Forms

Web Applications and Services

Spring Term

Naercio Magaia

UNIVERSITY OF SUSSEX

Contents

UNIVERSITY
OF SUSSEX

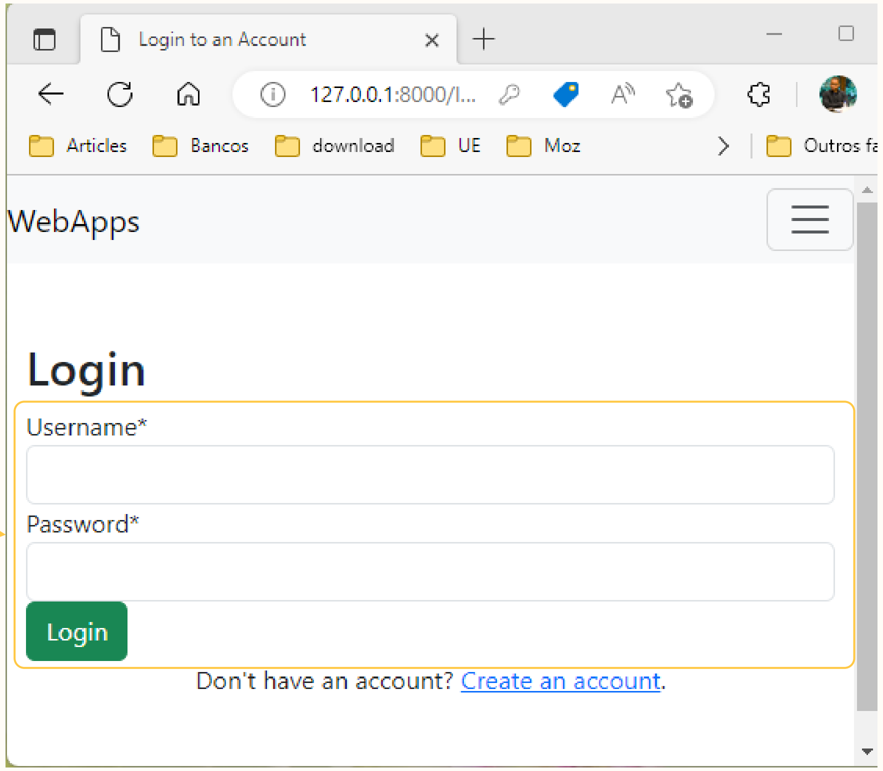
HTML forms

- It is a collection of elements inside `<form>...</form>` that
  - allow a visitor to enter text, select options, manipulate objects or controls, etc., and
  - then send that information back to the server
- Examples of form interface elements are
  - text input or checkboxes, which are **simple** and built into HTML itself
  - popping up a date picker or moving a slider or manipulating controls, which are more **complex** and use JavaScript and CSS besides HTML form `<input>` elements to achieve these effects

HTML forms

- A form must specify two things
  - where? i.e., the URL to which the data corresponding to the user's input should be returned
  - how? i.e., the HTTP method the data should be returned by
- The login form contains several <input> elements
  - one of `type="text"` for the username
  - one of type=`"password"` for the password, and
  - one of `type="submit"` for the "Login"

- Login page of the Comment Store App



Login to an Account

127.0.0.1:8000/l...

Articles Bancos download UE Moz Outros fa

WebApps

Login

Username*

Password*

Login

Don't have an account? Create an account.

GET and POST

- These methods are the only ones used when dealing with forms
  - For example, Django’s login form is returned using the POST method
- POST
  - Should be used in any request that could **change the state of the system**
  - Normally, the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response
- GET
  - should only be used for requests that **do not affect the state of the system**
  - Normally, the browser bundles the submitted data into a string and uses this to compose a URL
    - The URL contains the address where the data must be sent, as well as the data keys and values

GET is unsuitable for a password form

UNIVERSITY OF SUSSEX

Forms in Django

- A Django Form class describes a form and determines how it works and appears
- Its fields map to HTML form `<input>` elements
  - A *ModelForm* maps a model class's fields to HTML form `<input>` elements via a Form
- A form's fields are themselves classes
  - they manage form data and perform validation when a form is submitted (e.g., a *DateField* and a *FileField*)
- A form field is represented to a user in the browser as an HTML "widget"
  - Each field type has an appropriate default Widget class, but these can be overridden as required

UNIVERSITY OF SUSSEX

Forms in Django

- Steps to render an object in Django
  1. access it in the view (e.g., getting it from the database)
  2. pass it to the template context
  3. expand it to HTML markup using template variables
- When dealing with a form, it is common to instantiate it in the view, leave it empty or prepopulate it with
  - data received from a previous HTML form submission
    - it enables users to either read a website or to send information back to it too
  - data collated from other sources

UNIVERSITY OF SUSSEX

Building a form

- What does the HTML form below do?

the browser returns the form data to the URL and HTTP method specified

It will display a label and a button

```
<form action="/your-name/" method="post">
    <label for="your_name">Your name: </label>
    <input id="your_name" type="text" name="your_name" value="{{ current_name }}">
    <input type="submit" value="OK">
</form>
```

the variable will be used to pre-fill the *your_name* field.

- Once the form is submitted, the POST request will contain the form data.

UNIVERSITY OF SUSSEX

Building a form

- There should be a view matching the `/your-name/` URL
  - It will find the appropriate key/value pairs in the request, and then process them.
- Usually, a form contains dozens or hundreds of fields, and some might need to be prepopulated
- The browser might need to perform validations, e.g., before submitting the form
  - It may be desirable to use much more complex fields, e.g., allowing the user to pick dates from a calendar, etc.

Building a form in Django

- The starting point is to edit the *forms.py* file

```
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

It defines a Form class with the *your_name* field

The field's maximum allowable length is defined by *max_length*

- A Form instance has an `is_valid()` method
  - runs validation routines for all its fields
    - return True, if all fields contain valid data
    - place the form's data in its *cleaned_data* attribute.

UNIVERSITY OF SUSSEX

Building a form in Django

- When rendered it will look like:

```
<label for="your_name">Your name: </label>
<input id="your_name" type="text" name="your_name" maxlength="100" required>
```

- Generally, it does not include the `<form>` tags, or a submit button.
  - It is the developer's responsibility to provide them in the template.

UNIVERSITY OF SUSSEX

The view

- To handle the form, it is necessary to instantiate it in the view for the URL where it should be published:

Importing the *NameForm* class of the *.forms* module

If POST request, process the form data

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

from .forms import NameForm

def get_name(request):
    if request.method == 'POST':
        form = NameForm(request.POST)
        if form.is_valid():
            # process the data in form.cleaned_data as required
            return HttpResponseRedirect('/home/')
    else:
        form = NameForm()

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

create a form instance and populate it

check if the created form is valid

create a blank form for any other method

The template

The context

The template

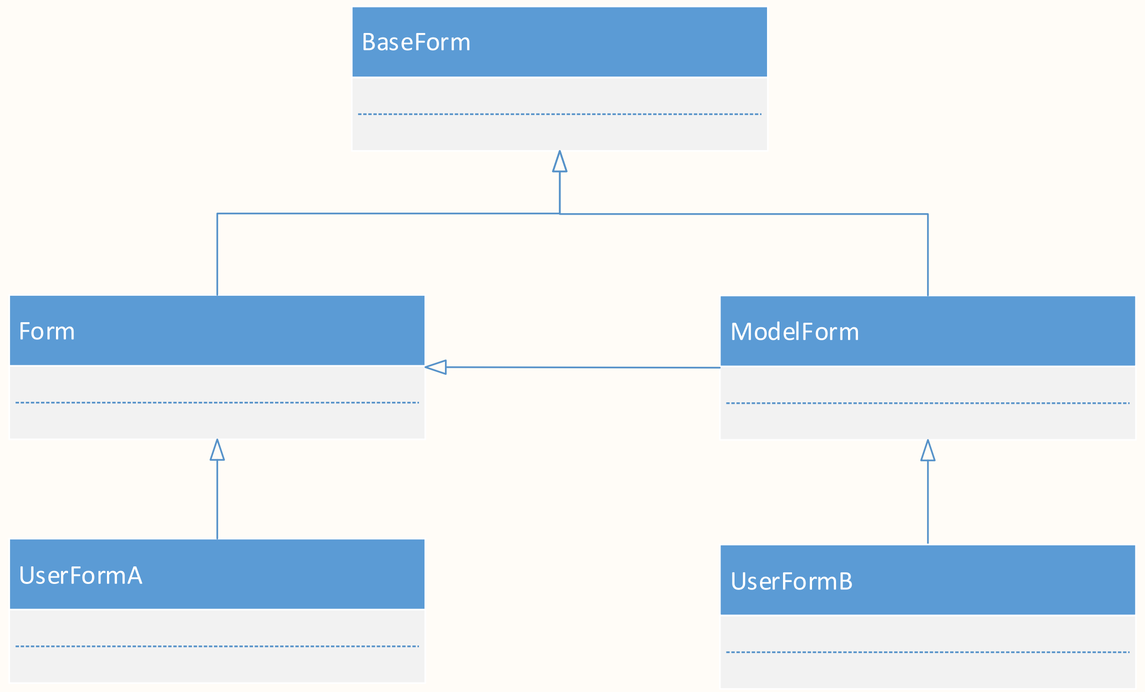
- It isn't necessary to do much in the *name.html* template

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

All fields and their attributes are unpacked into HTML markup by Django's template language.

- There is now a working web form
  - described by a Django Form
  - processed by a view, and
  - rendered as an HTML <form>

More about Django forms



- An *unbound form* has no data associated with it
  - It will be empty or will contain default values when rendered to the user
- A *bound form* has submitted data, and hence it is possible to check if the data is valid
  - If an invalid bound form is rendered, it can include inline error messages telling the user what data to correct

More about Django forms

- Consider the following form

```
from django import forms

class CommentForm(forms.Form):
    name = forms.CharField(max_length=100)
    visit_date = forms.DateField()
    comment_str = forms.CharField(widget=forms.Textarea)
```

Form fields

Textarea widget, which is larger, is used

- Each form field has a corresponding Widget class
  - a `CharField` will have a `TextInput` that produces an `<input type="text">`

UNIVERSITY OF SUSSEX

More about Django forms

- Validated form data can be accessed in the `form.cleaned_data` dictionary
- In the comment form example, the form data could be processed in the view as follows

```python
from django.http import HttpResponseRedirect

if form.is_valid():
    name = form.cleaned_data['name']
    visitdate = form.cleaned_data['visit_date']
    commentstr = form.cleaned_data['comment_str']
    store.insertcomment(name, visitdate, commentstr)

return HttpResponseRedirect('/thanks/')
```

UNIVERSITY OF SUSSEX

Working with form templates

- To get a form into a template it is to place the form instance into the template context
  - if the form is called `form` in the context, `{{ form }}` will render its `<label>` and `<input>` elements appropriately.
- It is possible to control the rendering of a form to generate an HTML output via a template
  - Create an appropriate template file
  - Set a custom `FORM_RENDERER` to use that `form_template_name` site-wide

Working with form templates

- It is also possible to customize per-form by
  - overriding the form's `template_name` attribute to render the form using the custom template, or
  - passing the template name directly to `Form.render()`
- Let us now render `{{ form }}` as the output of the `form_snippet.html` template

```
# In a template:
{{ form }}
```

```
# In the form_snippet.html:
{% for field in form %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
    </div>
{% endfor %}
```

Working with form templates

- Then

1. Configure the FORM_RENDERER setting in the *settings.py* file, or

```python
from django.forms.renderers import TemplatesSetting

class CustomFormRenderer(TemplatesSetting):
    form_template_name = "form_snippet.html"

FORM_RENDERER = "project.settings.CustomFormRenderer"
```

2. for a single form, or

```python
class MyForm(forms.Form):
    template_name = "form_snippet.html"
    ...
```

Working with form templates

3. for a single render of a form instance, pass in the template name to the `Form.render()`:

```
def index(request):
    form = MyForm()
    rendered_form = form.render("form_snippet.html")
    context = {'form': rendered_form}
    return render(request, 'index.html', context)
```

- Form rendering options for the `<label>/<input>` pairs
  - `{{ form.as_div }}` renders them wrapped in `<div>` tags.
  - `{{ form.as_table }}` renders them as table cells wrapped in `<tr>` tags.
  - `{{ form.as_p }}` renders them wrapped in `<p>` tags.
  - `{{ form.as_ul }}` renders them wrapped in `<li>` tags.

UNIVERSITY OF SUSSEX

Working with form templates

- The `CommentForm` output for the `{{ form.as_p }}` option is

```
<p><label for="id_name">Name:</label>
    <input id="id_name" type="text" name="name" maxlength="100" required></p>
<p><label for="id_visit_date">Visit date:</label>
    <input type="visit_date" name="visit_date" id="id_visit_date" required></p>
<p><label for="id_comment_str">Comment:</label>
    <textarea name="comment_str" id="id_comment_str" required></textarea></p>
```

UNIVERSITY OF SUSSEX

Working with form templates

- Instead of letting Django unpack the form’s fields, it can be done manually enabling to reorder them
  - Each field is available as an attribute of the form using `{{ form.name_of_field }}`

```
{{ form.non_field_errors }}
<div class="fieldWrapper">
    {{ form.name.errors }}
    <label for="{{ form.name.id_for_label }}">Your name:</label>
    {{ form.name }}
</div>
<div class="fieldWrapper">
    {{ form.visit_date.errors }}
    <label for="{{ form.visit_date.id_for_label }}">Your visit date:</label>
    {{ form.visit_date }}
</div>
```

Rendering form error messages

Working with form templates

- Using `{{ form.name_of_field.errors }}` displays a list of form errors, rendered as an unordered list
- The CSS class of `errorlist` allows to style its appearance
- To further customize the display of errors, loop over them

```
{% if form.subject.errors %}
    <ol>
    {% for error in form.subject.errors %}
        <li><strong>{{ error|escape }}</strong></li>
    {% endfor %}
    </ol>
{% endif %}
```

UNIVERSITY OF SUSSEX

Working with form templates

- Looping over each field in turn using `{% for %}` loop may reduce duplicate code

```
{% for field in form %}
    <div class="fieldWrapper">
        {{ field.errors }}
        {{ field.label_tag }} {{ field }}
        {% if field.help_text %}
        <p class="help">{{ field.help_text|safe }}</p>
        {% endif %}
    </div>
{% endfor %}
```

Outputs a `<ul class="errorlist">` containing any validation errors corresponding to this field.

The field's label wrapped in the appropriate HTML `<label>` tag.

Any help text that has been associated with the field.

- Useful attributes on `{{ field }}` include

UNIVERSITY OF SUSSEX

Next Lecture …

- ✓ Introduction
- ✓ HTTP, Caching, and CDNs
- ✓ Views
- ✓ Templates
- ✓ Forms
- ➢ **Models**
- Security
- Transactions
- Remote Procedure Call
- Web Services
- Time
- Elections and Group Communication
- Coordination and Agreement

UNIVERSITY OF SUSSEX