deti universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Technologies and Web Programming

Django Framework

Django Framework

Sending and Receiving Data

Forms

Receiving Data

- The “HttpRequest” type object “request” allows to access to a wide set of data, received by the web server
- This data can be directly accessed, using some methods and attributes, like:
  - request.path, request.get_host(), request.is_secure()
- Or can be accessed through the “request.META” dictionary, which contains all information present in the HTTP protocol header

Receiving Data

- Example of a view showing all the data present in the HTTP protocol header

views.py

```
from django.shortcuts import render, render_to_response
from django.http import HttpResponse

def info(request):
    values = request.META.items()
    html = []
    for k, v in values:
        html.append('<tr><td>%s</td><td>%s</td></tr>' % (k, v))
    return HttpResponse('<table>%s</table>' % '\n'.join(html))
```

Forms

- Forms are the HTML elements, by excellency, to send and receive data from the client to the server
- From the client side (browser), the Form can use the methods Get or Post to send the data it has
- From the server side, the called view can use the “request.GET” and “request.POST” dictionaries to access received data

Forms - example

- Form creation to search books by their titles in the previously created data model
- Defining the URL:

urls.py

```
from django.contrib import admin
from django.urls import path

from app import views

urlpatterns = [
    path('booksearch/', views.booksearch, name='booksearch'),
    path('insauthor/', views.authorins, name='authorins'),
```

Forms - example

- Defining the *view*:

views.py

```
5   from app.models import Author, Publisher, Book
6
7
8   def booksearch(request):
9       if 'query' in request.POST:
10          query = request.POST['query']
11          if query:
12              books = Book.objects.filter(title__icontains=query)
13              return render(request, 'booklist.html',{'boks':books, 'query':query})
14          else:
15              return render(request, 'booksearch.html', {'error':True})
16      else:
17          return render(request, 'booksearch.html', {'error':False})
```

Forms - example

- Defining the *template* for searching:

booksearch.html

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

{% block content %}

{% if error %}
    <p style="...">ERROR: Insert a query term.</p>
{% endif %}

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

{% endblock %}
```

Forms - exemplo

- Defining the template to show the results:

booklist.html

```
{% extends "layout.html" %}
{% block content %}
<p>Search by: <strong>{{ query }}</strong></p>
{% if boks %}
<p>Found {{ boks|length }} book{{ boks|pluralize }}.</p>
<ul>
    {% for bok in boks %}
        <li>{{ bok.title }}</li>
        <ul>
            <li>{{ bok.publisher }}</li>
            <li>{{ bok.date }}</li>
            <ul>
            {% for aut in bok.authors.all %}
                <li>{{ aut }}</li>
            {% endfor %}
            </ul>
        </ul>
    {% endfor %}
</ul>
{% else %}
    <p>Not found any result.</p>
{% endif %}
{% endblock %}
```

Django Framework

Form Classes and 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

```
1
2   <form action="." method="post">
3       {% csrf_token %}
4       {{ form }}
5       <input type="submit" value="Search">
6   </form>
7
```

- 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

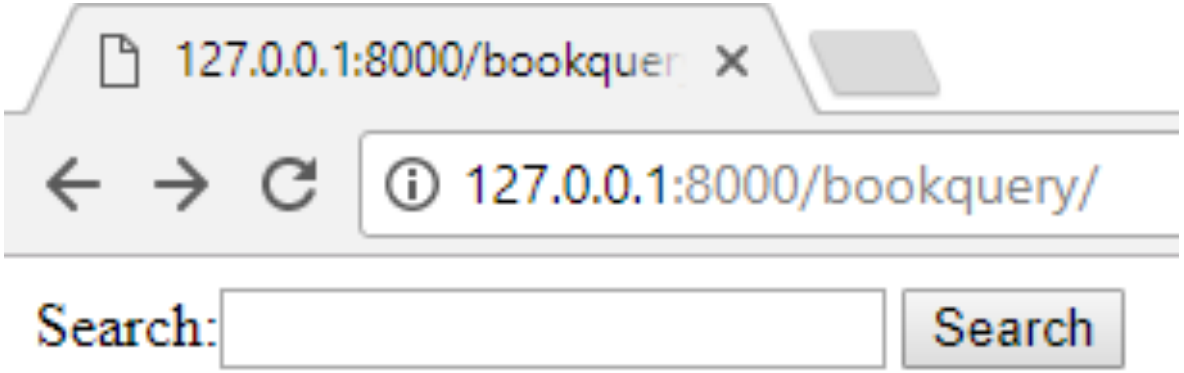
- The Data Field
  - Data submitted with a form, using Form Fields, can be validated through is_valid() function.
  - After validation, data can be accessed in form.cleaned_data dictionary.
  - The data in this dictionary is already converted into Python types, for immediate use.

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/?.?/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.