

Formation Django

TP4 : Django Forms

On souhaite créer un Formulaire de Contact dans notre application et stocker les messages reçus dans la base de données.

Prérequis :

- 1) Créer un modèle nommé `Message` contenant les attributs suivants : `name` (string), `email` (email), `subject` (string) et `content` (Text).
- 2) Créer les migrations et valider-les (`makemigrations` & `migrate`)
- 3) Créer une vue nommée `list_messages()` et son template correspondant nommée `"app/templates/messages_list.html"` pour pouvoir afficher les messages enregistrés dans la base de données.
- 4) Créer une route dans le fichier `urls.py` qui pointe vers la vue `list_messages()`

I. Formulaire HTML classique

A. Formulaire HTML

- 1) Créer un fichier HTML nommé `"contact_htmlform.html"` dans le dossier `app/templates` et y mettre un formulaire HTML qui ressemble à l'image. Ne pas oublier la balise `{% csrf_token %}`.
- 2) Créer une nouvelle vue dans le fichier `app/views.py` nommé `handle_contact()` qui permet de :
 - Afficher la page du formulaire si la requête est faite avec un verbe HTTP `GET`.
 - Récupérer les informations du formulaire (depuis l'attribut `request.POST`) et créer une nouvelle objet `Message` et le sauvegarder dans la base de données et rediriger finalement l'utilisateur vers la liste des messages.
- 3) Valider les données en respectant les règles suivantes :
 - Le titre ne doit pas dépasser **15 caractères**
 - Tous les attributs sont obligatoires sauf le `subject`
 - S'il y a une erreur de validation, rediriger l'utilisateur vers la page du formulaire.
- 4) Créer une route dans le fichier `urls.py` qui pointe vers la vue `handle_contact()` au chemin `"form"`
- 5) Enregistrer les modifications et tester le formulaire dans le navigateur en visitant l'URL : <http://127.0.0.1:8000/form>



B. Formulaire HTML avec Upload des Fichiers

- 1) Modifier le formulaire de l'exercice précédent en ajoutant un champ "Attachement" qui permet d'ajouter un fichier dans le message :
 - a) Ajouter l'attribut `attachement_file` dans le fichier modèle de message, relancer les migrations et appliquez-les.
 - b) Ajouter le champ attachement dans le formulaire HTML de contact (N.B: veiller à changer l'entype du formulaire)
 - c) Modifier le code de la vue `handle_contact()` pour prendre en charge le fichier chargé.

- 2) On souhaite permettre à l'utilisateur d'envoyer plusieurs fichiers attachements à la fois. Pour y arriver :
 - a) Créer un nouveau modèle nommé **Attachement** et y assigner une relation Many-to-One avec **Message**. Le modèle contient les attributs suivants :
 - **name** : CharField, max_length=200
 - **file** : FileField
 - b) Lancer la création des migrations et appliquer les.
 - c) Modifier le formulaire pour permettre l'envoi de fichiers multiple.
 - d) Modifier la vue pour gérer le traitement de fichiers multiple.

II. Formulaire Django

A. Utilisation de La classe Form

- 1) Dans un fichier `app/forms.py`, créer un formulaire de contact **ContactForm** en utilisant la classe **Form** et les mêmes attributs de l'exercice précédent sauf le champ d'attachements
- 2) Créer une deuxième vue nommée `handle_contact_form()` et son point d'entrée dans le fichier `urls.py` vers `"/form2"`
- 6) Créer un fichier template nommé `form_djangoform.html` affichant le formulaire sous forme de paragraphe (`form.as_p`) en utilisant la méthode **POST**. Ne pas oublier le tag `{%csrf_token%}`.
- 3) Dans la vue `handle_contact_form()` :
 - Afficher le formulaire vide si la méthode utilisée est **GET**
 - Dans le cas contraire, Remplir le formulaire par les données de `request.POST`
 - Valider le formulaire et enregistrer la nouvelle instance de **Message**.
 - Rediriger l'utilisateur vers la page de liste des messages.
- 4) Enregistrer les modifications et Tester le Nouveau Formulaire dans le navigateur en navigant vers <http://127.0.0.1:8000/form2>
- 5) Remplir le formulaire et Tester la soumission des messages.
- 6) Tester les différents cas de validations (nom de taille supérieure à 15, pas d'email, etc.)
- 7) Utiliser "**Inspector Élément**" dans Votre navigateur pour supprimer la validation des données coté client (supprimer manuellement les attribut HTML `required`, `maxlength` et `type`)
- 8) Personnaliser le formulaire **ContactForm** en ajoutant les caractéristiques :
 - **email** : ajouter le texte d'aide (`help_text`) suivant : "**Veillez saisir votre Email**"
 - **content** : Ajouter une Valeur initiale : "**N/A**"
 - **name** : Changer le texte de l'erreur "`required`" à "**SVP La saisie du nom est obligatoire**"
- 9) On souhaite remplacer l'attribut nom pas une instance de modèle nommée **User** qui représente un utilisateur de l'application.
 - a) Créer un Modèle nommé **AppUser** contenant les champs suivants :
 - **name**: string, max_length=20
 - **email**: email
 - **biography**: texte
 - b) Créer les migrations et appliquer-les
 - c) En utilisant le Shell Django (`python manage.py shell`) créer quelques instances de **AppUser** et sauvegarder les dans la base de données.
 - d) Modifier le Formulaire pour remplacer le champ `name` par un champ de type **ModelChoiceField** qui affiche la liste de tous les utilisateurs (`queryset=AppUser.objects.all()`)

- e) Modifier le code de la vue `handle_contact_form()` pour récupérer le nom de l'utilisateur sélectionné de `AppUser` puis sauvegarder le Message correctement.
- f) Actualiser la page du formulaire puis soumettre un nouveau message. Que constatez-vous ?

B. Validation des données de Formulaire

En utilisant le même formulaire de l'activité précédente :

- 1) Ajouter une méthode de validation personnalisée `clean_content()` pour le champ `content` qui force l'utilisateur à saisir plus de 3 mots.

```
def clean_content(self):
    content = self.data.dict().get('content')
    if len(content.split()) < 3:
        raise ValidationError('You must add more content')
    return content
```

- 2) Tester à nouveau l'envoi d'un formulaire avec une valeur incorrecte dans le champ `content` (par exemple : "**Hello World**"). Que se passe-t-il ?
- 3) Ajouter une méthode de validation du champ `subject` de tel façon à vérifier que la valeur ne contient pas le signe "@".
- 4) Modifier l'affichage du formulaire dans le fichier template "`form_djangoform.html`" pour afficher les champs individuellement.
- 5) Afficher les erreurs de validation en dessus du formulaire en utilisant une balise `{% for %}` Et dictionnaire d'erreurs `form.errors.values`.
- 6) Créer un template HTML nommée `app/common_errors.html` contenant l'affichage des erreurs du formulaire et l'inclure dans le fichier "`form_djangoform.html`" grâce à la balise `{%include%}`

C. Composants du Formulaire (widgets)

En utilisant le même formulaire de l'activité précédente :

```
subject = forms.CharField(widget=forms.TextInput(attrs={'style': 'background-color:dodgerblue'}))
```

- 1) Changer la couleur de fond du champ `subject` en **blue** en utilisant le widget `TextInput` avec argument `attrs` personnalisé.
- 2) Sauvegarder les changements et actualiser la page du formulaire. Que constatez-vous ?
- 3) Ajouter au formulaire un Champ `date_envoi` en utilisant un champ de type `DateField` avec un composant `SelectDateWidget`. Limiter les choix des années à 2020 et 2021 en utilisant le paramètre `years` de `SelectDateWidget`.

D. Formulaire à partir des modèles

On souhaite réutiliser le modèle `Message` pour générer un formulaire `MessageForm` sans avoir à définir les champs manuellement.

- 1) Dans le fichier `app/forms.py`, créer un nouveau formulaire `MessageForm` qui hérite de la classe **`ModelForm`**
- 2) Définir les attributs de la classe `Meta` pour définir le modèle sur `Message` et les champs concernés comme `__all__`
- 3) Redéfinir une nouvelle vue nommée `handle_contact_modelform()` qui utilise ce nouveau formulaire en utilisant le même fichier template que dans l'exercice précédent.
N. B : sauvegarder le message crée en utilisant la méthode `form.save()`
- 4) Définir un point d'entrée à la vue `handle_contact_modelform()` vers l'URL : `"/form3"`
- 5) Tester le formulaire **`MessageForm`** en visitant l'URL: <http://127.0.0.1:8000/form3>
- 6) Changer le type de widget utilisé pour `subject` en `Textarea` et la couleur de l'arrière-plan en couleur grise.