• Using Django’s built-in form handling.

Django’s built-in form handling provides a structured and convenient way to manage user input, validate data, and handle errors:

**1. Defining Forms**

Django provides a forms module that allows you to define forms as Python classes. These classes map directly to HTML form elements.

**Example:**

* python

from django import forms

class ContactForm(forms.Form):

name = forms.CharField(max\_length=100)

email = forms.EmailField()

message = forms.CharField(widget=forms.Textarea)

Here:

* CharField and EmailField define input fields.
* widget=forms.Textarea specifies the use of a <textarea> for the message.

**2. Rendering Forms in Templates**

To display the form on a webpage, pass it to a template and render it.

**Example View:**

* python

from django.shortcuts import render

from .forms import ContactForm

def contact\_view(request):

form = ContactForm()

return render(request, 'contact.html', {'form': form})

**Example Template (contact.html):**

* html

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Submit</button>

</form>

* {{ form.as\_p }} renders the form fields with <p> tags.
* You can also use {{ form.as\_table }} or {{ form.as\_ul }} for alternative layouts.

**3. Handling Form Submission**

When a form is submitted, validate the input data in your view.

**Example View:**

* python

def contact\_view(request):

if request.method == 'POST':

form = ContactForm(request.POST)

if form.is\_valid():

# Process the data

name = form.cleaned\_data['name']

email = form.cleaned\_data['email']

message = form.cleaned\_data['message']

# Add logic to save or send data

return render(request, 'success.html')

else:

form = ContactForm()

return render(request, 'contact.html', {'form': form})

* form.is\_valid() checks whether the form data meets validation criteria.
* form.cleaned\_data provides access to validated data.

**4. Using Model Forms**

Django’s ModelForm automates form creation for models, linking form fields to model fields.

**Example:**

* python

from django import forms

from .models import Contact

class ContactForm(forms.ModelForm):

class Meta:

model = Contact

fields = ['name', 'email', 'message']

This eliminates the need to define fields manually.

**5. Adding Custom Validation**

You can add custom validation methods for specific fields or the entire form.

**Field Validation:**

* python

class ContactForm(forms.Form):

email = forms.EmailField()

def clean\_email(self):

email = self.cleaned\_data.get('email')

if not email.endswith('@example.com'):

raise forms.ValidationError('Email must be from example.com domain.')

return email

**Form-wide Validation:**

* python

class ContactForm(forms.Form):

name = forms.CharField(max\_length=100)

email = forms.EmailField()

def clean(self):

cleaned\_data = super().clean()

name = cleaned\_data.get('name')

email = cleaned\_data.get('email')

if name and email and "admin" in name.lower():

raise forms.ValidationError("Admin cannot be part of the name.")

**6. Handling Errors**

Errors are automatically added to the form and can be displayed in the template using {{ form.errors }}.

• Implementing Django’s authentication system (sign up, login, logout, password management).

Implementing Django’s authentication system involves several steps to manage user authentication, including sign up, login, logout, and password management:

**1. Setting Up Django Authentication**

First, ensure you have Django installed and a project set up. If not, you can start by creating a new project and app:

django-admin startproject myproject

cd myproject

python manage.py startapp myapp

**2. Configuring Settings**

Ensure that myapp is added to your INSTALLED\_APPS in settings.py. Also, configure the AUTHENTICATION\_BACKENDS if needed.

* python

# myproject/settings.py

INSTALLED\_APPS = [

...

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'myapp',

...

]

AUTHENTICATION\_BACKENDS = [

'django.contrib.auth.backends.ModelBackend',

]

**3. Creating User Sign-Up View**

Create a form for user registration using Django’s built-in UserCreationForm.

**forms.py**:

* python

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

class SignUpForm(UserCreationForm):

email = forms.EmailField(required=True)

class Meta:

model = User

fields = ('username', 'email', 'password1', 'password2')

**views.py**:

* python

from django.shortcuts import render, redirect

from django.contrib.auth import login, authenticate

from .forms import SignUpForm

def signup\_view(request):

if request.method == 'POST':

form = SignUpForm(request.POST)

if form.is\_valid():

user = form.save()

user.refresh\_from\_db() # Load the profile instance

user.save()

raw\_password = form.cleaned\_data.get('password1')

user = authenticate(username=user.username, password=raw\_password)

login(request, user)

return redirect('home')

else:

form = SignUpForm()

return render(request, 'signup.html', {'form': form})

**urls.py**:

* python

from django.urls import path

from .views import signup\_view

urlpatterns = [

path('signup/', signup\_view, name='signup'),

]

**4. Creating Login and Logout Views**

Use Django’s built-in views for login and logout.

**urls.py**:

* python

from django.contrib.auth import views as auth\_views

urlpatterns = [

path('login/', auth\_views.LoginView.as\_view(template\_name='login.html'), name='login'),

path('logout/', auth\_views.LogoutView.as\_view(), name='logout'),

]

**login.html**:

* html

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Login</button>

</form>

**5. Password Management**

Django provides built-in views for password change and password reset.

**urls.py**:

* python

urlpatterns = [

...

path('password\_change/', auth\_views.PasswordChangeView.as\_view(template\_name='password\_change.html'), name='password\_change'),

path('password\_change/done/', auth\_views.PasswordChangeDoneView.as\_view(template\_name='password\_change\_done.html'), name='password\_change\_done'),

path('password\_reset/', auth\_views.PasswordResetView.as\_view(template\_name='password\_reset.html'), name='password\_reset'),

path('password\_reset/done/', auth\_views.PasswordResetDoneView.as\_view(template\_name='password\_reset\_done.html'), name='password\_reset\_done'),

path('reset/<uidb64>/<token>/', auth\_views.PasswordResetConfirmView.as\_view(template\_name='password\_reset\_confirm.html'), name='password\_reset\_confirm'),

path('reset/done/', auth\_views.PasswordResetCompleteView.as\_view(template\_name='password\_reset\_complete.html'), name='password\_reset\_complete'),

]

**password\_change.html**:

* html

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Change Password</button>

</form>