Django Model Form

It is a class which is used to create an HTML form by using the Model. It is an efficient way to create a form without writing HTML code.

Django automatically does it for us to reduce the application development time. For example, suppose we have a model containing various fields, we don't need to repeat the fields in the form file.

For this reason, Django provides a helper class which allows us to create a Form class from a Django model.

Csrf_token : **CSRF** stands for Cross-Site Request Forgery and it's a type of Cross Site Scripting attack that can be sent from a malicious site through a visitor's browser to your server.

Django ModelForm Example

First, create a model that contains fields name and other metadata. It can be used to create a table in database and dynamic HTML form.

1. Create project
2. Create app
3. Apply migrations
4. **// model.py**

```
      from django.db import models

      # Create your models here.

      class Student(models.Model):
          rno = models.IntegerField()
          sname = models.CharField(max_length=25)
          address = models.TextField
          contact = models.IntegerField()

# class Meta:
     #db_table = "employee"
```

5. Form.py

```
      from django import forms
      from mediapp.models import Student

      class StudForm(forms.ModelForm):
          class Meta:
              model = Student
              fields = "__all__"
```

6. View.py

```
from django.shortcuts import render
from mediapp.form import StudForm

# Create your views here.

def index(request):
    s = StudForm()
    return render(request,'index.html',{'form':s})
```

7. url.py

```
Add path for written view module
```

8. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Model Form</title>
</head>
<body>
    <form method="post" class="post-form">
        {% csrf_token %}
        {{ form.as_p}}
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Django Form Validation

Django provides built-in methods to validate form data automatically. Django forms submit only if it contains CSRF tokens. It uses uses a clean and easy approach to validate data.

The **is_valid()** method is used to perform validation for each field of the form, it is defined in Django Form class. It returns True if data is valid and place all data into a cleaned_data attribute.

1. Create project
2. Create app
3. Apply migrations
4. **// model.py**

```
class Employee(models.Model):
    eid = models.CharField(max_length=20)
    ename = models.CharField(max_length=100)
    econtact = models.CharField(max_length=15)
    class Meta:
        db_table = "employee"
```

5. Form.py

```
from mediapp.models import Student,Employee

class EmployeeForm(forms.ModelForm):
    class Meta:
        model = Employee
        fields = "__all__"
```

6. View.py

```
from mediapp.form import StudForm,EmployeeForm

# Create your views here.
def emp(request):
    if request.method == "POST":
        form = EmployeeForm(request.POST)
        if form.is_valid():
            try:
                return redirect('/')
            except:
                pass
    else:
        form = EmployeeForm()
    return
render(request,'empin.html',{'form':form})
```

7. url.py

Add path for written view module

8. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Index</title>
</head>
<body>
<form method="POST" class="post-form"
enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-
default">Save</button>
</form>
</body>
</html>
```

Django File Upload

File upload to the server using Django is a very easy task. Django provides built-in library and methods that help to upload a file to the server.

The **forms.FileField()** method is used to create a file input and submit the file to the server. While working with files, make sure the HTML form tag contains **enctype="multipart/form-data"** property.

Let's see an example of uploading a file to the server. This example contains the following files.

1. Create project
2. Create app
3. Apply migrations
4. Create static directory → inside it upload directory
5. Create 'functions' directory in app and put functions.py into it.
6. **// model.py**

```
class Employee(models.Model):
    eid = models.CharField(max_length=20)
    ename = models.CharField(max_length=100)
    econtact = models.CharField(max_length=15)
    class Meta:
        db_table = "employee"
```

7. Form.py

```
from mediapp.models import Student,Employee

class StudentForm(forms.Form):
    firstname = forms.CharField(label="Enter first name",max_length=50)
    lastname  = forms.CharField(label="Enter last name", max_length = 10)
    email     = forms.EmailField(label="Enter Email")
    file      = forms.FileField() # for creating file input
```

8. View.py : Here, one extra parameter **request.FILES** is required in the constructor. This argument contains the uploaded file instance.

```
from django.shortcuts import render
from django.http import HttpResponse
from myapp.functions.functions import handle_uploaded_file
from myapp.form import StudentForm
def index(request):
    if request.method == 'POST':
        student = StudentForm(request.POST, request.FILES)
        if student.is_valid():
            handle_uploaded_file(request.FILES['file'])
.           return HttpResponse("File uploaded successfuly")
.   else:
.       student = StudentForm()
.       return render(request,"index.html",{'form':student})
```

9. url.py

```
Add path for written view module
```

10. index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Index</title>
</head>
<body>
<form method="POST" class="post-form" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
</form>
        </body>
        </html>
```

11. Functions.py

```
def handle_uploaded_file(f):
    with open('myapp/static/upload/'+f.name, 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```

Django Database Connectivity

The **settings.py** file contains all the project settings along with database connection details. By default, Django works with **SQLite,** database and allows configuring for other databases as well.

Database connectivity requires all the connection details such as database name, user credentials, hostname drive name etc.

To connect with MySQL, **django.db.backends.mysql** driver is used to establishing a connection between application and database. Let's see an example.

We need to provide all connection details in the settings file. The settings.py file of our project contains the following code for the database.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'djangoApp',
        'USER':'root',
        'PASSWORD':'mysql',
        'HOST':'localhost',
        'PORT':'3306'
    }
.}
```