Bringing forms to the front end: Forms and ModelForms

Faisal Memon

Forms in Django

- The real power of any scalable but application lies in allowing users to to submit the information into the database.
- Forms in the front end enable users apart from site managers submit information into your web application.

<form> tag has 2 important attributes

action→ Defines where the information is to be submitted

method→ Defines how the information is submitted

Example form

```
<form action="/save" method="post">
  <label for="first_name">First name:</label>
 <br>
  <input type="text" id="first_name" name="first_name">
 <br>
  <label for="last_name">Last name:</label>
 <br>
  <input type="text" id="last_name" name="last_name">
  <input type="submit">
</form>
```

Issues in our current way

- We do everything manually getting data in views, server side validation and so on.
- Validation becomes very complicated and manual.
- We will have to validations for things like maximum character length allowed, required values, email type validation and so on for both client side and server side.
- Manual handling needed to handle hygienic things like form gets cleared when we submit the data and if validation is not passed.

Django handles the following aspect of forms

- Generating HTML for the fields defined and rendering it on the browser.
- Binding data to forms and displaying.
- Handling data submissions done by users via forms.
- Cleaning data and making validations easy to implement for developers.
- Preparing and structuring data to make it ready for rendering it to the users.
- If data entered is invalid, then reload the form along with the saved user submitted information and error messages for respective fields.

How does Django do it?

- ModelForm
- Form

Summarizing our observations

- Forms do a very good job of generating HTML for the fields defined.
- Binding data to forms and displaying.
- Handling data submissions done by users via forms.
- Cleaning data and making validations easy to implement for developers.
- If data entered is invalid, then reload the form along with the saved user submitted information and error messages for respective fields.

Ways of rendering form

- form.as_p
- form.as_table
- form.as_ul

Validating fields

- clean_<field_name>
- Validators

Attributes of {{field}}

{{ field.field_name }} → allows you to render the field from the form.

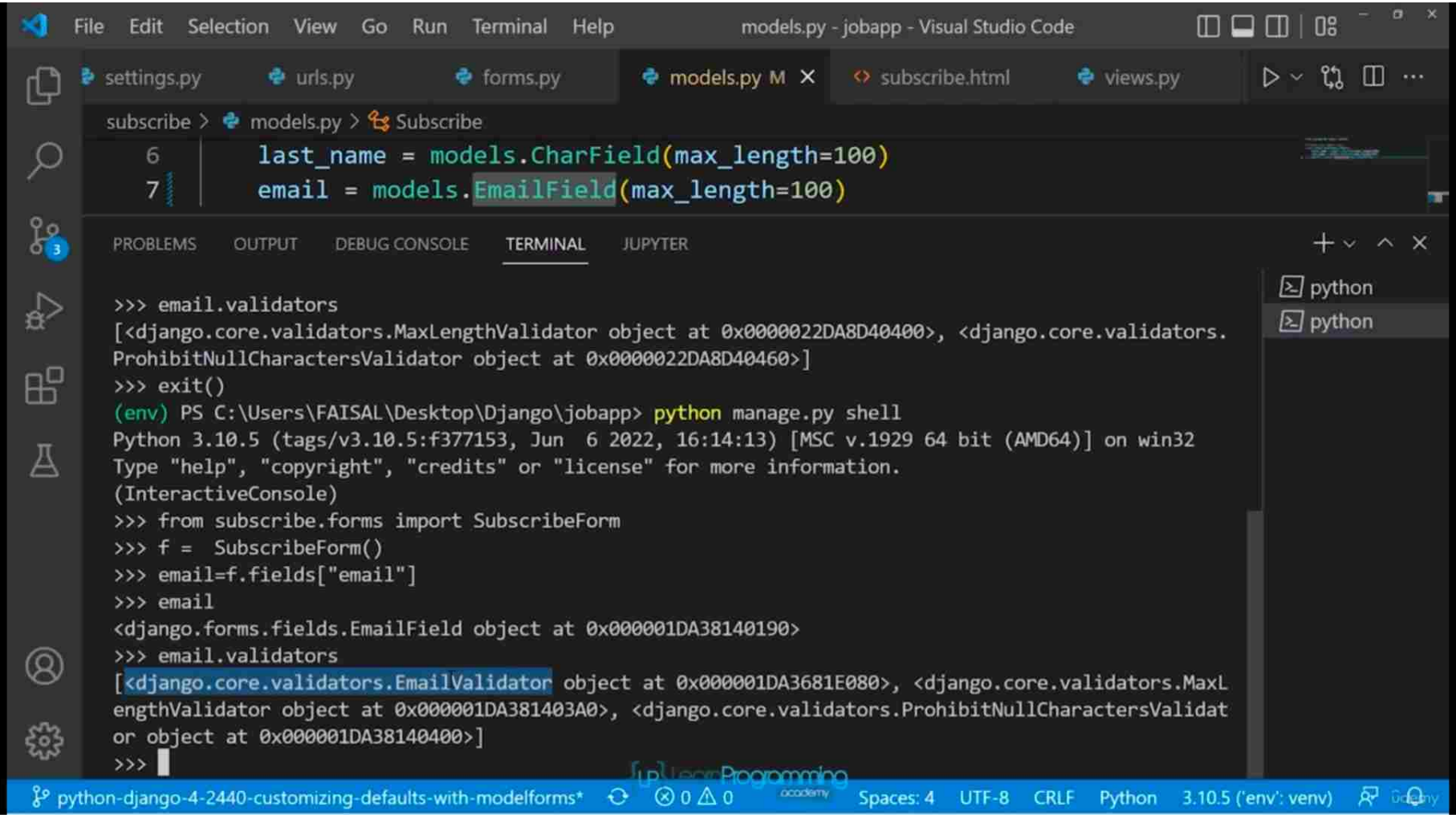
{{ field.errors }} → This renders a <ul class="errorlist"> containing any validation errors corresponding to this field.

{{ field.label }} → allows you to render the label of a field that is set in the form.

{{ field.id_for_label }} → this will return the id of the field for the label tag.

{{ field.help_text }} → allows you to access help text and render it manually.

{{ field.label_tag }} → this will generate the label tag that you wish to render in HTML for a particular field.


File Edit Selection View Go Run Terminal Help
models.py - jobapp - Visual Studio Code
settings.py
urls.py
forms.py
models.py M
subscribe.html
views.py
subscribe > models.py > Subscribe
6 last_name = models.CharField(max_length=100)
7 email = models.EmailField(max_length=100)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
>>> email.validators
[<django.core.validators.MaxLengthValidator object at 0x0000022DA8D40400>, <django.core.validators.
ProhibitNullCharactersValidator object at 0x0000022DA8D40460>]
>>> exit()
(env) PS C:\Users\FAISAL\Desktop\Django\jobapp> python manage.py shell
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from subscribe.forms import SubscribeForm
>>> f = SubscribeForm()
>>> email=f.fields["email"]
>>> email
<django.forms.fields.EmailField object at 0x000001DA38140190>
>>> email.validators
[<django.core.validators.EmailValidator object at 0x000001DA3681E080>, <django.core.validators.MaxL
engthValidator object at 0x000001DA381403A0>, <django.core.validators.ProhibitNullCharactersValidat
or object at 0x000001DA38140400>]
>>>
python
python
python-django-4-2440-customizing-defaults-with-modelforms*
0 0
Spaces: 4 UTF-8 CRLF Python 3.10.5 ('env': venv)

blank vs required vs null

blank and null options are to be used with Models and are used with fields that you define and required is a option that is usually used in forms.py.

blank option

- blank option accepts True and False.
- blank=True simply means that empty forms are accepted.
- Empty forms mean that the associated field where you are specifying this option is not required in a form.

null option

- null option accepts boolean True and False.
- null=True simply means that on the database level Python None values can be stored in the model and be saved.
- NULL values will be allowed in database tables.

required option

- required option accepts boolean True and False.
- This option is used at form level and is used to specify if the associated value is required or not.
- if you have required=False means that a particular form field is not required.

A form using Form class

```
from django import forms

class SubscribeForm(forms.Form):
    first_name = forms.CharField(max_length=100)
    last_name = forms.CharField(max_length=100)
    email = forms.EmailField(max_length=100)
```

Benefits of using Form class

- Forms do a very good job of generating HTML for the fields defined and rendering it on the browser
- Binding data to forms and displaying, which save some time for developers and binding data to forms is one of the common use case which developers have to do
- Forms handle data submissions done by users via forms
- Cleaning data and making validations easy to implement for developers
- If data entered is invalid, then the form retains the data entered in Form

A form using ModelForm class

```
from django import forms

from subscribe.models import Subscribe

class SubscribeForm(forms.ModelForm):
    class Meta:
        model=Subscribe
        fields = ['first_name','last_name','email']
```