

# TRAINING DAY7 REPORT

01 JULY 2025

## CREATING FORMS IN DJANGO

A form is an HTML element used to collect data from users. In Django, forms are used to receive input, validate data, and process it safely and efficiently.

### Ways to Create Forms in Django:

- Using HTML Forms (Manual)
- Using Django Forms (Form class)
- Using ModelForm (auto-generated from models)

#### a) Using HTML Forms

You can manually create an HTML form inside a template:

```
<form method="POST" action="/submit/">
    {% csrf_token %}
    <input type="text" name="name" placeholder="Your Name">
    <input type="email" name="email" placeholder="Your Email">
    <button type="submit">Submit</button>
</form>
```

- Requires handling the data manually in the view.
- Use `{% csrf_token %}` to prevent CSRF attacks.

#### b) Django Form Class

Create a form using forms.py:

```

blogapi > blog > forms.py > ...
1  from django import forms
2
3  class ContactForm(forms.Form):
4      name = forms.CharField(max_length=100)
5      email = forms.EmailField()
6      message = forms.CharField(widget=forms.Textarea)
7
8
9
10 |

```

- Provides built-in validation.
- Automatically renders HTML widgets.

### c) Django ModelForm

ModelForm is used when the form is directly linked to a database model.

```

blogapi > blog > forms.py > ...
1  from django.forms import ModelForm
2  from .models import Contact
3
4  class ContactForm(ModelForm):
5      class Meta:
6          model = Contact
7          fields = ['name', 'email', 'message']
8

```

- Simplifies code.
- Saves data to the database directly using form.save().

## HANDLING FORM INPUT

```
blogapi > blog > views.py > ...
1  from django.shortcuts import render
2  from django.http import HttpResponseRedirect
3  from .forms import ContactForm
4
5  def contact_view(request):
6      if request.method == "POST":
7          form = ContactForm(request.POST)
8          if form.is_valid():
9              form.save()
10             return HttpResponseRedirect("Thanks for contacting us!")
11      else:
12          form = ContactForm()
13      return render(request, 'contact.html', {'form': form})
14
```

### Steps:

1. Check if request is POST.
2. Bind data using `form = ContactForm(request.POST)`.
3. Validate using `form.is_valid()`.
4. Save to DB or process as needed.
5. On GET, show an empty form.

## GET AND POST METHODS

What is GET?

- Sends data via URL query parameters.
- Used for searching, filtering, or non-sensitive data.
- Does not modify server data.
- Example URL: `/search/?q=django`

What is POST?

- Sends data inside HTTP request body.
- Used for form submissions, user login, creating records.
- Data is not visible in the URL.
- Safer and supports large data.

## Differences Between GET and POST:

Feature	GET	POST
Visibility	URL (visible)	Request body (invisible)
Use Case	Read/search data	Submit/modify data
Data Limit	Limited (URL length)	No significant limit
Caching	Can be cached	Not cached
Security	Less secure (data in URL)	More secure (CSRF protected)

Get and Post examples -->

```
blogapi > blog > views.py > ...
15
16
17 def search_view(request):
18     query = request.GET.get('q')
19     results = BaseModel.objects.filter(name__icontains=query)
20     return render(request, 'search.html', {'results': results})
21
22
23 def submit_form(request):
24     if request.method == 'POST':
25         name = request.POST.get('name')
26         # Save or process data
27
```