deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Technologies and Web Programming

2019/2020

Django Framework

Django Framework

Django Forms

The Form Class

- The Form class describes a form and determines how it works and appears in the browser.
- The fields from Form class map to HTML form as <input> elements.
  - They are themselves classes; they manage form data and perform validation when a form is submitted.
  - They are represented in the browser as an HTML “widget”. Each field type has an appropriate default Widget class, but these can be overridden as required.

Form Class Instantiation

- In a Form class instantiation, we can opt to leave it empty or pre-populate it, for example with:
  - data from a saved model instance (as in the case of admin forms for editing);
  - data that we have collated from other sources;
  - data received from a previous HTML form submission.
- The last case is very useful, because it allows users to re-send information without to fill it again.

Building a Form

- To build a form, normally we write code as below in HTML.

bookquery.html

```html
<form action="." method="post">
    <label for="query">Search: </label>
    <input id="query" type="text" name="query">
    <input type="submit" value="Search">
</form>
```

- In fact, a form is generally much more complex, including several fields, fields types, restrictions and validation rules.
- So, it would be nice to get it easy.

Creating a Django Form (i)

- We start by creating a Form class with the needed fields, in module “forms.py”.

forms.py

```
from django import forms

# Create your forms here.
class BookQueryForm(forms.Form):
    query = forms.CharField(label='Search:', max_length=100)
```

- This is done like a data model in module “models.py”.
- In this case, the form will have a text input field with maximum length set to 100 and a user friendly label named “Search”.

Creating a Django Form (ii)

- Then, we create the template where the Form class will be represented.

bookquery.html

```
<form action="." method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Search">
</form>
```

- When rendered, the form will replace {{ form }} with the label and the input defined in the Form class.

Creating a Django Form (iii)

- The view will be like below.

views.py

```
4
5   from app.models import Author, Publisher, Book
6   from app.forms import BookQueryForm
7
8
9   def bookquery(request):
10      # if POST request, process form data
11      if request.method == 'POST':
12          # create form instance and pass data to it
13          form = BookQueryForm(request.POST)
14          if form.is_valid(): # is it valid?
15              query = form.cleaned_data['query']
16              books = Book.objects.filter(title__icontains=query)
17              return render(request, 'booklist.html', {'boks': books, 'query': query})
18      # if GET (or any other method), create blank form
19      else:
20          form = BookQueryForm()
21      return render(request, 'bookquery.html', {'form': form})
```

Access to form data, after its validation.

Django Form and its Fields

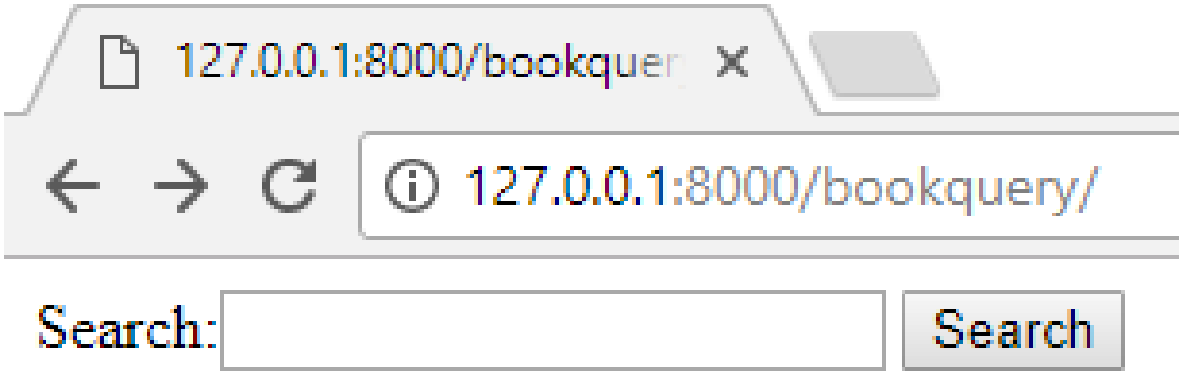
- The Data Field
  - Data submitted with a form, using Form Fields, can be validated through is_valid() function.
  - After validation, data can be accessed in form.cleaned_data dictionary.
  - The data in this dictionary is already converted into Python types, for immediate use.

Django Form and its Fields

- Examples of Data Fields and their representation in HTML.
  - BooleanField – as Check Box Input
  - CharField – as Text Input
  - IntegerField and FloatField – as Number Input
  - DateField, TimeField – as Text Input
  - ChoiceField – as Select
  - MultipleChoiceField – as Select Multiple
  - FileField – File Input
  - See more in: https://docs.djangoproject.com/en/2.2/ref/forms/fields/

Showing the Django Form

- The rendered form will be like below.



- This isn’t great, from aesthetic view point, but it runs properly and has automatic validation.

Control Django Form Rendering

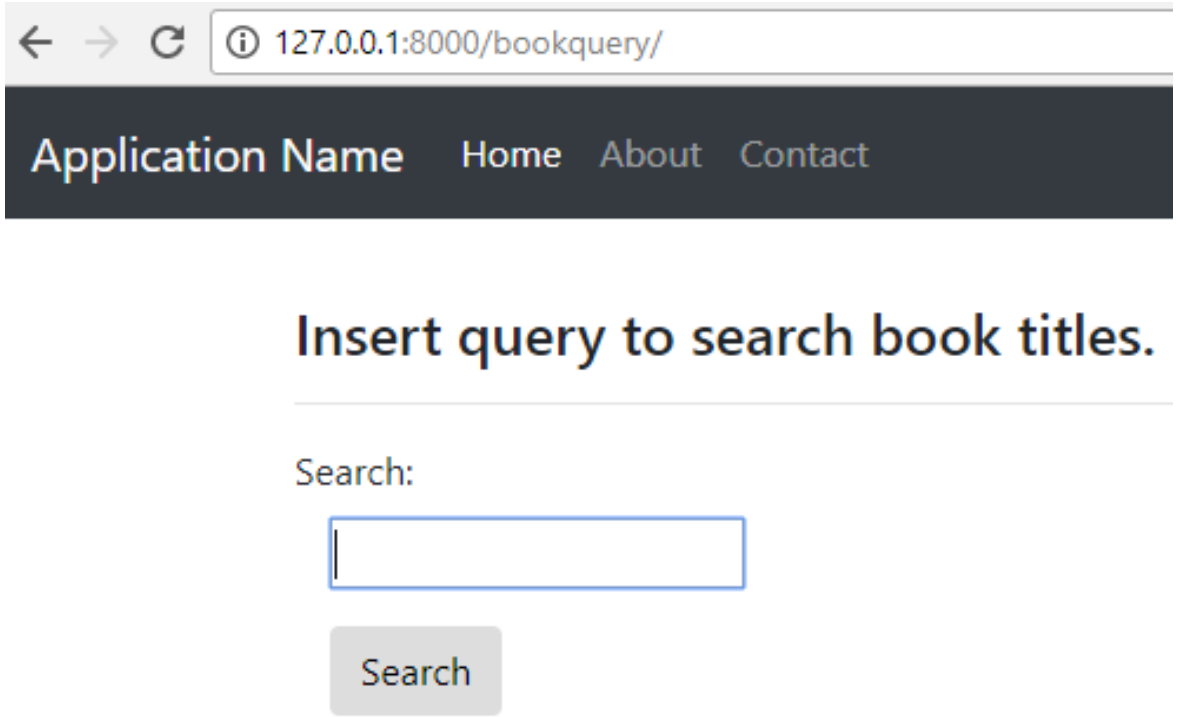
- It’s possible to render Form Fields individually.

bookquery.html

```
{% extends "layout.html" %}

{% block content %}

<h2>{{ title }}</h2>
<div class="row">
    <div class="col-md-8">
        <section id="insbookForm">
            <form action="." method="post" class="form-horizontal">
                {% csrf_token %}
                <h4>Insert query to search book titles.</h4>
                <hr />
                <div class="form-group">
                    {{ form.query.label_tag }}
                    <div class="col-md-10">
                        {{ form.query }}
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Search" class="btn btn-default" />
                    </div>
                </div>
            </form>
        </section>
    </div>
</div>

{% endblock %}
```

Showing Controlled Django Form

- In this case, the rendered form will be like below.



Django Framework

Django Authentication

Django Authentication

- Django comes with a user authentication system. It handles user accounts, groups, permissions and user sessions.
- It handles both <u>authentication</u> and <u>authorization</u>.
  - Authentication verifies if users are who they claim to be.
  - Authorization determines what authenticated users are allowed to do.
- It implements:
  - Users, Groups and Permissions
  - Forms and view tools for logging users, or restricting content.
  - A password hashing system.

Django Authentication

- To use Django authentication system, verify if the following modules are loaded at MIDDLEWARE_CLASSES and INSTALLED_APPS keys.

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
```

- Also, the database must be initialized. If not, run the command: *python manage.py migrate*.

Creating Login and Logout

- In the file “urls.py”, import the needed modules and define the needed urls:

urls.py

```
16
17  from django.contrib import admin
18  from django.contrib.auth import views as auth_views
19  from django.urls import path, include
20
21  from app import views
22
23  urlpatterns = [
24      path('login/', auth_views.LoginView.as_view(template_name='login.html'), name='login'),
25      path('logout', auth_views.LogoutView.as_view(next_page='/'), name='logout'),
26
```

Creating Login and Logout

- Use the file “login.html” to create the login template, and make the following modifications:

login.html

```
<form action="." method="post" class="form-horizontal">
    {% csrf_token %}
    <h4>Use a local account to log in.</h4>
    <hr />
    <div class="form-group">
      {{ form.username.label_tag }}
      <div class="col-md-10">
          {{ form.username }}
      </div>
    </div>
    <div class="form-group">
      {{ form.password.label_tag }}
      <div class="col-md-10">
          {{ form.password }}
      </div>
    </div>
```

Creating Login and Logout

- Use the file “loginpartial.html” to create an area where to show the login status. For that, modify “layout.html” file as follows:

layout.html

```
27      <div class="collapse navbar-collapse" id="navbarDefault">
28        <ul class="navbar-nav mr-auto">
29          <li class="nav-item active">
30            <a class="nav-link" href="{% url 'home' %}">Home</a>
31          </li>
32          <li class="nav-item">
33            <a class="nav-link" href="{% url 'about' %}">About</a>
34          </li>
35          <li class="nav-item">
36            <a class="nav-link" href="{% url 'contact' %}">Contact</a>
37          </li>
38        </ul>
39        {% include 'loginpartial.html' %}
40      </div>
41    </nav>
```

Authorization

- Authorization can be automatically managed by Django.
  - Through object “request.user”, it’s possible to verify if a given user is authenticated and have authorization to do operations.

views.py

```python
def authorins(request):
    if not request.user.is_authenticated or request.user.username != 'admin':
        return redirect('/login')
    # if POST request, process form data
    if request.method == 'POST':
        # create form instance and pass data to it
        form = AuthorInsForm(request.POST)
        if form.is_valid(): # is it valid?
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            a = Author(name=name, email=email)
            a.save()
            return HttpResponse("<h1>Author Inserted!!!</h1>")
    # if GET (or any other method), create blank form
    else:
        form = AuthorInsForm()
    return render(request, 'authorins.html', {'form': form})
```

Django Framework

Django Sessions

State

- HTTP protocol is a stateless protocol, which means that it doesn’t have any mechanism to save the connection state and as so it doesn’t allow sessions creation.
- To do this, some exterior mechanisms were developed in web clients and servers, which allow to save state data over multiple HTTP connections, producing artificial sessions.
- Mechanisms, like:
  - Cookies;
  - High level tools, using databases, to manage users, authentications and sessions.

Cookies

- A cookie is a little piece of information sent by a web server to its client, a browser, to save it while they are in communication.
- It's possible to save some kind of information in this cookie, like the user's username, for example.
- This cookies mechanism is in wide use by almost web sites, but it has some disadvantages:
  - Saving cookies in the browser is not compulsory, which doesn't allow to offer warranty of a good service;
  - They can't be used to save important information – they aren't secure;
  - The server can be inhibited, at some time, to access crucial information to continue the interaction with the client.

Django Sessions

- Django offers a high level mechanism for sessions establishment, which allows to save all kind of information in the server itself.
  - This information is saved in the database.
- To use this mechanism, verify the presence of the following lines in “settings.py” file.

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
```

Django Sessions (ii)

- Django manage automatically the sessions with its clients in a simple and clean way, through “request.session” object, which is a dictionary.

views.py

```python
def bookquery(request):
    # if POST request, process form data
    if request.method == 'POST':
        # create form instance and pass data to it
        form = BookQueryForm(request.POST)
        if form.is_valid(): # is it valid?
            query = form.cleaned_data['query']
            if 'searched' in request.session and request.session['searched'] == query:
                return HttpResponse('Query already made!!!')
            request.session['searched'] = query
            books = Book.objects.filter(title__icontains=query)
            return render(request, 'booklist.html', {'boks': books, 'query': query})
    # if GET (or any other method), create blank form
    else:
        form = BookQueryForm()
    return render(request, 'bookquery.html', {'form': form})
```