Day 78

Django Forms (Manual)

1. ✅ Creating an HTML Form
2. ✅ Handling POST requests in Django views
3. ✅ Accessing submitted form data
4. ✅ Displaying form results (optional feedback)

🧱 1. Create an HTML Form

Let’s create a **simple contact form** that asks for a user's name and message.

📄 `myapp/templates/myapp/contact.html`:

```
{% extends 'myapp/base.html' %}
{% load static %}

{% block content %}
  <h2>Contact Form</h2>

  <form method="POST" action="">
    {% csrf_token %}

    <label for="name">Name:</label><br>
    <input type="text" name="name" id="name"><br><br>

    <label for="message">Message:</label><br>
    <textarea name="message" id="message"></textarea><br><br>

    <button type="submit">Send</button>
  </form>
{% endblock %}
```

☑ Important:

- Use `method="POST"` for submitting data securely.
- `{% csrf_token %}` is required for Django's CSRF protection.

⚙ 2. Handle POST Request in Views

📄 `myapp/views.py`:

```
from django.shortcuts import render

def contact(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        message = request.POST.get('message')

        # You can now use these variables
        print("Name:", name)
        print("Message:", message)

        return render(request, 'myapp/contact.html', {
            'success': True,
            'name': name,
        })

    return render(request, 'myapp/contact.html')
```

📍 3. Display Submitted Data or Success Message

Update your template to show a message:

```
{% if success %}
  <p style="color:green;">Thanks, {{ name }}! Your message has been
sent.</p>
```

```
{% endif %}
```

🔗 4. Map the View to a URL

📄 `myapp/urls.py`:

```
from django.urls import path
from . import views

urlpatterns = [
    path('contact/', views.contact, name='contact'),
]
```

Visit: http://127.0.0.1:8000/contact/

✅ Summary

Task	Code/Action
Create form in HTML	Use `<form method="POST">` and `csrf_token`
Read data in view	`request.POST.get('fieldname')`
Detect method	`if request.method == 'POST'`
Show success message	Pass data back via `render()` context

Bonus 💡 : Validate Input (Basic)

You can add basic validation:

```
if not name or not message:
    return render(request, 'myapp/contact.html', {
        'error': 'Please fill out all fields.'
    })
```

And in the template:

```
{% if error %}
  <p style="color:red;">{{ error }}</p>
{% endif %}
```

Django Forms Using `ModelForm`

1. ✅ What is `ModelForm`?
2. ✅ Creating a `ModelForm` class
3. ✅ Rendering the form in templates
4. ✅ Validating and saving data
5. ✅ Custom error messages
6. ✅ Optional: Styling with `crispy-forms`

🧠 1. What is `ModelForm`?

- A **`ModelForm`** automatically creates a form based on a Django model.
- It saves time and reduces duplication between your `models.py` and `forms.py`.

☑️ 2. Create a `ModelForm`

Let's say you already have a model like this:

📄 `myapp/models.py`:

```
from django.db import models
```

```
class Contact(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    message = models.TextField()

    def __str__(self):
        return self.name
```

Now create a `forms.py`:

📄 `myapp/forms.py`:

```
from django import forms
from .models import Contact

class ContactForm(forms.ModelForm):
    class Meta:
        model = Contact
        fields = ['name', 'email', 'message']
```

✅ 3. Use the Form in Views

📄 `myapp/views.py`:

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

def contact(request):
    form = ContactForm()

    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            form.save()
            return render(request, 'myapp/contact.html', {
                'form': ContactForm(),  # fresh form
                'success': True
            })

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

✅ 4. Render Form in Template

📄 `myapp/templates/myapp/contact.html`:

```
{% extends 'myapp/base.html' %}
{% load static %}

{% block content %}
  <h2>Contact Form (ModelForm)</h2>

  {% if success %}
    <p style="color:green;">Your message was submitted
successfully!</p>
  {% endif %}

  <form method="POST" action="">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Submit</button>
  </form>
{% endblock %}
```

- `{{ form.as_p }}` renders each field wrapped in a <p> tag.
- You can also use `{{ form.as_table }}` or `{{ form.as_ul }}`.

✅ 5. Custom Validation & Error Messages

Add validations in `forms.py`:

```
class ContactForm(forms.ModelForm):
    class Meta:
        model = Contact
        fields = ['name', 'email', 'message']
        error_messages = {
            'name': {
                'required': 'Please enter your name.'
            },
            'email': {
                'required': 'We need your email to reply.',
                'invalid': 'Enter a valid email address.'
            },
        }

    def clean_name(self):
        name = self.cleaned_data['name']
        if any(char.isdigit() for char in name):
            raise forms.ValidationError("Name should not contain 
numbers.")
        return name
```

🎨 6. Optional: Use `django-crispy-forms` for better styling

✅ Step 1: Install crispy forms

```
pip install django-crispy-forms
```

pip install crispy-bootstrap5

Add to `settings.py`:

```
INSTALLED_APPS = [
    ...
    'crispy_forms',
   'crispy_bootstrap5',
]
```

CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5"

CRISPY_TEMPLATE_PACK = "bootstrap5"

✅ Step 2: Load in your template

```
{% extends 'myapp/base.html' %}
{% load static %}
{% load crispy_forms_tags %}

{% block content %}

<h2>Contact Form (ModelForm)</h2>

{% if success %}
<p style="color: green">Your message was submitted successfully!</p>
{% endif %}

<form method="POST" action="">
  {% csrf_token %} {{ form|crispy }}
  <button type="submit">Submit</button>
</form>

{% endblock %}
```

✓ Summary

Task	Tool/Code Example
Create ModelForm	`class ContactForm(forms.ModelForm):`
Render form in template	`{{ form.as_p }}` or `{{ form
Validate form	`if form.is_valid():`
Save form data	`form.save()`
Custom error message	Use `error_messages` in `Meta`
Field-specific validation	Define `clean_<fieldname>()` in form class
Form styling (optional)	Use `crispy-forms` and Bootstrap integration

DAY 78 TASKS

◇ SECTION 1: Manual Django Forms (Tasks 1–15)

1. **Create a basic HTML contact form with `name` and `message` fields.**
2. **Add `{% csrf_token %}` inside the form to protect against CSRF attacks.**
3. **Use the `POST` method in the form and confirm that `request.method == 'POST'`.**
4. **Extract the name and message from `request.POST` in the view.**
5. **Print the submitted data in the server console using `print()`.**
6. **Pass the submitted name back to the template using `render()` context.**
7. **Show a success message conditionally using `{% if success %}`.**
8. **Add basic form validation: check if either name or message is empty.**
9. **Display an error message if the form fields are not filled.**
10. **Include basic CSS styles in the form using static files.**
11. **Create a separate `contact.css` file and link it in your contact template.**
12. **Style the `<input>`, `<textarea>`, and `<button>` in your form using CSS.**
13. **Add a reset button that clears the input fields using HTML only.**
14. **Add a route `/contact/` in your `myapp/urls.py` mapped to the `contact()` view.**

15. Test form behavior by submitting various name/message combinations.

◇ SECTION 2: ModelForm Creation & Usage (Tasks 16–30)

16. Define a model named `Contact` with fields: `name`, `email`, and `message`.
17. Make migrations and migrate to create the database table.
18. Create a `forms.py` file in your app folder.
19. Create a `ContactForm` class using `forms.ModelForm`.
20. Use `Meta` class inside `ContactForm` to bind it with the `Contact` model.
21. Specify the fields to include: `['name', 'email', 'message']`.
22. Import and render the form in your `contact()` view.
23. Check for `POST` method and validate using `form.is_valid()`.
24. Save form data to the database using `form.save()`.
25. After saving, display a success message and reset the form in the template.
26. Render the form in the template using `{{ form.as_p }}`.
27. Replace `{{ form.as_p }}` with `{{ form.as_table }}` and observe changes.
28. Test the form by submitting multiple valid entries.
29. Verify data is stored by accessing `Contact` model via Django admin or shell.
30. Render only specific form fields manually (e.g., `{{ form.name }}`) in the template.

◇ SECTION 3: Form Validation & Custom Error Handling (Tasks 31–40)

31. Add `error_messages` inside the `Meta` class for name and email.
32. Provide a custom message when the email is empty or invalid.
33. Create a `clean_name()` method in your form and disallow numeric characters.
34. Test the custom validation by entering digits in the name field.
35. In the template, show individual field errors using `{{ form.name.errors }}`.
36. Use `{% if form.errors %}` to show a general form error message.
37. Create a `clean_message()` method that raises an error if message length < 10.

38. Test the validation by submitting a short message and verifying the error.
39. Add placeholder text in form fields using widgets in `ContactForm`.
40. Add max length attributes to inputs via form widgets and test input limits.

◇ SECTION 4: Crispy Forms & Styling (Tasks 41–50)

41. Install `django-crispy-forms` and `crispy-bootstrap5`.
42. Add `'crispy_forms'` and `'crispy_bootstrap5'` to `INSTALLED_APPS`.
43. Set `CRISPY_ALLOWED_TEMPLATE_PACKS` and `CRISPY_TEMPLATE_PACK` in `settings.py`.
44. Update your template to use `{% load crispy_forms_tags %}`.
45. Replace `{{ form.as_p }}` with `{{ form|crispy }}` in the form.
46. Use Bootstrap5 classes for form layout (e.g., form-control, btn btn-primary).
47. Add a Bootstrap alert for success message using: `<div class="alert alert-success">`.
48. Customize the `ContactForm` using `widgets` to add CSS classes to fields.
49. Create a responsive layout for the form using Bootstrap grid (row/col).
50. Use conditional rendering in the template to show success/error alerts with different colors.

Day 78 MINI PROJECTS

☑ 1. Contact Us Page (Manual Form + ModelForm)

Scenario: A company wants users to send feedback through a contact form.

Requirements:

- Create a `Contact` model: name, email, message
- Implement both manual and `ModelForm` versions
- Display success and error messages on submission
- Use `{% csrf_token %}` in manual form

- Add custom error message: "Name must not contain numbers"
- Use `crispy-forms` for the final styled form

✅ 2. Newsletter Signup Form

Scenario: Visitors subscribe with their name and email.

Requirements:

- `Subscriber` model: name, email
- ModelForm with custom `clean_email()` validation
- Manual form version also for practice
- Display success or "already subscribed" message
- Use `{{ form.as_table }}`
- Add basic CSS or Bootstrap

✅ 3. Job Application Form

Scenario: Applicants submit job requests.

Requirements:

- `JobApplication` model: name, email, position, cover_letter
- Validate name (no digits), email, and position selection
- Show validation errors using `form.errors`
- Store submissions in the database
- Add `crispy-forms` for UI enhancement

✅ 4. Event Registration Form

Scenario: Users register for an event.

Requirements:

- `Registration` model: full_name, email, session_type (choice field)
- Validate session_type (required)
- Display success message after save
- Template renders form with `{{ form.as_ul }}`
- Style the form with Bootstrap 5

✅ 5. Restaurant Table Booking

Scenario: Customers book a table online.

Requirements:

- `Reservation` model: name, email, date, time, guests
- Add `clean_name` to block names with numbers
- Manual form to handle POST and show feedback
- Custom error: “Please select valid date/time”
- Use `crispy-forms` for better UI

✅ 6. Student Feedback Form

Scenario: Students submit anonymous feedback.

Requirements:

- `Feedback` model: student_name (optional), course_name, feedback
- Manual + ModelForm implementations
- Show a thank-you message
- Custom error: “Feedback must be at least 20 characters.”
- Use `{{ form|crispy }}` for polished UI

✅ 7. Book Recommendation Suggestion Form

Scenario: Readers suggest books.

Requirements:

- `BookSuggestion` model: title, author, reason (min length validation)
- Reject suggestions missing fields
- Use `{% if error %}` in manual form
- Show saved entries in admin
- Use `widgets` to add placeholder text

✅ 8. Bug Report Submission

Scenario: Developers collect bug reports from users.

Requirements:

- `BugReport` model: name, email, issue_type (choice), description
- Validate description length and proper email
- Custom error messages using `Meta.error_messages`
- Display success alert with Bootstrap 5 styling
- Use crispy-form layout

✅ 9. Support Ticket Form

Scenario: Customers raise tickets for support.

Requirements:

- `SupportTicket`: name, email, issue_summary, message
- Manual form → POST data → show `success` or `error`
- Then replace with a `ModelForm`
- Add `clean_message()` to require at least 30 words

- Style with crispy-forms

✅ 10. Product Review Form

Scenario: Collect product reviews from users.

Requirements:

- `ProductReview`: name, product_name, rating (1–5), comment
- Custom error: rating must be between 1–5
- Show review submission result dynamically
- Use `{{ form.as_table }}`
- Form should work with and without crispy styling

✅ 11. Course Feedback

Scenario: Learners give feedback on courses.

Requirements:

- `CourseFeedback`: name, email, rating, comments
- Manual form first → handle via `request.POST`
- Add ModelForm with field-specific validation
- Add Bootstrap alert styling for feedback messages
- Use `{{ form|crispy }}` with Bootstrap 5

✅ 12. Volunteer Registration

Scenario: Users apply as volunteers for an NGO.

Requirements:

- `Volunteer`: name, email, area_of_interest

- Use `forms.Select()` for area choices
- Validate form manually and via ModelForm
- Add thank-you message on submission
- Use crispy-forms for layout

✅ 13. Internship Application Form

Scenario: Interns apply with resume details.

Requirements:

- `InternApplication`: name, email, field, motivation (min 50 chars)
- Validate inputs using `clean_field()`
- Display messages like “Invalid field” or “Thank you!”
- Render using `form.as_ul()`
- Add field placeholders using widgets

✅ 14. Suggest a Feature Form

Scenario: SaaS users suggest new features.

Requirements:

- `FeatureRequest`: name, email, feature_title, details
- Manual form: handle POST, validate inputs
- ModelForm: save to DB
- Display flash message after saving
- Enhance with crispy-forms

✅ 15. Complaint Form for a Local Service

Scenario: Citizens file complaints to municipality.

Requirements:

- `Complaint`: name, email, subject, description
- Add custom message: "Description cannot be empty"
- Use `{% if form.errors %}` to display all errors
- Render form using `form|crispy`
- Save complaint to DB

☑ 16. Book Lending Request

Scenario: Users request a book from library.

Requirements:

- `BookRequest`: name, book_title, author, request_reason
- Add a textarea for reason with validation
- Manual + ModelForm versions
- Use Django messages or success variable
- Test rendering using `form.as_p`

☑ 17. Appointment Request Form

Scenario: Patients request doctor appointments.

Requirements:

- `Appointment`: name, email, date, reason
- Validate email format and date is not in past
- Use ModelForm + `clean_date()`
- Show "Appointment requested!" on success
- Style with crispy-bootstrap5

☑ 18. Blog Post Submission (Guest Author)

Scenario: Guest writers submit blog ideas.

Requirements:

- `BlogPost`: author_name, email, title, content
- Validate title uniqueness
- Handle submission success and error in template
- Use custom widgets for styling inputs
- Form rendered with `form.as_p` and `crispy` toggle

☑ 19. Report a Problem Page

Scenario: General issue reporting form on website.

Requirements:

- `ProblemReport`: user_name, email, url, issue_description
- Validate that URL is properly formatted
- Use `{% if form.errors %}` to show error list
- Use both manual and model form implementations
- Enable crispy-forms for styled rendering

☑ 20. Hotel Room Booking Form

Scenario: Visitors book rooms via a form.

Requirements:

- `RoomBooking`: name, email, room_type (choices), check_in, check_out
- Custom validation: check-out date must be after check-in
- Display "Booking successful" with guest name
- Use widgets to add date pickers

- Add Bootstrap 5 classes via crispy-forms