

How to Style Your Django Forms

Spoiler: You have to use widgets.



Gustavo Maciel · Follow

Published in The Startup · 4 min read · Jan 13, 2021



251



1

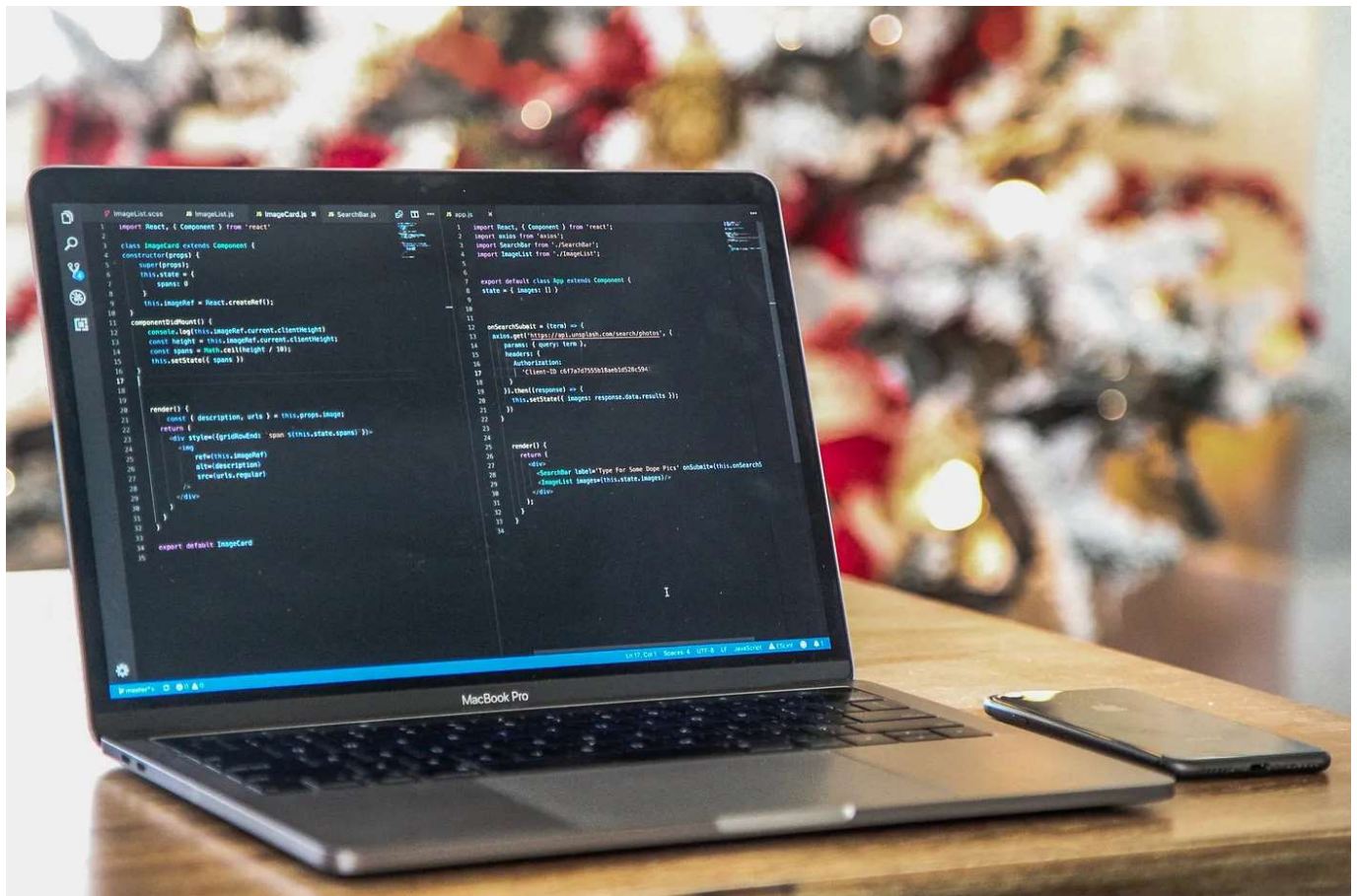


Photo by [Joshua Aragon](#) on [Unsplash](#)

Everyone who uses Django knows how good the Django forms can be. But when you are using it for the first time, a question arises: How can I style it? How can I add a class?

Well, there's a way (quite simple actually), you have to use widgets.

But what is that exactly? Let's see the definition of widget, according to Django Docs:

A widget is Django's representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget.

In other words, widget is just a way to define how that content will be rendered as HTML. So, for example, a **CharField** has a default widget of **TextInput** that renders as `<input type="text">` .

But widgets are customizable, so you can also set things like the size of that **textarea** or if that field is going to be a *required* field and so goes on...

So, let's try to build an example to show the widgets in action.

Suppose we have a form called **UserInfoForm** to get the name of the user and also his email.

```
1  from django import forms
2
3  class UserInfoForm(forms.Form):
4      name = forms.CharField()
5      email = forms.EmailField()
```

forms.py hosted with ❤ by GitHub

[view raw](#)

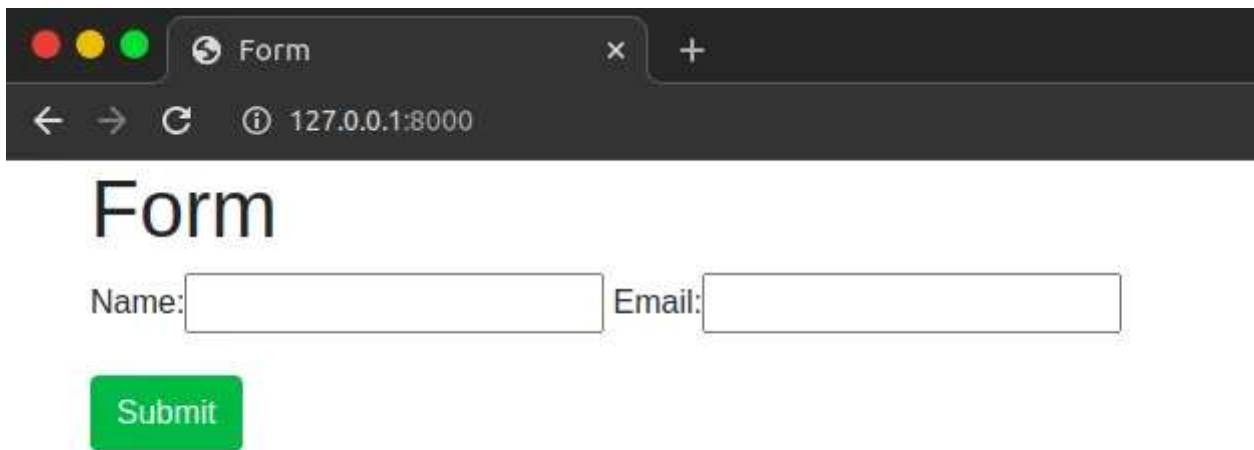
And the HTML looks like this:

```
1  <div class="container">
2      <h1>Form</h1>
3      <form action="{% url 'index' %}" method="post">
4          {% csrf_token %}
5          <div class="form-group">
6              {{ form }}
7          </div>
8          <div class="form-group">
9              <input class="btn btn-success" type="submit" value="Submit">
10          </div>
11      </form>
12  </div>
```

index.html hosted with ❤ by GitHub

[view raw](#)

Right now this form is using the default widgets, and it doesn't have any style, so, basically, it looks like this:



So, to change it, we need to customize the appearance. You can customize widgets in two ways — it can be via **widget instance** or **widget class**. For this first example, I'll be using widget instance. Basically you have to use the **Widget.attrs** argument, which is a dictionary containing HTML attributes to be set to the rendered widget, like the example below:

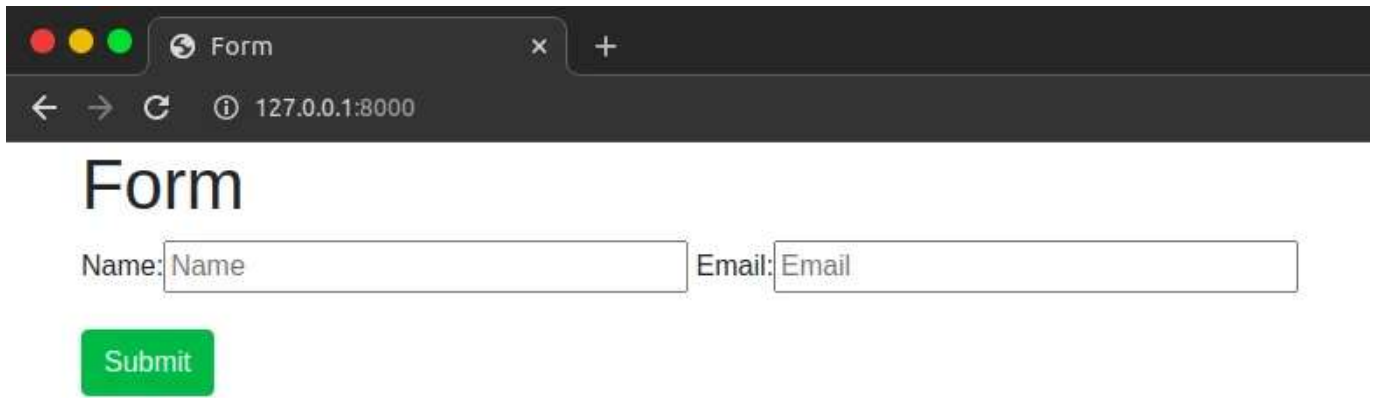
```
1 from django import forms
2 from django.forms import TextInput, EmailInput
3
4 class UserInfoForm(forms.Form):
5     name = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Name', 'style': 'width: 300px;'}))
6     email = forms.EmailField(widget=forms.EmailInput(attrs={'placeholder': 'Email', 'style': 'width: 300px;'}))
```

forms.py hosted with ❤️ by GitHub

[view raw](#)

And then we can see the **attrs** dictionary taking **placeholder** as a key and the **Name** as value as long with the **style** as key and the **width: 300px;** as value.

And this is the result:

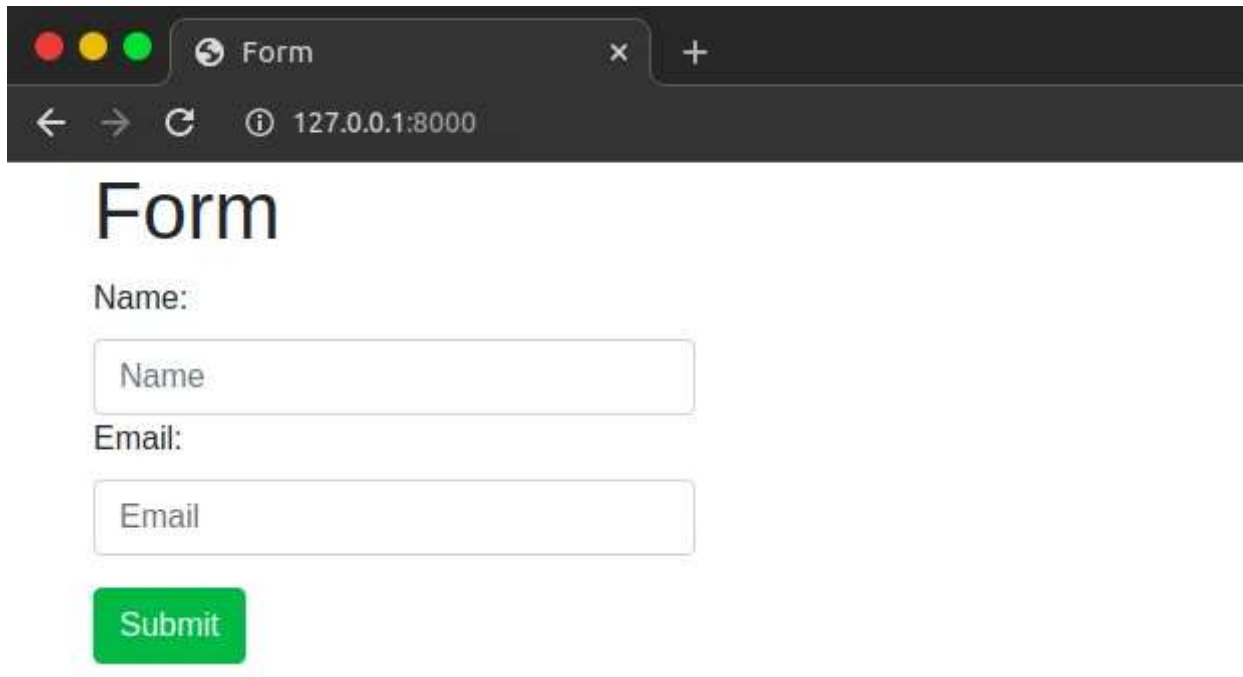


Much better, right? But it still can be improved, we can add a Bootstrap class to it. And we can do this by declaring a class inside the **attrs** dictionary.

```
1 from django import forms
2 from django.forms import TextInput, EmailInput
3
4 class UserInfoForm(forms.Form):
5     name = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Name', 'style': 'width: 100%;'}))
6     email = forms.EmailField(widget=forms.EmailInput(attrs={'placeholder': 'Email', 'style': 'width: 100%;'}))
```

[Open in app](#)[Sign up](#)[Sign in](#)[Write](#)

We've added a Bootstrap class and this is the result.



The screenshot shows a web browser window with a single tab titled 'Form'. The address bar displays '127.0.0.1:8000'. The page content features a large heading 'Form' at the top. Below the heading, there is a label 'Name:' followed by a text input field containing the placeholder text 'Name'. Underneath this is a label 'Email:' followed by another text input field containing the placeholder text 'Email'. At the bottom of the form is a green rectangular button with the text 'Submit' in white.

Now it's good!

But usually when we are using Django forms, those forms are related to some model — and right now, this form isn't. To that happen, we need to make a few more changes. We're going to use the **widget class** now. The widget class has a basic attribute **attrs**, just like the the example above. We also have to add a new class called **Meta** and specify the name of the model this form is related to, the fields we want to have and finally the widgets for those fields.

```
1  from django import forms
2  from django.forms import ModelForm, TextInput, EmailInput
3
4  from .models import User
5
6  class UserInfoForm(ModelForm):
7      class Meta:
8          model = User
9          fields = ['name', 'email']
10         widgets = {
11             'name': TextInput(attrs={
12                 'class': "form-control",
13                 'style': 'max-width: 300px;',
14                 'placeholder': 'Name'
15             }),
16             'email': EmailInput(attrs={
17                 'class': "form-control",
18                 'style': 'max-width: 300px;',
19                 'placeholder': 'Email'
20             })
21         }
```

forms.py hosted with ❤ by GitHub

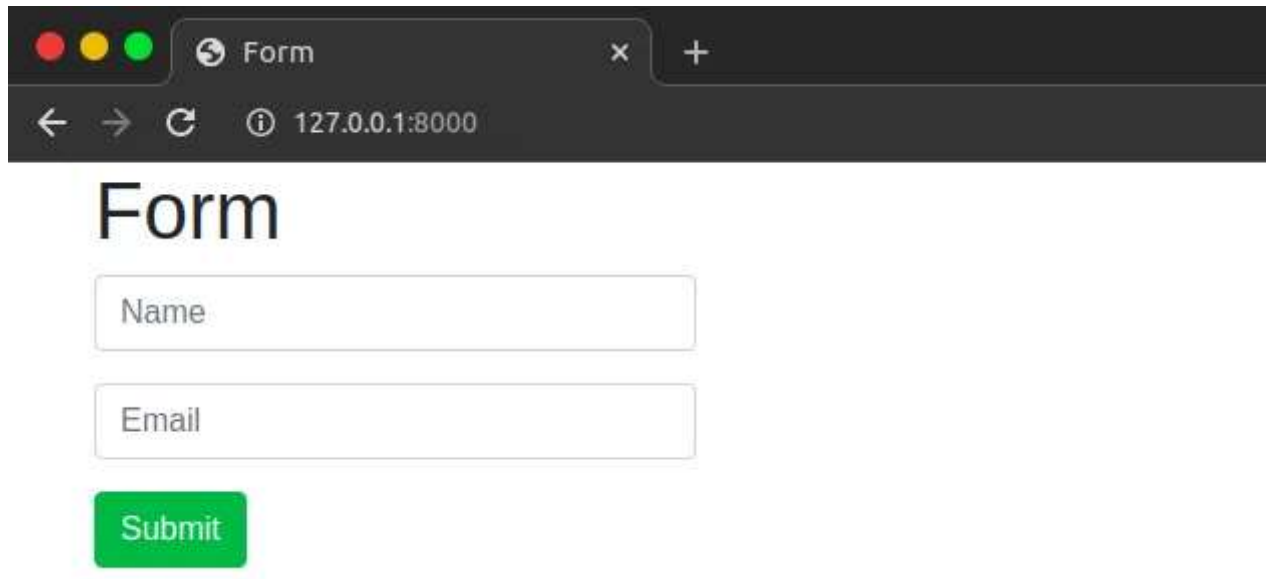
[view raw](#)

So, what's going on here? The **CharField** and the **EmailField** that we were using before are **Built-in Field classes**, but if we are going to work with widgets class, we need to use the **Built-in widgets** (and these widgets are **TextInput** and **EmailInput**, for this example). They will work the same in the end but the configuration will be slightly different.

This is basically the same form as before, it didn't change the way it gets rendered or anything, but now this form is *connected* to the **User** model, which is the one that was created to store the user info.

And the HTML looks like this:

The final result:

A screenshot of a web browser window. The browser's address bar shows the URL '127.0.0.1:8000'. The page title is 'Form'. The form itself consists of a title 'Form' in a large, bold, black font. Below the title are two text input fields, one labeled 'Name' and one labeled 'Email', both with light gray borders. Below these fields is a green rectangular button with the word 'Submit' in white text.

Conclusion

Learning how to use widgets is great because now you know how to make those forms look better and how simple it is to add a Bootstrap class, which can be handy.

You can check the source code of this project on [Github](#).

Of course there's much more you can do with widgets. And the best place to learn is from the official documentation — *aka* [Django docs](#).

[Programming](#)[Django](#)[HTML](#)[CSS](#)[Bootstrap](#)



Written by Gustavo Maciel

29 Followers · Writer for The Startup

Follow

I write about some things I find useful.

More from Gustavo Maciel and The Startup



Gustavo Maciel in Geek Culture

How to Deploy a Django App on Heroku

A step by step guide.

4 min read · Jan 22, 2021



121



1



Sinem Günel in The Startup

I Constantly Use MrBeast's Strategies to Grow My Business — Here's How

3 psychological biases you won't be able to unsee anymore



11 min read · Feb 13, 2024



4.4K



76



Michael Lim in The Startup

How Making A \$100 Per Day Through Your One-Person Business...

My life changed the day I made \$1 online.

4 min read

Jan 22, 2024

3.6K

96

Maryam Merchant

10 Habits That Are Damn Hard to Do, But Pay off Forever

Master it and you will find your way to your goals

8 min read

Jan 4, 2024

5K

80

See all from Gustavo Maciel

See all from The Startup

Recommended from Medium





Vaibhav Sharma

Unlocking the Magic of Single Page Applications with Django and HTMX

Django, renowned as the 'web framework for perfectionists with deadlines,' empowers...

7 min read · Sep 18, 2023



30



2



Martin de Jesus

Real-Time Progress Bar using Django Channels, React, and WebSockets

In the world of modern web applications, users expect more than just static content....

7 min read · Sep 4, 2023



93



Lists

General Coding Knowledge

20 stories · 930 saves



Coding & Development

11 stories · 453 saves



Stories to Help You Grow as a Software Developer

19 stories · 834 saves

ChatGPT

21 stories · 473 saves



TatibaevMurat in Python in Plain English

Creating Beautiful Forms in Django with django-crispy-forms

In the world of web development, forms are an essential part of collecting user data and...

3 min read · Sep 13, 2023



biswajit panda

Passing Variables from Django Function (view) to HTML page...

To know more details about folder structure and html rendering read this

7 min read · Nov 23, 2023



Nuno Bispo in Django Unleashed

Email Management in Django with Django-Mailbox

Django-Mailbox is an open-source Django app that allows you to build a mailbox within...

★ • 3 min read • Sep 5, 2023



JOKEN VILLANUEVA

Django Projects and Ideas with Free Source Code | Beginners to Advance...

Hello, Itsourcecoders. In today's topic, I'm going to share with you the Django System...

8 min read • Jan 30, 2024

See more recommendations