Chapitre XI : Les Formulaires

Enseignantes :

Naima Halouani & Hounaida Moalla

Plan

- Introduction
- Création de formulaire
- Activation
- Amélioration

Introduction

Utilité :

Les formulaires sont utilisés lorsqu'on veut interagir avec une base de données.

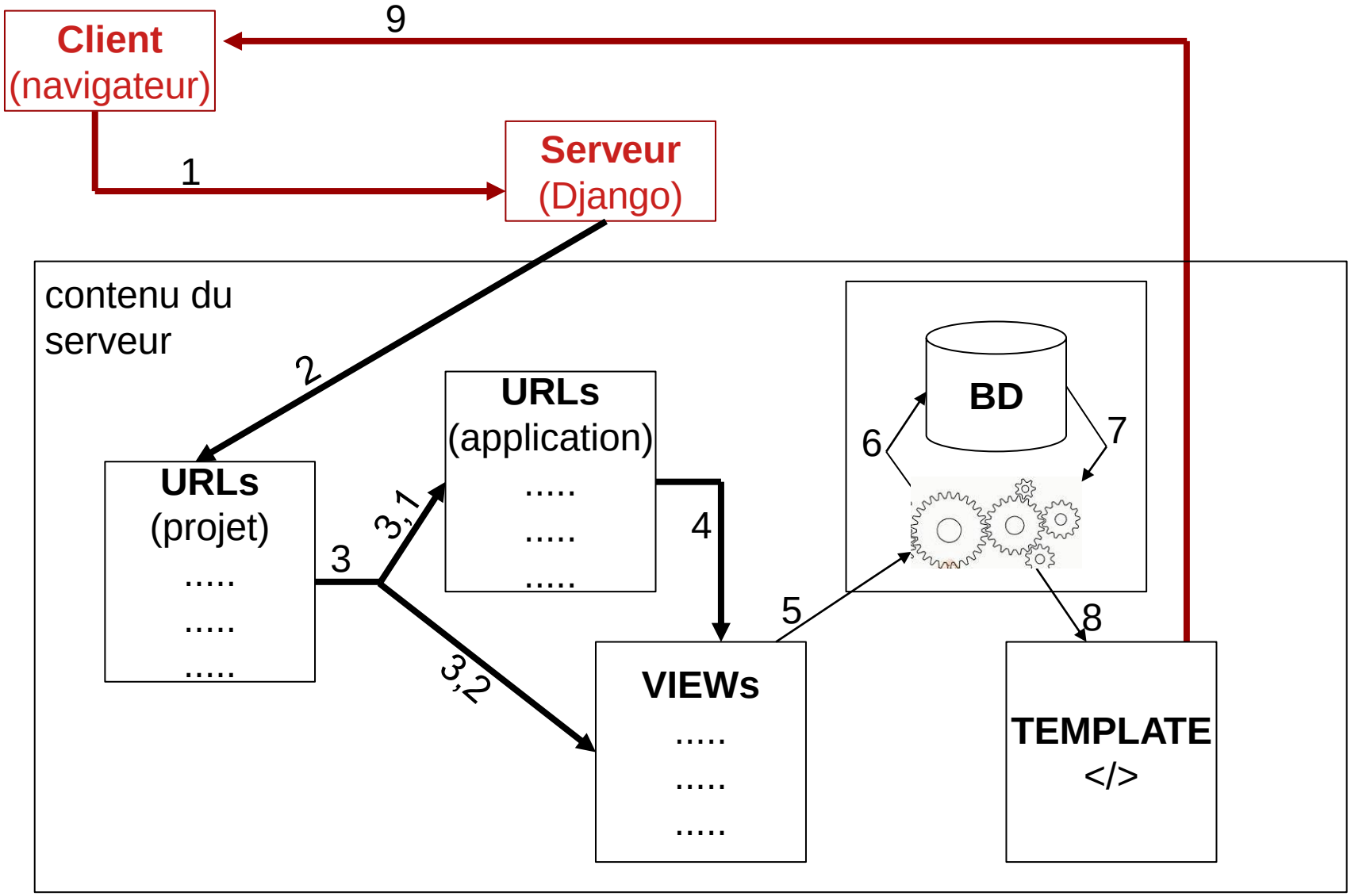
Points forts :

L'utilisation des formulaires simplifie à la fois :

- la production de formulaire HTML
- et aussi la validation de données.

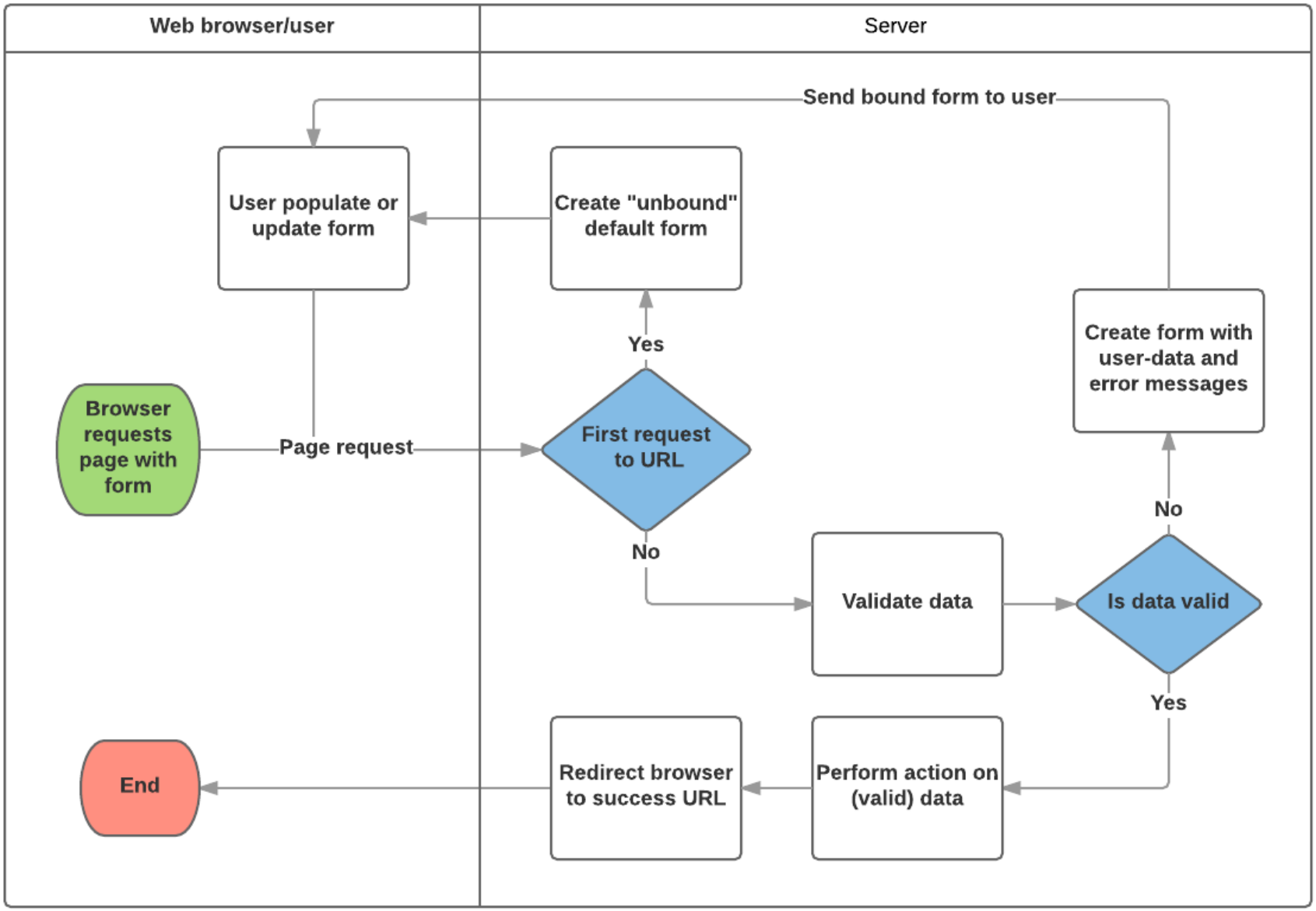
Introduction

Principe :



Introduction

un diagramme représentant les étapes de gestion d'un formulaire de requêtes



Introduction

En se basant sur la lecture du diagramme ci-dessus, les tâches principales dont s'aquitte Django à l'occasion de la gestion d'un formulaire sont :

1. Afficher le formulaire sous sa forme par défaut la première fois où il est demandé par l'utilisateur.
   - Le formulaire peut contenir des champs vides (par exemple si vous créez un nouvel enregistrement) ou peut être prérempli de valeurs initiales (par exemple si vous modifiez les valeurs d'un enregistrement existant, ou que ces champs ont des valeurs initiales utiles).
   - Le formulaire est qualifié à cette étape de formulaire libre, parce qu'il n'est associé à aucune donnée entré par l'utilisateur (bien qu'il puisse avoir des valeurs initiales) .
2. Recevoir des données d'une requete d'envoi de données et les lier au formulaire.
   - Lier les données au formulaire signifie que les données entrées par l'utilisateur, ainsi que les erreurs éventuelles sont accessibles lorsque nous avons besoin de réafficher le formulaire.
3. Nettoyer et valider les données
   - Le nettoyage de données consiste à désinfecter la saisie (par exemple en supprimant les caractères non valides, et qui pourraient être utilisés pour envoyer du contenu malveillant au serveur.) et à convertir ces données en types Python cohérents.
   - La validation vérifie que les valeurs envoyées sont appropriées au champ (par exemple dans le bon intervalle de dates, ni trop long ni trop court, etc.)
4. Si une donnée n'est pas valide, ré-affiche le formulaire, cette fois-ci avec les données déjà saisies par l'utilisateur et les messages d'erreur pour les champs en erreur.
5. Si toutes les données sont conformes, effectue les actions demandées (e.g. sauvegarde les données, envoyer un mail, renvoie le résultat d'une recherche, télécharge un fichier etc.)
6. Une fois toutes ces actions accomplies, redirige l'utilisateur vers une autre page.

Introduction

Les formulaires permettent de partager des informations avec le serveur de manière relativement sécurisée via des requêtes POST.

Toutes les classes de formulaires sont créées comme sous-classes de **django.forms.Form** ou de **django.forms.ModelForm**.

1. La classe **Form** crée et configure les formulaires manuellement.

   ➡ Utilisée pour les formulaires qui n'interagissent pas directement avec les modèles.

   Exemple : un formulaire de contact ou un formulaire d'inscription, où vous n'interagissez pas nécessairement avec la base de données.

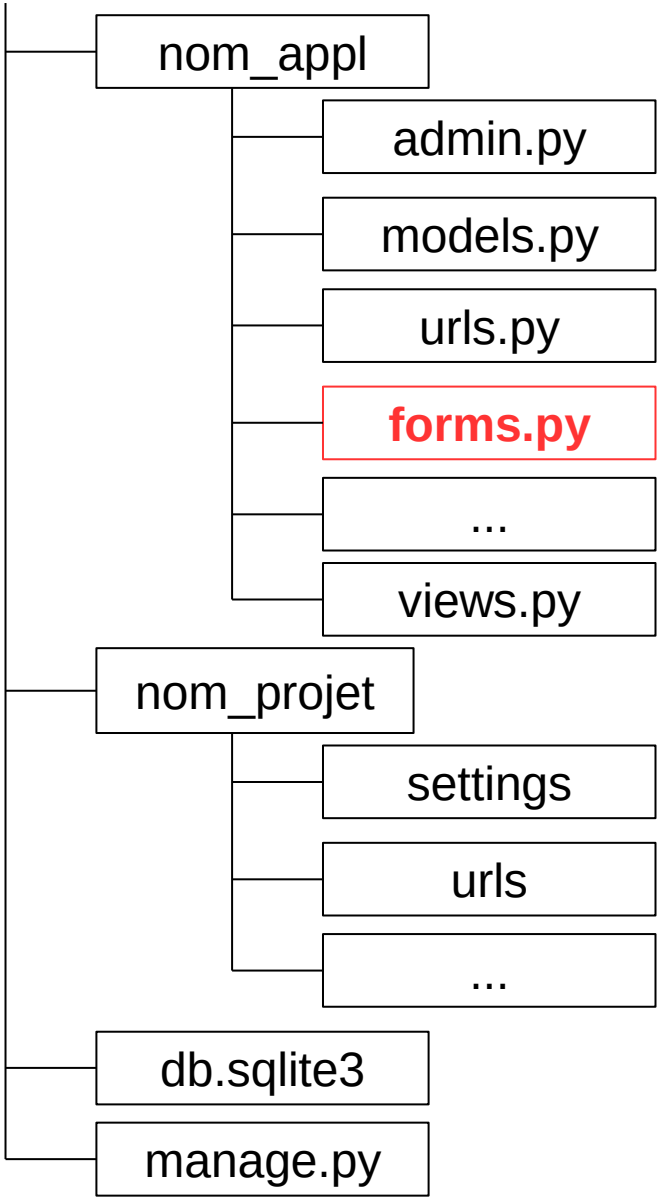
2. La classe **ModelForm** crée les formulaires à partir des modèles de l'application qui pourront être modifié par la suite.

Introduction

Architecture :

Tous les formulaires d'une application doivent être **regroupés** dans un seul emplacement.

Ajouter un fichier **forms.py** dans lequel seront regroupés tous les formulaires de l'application.



Création de formulaire avec la classe Form (1)

Définition

La classe **Form** est le cœur du système de gestion des formulaires de Django.

La classe **Form** décrit un formulaire et détermine son fonctionnement et son apparence.

Elle :

- **spécifie les champs** présents dans le formulaire,
- **affiche les composants** : widgets, les labels, les valeurs initiales, les valeurs valides et (après validation) **les messages d'erreur** associés aux champs invalides.
- **fournit des méthodes** pour se restituer elle-même dans les templates en utilisant des formats prédéfinis (tables, listes etc.) ou pour obtenir la valeur de chaque élément de formulaire (permettant un rendu manuel fin).

Création de formulaire avec la classe Form (2)

Rappel :

Le formulaire est défini en HTML comme une collection d'éléments enfermés entre deux balises **<form> ... </form>** contenant des composants de l'interface et un bouton "submit":

```
<form action="url" method="post">
    ...
    <input type="submit" value="OK">
</form>
```

- **action :** Il s'agit de la destination (ressource ou URL) où sont envoyées les données lorsque le formulaire est soumis.
- **method :** La méthode HTTP utilisée pour envoyer les données : POST ou GET.

Création de formulaire avec la classe Form (3)

Exemple: Un formulaire qui affiche des champs de saisie pour une inscription :

testForm.html

```
<div class="col-md-12 text-center">
        <h2>Saisie de forunisseur ..... </h2>
    </div>

    <form method='POST' class="post-form">
        {% csrf_token %}

        <label >Nom : </label>
        <input id="nom" type="text" name="name_field" > <br/>

        <label >Adresse : </label>
        <input id="adr" type="text" name="name_field" > <br/>

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

Le tag **{% csrf-token %}** permet de sécuriser la requête tout simplement (création de clé).

Création de formulaire avec la classe Form (4)

La syntaxe de déclaration pour **un Form** est très semblable à celle utilisée pour déclarer **un Model**, et partage les mêmes types de champs.

forms.py

```
from django import forms

class FrsForm(forms.Form):
        nom=forms.CharField(max_length=30)
    adr=forms.CharField(widget=forms.Textarea)
```

Permet double contrôle :

- le navigateur empêchera automatiquement la saisie exagérée ;

- et Django validera que les données respectent cette longueur

À chaque champ de formulaire correspond une classe Widget, qui elle-même se réfère à un composant de formulaire HTML comme par exemple <input type="textarea">.

Dans la plupart des cas, le composant par défaut du champ convient bien. Par exemple, le composant par défaut du champ ***CharField*** est ***TextInput***, qui produit une balise ***<input type="text">*** en HTML.

Si une balise ***<textarea>*** est utile, il faut définir le composant approprié lors de la définition du champ du formulaire (le champ message de type ***Textarea***).

Création de formulaire avec la classe Form (5)

La vue doit être en mesure de savoir si elle est appelée pour la première fois

views.py

```
from .forms import FrsForm
def index(request):
    if request.method == "POST" :
        form = FrsForm(request.POST)

        if form.is_valid():
            nom = form.cleaned_data['nom']
            adr = form.cleaned_data['adr']
                        frs=Fournisseur()
            frs.nom=nom
            frs.adr=adr
            form.save()
    else :
        form = FrsForm()

    return
render(request,'magasin/testForm.html',{'form':form})
```

pour repérer des requêtes de type validation de formulaire

crée une instance de formulaire et la complète avec les données reçues à partir de la requête (lier les données au formulaire).

traiter les données dans **form.cleaned_data** selon les besoins.

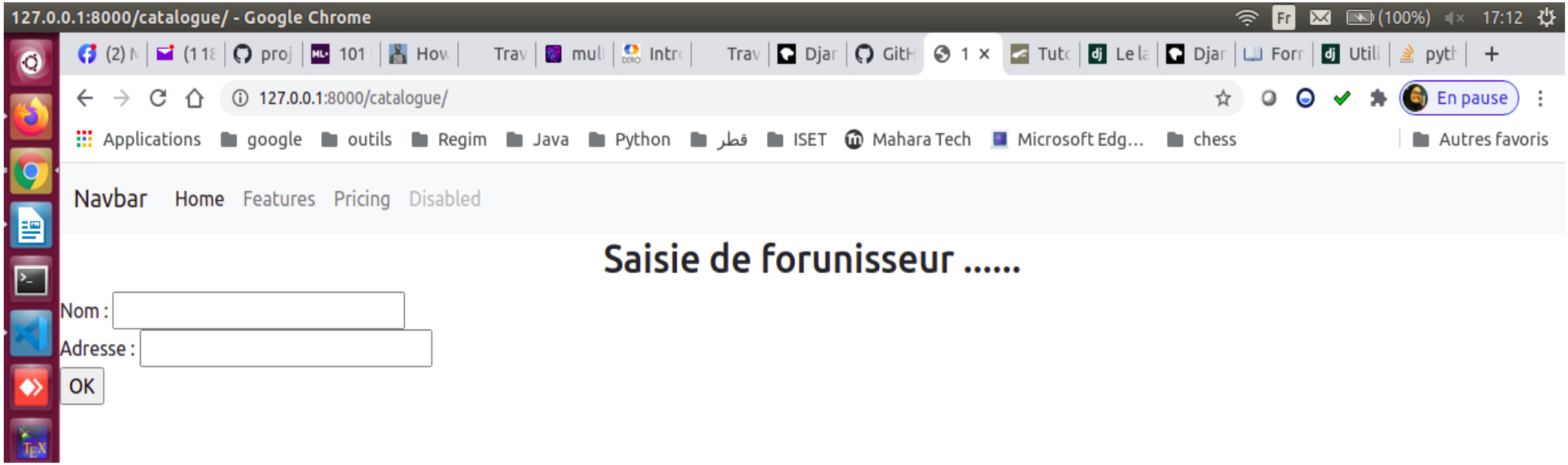
pour identifier une requête initiale de création de formulaire

crée une instance de formulaire vierge. C'est à quoi l'on peut s'attendre lors du premier accès.

is_valid() : permet de vérifier si les données du formulaire sont valides ou non. Lorsque les données ont été validées avec succès (is_valid()=True), les données du formulaire sont mises dans le dictionnaire **form.cleaned_data**.

Ces données peuvent être utilisées pour mettre à jour la base de données ou effectuer d'autres opérations avant de renvoyer au navigateur une redirection

Création de formulaire avec la classe Form (6)



Remarquer que le save n'a aucune influence sur la base de données.

Création de formulaire avec la classe ModelForm (1)

ModelForm peut être considéré comme une sous-classe de **Form**.

Form et **ModelForm** héritent effectivement de comportements communs provenant de la classe (privée) **BaseForm**.

si un formulaire est destiné à ajouter ou modifier directement un modèle Django, un formulaire **ModelForm** peut vous économiser beaucoup de temps, d'effort et de code, car il s'occupe de construire un formulaire avec tous les champs et attributs appropriés à partir d'une classe **Model**.

Création de formulaire avec la classe ModelForm (2)

Etape 1 : Un **formulaire** est défini sous forme d'une classe qui hérite de la classe **ModelForm**.

forms.py

```
from django.forms import ModelForm
from .models import Produit
class ProduitForm(ModelForm):
        class Meta :
                model = Produit
                #fields = "__all__"
                fields=['libelle','description']
```

- **fields** : précise les champs à afficher dans le formulaire
- si **fields="__all__"**, il s'agit d'afficher tous les champs du modèle
- La classe **Meta** indique à Django qu'on va utiliser le modèle **Produit**. Donc le formulaire qui sera créé va contenir les mêmes champs définis dans le modèle.

Création de formulaire avec la classe ModelForm (3)

Etape 2 :

Les données du formulaire sont traitées par une vue

views.py

```
from .models import Produit
from .formulaire import ProduitForm

def index(request):
    form = ProduitForm()
    return render(request,'magasin/index.html',{'form':form})
```

ProduitForm() créer une instance du formulaire dans la vue à destination de l'URL à laquelle il doit apparaître.

Création de formulaire avec la classe ModelForm (4)

Etape 3 : Préparation du formulaire HTML à afficher :

index.html

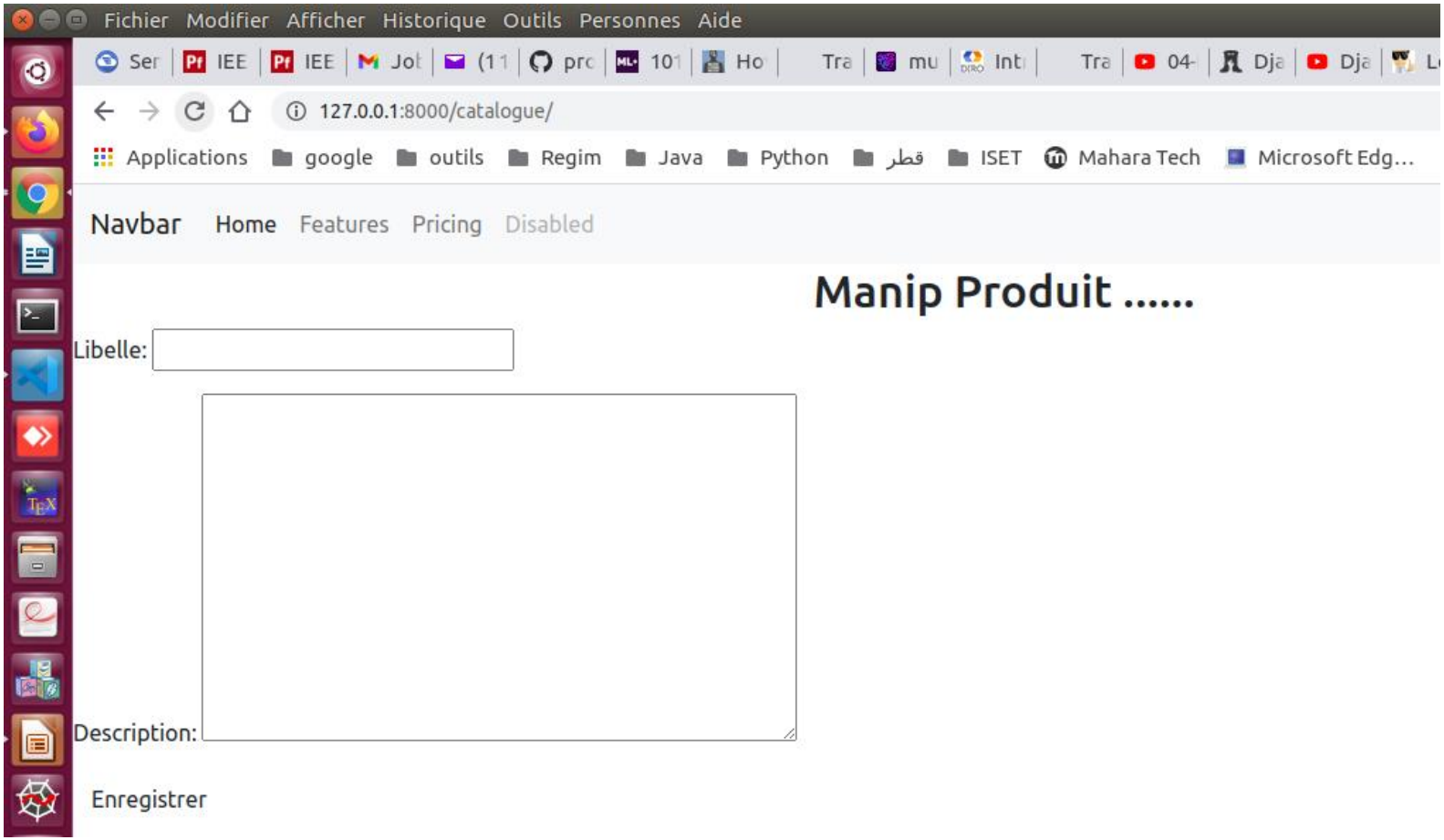
```
{% block body %}
    <div class="col-md-12 text-center">
        <h2>Manip Produit ..... </h2>
    </div>
    <form method='POST' class="post-form">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="sumit" class="btn btn-default">Enregistrer</button>
    </form>
{% endblock %}
```

{{ form.as_p }} : le contenu de **form** sera inséré dans le formulaire. **as_p()** convertit les champs du formulaire et leurs attributs en balises HTML à partir de {{ form }}.

Lorsque le formulaire aura été envoyé, la requête POST envoyée au serveur contiendra les données du formulaire.

Création de formulaire avec la classe ModelForm (5)

Résultat :



NB : ce formulaire n'est pas encore fonctionnel car nous n'avons pas encore traité la requête POST qui est précisée dans **index.html**

POST est utilisée si l'utilisateur a saisie des données et a pressé un bouton envoyer ou enregistrer

Formulaire lié

formulaires liées et non liées :

- **Un formulaire non renseigné** n'a aucune donnée associée. Lorsqu'il est présenté à l'utilisateur, il sera vide ou ne contiendra que des valeurs par défaut.
- **Un formulaire renseigné** contient des données envoyées et il est donc possible de lui demander si ces données sont valides.

  Si un formulaire renseigné non valide est affiché, il peut contenir des messages d'erreur intégrés indiquant à l'utilisateur quelles sont les données à corriger.

L'attribut **is_bound** d'un formulaire indique si des données ont été liées au formulaire ou pas.

Activation

Etape 4 : insérer le formulaire dans le **views.py**

views.py

```
from .formulaire import ProduitForm
from django.shortcuts import render, redirect
def index(request):
    if request.method == "POST" :
        form = ProduitForm(request.POST).save()
        return redirect('/magasin')
    else :
        form = ProduitForm()   #créer formulaire vide
    return render(request,'magasin/index.html',{'form':form})
```

- Le **if** permet de créer un formulaire avec enregistrement de données
- La méthode **save()** permet d'enregistrer les données postées
- Le **return** permet de rediriger l'utilisateur.

Si le formulaire est envoyé par une requête **POST**, la vue crée également une instance de formulaire et la complète avec les données reçues à partir de la requête : **form = NameForm(request.POST)**. C'est ce qu'on appelle « lier les données au formulaire »

Amélioration

Etape 5 : Si nous voulons afficher la liste des produits en bas du formulaire :

views.py

```
from .formulaire import ProduitForm
from django.shortcuts import render, redirect

def index(request):
    if request.method == "POST" :
        form = ProduitForm(request.POST).save()
        return redirect('/magasin')
    else :
        form = ProduitForm()

    list=Produit.objects.all()
    return render(request,'magasin/index.html',{ 'form':form, 'list':list })
```

Amélioration

Etape 6 : Puis la récupération et l'affichage de la liste des produits.

index.html

```
{% block body %}
    ...
    <form method='POST' class="post-form">
       ....
    </form>

    {% for article in list %}
        {{article.libelle}}<br/>
        {{article.description}}<br/>
        ------------------------<br/>
    {% endfor %}
{% endblock %}
```

NB : Il est possible de mettre en forme des enregistrements sous forme de tableau.