SDEV 2401

Rapid Backend App Development

Forms Sanitization Validation - 1

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.

- Introduction to Forms, Validation, and Sanitization
- Creating a Job Application Form
- Rendering the Form in a Template
- Handling Form Submission in the View
- Custom Validation
- Sanitizing and Using Cleaned Data
- Challenge Exercise
- Key Takeaways

Introduction to Forms, Validation, and Sanitization

- Web applications often need to collect user input.
- Validating and sanitizing this input is essential for:
  - Security (preventing SQL injection, XSS, etc.)
  - Data integrity
  - User trust
- Today, we'll explore these concepts using a **Job Application** form as our example in slides.

Creating a Job Application Form

- Let’s define a form for users to apply for a job.
- Fields: `full_name`, `email`, `resume_link`, `cover_letter`.

```
from django import forms

class JobApplicationForm(forms.Form):
    full_name = forms.CharField(max_length=100, required=True)
    email = forms.EmailField(required=True)
    resume_link = forms.URLField(required=True)
    cover_letter = forms.CharField(widget=forms.Textarea, required=True)
```

- Each field type provides built-in validation and sanitization.
- Django automatically handles common validation (e.g., email format, URL format), this is a huge help in preventing bad data from entering your application.

Rendering the Form in a Template

- Use Django's template system to render the form.
- Example: `job_application.html`

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Apply</button>
</form>
```

- `csrf_token` is required for security.
- `form.as_p` renders fields as paragraphs.
- In our example we'll take a look at how to render a form with more customization of field rendering.

Handling Form Submission in the View

- Process both GET and POST requests on a single view function.

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

def apply_for_job(request):
    if request.method == "POST":
        form = JobApplicationForm(request.POST)
        if form.is_valid():
            # Process the cleaned data
            # (e.g., send email, save to database)
            return render(request, "job_application.html", {"form": JobApplicationForm(), "success": True})
    else:
        # GET request
        form = JobApplicationForm()
    return render(request, "job_application.html", {"form": form})
```

- On success, show a confirmation message, and reset the form. We'll take a look at this in our example as well.

Custom Validation

- Add custom validation for the `cover_letter` field (e.g., minimum length).

```
class JobApplicationForm(forms.Form):
    # ... fields ...
    def clean_cover_letter(self):
        letter = self.cleaned_data.get('cover_letter')
        if len(letter) < 50:
            raise forms.ValidationError("Cover letter must be at least 50 characters.")
        return letter
```

- Use `clean_<fieldname>` for field-specific validation.
  - with this you'll either raise a `ValidationError` or return the cleaned data.
- Use `clean()` for cross-field validation.
  - We'll take a look at this more in depth with model forms (later on)

Sanitizing and Using Cleaned Data

- After `form.is_valid()`, use `form.cleaned_data` for safe, validated input.
- Example: Save to database or send an email.

```
form = JobApplicationForm(request.POST)
if form.is_valid():
    data = form.cleaned_data # use this
    # Save data['full_name'], data['email'], etc.
```

- Never trust `request.POST` directly—always use cleaned data, normally called `form.cleaned_data`.
  - This prevents security vulnerabilities.
  - Ensures data integrity.
- This is one of the most important reasons to use a web framework like django, as they have developed these security features over many years of web development.

Challenge Exercise

- Create a **Feedback** form for your site.
  - Fields: `username`, `email`, `feedback_text`
  - Validate that `feedback_text` is not empty and at least 20 characters.
  - On submission, display a thank you message and email the feedback to the admin.

Key Takeaways

- Always validate and sanitize user input.
- Use Django forms for both rendering and validation.
- Custom validation helps enforce business rules.
- Cleaned data is safe to use in your application logic.

Example

Let’s go do an example together