Chapter-5

Working with Django Forms

Django Forms:

- It is the very important concept in web development.
- The main purpose of forms is to take user input.
  Eg: login form, registration form, enquiry form etc
- From the forms we can read end user provided input data and we can use that data based on requirement. We may store in the database for future purpose. We may use just for validation/authentication purpose etc
- Here we have to use Django specific forms but not HTML forms.

Advantages of Django Forms over HTML Forms:

1) We can develop forms very easily with python code
2) We can generate HTML Form widgets/components (like textarea, email, pwd etc) very quickly
3) Validating data will become very easy
4) Processing data into python data structures like list, set etc will become easy
5) Creation of Models based on forms will become easy etc.

Process to generate Django Forms:

Step-1: Creation of forms.py file in our application folder with our required fields.

forms.py:

```python
from django import forms
class StudentForm(forms.Form):
    name=forms.CharField()
    marks=forms.IntegerField()
```

Note: name and marks are the field names which will be available in html form

Step-2: usage of forms.py inside views.py file:

views.py file is responsible to send this form to the template html file

views.py:

```python
from django.shortcuts import render
from . import forms

# Create your views here.
def studentinputview(request):
    form=forms.StudentForm()
```

```
    my_dict={'form':form}
    return render(request,'testapp/input.html',context=my_dict)
```

# Alternative Short Way:

```
def studentinputview(request):
    form=forms.StudentForm()
    return render(request,'testapp/input.html',{'form':form})
```

Note: context parameter is optional.We can pass context parameter value directly without using keyword name 'context'

Step-3: Creation of html file to hold form:

Inside template file we have to use template tag to inject form {{form}}

It will add only form fields. But there is no <form> tag and no submit button.
Even the fields are not arranged properly.It is ugly form.

We can make proper form as follows

```
<h1>Registration Form</h1>
 <div class="container" align="center">
   <form  method="post">
  {{form.as_p}}
  <input type="submit" class="btn btn-primary" name="" value="Submit">
   </form>
 </div>
```

input.html:

```
<!DOCTYPE html>
{%load staticfiles%}
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="{%static "css/bootstrap.css"%}">
    <link rel="stylesheet" href="{%static "css/demo2.css"%}">
    <title></title>
  </head>
  <body>
    <h1>Registration Form</h1>
    <div class="container" align="center">
      <form  method="post">
          {{form.as_p}}
```

```
            <input type="submit" class="btn btn-primary" name="" value="Submit">
        </form>
      </div>
    </body>
  </html>
```

If we submit this form we will get 403 status code response
Forbidden (403)
CSRF verification failed. Request aborted.

Help
Reason given for failure:

  CSRF token missing or incorrect.

CSRF (Cross Site Request Forgery) Token:

- Every form should satisfy CSRF (Cross Site Request Forgery) Verification, otherwise Django won't accept our form.
- It is meant for website security. Being a programmer we are not required to worry anything about this. Django will takes care everything.
- But we have to add csrf_token in our form.

```
<h1>Registration Form</h1>
    <div class="container" align="center">
        <form   method="post">
         {{form.as_p}}
         {% csrf_token %}
         <input type="submit" class="btn btn-primary" name="" value="Submit">
        </form>
    </div>
```

If we add csrf_token then in the generate form the following hidded field will be added,which makes our post request secure

```
<input type='hidden' name='csrfmiddlewaretoken'
value='1ZqIJJqTLMVa6RFAyPJh7pwzyFmdiHzytLxJIDzAkKULJz4qHcetLoKEsRLwyz4h'/>
```

The value of this hidden field is keep on changing from request to request.Hence it is impossible to forgery of our request.

If we configured csrf_token in html form then only django will accept our form.

How to process Input Data from the form inside views.py File:

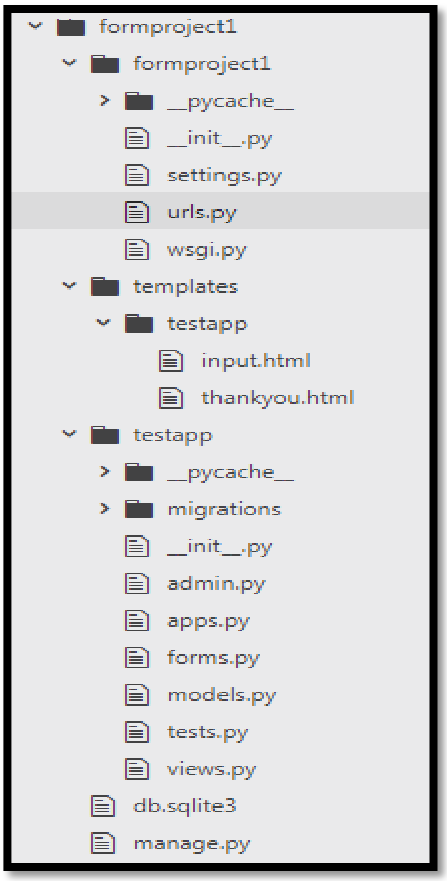
We required to modify views.py file. The end user provided input is available in a dictionary named with 'cleaned_data'

views.py:

```python
from django.shortcuts import render
from . import forms

# Create your views here.
def studentinputview(request):
    form=forms.StudentForm()
    if request.method=='POST':
        form=forms.StudentForm(request.POST)
        if form.is_valid():
            print('Form validation success and printing data')
            print('Name:',form.cleaned_data['name'])
            print('Marks:',form.cleaned_data['marks'])
    return render(request,'testapp/input.html',{'form':form})
```

Student Name and Marks Submission Form Project (formproject1):



```
formproject1
    formproject1
        __pycache__
        __init__.py
        settings.py
        urls.py
        wsgi.py
    templates
        testapp
            input.html
            thankyou.html
    testapp
        __pycache__
        migrations
        __init__.py
        admin.py
        apps.py
        forms.py
        models.py
        tests.py
        views.py
    db.sqlite3
    manage.py
```

forms.py

```python
from django import forms
class StudentForm(forms.Form):
    name=forms.CharField()
    marks=forms.IntegerField()
```

views.py

```python
from django.shortcuts import render
from testapp.forms import StudentForm
# Create your views here.
def student_input_view(request):
    sent=False
    if request.method=='POST':
        form=StudentForm(request.POST)
        if form.is_valid():
            print('Form Validation Success and printing data')
            print('Name:',form.cleaned_data['name'])
            print('Marks:',form.cleaned_data['marks'])
            sent=True
    form=StudentForm()
    return render(request,'testapp/input.html',{'form':form,'sent':sent})
```

input.html

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
    <!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
  </head>
  <body>
    <div class="container" align='center'>
      {% if sent %}
          <h1>Thanks for Providing Information</h1>
          <h2>Enter Next Student Marks</h2>
      {%else%}
          <h1>Student Marks Submit Form</h1>
      {%endif%}<hr>
```

```
    <form method='POST'>
      {{form.as_p}}
      {%csrf_token%}
      <input type="submit" name="" class='btn btn-lg btn-primary' value="Submit Marks">
    </form>
  </div>
 </body>
</html>
```

thankyou.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <h1>Thanks for providing your Name and Marks</h1>
  </body>
</html>
```

urls.py

```
from django.conf.urls import url
from django.contrib import admin
from testapp import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^input/', views.student_input_view),
]
```

Student FeedBack Form Project

forms.py:

```
from django import forms

class FeedBackForm(forms.Form):
    name=forms.CharField()
    rollno=forms.IntegerField()
    email=forms.EmailField()
```

```
    feedback=forms.CharField(widget=forms.Textarea)
```

views.py

```
from django.shortcuts import render
from import forms

def feedbackview(request):
   form=forms.FeedBackForm()
   if request.method=='POST':
      form=forms.FeedBackForm(request.POST)
      if form.is_valid():
         print('Form Validation Success and printing information')
         print('Name:',form.cleaned_data['name'])
         print('Roll No:',form.cleaned_data['rollno'])
         print('Email:',form.cleaned_data['email'])
         print('FeedBack:',form.cleaned_data['feedback'])
   return render(request,'testapp/feedback.html',{'form':form})
```

feedback.html

```
<!DOCTYPE html>
{% load staticfiles%}
<html lang="en" dir="ltr">
 <head>
   <meta charset="utf-8">
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/css/bootstrap.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
   <link rel="stylesheet" href="{% static "css/demo4.css" %}">
   <title></title>
 </head>
 <body>
   <div class="container" align='center'>
     <h1>Student Feedback Form</h1><hr>
       <form class="" action="index.html" method="post">
         {{form.as_p}}
         {% csrf_token %}
         <input type="submit"  class="btn btn-primary" value="Submit Feedback">
       </form>
   </div>
 </body>
</html>
```

Form Validations:

Once we submit the form we have to perform validations like

1) Length of the field should not be empty
2) The max number of characters should be 10
3) The first character of the name should be 'd' etc

We can implement validation logic by using the following 2 ways.

1) Explicitly by the programmer by using clean methods
2) By using Django inbuilt validators

Note: All validations should be implemented in the forms.py file

1) Explicitly by the Programmer by using Clean Methods:

- The syntax of clean method: clean_fieldname(self)
- In the FormClass for any field if we define clean method then at the time of submit the form, Django will call this method automatically to perform validations. If the clean method won't raise any error then only form will be submitted.

```python
from django import forms

class FeedBackForm(forms.Form):
    name=forms.CharField()
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)

    def clean_name(self):
        inputname=self.cleaned_data['name']
        if len(inputname) < 4:
            raise forms.ValidationError('The Minimum no of characters in the name field should be 4')
        return inputname
```

- The returned value of clean method will be considered by Django at the time of submitting the form.

forms.py

```python
from django import forms
from django.core import validators

class FeedBackForm(forms.Form):
    name=forms.CharField()
```

```
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)

    def clean_name(self):
        print('validating name')
        inputname=self.cleaned_data['name']
        if len(inputname) < 4:
            raise forms.ValidationError('The Minimum no of characters in the name field should be 4')
        return inputname+'durga'
    def clean_rollno(self):
        inputrollno=self.cleaned_data['rollno']
        print('Validating rollno field')
        return inputrollno
    def clean_email(self):
        inputemail=self.cleaned_data['email']
        print('Validating email field')
        return inputemail
    def clean_feedback(self):
        inputfeedback=self.cleaned_data['feedback']
        print('Validating feedback field')
        return inputfeedback
```

Server Console:

```
validating name
Validating rollno field
Validating email field
Validating feedback field
Form Validation Success and printing information
Name: Durgadurga
Roll No: 101
Email: durgaadvjava@gmail.com
FeedBack: This is sample feedback
```

Note:

1) Django will call these filed level clean methods automatically and we are not required to call explicitly.
2) Form validation by using clean methods is not recommended.

2) Django's Inbuilt Core Validators:

- Django provides several inbuilt core validators to perform very common validations. We can use these validators directly and we are not required to implement.
- Django's inbuilt validators are available in the django.core module.

- from django.core import validators
- To validate Max number of characters in the feedback as 40,we have to use inbuilt validators as follows.

<u>**forms.py**</u>

```
from django import forms
from django.core import validators

class FeedBackForm(forms.Form):
    name=forms.CharField()
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea,validators=
[validators.MaxLengthValidator(40)])
```

<u>**Note:**</u> **We can use any number of validators for the same field**

```
feedback = forms.CharField(widget = forms.Textarea,validators =
[validators.MaxLengthValidator(40),validators.MinLengthValidator(10)])
```

<u>**Note:**</u> **Usage of built in validators is very easy when compared with clean methods.**

How to implement Custom Validators by using the same Validator Parameter:

The value of name parameter should starts with 'd' or 'D'. We can implement this validation as follows

```
def starts_with_d(value):
    if value[0].lower() != 'd':
        raise forms.ValidationError('Name should be starts with d | D')

class FeedBackForm(forms.Form):
    name=forms.CharField(validators=[starts_with_d])
    rollno=forms.IntegerField()
```

Validation of Total Form directly by using a Single Clean Method:

Whenever we are submitting the form Django will call clean() method present in our Form class. In that method we can implement all validations.

forms.py

```python
from django import forms
from django.core import validators

class FeedBackForm(forms.Form):
   name=forms.CharField()
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)

    def clean(self):
        print('Total Form  Validation...')
        total_cleaned_data=super().clean()
        inputname=total_cleaned_data['name']
        if inputname[0].lower() != 'd':
            raise forms.ValidationError('Name parameter should starts with d')
        inputrollno=total_cleaned_data['rollno']
        if inputrollno <=0:
            raise forms.ValidationError('Rollno should be > 0')
```

How to Check Original pwd and reentered pwd are same OR not in Signup Form:

forms.py

```python
from django import forms
from django.core import validators

class FeedbackForm(forms.Form):
    name=forms.CharField()
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)

    def clean(self):
        print('Total Form Validation by using single clean method...')
        total_cleaned_data=super().clean()
        inputname=total_cleaned_data['name']
        if inputname[0].lower() !='d':
            raise forms.ValidationError('Name should starts with d|D')
        inputrollno=total_cleaned_data['rollno']
        if inputrollno<=0:
            raise forms.ValidationError('Rollno should be >0')
```

```python
class SignupForm(forms.Form):
    name=forms.CharField()
    password=forms.CharField(widget=forms.PasswordInput)
    rpassword=forms.CharField(label='Reenter Password',widget=forms.PasswordInput)
    email=forms.EmailField()

    def clean(self):
        total_cleaned_data=super().clean()
        pwd=total_cleaned_data['password']
        rpwd=total_cleaned_data['rpassword']
        if pwd != rpwd:
            raise forms.ValidationError('Both Passwords must be same')
```

views.py

```python
from django.shortcuts import render
from testapp.forms import FeedbackForm,SignupForm

# Create your views here.
def feedback_form_view(request):
    form=FeedbackForm()
    # feedback_submitted=False
    if request.method=='POST':
        form=FeedbackForm(request.POST)
        if form.is_valid():
            print('Basic Validation Completed and Printing Data')
            print('Name:',form.cleaned_data['name'])
            print('RollNo:',form.cleaned_data['rollno'])
            print('Email:',form.cleaned_data['email'])
            print('Feedback:',form.cleaned_data['feedback']) # feedback_submitted=True
    return render(request,'testapp/feedback.html',{'form':form})

def signup_form_view(request):
    form=SignupForm()
    if request.method=='POST':
        form=SignupForm(request.POST)
        if form.is_valid():
            print('Basic Validation Completed and Printing Data')
            print('Name:',form.cleaned_data['name'])
            print('Password:',form.cleaned_data['password'])
            print('Email:',form.cleaned_data['email'])
    return render(request,'testapp/signup.html',{'form':form})
```

signup.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title></title>
    <!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
  </head>
  <body>
    <div class="container" align='center'>
        <h1>Student Signup Form</h1><hr>
        <form method="post">
          {{form.as_p}}
          {%csrf_token%}
          <input type="submit" class='btn btn-lg btn-warning' value="Signup">
        </form>
    </div>
  </body>
</html>
```

Note: If we want to perform validations by comparing multiple input fields then we should go for single clean() method.

How to prevent Requests from BOT:

Generally form requests can be send by end user. Sometimes we can write automated programming script which is responsible to fill the form and submit. This automated programming script is nothing but BOT.

The main objectives of BOT requests are:

1) To create unnecessary heavy traffic to the website, which may crash our application.
2) To spread malware (viruses)

Being web developer compulsory we have to think about BOT requests and we have to prevent these requests.

How to prevent BOT Requests:

- In the form we will place one hidden form field, which is not visible to the end user. Hence there is no chance of providing value to this hidden field.
- But BOT will send the value for this hidden field also.If hidden field got some value means it is the request from BOT and prevent that form submission.

```
from django import forms
from django.core import validators

class FeedBackForm(forms.Form):
    name=forms.CharField()
    password=forms.CharField(widget=forms.PasswordInput)
    rpassword=forms.CharField(label='Re Enter Password',
widget=forms.PasswordInput)
    rollno=forms.IntegerField()
    email=forms.EmailField()
    feedback=forms.CharField(widget=forms.Textarea)
    bot_handler=forms.CharField(required=False,widget=forms.HiddenInput)

    def clean(self):
        print('validating passwords match...')
        total_cleaned_data=super().clean()
        fpwd=total_cleaned_data['password']
        spwd=total_cleaned_data['rpassword']
        if fpwd != spwd:
            raise forms.ValidationError('Both passwords must be matched')
        bot_handler_value=total_cleaned_data['bot_handler']
        if len(bot_handler_value)>0:
            raise forms.ValidationError('Request from BOT...cannot be submitted!!!')
```

Notes with BOT Field:

```
class FeedBackForm(forms.Form):
    normal fields..
    bot_handler=forms.CharField(required=False,widget=forms.HiddenInput)

    def clean(self):
        total_cleaned_data=super().clean()
        bot_handler_value=total_cleaned_data['bot_handler']
        if len(bot_handler_value)>0:
            raise forms.ValidationError('Request from BOT...cannot be submitted!!!')
```

Note: Other ways to prevent BOT requests

1) By using Captchas
2) By using image recognizers
   (like choose 4 images where car present)

Note: Other way to create project → py -m django startproject modelformproject