

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 x
from django.shortcuts import render, render_to_response
from django.http import HttpResponseRedirect

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 HttpResponseRedirect('<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 x
16
17 from django.contrib import admin
18 from django.urls import path
19
20 from app import views
21
22 urlpatterns = [
23     path('booksearch/', views.booksearch, name='booksearch'),
24     path('insauthor/', views.authorins, name='authorins'),
25 ]
```

Forms - example



- Defining the *view*:

```
views.py x
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', {'books': 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 x
1  {% extends "layout.html" %}
2
3  {% block content %}
4
5  {% if error %}
6      <p style="...">ERROR: Insert a query term.</p>
7  {% endif %}
8
9  <form action="." method="post">
10      {% csrf_token %}
11      <input type="text" name="query">
12      <input type="submit" value="Search">
13  </form>
14
15  {% endblock %}
16
```


Forms - exemplo



- Defining the template to show the results:

```
booklist.html x
1 {% extends "layout.html" %}
2 {% block content %}
3 <p>Search by: <strong>{{ query }}</strong></p>
4 {% if boks %}
5 <p>Found {{ boks|length }} book{{ boks|pluralize }}.</p>
6 <ul>
7     {% for bok in boks %}
8         <li>{{ bok.title }}</li>
9         <ul>
10             <li>{{ bok.publisher }}</li>
11             <li>{{ bok.date }}</li>
12             <ul>
13                 {% for aut in bok.authors.all %}
14                     <li>{{ aut }}</li>
15                 {% endfor %}
16             </ul>
17         </ul>
18     {% endfor %}
19 </ul>
20 {% else %}
21 <p>Not found any result.</p>
22 {% endif %}
23 {% 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 x
1
2 <form action="." method="post">
3     <label for="query">Search: </label>
4     <input id="query" type="text" name="query">
5     <input type="submit" value="Search">
6 </form>
7
```

- 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 x
1  from django import forms
2
3  # Create your forms here.
4  class BookQueryForm(forms.Form):
5      query = forms.CharField(label='Search:', max_length=100)
6
```

- 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.

```
1  <form action="." method="post">
2      {% csrf_token %}
3      {{ form }}
4      <input type="submit" value="Search">
5  </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 x
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', {'books': 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



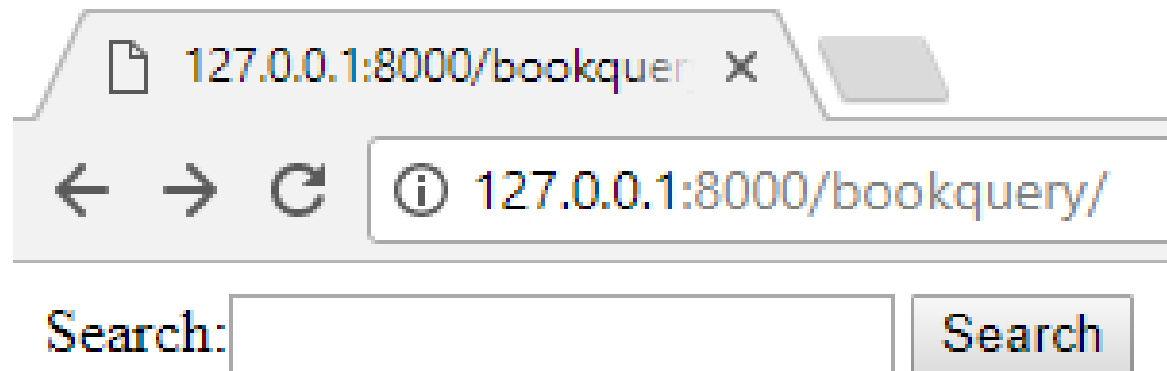
- Examples of Data Fields and their representation in HTML.
 - BooleanField – as Check Box Input
 - CharField – as Text Input
 - IntegerField and FloatField – as Number Input
 - DateField, TimeField – as Text Input
 - ChoiceField – as Select
 - MultipleChoiceField – as Select Multiple
 - FileField – File Input
 - See more in:

<https://docs.djangoproject.com/en/3.1/ref/forms/fields/>

Showing the Django Form



- The rendered form will be like below.



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/bookquer` with a close button. Below the address bar, there are navigation buttons (back, forward, refresh) and an information icon. The main content area displays a search form with the label "Search:" followed by a text input field and a "Search" button.

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

Control Django Form Rendering

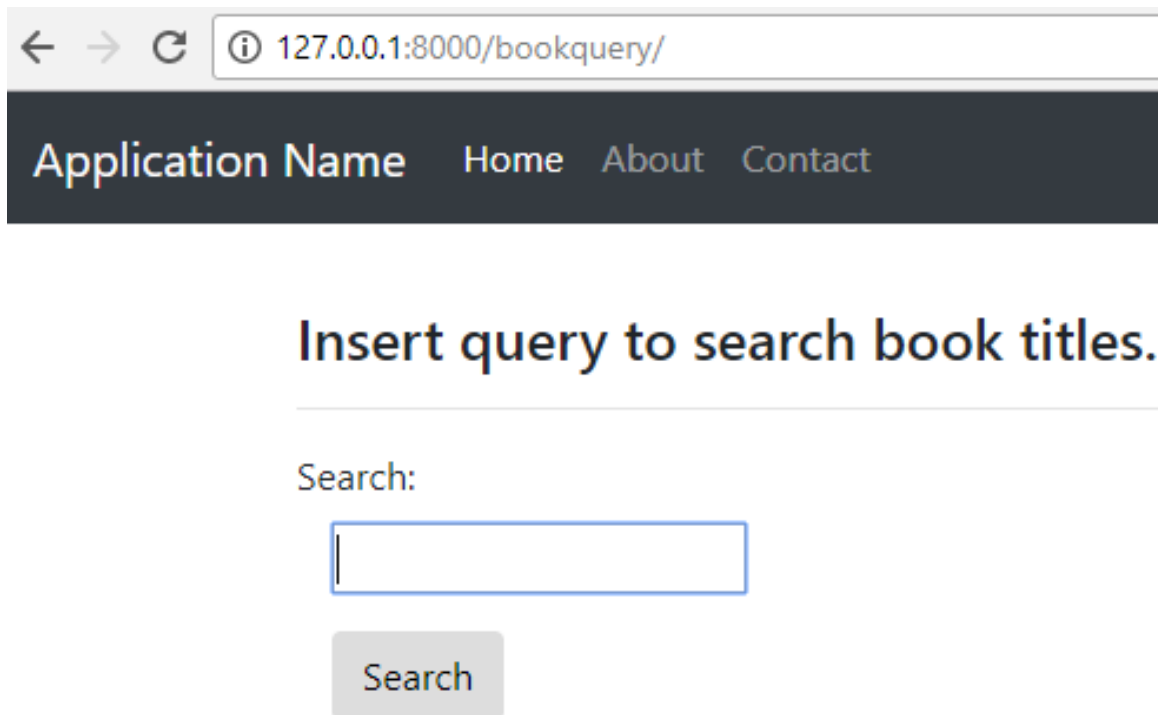


- It's possible to render Form Fields individually.

```
bookquery.html x
2      {% extends "layout.html" %}
3
4      {% block content %}
5
6          <h2>{{ title }}</h2>
7
8          <div class="row">
9
10             <div class="col-md-8">
11
12                 <section id="insbookForm">
13
14                     <form action="." method="post" class="form-horizontal">
15
16                         {% csrf_token %}
17                         <h4>Insert query to search book titles.</h4>
18                         <hr />
19                         <div class="form-group">
20                             {{ form.query.label_tag }}
21                             <div class="col-md-10">
22                                 {{ form.query }}
23                             </div>
24                         </div>
25                         <div class="form-group">
26                             <div class="col-md-offset-2 col-md-10">
27                                 <input type="submit" value="Search" class="btn btn-default" />
28                             </div>
29                         </div>
30                     </form>
31                 </section>
32             </div>
33         </div>
34     {% endblock %}
```

Showing Controlled Django Form

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



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/bookquery/". The page has a dark navigation bar with links for "Application Name", "Home", "About", and "Contact". Below the navigation bar, the text "Insert query to search book titles." is displayed. Underneath this text is a "Search:" label, a text input field, and a "Search" button.

← → ↻ ⓘ 127.0.0.1:8000/bookquery/

Application Name Home About Contact

Insert query to search book titles.

Search:

Search