Python

Full stack Skills Bootcamp

Introduction to Django Forms

- Goal

Learn how to create and edit blog posts using Django forms.

- Customize the user interface beyond Django's admin.
- Use forms for better control over data entry and display.

- Note: We'll use ModelForm to link the form to our Post model

Creating Forms with ModelForm

What is a ModelForm?

- A form that links directly to a Django model
- Automatically saves form data to the associated model.

```python
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text')
```

Using the Form in a View

Steps:

- Create a link in base.html to the form page
- Define a URL for creating a new post.

```
# urls.py
path('post/new/', views.post_new, name='post_new')
```

- Define a View for creating a new post :

```
# views.py
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

Linking to the Form in base.html

Adding a Link and Icon :

- Add this in head section of your HTML:

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css">

```html
<a href="{% url 'post_new' %}" class="top-menu">
    <i class="bi bi-file-earmark-plus"></i>
</a>
```

Defining the Form Template

- Create a new Html file to define the form for separation of concern.

Steps:

- Display the form with {{ form.as_p }}
- Wrap in <form method="POST">...</form>
- Include a Save button
- Add {% csrf_token %} for security.

```html
<form method="POST" class="post-form">{% csrf_token %}
    {{ form.as_p }}
    <button type="submit" class="save btn btn-secondary">Save</button>
</form>
```

Handling Form Submission in Views

Form Submission Process:

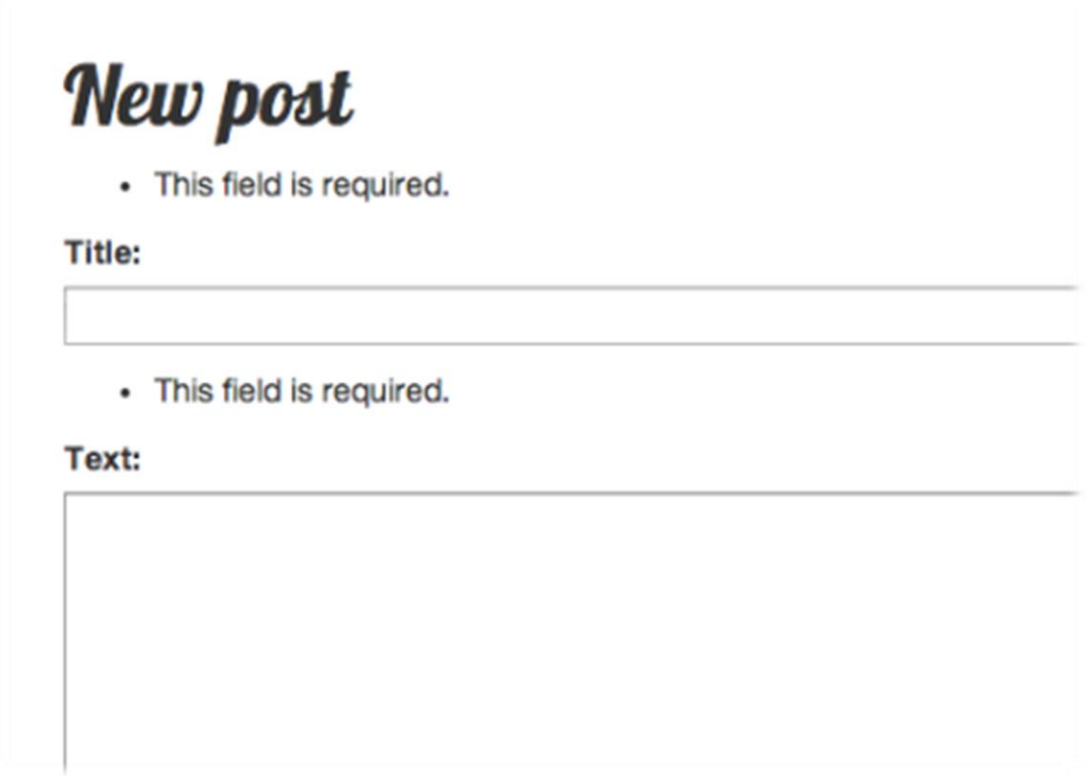
- POST method processes form data in request.POST
- Check form validity with form.is_valid()
- Save and redirect to post_detail.

```python
def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

Form Validation

Built-in Validation:

- Required fields like title and text are checked automatically.
- If fields are missing, form will not submit.



Editing Existing Data

- Reuse form template to edit existing posts.
- Define post_edit URL and view for editing existing data.

```python
# urls.py
path('post/<int:pk>/edit/', views.post_edit, name='post_edit')

# views.py
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

Summary

Key Takeaways:

- Use Django forms to customize data entry
- ModelForm connects forms to Django models
- Secure forms with CSRF tokens
- Use conditional logic to handle new and existing posts in views