SDEV 2401

Rapid Backend App Development

Forms Sanitization Validation - 2

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

Agenda

On the right is what we will cover today.

- Model Forms in Django
- Creating a Book Submission Model Form
- Rendering the Book Submission Form
- Handling Form Submission and Saving Data
- Custom Field and Cross-Field Validation
- Displaying Non-Field Errors
- Challenge Exercise
- Key Takeaways

Model Forms in Django

- Model forms connect Django forms directly to database models.
- They simplify creating, updating, and validating model instances.
- Today, we'll use a **Book Submission** example.

Creating a Book Submission Model Form (the model)

- Define a model form for users to submit new books.

```
from django.models import Model

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    isbn = models.CharField(max_length=13, unique=True)
    summary = models.TextField()

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

Creating a Book Submission Model Form

- Fields are generated from the model, including built-in validation.

```
from django import forms
from .models import Book

class BookSubmissionForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = ['title', 'author', 'isbn', 'summary']
```

- you can see here that we're using the Meta class to specify which model to use and which fields to include in the form.
- The fields will automatically have validation based on the model field types and constraints.

Rendering the Book Submission Form

- Create a template to render the form. This is review from last class.

```
<form method="post">
    {% csrf_token %}
    {% for field in form %}
        <div>
            <label>{{ field.label }}</label>
            {{ field }}
            {% if field.errors %}
                <div class="error">{{ field.errors }}</div>
            {% endif %}
        </div>
    {% endfor %}
    <button type="submit">Submit Book</button>
</form>
```

Handling Form Submission and Saving Data

- you can see here that when the form is valid, we call `form.save()` to create a new Book instance in the database.
- you can access the created instance via `form.instance` if needed, and here we're passing the instance to the template with the `"book"` context variable to show a success message with the book title.

```
def submit_book(request):
    if request.method == "POST":
        form = BookSubmissionForm(request.POST)
        if form.is_valid():
            # save to database
            form.save()
            # get the instance
            book = form.instance
            return render(request, "submit_book.html", {"form": BookSubmissionForm(), "book": book, "success": True})
    else:
        form = BookSubmissionForm()
    return render(request, "submit_book.html", {"form": form})
```

Custom Field and Cross-Field Validation

- `clean_isbn` validates the ISBN field specifically.
- `clean` method performs cross-field validation (e.g., summary length).

```
class BookSubmissionForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = ['title', 'author', 'isbn', 'summary']

    def clean_isbn(self):
        isbn = self.cleaned_data.get('isbn')
        if len(isbn) != 13 or not isbn.isdigit():
            raise forms.ValidationError("ISBN must be a 13-digit number.")
        return isbn

    def clean(self):
        cleaned_data = super().clean()
        summary = cleaned_data.get('summary', '')
        if len(summary) < 50:
            self.add_error('summary', "Summary must be at least 50 characters.")
        return cleaned_data
```

Displaying Non-Field Errors

Non field errors will be shown whenever you use the `clean()` method to raise validation errors that are not specific to a single field.

Tod display them in the template, you can use the following code snippet:

```
{% if form.non_field_errors %}
    <div class="error">
        {% for error in form.non_field_errors %}
            <p>{{ error }}</p>
        {% endfor %}
    </div>
{% endif %}
```

Challenge Exercise

- Add a `genre` field to the `Book` model and form.
- Validate that the genre is not "Unknown".
- Bonus: Add a `publication_year` field and ensure it is not in the future.

Key Takeaways

- Model forms reduce boilerplate and keep forms in sync with models.
- Use field-level and cross-field validation for robust data entry.
- Always display both field and non-field errors to users.
- Practice: Try building forms for other models (e.g., Movie, Event).

Example

Let’s go do an example together