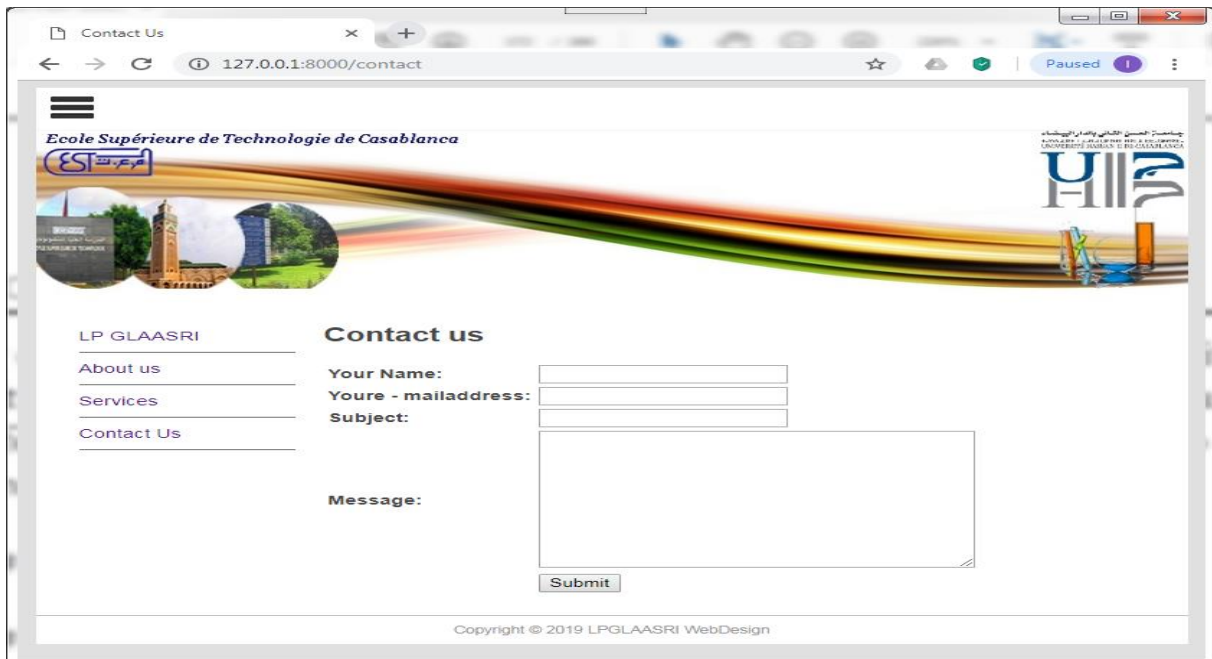
▶ Local vars

C:\Users\hassouni\ Django2019\mysite2019\pages\views.py in contact
27.         assert False ...
```

▼ Local vars

Variable	Value
cd	{'email': 'moha@gmail.com', 'message': 'La planete Mars est entièrement conquise par des individus de ', 'race inconnue.', 'subject': 'Plane Mars', 'yourname': 'Moha'}
form	<ContactForm bound=True, valid=True, fields=(yourname;email;subject;message)>
request	<WSGIRequest: POST '/contact'>
submitted	False

Une fois que vous avez vérifié que le formulaire fonctionne correctement, cliquez sur le lien «Contact-us» dans le menu pour revenir au formulaire vide.



Notre formulaire de contact fonctionne très bien, mais il a toujours l'air un peu simple: les champs ne sont pas bien alignés et les messages d'erreur ne se démarquent pas vraiment.

Modifions le fichier CSS pour donner à notre formulaire un meilleur format. Ajoutez ce qui suit à la fin de votre fichier main.css:

C:\Users\hassouni\Django2019\mysite2019\mysite2019\static\main.css

```
.....
ul.errorlist {
    margin: 0;
    padding: 0;
}
.errorlist li {
    border: 1px solid red;
    color: red;
    background: rgba(255, 0, 0, 0.15);
    list-style-position: inside;
    display: block;
    font-size: 1.2em;
    margin: 0 0 3px;
    padding: 4px 5px;
    text-align: center;
    border-radius: 3px;
}

input, textarea {
    width: 100%;
    padding: 5px !important;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    border-radius: 3px;
    border-style: solid;
    border-width: 1px;
```

```

        border-color: rgb(169,169,169)
    }

    input {
        height: 30px;
    }

    .success {
        background-color: rgba(0, 128, 0, 0.15);
        padding: 10px;
        text-align: center;
        color: green;
        border: 1px solid green;
        border-radius: 3px;
    }
}

```

Une fois que vous avez enregistré les modifications dans votre fichier CSS, actualisez le navigateur et soumettez le formulaire vide.

LP GLAASRI

About us

Services

Contact Us

### Contact us

Your Name: This field is required.

Your email address:

Subject: This field is required.

This field is required.

Message:

Submit

Copyright © 2019 LPGLAASRI WebDesign

Non seulement votre formulaire devrait-il être mieux présenté, mais affiche également de jolis messages d'erreur (Figure ci-dessus).

Lorsque vous entrez des données valides et soumettez le formulaire, la view contact est redirigée vers la page de contact avec «submitted = True» comme paramètre GET - <http://127.0.0.1:8000/contact?submitted=True>.

LP GLAASRI

About us

Services

Contact Us

**Contact us**

Your Name:

Your email address:

Subject:

Message:

Copyright © 2019 LPGLAASRI WebDesign

En cliquant sur le bouton "submit", nous obtenons la page ci-dessous.

LP GLAASRI

About us

Services

Contact Us

**Contact us**

Your message was submitted successfully. Thank you.

Copyright © 2019 LPGLAASRI WebDesign

## Envoi des données du formulaire par Email.

Notre formulaire de contact fonctionne bien et a une belle apparence, mais il n'est pas très utile pour l'instant car nous ne faisons rien avec les données du formulaire.

S'agissant d'un formulaire de contact, le moyen le plus courant de gérer les données soumises consiste à les envoyer par courrier électronique à un administrateur de site ou à une autre personne de contact au sein de l'organisation.

Configurer un serveur de messagerie pour tester les e-mails en développement peut être une véritable galère. Heureusement, c'est un autre problème pour lequel les développeurs Django ont fourni une solution pratique. Django fournit un certain nombre de moteurs de courrier électronique, y compris quelques-uns spécifiquement conçus pour être utilisés pendant le développement.

Nous allons utiliser le backend de la console. Ce backend est particulièrement utile en développement car il ne vous oblige pas à configurer un serveur de messagerie pendant que vous développez une application Django. Le backend de la console envoie un courrier électronique au terminal (console). Vous pouvez vérifier cela dans la fenêtre de votre terminal après avoir envoyé votre formulaire.

Il existe d'autres back-end de courrier électronique à tester - filebased, locmem et dummy, qui envoient vos emails dans un fichier de votre système local, les enregistrent dans un attribut en mémoire ou les envoient à un back-end factice.

Vous trouverez plus d'informations dans la documentation de Django à l'adresse:

<https://docs.djangoproject.com/en/2.1/topics/email/#topic-email-backends>

Modifions la view contact pour envoyer des courriels (modifications en gras):

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
#-----
from django.core.mail import send_mail, get_connection
#-----

from .models import Page
from .forms import ContactForm

def index(request, pagename):
    pagename = '/' + pagename
    pg = get_object_or_404(Page, permalink=pagename)
    #pg = Page.objects.get(permalink=pagename)
    context = {
        'title': pg.title,
        'content': pg.bodytext,
        'last_updated': pg.update_date,
        'page_list': Page.objects.all(),
    }
    #assert False
    return render(request, 'pages/page.html', context)

def contact(request):
    submitted = False
    if request.method == 'POST':
        form = ContactForm(request.POST)
```



```

        if form.is_valid():
            cd = form.cleaned_data
            #assert False

#-----
            con = get_connection('django.core.mail.backends.console.EmailBackend')
            send_mail(cd['subject'],cd['message'], cd.get('email', 'noreply@gmail.com'),
                    ['lpglasri@gmail.com'],connection = con)
#-----

            return HttpResponseRedirect('/contact?submitted=True')
        else:
            form = ContactForm()
            if 'submitted' in request.GET:
                submitted = True

            return render(request, 'pages/contact.html', {'form': form, 'page_list':
Page.objects.all(), 'submitted': submitted})

```

Regardons les changements que nous avons apportés:

Nous commençons par importer les fonctions **send\_mail** et **get\_connection** à partir de **django.core.mail**.

```
from django.core.mail import send_mail, get_connection
```

Nous appelons la fonction **get\_connection** pour créer un objet connection au "pseudo-serveur de messagerie":

```
con = get_connection('django.core.mail.backends.console.EmailBackend')
```

Nous appelons par la suite la fonction **send\_mail** pour envoyer les données du dictionnaire **cd** qui contient les données du formulaire.

```
send_mail(cd['subject'],cd['message'], cd.get('email', 'noreply@gmail.com'),
        ['lpglasri@gmail.com'],connection = con)
```

C'est tout ce dont vous avez besoin pour envoyer un email en Django. S'il s'agissait d'un véritable site Web, tout ce dont vous auriez besoin pour la production consiste à modifier le backend et à ajouter vos paramètres de serveur de messagerie à **settings.py**.

Testez la view en remplissant le formulaire et en le soumettant. Si vous regardez dans la fenêtre de la console (invite de commande), vous verrez que la view a envoyé le courrier électronique codé directement à la console.

Content-Transfer-Encoding: 8bit

Subject: Plane Mars

From: moha@gmail.com

To: lpglasri@gmail.com

Date: Thu, 28 Feb 2019 15:43:42 -0000

Message-ID: <155136862208.7904.5489047946883250718@hassouni-PC>

La planete Mars a été conquise par des individus d'une race inconnue.

-----  
[28/Feb/2019 16:43:42] "POST /contact HTTP/1.1" 302 0

[28/Feb/2019 16:43:42] "GET /contact?submitted=True HTTP/1.1" 200 1216

