Form API

In this reading item, you'll explore the important features of the Form class defined in **`django.forms`** module.

HTML Form

Almost every web application collects certain data from the user by presenting them with a form to fill out and submit. A form is a document wherein the user enters their responses at certain labeled placeholders. For example, a form that is used to register on a certain site, or the one used to fill your travel details for booking a ticket, or a job application form.

You can construct a form using HTML's form tag and its various input elements (such as text, radio, and checkboxes, dropdowns and lists).

```html
<form action="/form/" method="POST">
    <label for="Name">Name of Applicant</label>
    <input type="text" id="name" name="name">

    <label for="Address">Address</label>
    <input type="text" id="add" name="add">

    <label for="Post">Post</label>
    <select id="Post" name="Post">
      <option value="Manager">Manager</option>
      <option value="Cashier">Cashier</option>
      <option value="Operator">Operator</option>
    </select>

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

HTML has very little validation support for these elements. Hence, you need to use JavaScript functions to validate the data before submitting it to the server.

Django Form

Django framework includes a Form class, its attributes, and methods in the **`django.forms`** module. This class is used as a base for a user-defined form design.

```python
from django import forms

class ApplicationForm(forms.Form):
   pass
```

The attributes of the form are Field class objects. The forms module has a collection of Field types. These fields correspond to the HTML elements they eventually render on the user's browser. For example, the **`forms.CharField`** is translated to HTML's text input type. Similarly, the **`ChoiceField`** is equivalent to **`SELECT`** in HTML. The Django form class representing the above Application form would be:

```python
from django import forms

class ApplicationForm(forms.Form):
    name = forms.CharField(label='Name of Applicant', max_length=50)
    address = forms.CharField(label='Address', max_length=100)
    posts = (('Manager', 'Manager'),('Cashier', 'Cashier'),('Operator', 'Operator'))
    field = forms.ChoiceField(choices=posts)
```

By convention, the user-defined form classes are stored in a forms.py file in the app's package folder. So, if you have a Django app called **`myapp`**, the above code is kept in **`myapp/forms.py`**.

Django Form Fields

Out of the many available form field types, some of the most frequently used are as follows:

CharField: Translates to input **`type=text`** HTML form element. If you want to accept a longer multiline text, set its widget property to **`forms.Textarea`**

```python
forms.CharField(label="Enter first name",max_length=50)
```

IntegerField: Similar to a CharField but customized to accept only integer numbers. You can limit the value entered by setting `min_value` and `max_value` parameters.

```
age = forms.IntegerField(min_value=20, max_value=60)
```

FloatField: A text input field that validates if the input is a valid float number. Its variant `DecimalField` accepts a float number of specified decimal places.

```
price = forms.FloatField()
```

FileField: Presents an input `type=file` element on the HTML form.

```
upload = forms.FileField(upload_to ='uploads/')
```

ImageField: Similar to `FileField`with added validation to check if the uploaded file is an image. The pillow library must be installed for this field type to be used.
EmailField: A `CharField` that can validate if the text entered is a valid email ID.

```
email = forms.EmailField(max_length = 254)
```

ChoiceField: Emulates the HTML's SELECT element. Populates the drop-down list with a choice parameter whose value should be a sequence of two item tuples.

```
gender = forms.CharField(max_length=1, choices=GENDER_CHOICES)
```

The instance of the Form class translates to the HTML script of a form. When returned to the browser, the form is rendered.
Let us open the Django shell and check what the form object returns.

```
>>> from myapp import forms
>>> f = ApplicationForm()
>>> print(f)
```

You'll see the following HTML script:

```html
<tr>
    <th><label for="id_name">Name of Applicant:</label></th>
    <td>

        <input type="text" name="name" maxlength="50" required id="id_name">

    </td>
</tr>
<tr>
    <th><label for="id_address">Address:</label></th>
    <td>
        <input type="text" name="address" maxlength="100" required id="id_address">

    </td>
</tr>
<tr>
    <th><label for="id_field">Field:</label></th>
    <td>
        <select name="field" id="id_field">
            <option value="Manager">Manager</option>
            <option value="Cashier">Cashier</option>
            <option value="Operator">Operator</option>
        </select>
    </td>
</tr>
```

Form Template
The form object thus translates to HTML script of form – minus the **`<form>`** as well as the **`<table>`** tag. To render it on a browser, youhave to first write an HTML template and put the form object in **`jinja2`** tag. Let us save the following **`form.html`** file in the project's templates folder.

```html
<html>
<body>
    <form action="/form" action="POST">
        {% csrf_token %}
        <table>
            {{ f }}
        </table>
</body>
<html>
```

Let there be a following view in the app's **`views.py`** file which renders the **`forms.html`** template and sends the **`ApplicationForm`** object as a context.

```python
from .forms import ApplicationForm

def index(request):
    form = ApplicationForm()

    return render(request, 'form.html', {'form': form})
```

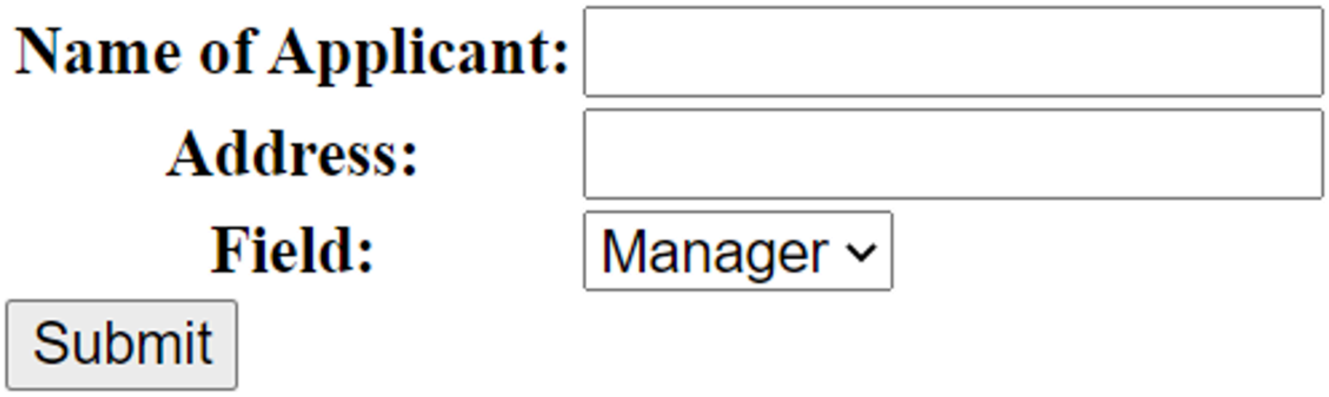
Inside the template, the form can be rendered in different ways. Instead of a tabular presentation of the form elements you can use the following variations:
`{{ form.as_table }}` will render them as table cells wrapped in**`<tr>`** tags. The form is rendered as a table by default.

```html
<table>
    <tr>
        <th><label for="id_name">Name of Applicant:</label></th>
        <td>
            <input type="text" name="name" maxlength="50" required id="id_name">
        </td>
    </tr>

    <tr>
        <th><label for="id_address">Address:</label></th>
        <td>
            <input type="text" name="address" maxlength="100" required id="id_address
        </td>
    </tr>

    <tr>
        <th><label for="id_field">Field:</label></th>
        <td>
            <select name="field" id="id_field">
                <option value="Manager">Manager</option>
                <option value="Cashier">Cashier</option>
                <option value="Operator">Operator</option>
            </select>
        </td>
    </tr>
</table>
<input type="submit">
```

Output:



`{{ form.as_p }}` will render them wrapped in**`<p>`** tags.

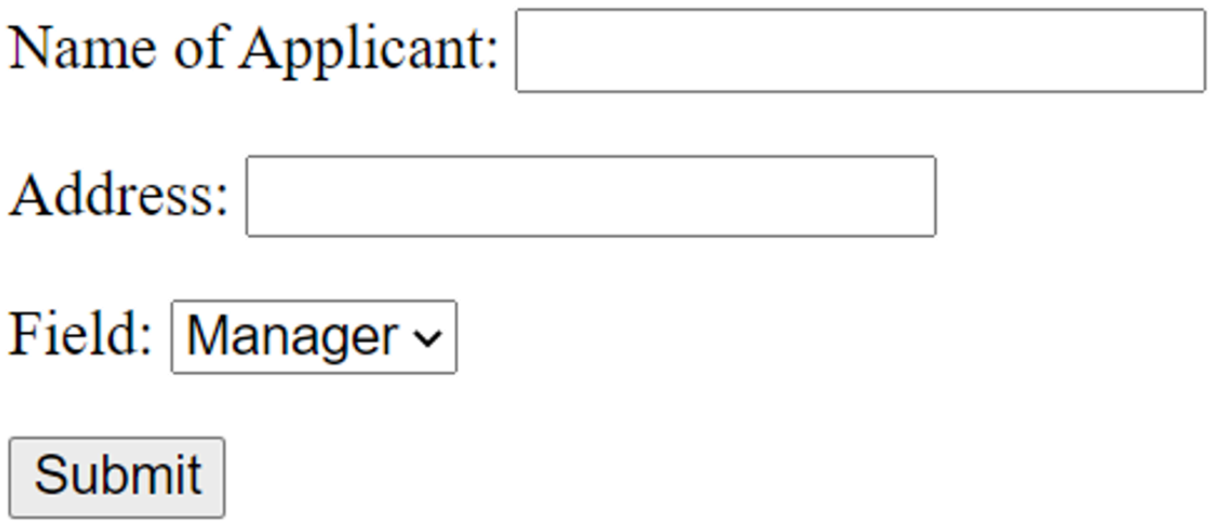
```html
<!-- HTML rendered in the browser -->

<p>
    <label for="id_name">Name of Applicant:</label>
    <input type="text" name="name" maxlength="50" required id="id_name">
</p>

<p>
    <label for="id_address">Address:</label>
    <input type="text" name="address" maxlength="100" required id="id_address">
</p>

<p>
    <label for="id_field">Field:</label>
    <select name="field" id="id_field">
        <option value="Manager">Manager</option>
        <option value="Cashier">Cashier</option>
        <option value="Operator">Operator</option>
    </select>
</p>
```

The form is displayed in the browser:



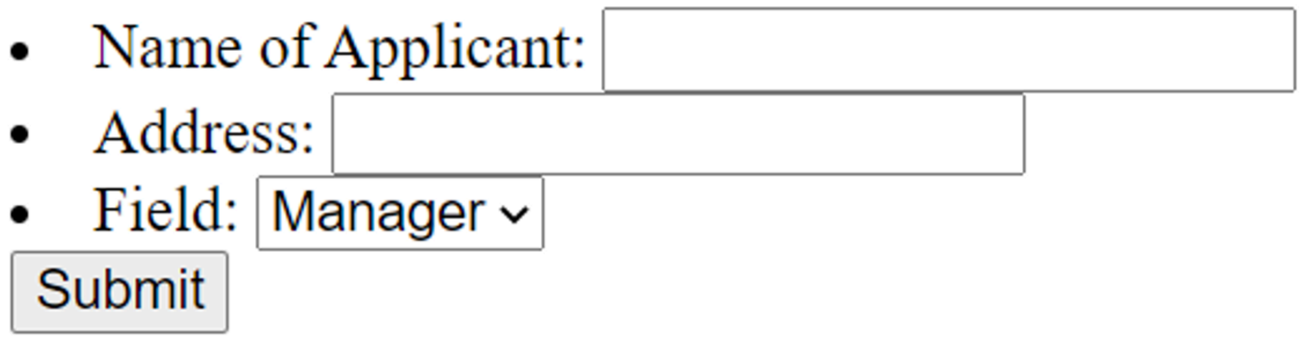
`{{ form.as_ul }}` will render them wrapped in **`<li>`** tags.

```html
<li>
    <label for="id_name">Name of Applicant:</label>
    <input type="text" name="name" maxlength="50" required id="id_name">
</li>

<li>
    <label for="id_address">Address:</label>
    <input type="text" name="address" maxlength="100" required id="id_address">
</li>

<li>
    <label for="id_field">Field:</label>
    <select name="field" id="id_field">
        <option value="Manager">Manager</option>
        <option value="Cashier">Cashier</option>
        <option value="Operator">Operator</option>
    </select>
</li>
<input type="submit">
```

The form is renders in the browser:



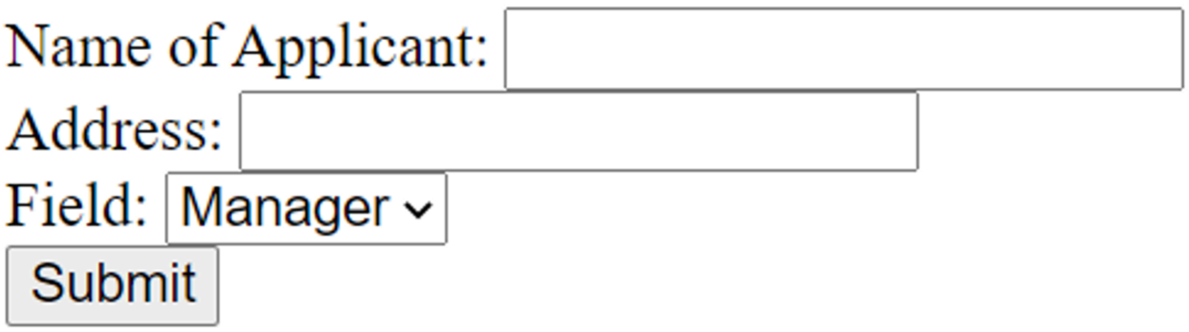
{{ form.as_div }} will render them wrapped in **<div>** tags.

```
<div>
    <label for="id_name">Name of Applicant:</label>
    <input type="text" name="name" maxlength="50" required id="id_name">
</div>

<div>
    <label for="id_address">Address:</label>
    <input type="text" name="address" maxlength="100" required id="id_address">
</div>

<div>
    <label for="id_field">Field:</label>
    <select name="field" id="id_field">
        <option value="Manager">Manager</option>
        <option value="Cashier">Cashier</option>
        <option value="Operator">Operator</option>
    </select>
</div>
<input type="submit">
```

The form is renders in the browser:



Reading Form Contents
Note that the form's action attribute is set to "**/form**" path. So, you'll provide a **form()** view mapped to this URL. This function fetches the data submitted by the user. It may be used to add a new row in the database table, or for any other processing.
Inside the view, populate the form object with the POST data and check it is valid. The Form class provides **is_valid()** method. It runs validation on each field and returns True if all field validations are passed.

```
from .forms import ApplicationForm

def index(request):

    if request.method == 'POST':
        form = ApplicationForm(request.POST)
        # check whether it's valid:
        if form.is_valid():
            # process the data
            # ...
            # ...
            return HttpResponse('Form successfully submitted')
```

Once the Form instance is validated, you can now access the data in individual field via the **cleaned_data** attribute. It ensures that the field contains the output in consistent form. In our example, the three values in the three input elements are parsed to Python variables like this:

```
name = form.cleaned_data['name']
```

```
add = form.cleaned_data['address']
post = form.cleaned_data['posts']
```

In this reading item on Forms API, you learned about Django Form and form fields. You also learned how the form is rendered and processed.