# Dynamically Add Forms in Django with Formsets and JavaScript

This tutorial demonstrates how multiple copies of a form can be dynamically added to a page and processed using Django formsets and JavaScript.

In a web app, a user may need to submit the same form multiple times in a row if they are inputting data to add objects to the database. Instead of having to submit the same form over and over, Django allows us to add mutliple copies of the same form to a web page using formsets.

We'll demonstrate this through a bird watching app to keep track of the birds a user has seen. It will include only two pages, a page with a list of birds and a page with a form to add birds to the list. You can see the completed source code for this example on GitHub.

For this tutorial, it would be helpful to have a basic understanding of:

- Django forms
- Django class based views
- JavaScript DOM manipulation

## Setup

### Model

The app will only have one model, `Bird`, that stores information about each bird

we've seen.

```
# models.py
from django.db import models

class Bird(models.Model):
    common_name = models.CharField(max_length=250)
    scientific_name = models.CharField(max_length=250)

    def __str__(self):
      return self.common_name
```

The model is composed of two `CharFields` for the common name and the scientific name of the bird

## URLs

We will manage URLs with a project level `urls.py` file set to include the app level `urls.py` file.

```
# project level urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('birds.urls')),
]
```

The app level `urls.py` will include the paths to the two pages of the app.

```
# app level urls.py
from django.urls import path
from .views import BirdAddView, BirdListView
```

```
urlpatterns = [
    path('add', BirdAddView.as_view(), name="add_bird"),
    path('', BirdListView.as_view(), name="bird_list")
]
```

### Views

Initially, we'll set up the view for just the page that lists the birds. This view will use a generic `ListView` to get all bird instances and make them available for display in the specified template.

```
# views.py
from django.views.generic import ListView
from .models import Bird

class BirdListView(ListView):
    model = Bird
    template_name = "bird_list.html"
```

We will add the view for the form page later on.

## Formsets

At this point, we could create a form and a view that would provide the form to the template so the user could add a new bird to the list. But the user would only be able to add one bird at a time. To add multiple birds at one time we need to use a formset instead of a regular form.

Formsets allow us to have multiple copies of the same form on a single page. This is helpful if the user should be able to submit multiple forms at once to create multiple model instances. In our example, this would allow the user to be able to add multiple birds to their list at one time without having to submit the form for each individual bird.

### Model Formsets

Formsets can be created for different types of forms, including model forms. Since we want to create new instances of the `Bird` model, a model formset is a natural fit.

If we just wanted to add a single model form to our page, we would create it like this:

```
# forms.py
from django.forms import ModelForm
from .models import Bird

# A regular form, not a formset
class BirdForm(ModelForm):
    class Meta:
      model = Bird
      fields = [common_name, scientific_name]
```

The model form requires we specify the model we want the form to be associated with and the fields we want to include in the form.

To create a model formset, we don't need to define the model form at all. Instead, we use Django's `modelformset_factory()` which returns a formset class for a given model.

```
# forms.py
from django.forms import modelformset_factory
from .models import Bird

BirdFormSet = modelformset_factory(
    Bird, fields=("common_name", "scientific_name"), extra=1
)
```

`modelformset_factory()` requires the first argument be the model the formset is for. After specifying the model, different optional arguments can be specified.

We'll use two optional arguments - `fields` and `extra`.

- `fields` - specifies the fields to be displayed in the form
- `extra` - the number of forms to initially display on the page

By setting `extra` to 1, we will be passing only one form to the template initially. One is the default value, but we'll explicitly specify it for clarity. Setting `extra` to any other number will provide that number of forms to the template.

`Extra` allows us to display multiple copies of the form to the user, but we may not know how many birds the user wants to add at one time. If we show too few forms, then the user would still have to submit the form multiple times. If we show too many forms, the user may have to scroll far down the page to find the submit button. Luckily, by setting `extra` we aren't limiting ourselves to only having that many copies of the form. We can add as many copies of the form as we realistically would need (up to 1000 copies) to our page even if `extra` is set to 1. We can add the other copies of the form to the page dynamically with JavaScript. We'll do that when we create our template for displaying the form. Before we can do that, we need to create the view.

## View

We need a view that can handle the GET request to initially display the form, and the POST request when the form in submitted. We'll use the class-based `TemplateView` for the base view functionality and add methods for GET and POST requests.

The GET request requires that we create an instance of the formset and add it to the context.

```
# views.py
from django.views.generic import ListView, TemplateView # Import Te
from .models import Bird
from .forms import BirdFormSet # Import the formset

class BirdListView(ListView):
    model = Bird
    template_name = "bird_list.html"
```

```
# View for adding birds
class BirdAddView(TemplateView):
    template_name = "add_bird.html"

    # Define method to handle GET request
    def get(self, *args, **kwargs):
        # Create an instance of the formset
        formset = BirdFormSet(queryset=Bird.objects.none())
        return self.render_to_response({'bird_formset': formset})
```

To create the formset instance, we first import the formset and then we call it in the `get` method. If we just call the formset with no arguments, we will get a formset that contains a form for all `Bird` instances in the database. Since we want this view to only add new birds we need to prevent the displayed forms from being pre-populated with `Bird` instances. To do that we specific a custom queryset. We set the queryset argument to `Bird.objects.none()`, which creates an empty queryset. This way no birds will be pre-populated in the forms.

Once we have created the formset instance, we call `render_to_response` with the argument being a dictionary that has the formset assigned to the `bird_formset` key.

For the POST request, we need to define a `post` method that handles the form when it is submitted.

```
# views.py
from django.views.generic import ListView, TemplateView
from .models import Bird
from .forms import BirdFormSet
from django.urls import reverse_lazy
from django.shortcuts import redirect

class BirdListView(ListView):
```

```
    model = Bird
    template_name = "bird_list.html"

class BirdAddView(TemplateView):
    template_name = "add_bird.html"

    def get(self, *args, **kwargs):
        formset = BirdFormSet(queryset=Bird.objects.none())
        return self.render_to_response({'bird_formset': formset})

    # Define method to handle POST request
    def post(self, *args, **kwargs):

        formset = BirdFormSet(data=self.request.POST)

        # Check if submitted forms are valid
        if formset.is_valid():
            formset.save()
            return redirect(reverse_lazy("bird_list"))

        return self.render_to_response({'bird_formset': formset})
```

First, we create an instance of the `BirdFormSet`. This time we include the submitted data from the request via `self.request.POST`. Once the formset is created, we have to validate it. Similar to a regular form, `is_valid()` can be called with a formset to validate the submitted forms and form fields. If the formset is valid, then we call `save()` on the formset which creates new `Bird` objects and adds them to the database. Once that is complete, we redirect the user to the page with the list of birds. If the formset is not valid, the formset is returned to the user with the appropriate error messages.

## Template

Now that we have the views set up, we can create the templates that will be rendered to the user.

To display the list of birds, we will loop through `object_list` which is provided by `ListView` and includes all bird instances in the database.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scal
    <title>Bird List</title>
</head>
<body>
    <h1>Bird List</h1>
    <a href='{% url "add_bird" %}'>Add bird</a>
    {% for bird in object_list %}
        <p>{{bird.common_name}}: {{bird.scientific_name}}</p>
    {% endfor %}
</body>
</html>
```

We'll display the common name and the scientific name for each bird. and include a link to the page with the form to add a bird to the list.

The template for our add bird page will initially just show the single form we specified with `extra`.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scal
    <title>Add bird</title>
</head>
<body>
    <h1>Add a new bird</h1>
```

```
    <form id="form-container" method="POST">
        {% csrf_token %}
        {{bird_formset.management_form}}
        {% for form in bird_formset %}
        <div class="bird-form">
        {{form.as_p}}
        </div>
        {% endfor %}
        <button type="submit">Create Birds</button>
    </form>
    </body>
</html>
```

We create an HTML `<form>` with an id of `form-container` and method of `POST`. Like a regular form, we include the `csrf_token`. Unlike a regular form, we have to include `{{bird_formset.management_form}}`. This inserts the management form into the page which includes hidden inputs that contain information about the number of forms being displayed and is used by Django when the form is submitted.

```
<input type="hidden" name="form-TOTAL_FORMS" value="1" id="id_form-
<input type="hidden" name="form-INITIAL_FORMS" value="0" id="id_for
<input type="hidden" name="form-MIN_NUM_FORMS" value="0" id="id_for
<input type="hidden" name="form-MAX_NUM_FORMS" value="1000" id="id_
```

The `form-TOTAL_FORMS` input contains the value of the total number of forms being submitted. If it doesn't match the actual number of forms Django receives when the forms are submitted, an error will be raised.

Forms in the formset are added to the page using a `for` loop to go through each form in the formset and render it. We've chosen to display the form in `<p>` tags by using `{{form.as_p}}`. The forms are rendered in HTML as:

```
<p>
```

```
    <label for="id_form-0-common_name">Common name:</label>
    <input type="text" name="form-0-common_name" maxlength="250" id="
  </p>
  <p>
    <label for="id_form-0-scientific_name">Scientific name:</label>
    <input type="text" name="form-0-scientific_name" maxlength="250"
    <input type="hidden" name="form-0-id" id="id_form-0-id">
  </p>
```

Each field in the form gets a `name` and `id` attribute that contains a number and the name of the field. The field for the common name has a `name` attribute of `form-0-common_name` and an `id` of `id_form-0-common_name`. These are important because the number in both of these attributes is used to identify what form the field is a part of. For our single form, each field is part of form 0, since form numbering starts at 0.

While this template will work and allow the user to submit one bird at a time, it still doesn't allow us to add more forms if the user wants to add more than one bird. We could have added more forms by setting `extra` to a different number. Instead, we are going to use JavaScript to allow the user to choose how many forms they want to submit.

## Making Formsets Dynamic

Since we are using formsets and have set up our views to handle a formset instead of a regular form, users can submit as many forms as they want at one time. We just need to make those forms available to them on the page. This can be done using DOM manipulation with JavaScript.

Adding additional forms requires using JavaScript to:

- Get an existing form from the page
- Make a copy of the form
- Increment the number of the form
- Insert the new form on the page
- Update the number of total forms in the management form

Before we get to JavaScript, we'll add a another button to the HTML form that

will be used to add additional forms to the page.

```
<button id="add-form" type="button">Add Another Bird</button>
```

This button will be added directly before the "Create Birds" button.

To get the existing form on the page, we use `document.querySelectorAll('.bird-form')` where `bird-form` is the class given to the `div` that contains the fields of each form. While `document.querySelectorAll('.bird-form')` will return a list of all elements on the page that have a class of `bird-form`, we only have one form on the page at this point, so it will return a list that includes only the one form. We will use `document.querySelector()` to get the other elements on the page necessary to perform all the steps, namely the whole HTML form, the button the user can click to add a new form to the page, and the total forms input of the management form.

```
let birdForm = document.querySelectorAll(".bird-form")
let container = document.querySelector("#form-container")
let addButton = document.querySelector("#add-form")
let totalForms = document.querySelector("#id_form-TOTAL_FORMS")
```

We also need to get the number of the last form on the page. Again, since we know only one form will be initially displayed on the page, we could just say the last form is form 0. But in the case where we wanted more than one form on the page initially, we can find it by taking the number of forms stored in `birdForm` and subtracting one to account for form numbering starting at 0.

```
let formNum = birdForm.length-1 // Get the number of the last form
```

We only want a new form to be added to the page when the user clicks on the "Add Another Bird" button. This means the rest of the steps should only be executed when the button is pressed. We need to create a function that

performs the remaining steps and attach it to the button press with an event listener. We'll call our function `addForm` so it can be associated to the button click by

```
addButton.addEventListener('click', addForm)
```

The `addForm` function will look like this

```
function addForm(e) {
    e.preventDefault()

    let newForm = birdForm[0].cloneNode(true) //Clone the bird form
    let formRegex = RegExp(`form-(\\d){1}-`,'g') //Regex to find al

    formNum++ //Increment the form number
    newForm.innerHTML = newForm.innerHTML.replace(formRegex, `form-
    container.insertBefore(newForm, addButton) //Insert the new for

    totalForms.setAttribute('value', `${formNum+1}`) //Increment th
}
```

First, we prevent the default action of the button click so only our `addForm` function is executed. Then we create a new form by cloning `birdForm` using `.cloneNode()`. We pass `true` as an argument so all child nodes of `birdForm` are also copied to `newForm`.

Since we created a copy, `newForm` includes the exact same attribute values as the form already on the page. This means the form numbers are the same. Since we cannot submit two forms with the same number, we need to increment the number of the form. We do this by incrementing `formNum` then using a regular expression to find and replace all instances of the form number in the HTML of `newForm`.

By looking at the HTML of the form, we can see that all attributes which include

the form number contain a common pattern of `form-0-`. We'll create a regular expression to match this pattern and save it to `formRegex`.

Next, we'll identify all instances that match `formRegex` in the HTML of `newForm` and replace it with a new string. We want the replacement string to be the same as the matched pattern expect the number will now be the value of the recently incremented `formNum`. By accessing the HTML of `newForm` with `.innerHTML` and then using `.replace()`, we are able to match the regular expression and perform the replacement.

Now that we have the correct form number in `newForm`, we can add it to the page. We'll use `.insertBefore()` and add the new form before `addButton`.

With the form now on the page, we just need to make sure the management form is properly updated with the correct amount of forms that will be submitted. We need to increment the value of the `form-TOTAL_FORMS` hidden field. We already saved this field in `totalForms`. To update it, we'll use `.setAttribute()` to set the `value` attribute to one greater than the current form number. We do one greater because `form-TOTAL_FORMS` includes the number of forms on the page starting at 1, while individual form numbering starts at 0.

We can now add all of the JavaScript to a `<script>` tag before the closing `<body>` tag to get our final template.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scal
    <title>Add bird</title>
</head>
<body>
    <h1>Add a new bird</h1>
    <form id="form-container" method="POST">
        {% csrf_token %}
```

```
        {{bird_formset.management_form}}
        {% for form in bird_formset %}
        <div class="bird-form">
        {{form.as_p}}
        </div>
        {% endfor %}
        <button id="add-form" type="button">Add Another Bird</butto
        <button type="submit">Create Birds</button>
    </form>


    <script>
        let birdForm = document.querySelectorAll(".bird-form")
        let container = document.querySelector("#form-container")
        let addButton = document.querySelector("#add-form")
        let totalForms = document.querySelector("#id_form-TOTAL_FOR

        let formNum = birdForm.length-1
        addButton.addEventListener('click', addForm)

        function addForm(e){
            e.preventDefault()

            let newForm = birdForm[0].cloneNode(true)
            let formRegex = RegExp(`form-(\\d){1}-`,'g')

            formNum++
            newForm.innerHTML = newForm.innerHTML.replace(formRegex
            container.insertBefore(newForm, addButton)

            totalForms.setAttribute('value', `${formNum+1}`)
        }
    </script>
    </body>
</html>
```

That's it! Now the user can add as many forms as they want to the page by clicking the "Add Another Bird" button and when the form is submitted they will be saved to the database, the user will be redirected back to the list of birds, and all of the newly submitted birds will be displayed.

## Summary

Django formsets allow us to include multiple copies of the same form on a single page and correctly process them when submitted. Additionally, we can let the user choose how many forms they want to submit by using JavaScript to add more forms to the page. While this tutorial used modelformsets formsets for other types of forms can also be used.

## Helpful Resources

- Django Documentation on Formsets
- Django Documentation on Model Formsets

 brennan@brennantymrak.com

Questions, comments, still trying to figure it out? - Let me know!