# Django Forms Concepts

- A form is a collection of fields and widgets which are used to get the information from user.
- It is very important concept in web development. The main purpose of forms is to take user input.
- Forms provides communication between user and admin or management of website.
- We can create different kind of forms using HTML or django forms module.
- **For example**: Login form, Registration form, Enquiry form, Create 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 this data 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.

## Django Forms:

- Django providing the "forms" module to developing the required form fields with out using HTML tags code.
- We can import forms module like below,

  **from django import forms**
- **Generally Django supports two types of form classes. They are,**
  1. forms.Form
  2. forms.ModelForm

- When we use **forms.Form** then we have to specify all model fields into **forms.py** which will lead to duplicate coding, it also takes more time to repeate the same fields and fieldtypes in the forms file.
- If we want use the same model fields and with fieldtypes in the form class then we can go for forms.ModelForm class.
- If we use **forms.ModelForm** then we can reduce the code in **forms.py** file and in views.py also we reduce lot of code to directly access the form data and also directly we can save the data by using .save().

## Advantages of Django Forms over HTML forms:

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

**Note : Create a form class code is like creating the model class code:**

**For example:**

open models.py

```
from django.db import models
class Employee(models.Model):
    ename = models.CharField(max_length=50)
    salary = models.FloatField()
    address = models.TextField(max_length=200)
    mobile = models.BigIntegerField()
```

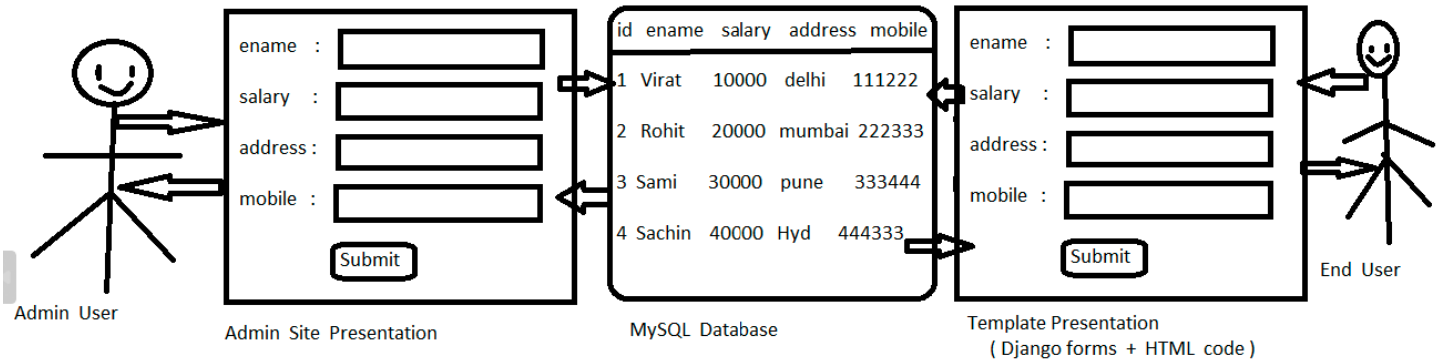
open forms.py

```
from django import forms
class EmployeeForm(forms.Form):
    ename = forms.CharField()
    salary = forms.FloatField()
    address = forms.CharField()
    mobile = forms.IntegerField()
```

- When we run the models then the **models.py code converted into SQL Code** and **creates database table** in the db.
- When we run the forms then the **form.py code will convert into HTML Code** and **creates a html form** on the browser.

Admin Users and End Users of our application can communicate the Database to managing data of our application.

For example, see the below diagram




- Admin Users by using admin site presentation , will interact with database of our application.
- End Users by using django template presentation , will interact with database of our application.
- Django template presentation by using pure HTML code and django forms code we can design.

## Create a forms.py file to check how form.py code converted into HTML code

- Create one python file name as **forms.py** inside our application name.

  **For example:**

```
applicationName
      ....
      ....
      forms.py
      ...
      ...
```

- Now Create Userdefined form classes using these Predefined form classes inside form.py.

  **For example:**

```
from django import  forms
class EmployeeForm(forms.Form):
    ename = forms.CharField()
    salary = forms.FloatField()
    address = forms.CharField()
    mobile = forms.IntegerField()
```

Step1 : Open the python console or python shell using pycharm or command prompt.

Step2 : Import the Usedefined form class into python console and create the object for this form class.

Step3: Now print this form class object then we will get form fields related HTML code on python console.

**For example:**

**Terminal>** **python manage.py shell**

```
>>>
>>> from  applicationName.forms import EmployeeForm
>>> form = EmployeeForm ()
>>> print(form)
```

**Output:**

```
<tr>
    <th><label for="id_ename">Ename:</label></th>
    <td><input type="text" name="ename" required id="id_ename"> </td>
</tr>

<tr>
  <th><label for="id_salary">Salary:</label></th>
  <td><input type="number" name="salary" step="any" required id="id_salary"></td>
</tr>

<tr>
    <th><label for="id_address">Address:</label></th>
    <td><input type="text" name="address" required id="id_address"></td>
</tr>

<tr>
  <th><label for="id_mobile">Mobile:</label></th>
  <td><input type="number" name="mobile" required id="id_mobile"></td>
</tr>
```

## Process to generate Django forms:

**Step-1: Create the forms.py file in our application folder with our required fields.**

**forms.py:**

```
from django import  forms
class EmployeeForm(forms.Form):
    ename = forms.CharField()
    salary = forms.FloatField()
    address = forms.CharField()
    mobile = forms.IntegerField()
```

**Note**: ename, salary, address and mobile fields 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 object code to the template html file
- import our Form class from forms.py into views.py file
- Create form object using Form class to generating HTML code for Form class fields.

**views.py:**

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

def EmployeeFormView(request):
    form = EmployeeForm()
    context = {
        "form" : form
    }
    return render(request, 'employeeapp/employee_form.html' , context)
```

**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 or variable name syntax to inject form object like {{ form }} .
- It will add only form fields. But there is no **<form>** tag and no **<input type="submit">** button. Even the fields are not arranged properly. It is ugly form.

  **For example:** open **employee_form.html** file

```
<div class="container" align="center">
       {{ form }}
</div>
```

  Note: It display like below

Ename: Salary: Address: Mobile:

- We can make proper form by adding **<form>, <input type="submit">** tags and **{{ form.as_p }}** structure. Here **as_p** converts every form tag field into paragraph.

  **For example:** open **employee_form.html** file

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

  Note: It display like below

Ename:

Salary:

Address:

Mobile:

Submit

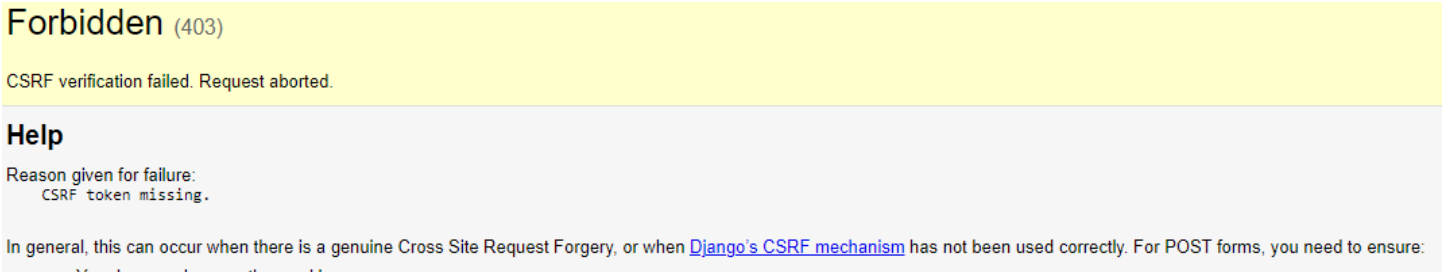
- If we submit this form we will get 403 status code response like **Forbidden (403)**

CSRF verification failed. Request aborted.

**Help**

Reason given for failure:

CSRF token missing or incorrect.




- Every form should satisfy **CSRF (Cross Site Request Forgery**) Verification, otherwise Django won't accept our form data with POST method request.
- It is meant for website security. Being a programmer we are not required to worry anything about this.
- Django will takes care everything. But to over come this above problem then we have to add csrf_token in our <form> like **{% csrf_token %}**

  **For example:**

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

- **NOTE:** If we add csrf_token then in the generate form will have following hidded field added, which makes our post request more 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.
- **Note:** 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**'
- **Now open views.py and create below view**

```
def  EmployeeView(request):
    if request.method == 'POST':
        form = EmployeeForm(request.POST)
        if form.is_valid():
            print('Form validation success and printing data')
```

```
            print('Employee Name:', form.cleaned_data['ename'])
            print('Employee Salary:', form.cleaned_data['salary'])
            print('Employee mobile:', form.cleaned_data['mobile'])
            print('Employee Address:', form.cleaned_data['address'])
        else:
            return HttpResponse('Form data not validated')
    else:
        form = forms.EmployeeForm()
        return render(request,'employeeapp/employee_form.html',{'form':form})
```

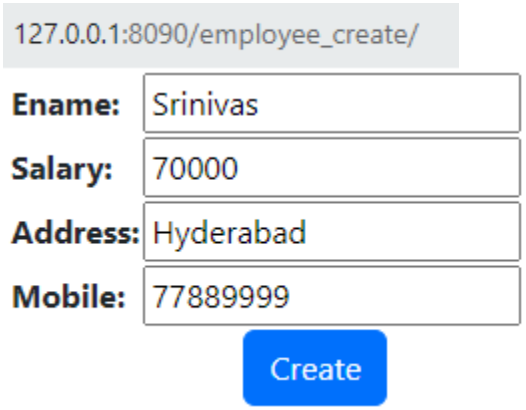
- **Open urls.py file and create the required urls to execute the required views**

```
from employeeapp  import  views

urlpatterns = [
   '------',
   path('employee_create/', views.EmployeeView)

   '------'
]
```

- **Now execute the runserver command and execute url from browser like below**




- **Now Enter required data  and submit data  then we will get like below on python console**

**Output:**

```
Form validation success and printing data
Employee Name: Srinivas
Employee Salary: 70000
Employee mobile: 77889999
Employee Address: Hyderabad
```