Django Forms

Web Programming 2

Dr Aaron Mininger

14 Mar 2024

Models

Note, you can run django in a python shell for testing and doing things with the database

```
(venv)$ python manage.py shell
```

Model Actions

Common actions with models:

- View
- Add
- Change
- Delete

(Similar to CRUD: Create, Read, Update, Delete)

Model Actions

We can do all these things programmatically using objects

Viewing an object

- Remember that the object's attributes are the values of the DB fields

```
post = BlogPost.objects.get(id=5)

print(post.id)    # 5
print(post.title) # title
```

Model Actions

We can do all these things programmatically using objects

Adding an object

- We can create a new object, then call the `save()` method

```
new_post = BlogPost(title='Pili', content='A very spicy chili oil')
new_post.save() # This adds to the database
```

Model Actions

We can do all these things programmatically using objects

Changing an object

- We can change an object's attribute, then call the `save()` method

```
pili = BlogPost.objects.get(title='Pili')
pili.content = 'A very spicy chili oil made with habanero peppers')

pili.save() # This will save to the database
```

Model Actions

We can do all these things programmatically using objects

Deleting an object

- Model objects also have a delete() method

```
pili = BlogPost.objects.get(title='Pili')

pili.delete()
```

Url Structure

Usually, we want to support these actions through pages on the website

Example: Our blog model will have the following urls:

- `blog/post/add/` - add new post
- `blog/post/<id>/` - view specific post
- `blog/post/<id>/change` - change the given post
- `blog/post/<id>/delete` - delete the given post

View

We have already implemented our blog post view page

`blog/urls.py`

```
    path('post/<int:post_id>/', views.blog_post_detail, name='post-detail')
```

`blog/views.py`

```
def blog_post_detail(request, post_id):
    blog_post = BlogPost.objects.get(id=post_id)

    data = {
        'post': blog_post
    }

    return render(request, 'blog/blog-post-detail.html', data)
```

View

`blog/templates/blog/blog-post-detail.html`

```
{% extends 'base-template.html' %}

{% block 'title' %}{{ post.title }}{% endblock %}

{% block 'content' %}
    <a href="{% url 'blog:home' %}">Home</a>

    <h1>{{ post.title }}</h1>

    <p>Published: <em>{{ post.pub_date }}</em></p>

    <p>{{ post.content }}</p>

    <p><em>{{ post.num_views }} views</em></p>
{% endblock %}
```

Handling Errors

What happens if the object does not exist?

- `BlogPost.objects.get(id=post_id)` will raise a `DoesNotExist` exception

`blog/views.py`

```
def blog_post_detail(request, post_id):
    blog_post = BlogPost.objects.get(id=post_id)
    data = {
        'post': blog_post
    }

    return render(request, 'blog/blog-post-detail.html', data)
```

Handling Errors

What happens if the object does not exist?

The best practice is to return an Http404 error

`blog/views.py`

```
import django.http.Http404

def blog_post_detail(request, post_id):
    try:
        blog_post = BlogPost.objects.get(id=post_id)
    except:
        raise Http404('Blog post does not exist')

    data = {
        'post': blog_post
    }

    return render(request, 'blog/blog-post-detail.html', data)
```

get_object_or_404

This is so common, that Django provides a shortcut function

`get_object_or_404(Model, parameter)`

- Will return the object matching the parameters
- If the object does not exist, raises an 404 error

```
from django.shortcuts import render, get_object_or_404

def blog_post_detail(request, post_id):
    blog_post = get_object_or_404(BlogPost, id=post_id)

    data = {
        'post': blog_post
    }

    return render(request, 'blog/blog-post-detail.html', data)
```

Add

Let's create a page that lets us create a new blog post

Url: `blog/post/add/`

How do we get input on an html?

Forms

We need to use Forms

- Django has a lot of tools to work with forms
- Django does form and data validation

Forms

With forms, we need to understand HTTP GET and POST

- GET is for receiving data from the server
- POST is for sending data to the server

Forms

With forms, we need to understand HTTP GET and POST

- GET is for receiving data from the server
- POST is for sending data to the server

When the user first loads the page, they GET an empty form

When they fill in the form and press (Submit) button, it does a POST with the data

Forms

Django has a Form class that defines the elements of the form

- It works similar to Models, you define the fields

Forms

Let's see an example Form where the user can input their country of residence

How would we write this in html?

Forms

Let's see an example Form where the user can input their country of residence

- It has one text input: the country
- When they click submit, it will POST the data to `/blog/send-country/`

`blog/templates/blog/send-country.html`

```
<form action="/blog/send-country/" method="post">
    <label for="country_in">Tell me what country you live in!</label>
    <input id="country_in" type="text" name="country">

    <input type="submit" value="Submit">
<form>
```

Forms

Let's create a Django form that does the same thing:

`blog/forms.py`

```
from django import forms

class SendCountryForm(forms.Form):
    country = forms.CharField(label="Tell me what country you live in!", max_length=50)
```

Equivalent to:

`blog/templates/blog/send-country.html`

```
<form action="/blog/send-country/" method="post">
    <label for="country_in">Tell me what country you live in!</label>
    <input id="country_in" type="text" name="country">

    <input type="submit" value="Submit">
<form>
```

Forms

We need to create a url for the view:

`blog/urls.py`

```
    path('send-country/', blog_views.send_country, name='send_country')
```

Forms

We need to create a view

`blog/views.py`

Often we use the following pattern:

- If the request is a GET - return an empty form
- If the request is a POST - process the form data and redirect

Forms

Here is our view:

```
def log_country(country):
    with open('countries.log', 'w+') as f:
        f.write(country)

def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():
            log_country(form.cleaned_data['country'])
            return redirect('/blog/')

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    return render(request, 'blog/send-country.html', { "form": form })
```

Forms

The `request` object has a property: `method`

- normally either 'GET' or 'POST'

```
def send_country(request):
    # request is POST - process the data
    if request.method == 'POST':
        ### ...

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    return render(request, 'blog/send-country.html', { "form": form })
```

Forms

If the request.method is GET:

- create a new blank form, and render the page

```
def send_country(request):
    # request is POST - process the data
    if request.method == 'POST':
        ### ...

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    return render(request, 'blog/send-country.html', { "form": form })
```

Forms

If the request is a POST, we initialize the form using the POST data

```
def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    return render(request, 'blog/send-country.html', { "form": form })
```

Forms

Then, we can call the method `form.is_valid()`

- This does form validation and returns True if valid

```
def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    return render(request, 'blog/send-country.html', { "form": form })
```

Forms

If the form is valid, we can use the data (via `form.cleaned_data` dictionary)

- `country = form.cleaned_data['country']`

```
def log_country(country):
    with open('countries.log', 'w+') as f:
        f.write(country)

def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():
            country = form.cleaned_data['country']
            log_country(country)

    # request is GET - return an empty form
    else:
        form = SendCountryForm()
```

Forms

If the form is valid, we can use the data (via `form.cleaned_data` dictionary)

- Here we write the country to a log file

```
def log_country(country):
    with open('countries.log', 'w+') as f:
        f.write(country)

def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():
            country = form.cleaned_data['country']
            log_country(country)

    # request is GET - return an empty form
    else:
        form = SendCountryForm()
```

Forms

Once we are finished processing the form, we return a **redirect**

- A redirect tells the browser to go to a different page
- Here we will tell the browser to return to the blog home page
- Return `django.shortcuts.redirect(url)`

```
from django.shortcuts import render, redirect # Need to import!

def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():
            country = form.cleaned_data['country']
            log_country(country)
            return redirect('/blog/')

    # request is GET - return an empty form
    else:
        form = SendCountryForm()
```

Forms

What if the form is not valid?

- Example: leaving an empty country

Then we will stay on the same page, but Django will show an error message

```
def send_country(request):
    # request is POST - process the data
    if request.method == "POST":
        form = SendCountryForm(request.POST)  # Fill form data with POST values
        if form.is_valid():
            # Use form

    # request is GET - return an empty form
    else:
        form = SendCountryForm()

    # If form is not valid, render the errors here
    return render(request, 'blog/send-country.html', { 'form': form })
```

Forms

Use the following structure for form pages:

```
def send_country(request):
    if request.method == "POST":
        form = Form(request.POST)  # Fill form data with POST values
        if form.is_valid():
            # Use form.cleaned_data
            # return redirect

    else:
        form = Form()   # GET - create empty form

    return render(request, 'app/template-name.html', { 'form': form })
```

Forms

Django will also write our form inputs for us

`blog/templates/blog/send-country.html`

```
{% block content %}
<form action="{% url 'blog:send-country' %}" method="post">
    {% csrf_token %}
    {{ form }}

    <input type="submit" value="Submit">
<form>
```

Forms

!!! IMPORTANT !!!

Always include the `{% csrf_token %}` in every form

- It protects your site from cross-site request forgeries

`blog/templates/blog/send-country.html`

```
{% block content %}
<form action="{% url 'blog:send-country' %}" method="post">
    {% csrf_token %}
    {{ form }}

    <input type="submit" value="Submit">
<form>
```

Forms

Summary:

1. Create a class that extends `django.forms.Form`
2. Create a url for the page with the form
3. Create a view that accepts both GET/POST and creates the Form
4. Have the view render the template with the form as a parameter
5. Create the template an include the `{{ form }}` inside a `<form>` tag
6. Don't forget the `{% csrf_token %}`

Model Forms

Django can automatically generate forms for a specific Model

These are called **ModelForms** (`django.forms.ModelForm`)

- These add additional validation from your model

Model Forms

A Model Form must extend `django.forms.ModelForm`

- Let's create one for a BlogPost

`blog/forms.py`

```
from django import forms

class BlogPostForm(forms.ModelForm):


```

Model Forms

We must define `Meta` information

- `model = BlogPost` - the Model this form is based on
- `fields = [ ]` - the fields of the model to include

`blog/forms.py`

```
from django import forms

from .models import BlogPost

class BlogPostForm(forms.ModelForm):
    class Meta:
        model = BlogPost
        fields = [ 'title', 'content', 'is_published' ]
```

Model Forms

Example: Let's create a page for writing a new Blog Post

`blog/urls.py`

```
    path('post/add/', blog_views.blog_post_add, 'post-add')
```

`blog/views.py`

```
def blog_post_add(request):
    if request.method == "POST":
        form = BlogPostForm(request.POST)
        if form.is_valid():
            return redirect('/blog/')

    else:
        form = BlogPostForm()

    return render(request, 'blog/blog-post-add.html', { 'form': form })
```

Model Forms

Example: Let's create a page for writing a new Blog Post

`blog/templates/blog/blog-post-add.html`

```
{% extends 'base-template.html' %}

{% block 'title' %}Create a new Blog Post{% endblock %}

{% block 'content' %}
    <a href="{% url 'blog:home' %}">Home</a>

    <form action="{% url 'blog:post-add' %}" method="post">
        {% csrf_token %}
        {{ form }}

        <input type="submit" value="Submit">
    </form>
{% endblock %}
```

Model Forms

Model forms have a useful function called save()

- This will try to add or change the object represented by the form

`blog/views.py`

```
def blog_post_add(request):
    if request.method == "POST":
        form = BlogPostForm(request.POST)
        if form.is_valid():
            blog_post = form.save()
            return redirect('/blog/')

    else:
        form = BlogPostForm()

    return render(request, 'blog/blog-post-add.html', { 'form': form })
```

Model Forms

Note: We can give a Model a function `get_absolute_url` to return the default url view for an object

`blog/models.py`

```
class BlogPost(Model):

    def get_absolute_url(self):
        return "/blog/post/" + str(self.id) + "/"
```

Model Forms

If `get_absolute_url` is defined, we can pass the object to `redirect`

`blog/views.py`

```
def blog_post_add(request):
    if request.method == "POST":
        form = BlogPostForm(request.POST)
        if form.is_valid():
            blog_post = form.save()
            return redirect(blog_post)  # redirects to blog_post.get_absolute_url()

    else:
        form = BlogPostForm()

    return render(request, 'blog/blog-post-add.html', { 'form': form })
```

Model Forms

We can also change our `blog/templates/blog/home.html` to use this:

- We can use this in our href address

```
{% block 'content' %}
    <h1>Chakula Chat Blog</h1>

    <ul>
        {% for post in blog_posts %}
        <a href="{{ post.get_absolute_url }}">
            <li>{{ post.title }} (<em>{{ post.pub_date }}</em>)</li>
        </a>
        {% endfor %}
    </ul>
{% endblock %}
```

Model Forms

We can make a page to let the user edit a post (change)

`blog/urls.py`

```
    path('post/<int:post_id>/change/', blog_views.blog_post_change, name='post-change'),
```

Now, our view is working with a specific object,

it takes the id as a second parameter

`blog/views.py`

```
def blog_post_change(request, post_id):
```

Model Forms

The view is similar to the add view

1. We need to get the object at the beginning (the object to change)
2. We need to pass the object instance to the forms (`instance=post`)
3. We need to pass the object to the template as well

`blog/views.py`

```
def blog_post_change(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id) # Need to fetch the specific object
    if request.method == "POST":
        form = BlogPostForm(request.POST, instance=post)
        if form.is_valid():
            blog_post = form.save()   # This will update the object
            return redirect('/blog/')

    else:
        form = BlogPostForm(instance=post)

    return render(request, 'blog/blog-post-change.html', { 'form': form, 'post': post })
```

Model Forms

The template is also very similar to the add template

1. The action url needs to include the post id

`blog/templates/blog/blog-post-change.html`

```
{% extends 'base-template.html' %}

{% block 'title' %}Edit Post: {{ post.title }}{% endblock %}

{% block 'content' %}
    <a href="{% url 'blog:home' %}">Home</a>

    <h1>Edit Blog Post</h1>

    <form action="{% url 'blog:post-change' post.id %}" method="post">
        {% csrf_token %}
        {{ form }}

        <input type="submit" value="Submit">
    </form>
{% endblock %}
```

Linking Pages

Let's add some additional links in our pages:

- Our blog homepage will have a link to add a new post

`blog/templates/blog/home.html`

```
<a href="{% url 'blog:post-add' %}">Create a New Post</a>
```

Linking Pages

Let's add some additional links in our pages:

- A blog post page will have a link to edit it

`blog/templates/blog/blog-post-detail.html`

```
<h1>{{ post.title }}</h1>

<p>
    <a href="{% url 'blog:post-change' post.id %}">Edit Post</a>
</p>
```

Linking Pages

Let's add some additional links in our pages:

- The change page will have a back link

`blog/templates/blog/blog-post-change.html`

```
<p>
    <a href="{% url 'blog:post-detail' post.id %}">Back</a>
</p>
```

Deleting

Finally, let's create a way to delete a post

`blog/urls.py`

```
    path('post/<int:post_id>/delete/', blog_views.blog_post_delete, 'post-delete')
```

`blog/views.py`

```
def blog_post_delete(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    # ...
```

Deleting

Note: we don't need a form, since this is just a simple action

- All Model objects have a `delete()` method

`blog/views.py`

```
def blog_post_delete(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    if request.method == "POST":
        post.delete()
        return redirect('/blog/')

    return render(request, 'blog/blog-post-delete.html', { 'post': post })
```

Deleting

Finally, let's create a way to delete a post

- Our page will have confirm/cancel buttons

`blog/templates/blog/blog-post-delete.html`

```
{% extends 'base-template.html' %}

{% block 'title' %}Delete Post: {{ post.title }}{% endblock %}

{% block 'content' %}
    <a href="{% url 'blog:home' %}">Home</a>

    <h1>Delete Post: {{ post.title }}</h1>

    <form action="{% url 'blog:post-delete' post.id %}" method="post">
        {% csrf_token %}

        <p>Are you sure you want to delete this post?</p>
        <input type="submit" value="Yes">
        <a href="{{ post.get_absolute_url }}">
            <input type="button" value="Cancel">
```

Summary

We have created the following 4 views:

- `blog_post_detail`
- `blog_post_add`
- `blog_post_change`
- `blog_post_delete`

Publish

Let's create another action: Publish

By default, posts are hidden (not published)

Let's add a special button to publish them

- We will also set the publish time

Publishing

Finally, let's create a way to publish a post

`blog/urls.py`

```
    path('post/<int:post_id>/publish/', blog_views.blog_post_publish, 'post-publish')
```

`blog/views.py`

```
def blog_post_publish(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    if request.method == "POST":
        # publish the article

    # Redirect to the post's detail page
    return redirect(post)
```

Publishing

Finally, let's create a way to publish a post

- Note: we don't need a template, since this is just an action
- We can edit the object and then call the save() method to update it

`blog/views.py`

```
from datetime import datetime

def blog_post_publish(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    if request.method == "POST":
        post.is_published = True
        post.pub_date = datetime.now()
        post.save()

    # Redirect to the post's detail page
    return redirect(post)
```

Publishing

Let's add a button to the detail page to publish an article

`blog/templates/blog/blog-post-detail.html`

```
{% block 'content' %}
    <h1>{{ post.title }}</h1>

    <p>
        <a href="{% url 'blog:post-change' post.id %}">
            <button>Edit</button></a>
        <a href="{% url 'blog:post-delete' post.id %}">
            <button>Delete</button></a>
    </p>

    <form action="{% url 'blog:post-publish' post.id %}" method="post">
        {% csrf_token %}
        <input type="submit" value="Publish">
    </form>

    <p>Published: <em>{{ post.pub_date }}</em></p>
{% endblock %}
```

Publishing

Problem: We only want the button if it is not published yet

- Solution: Template if statements

`blog/templates/blog/blog-post-detail.html`

```
{% block 'content' %}
    <h1>{{ post.title }}</h1>

    <p>
        <a href="{% url 'blog:post-change' post.id %}">
            <button>Edit</button></a>
        <a href="{% url 'blog:post-delete' post.id %}">
            <button>Delete</button></a>
    </p>

    <form action="{% url 'blog:post-publish' post.id %}" method="post">
        {% csrf_token %}
        <input type="submit" value="Publish">
    </form>

    <p>Published: <em>{{ post.pub_date }}</em></p>
```

Publishing

Problem: We only want the button if it is not published yet

- Solution: Template if statements

```
{% if post.is_published %}

{% else %}

{% endif %}
```

Publishing

Problem: We only want the button if it is not published yet

- Solution: Template if statements

`blog/templates/blog/blog-post-detail.html`

```
{% block 'content' %}
    <h1>{{ post.title }}</h1>

    <p>
        <a href="{% url 'blog:post-change' post.id %}">
            <button>Edit</button></a>
        <a href="{% url 'blog:post-delete' post.id %}">
            <button>Delete</button></a>
    </p>

    {% if post.is_published %}
        <p>Published: <em>{{ post.pub_date }}</em></p>
    {% else %}
        <form action="{% url 'blog:post-publish' post.id %}" method="post">
            {% csrf_token %}
            <input type="submit" value="Publish">
```