Django Forms

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#python-web

Web Forms

Web/HTML Form (1)

- It is an online page
  - Which allows users to **enter data**
  - The data is then **sent to a server** for processing
- It mimics a paper document where **users fill out** particular **fields**
  - It may contain text boxes, checkboxes, select options, a submit button, etc.

Web/HTML Form (2)

- In HTML, forms are enclosed in the **<form>** tag
- **GET** and **POST** are the only HTTP methods to use when dealing with forms

```
<body>

<form action="/your-name/" method="post">
    <-- input elements --/>
    <-- submit button --/>
</form>

</body>
```

Forms in Django

Django Forms Advantages (1)

- Django provides a range of **tools** and **libraries** to
  - Create forms using **python code**
  - Support all features of **HTML forms** in a **pythonic way**
  - **Simplify** and **automate** vast portions of the process

Django Forms Advantages (2)

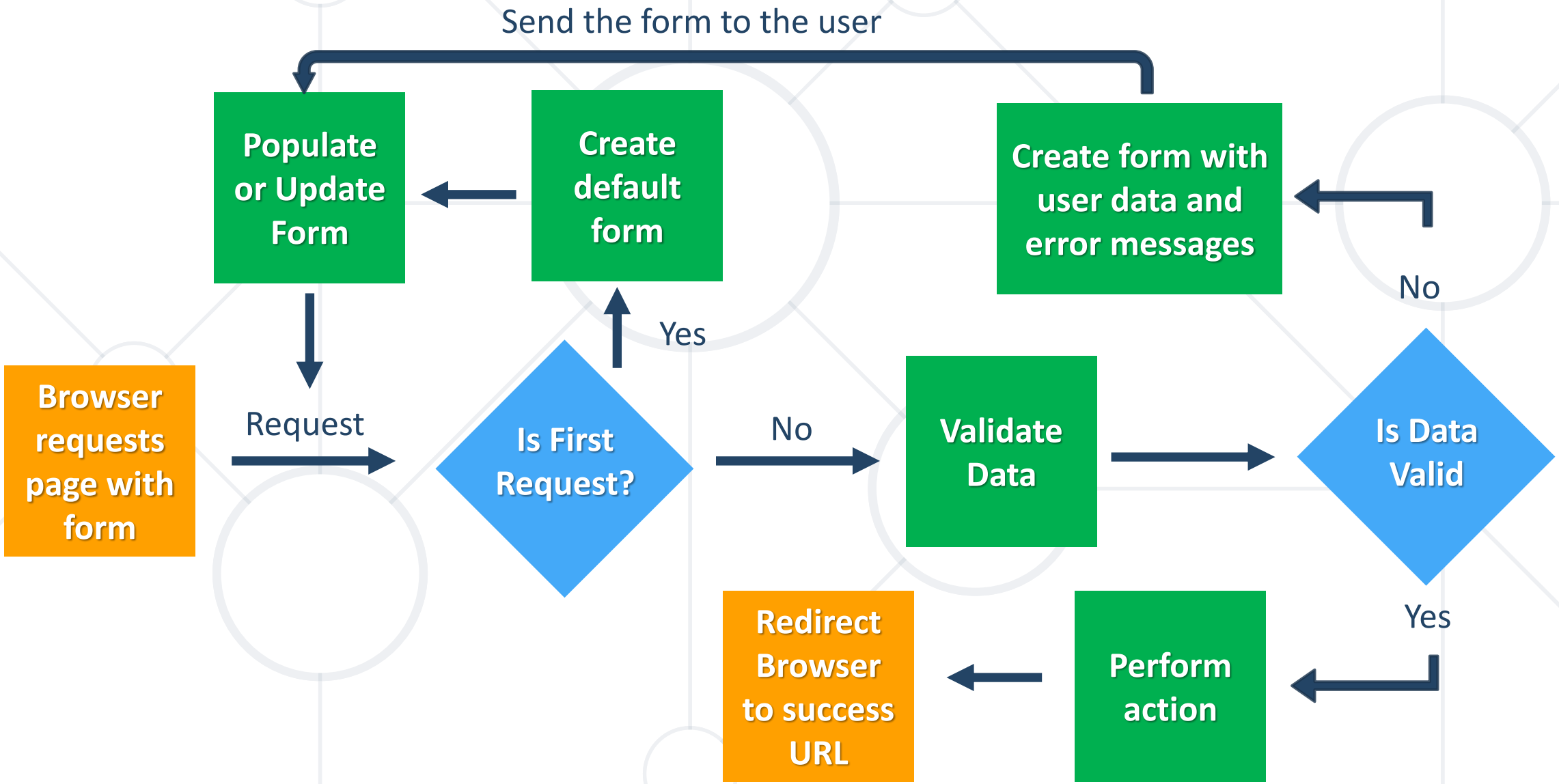
- E.g., the form fields map to HTML form **<input>** elements

```
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your Name', max_length=50)
```

```
<form action="/your-name/" method="post">
    <label for="your_name">Your name: </label>
    <input type="text" id="your_name" name="your_name"
           maxlength="50" required>
    <input type="submit" value="OK">
</form>
```

Django Forms Handling



```
class Form
```

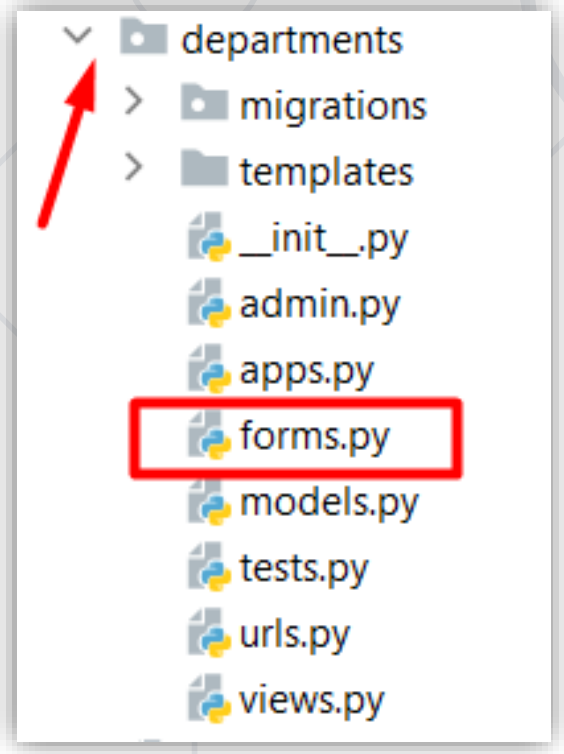
Django Form Class

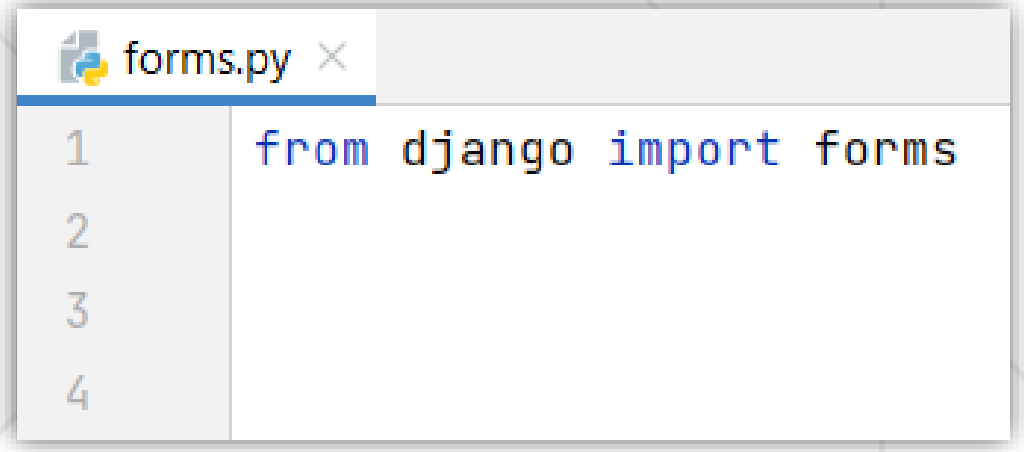
The Django Form Class

- The Django form
  - Describes the **form fields**
  - Determines how the form **works** and **appears**
  - Perform **validation** when the form is submitted

Create a Django Form (1)

- First, create a **forms.py file** in the app directory
- Import the **forms library**





Create a Django Form (2)

- To create a form:
  - **Inherit** from the **Form** class
  - **Add** the form **fields**

```
from django import forms

class NameForm(forms.Form):
    name = forms.CharField()
```

- Note: the Form and the **Model classes** share **most field types** and some **common arguments**

Create a Django Form (3)

- Create a **view** with a corresponding URL path

views.py

```
from .forms import NameForm

def add_new_name(request):
    if request.method == "GET":
        form = NameForm()

    if request.method == "POST":
        form = NameForm(request.POST)
        if form.is_valid():
            #  do something with the data
            #  redirect to the desired page

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

Generate an empty form

Binds the collected data to the form

Check if the data is valid

Return an empty form or invalid data with errors

Flat is Better than Nested

- Most of the time, you can use **one Form instantiation**

views.py

```
from .forms import NameForm

def add_new_name(request):
    form = NameForm(request.POST or None)
    if form.is_valid():
        #  do something with the data
        #  redirect to the desired page

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

If no post request, generate an empty form

Create a Django Form (4)

- Create a **template** with the form

index.html

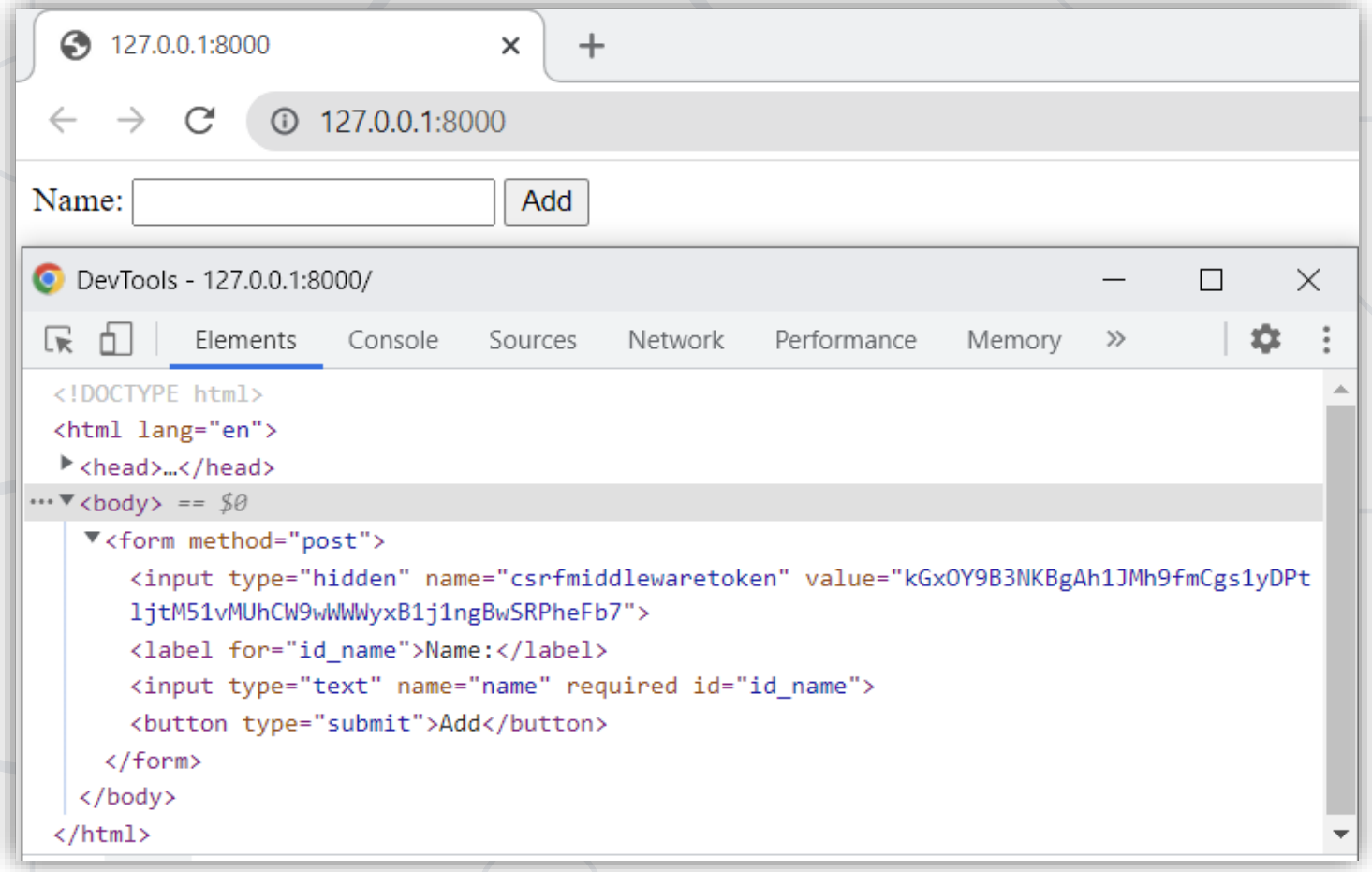
```
<body>
  <form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Add</button>
  </form>
</body>
```

The Django form

Cross-Site Request Forgery secure random token

Create a Django Form (5)

- Start the development server



Django Form Fields

Django Form Fields

- The most important part is defining the fields of the form
- Each field has **custom validation logic**
- Each field takes some **common arguments**
- Some fields take **field-specific arguments**

Full list of form fields: https://docs.djangoproject.com/en/4.1/ref/forms/fields/#built-in-field-classes

Form Field Arguments (1)

- By default, each Field class assumes the **value is required**
  - If you pass an **empty value**, it will raise a **ValidationError**
- You can **specify** that a field is **not required**

```
first_name = forms.CharField(required=False)
```

Form Field Arguments (2)

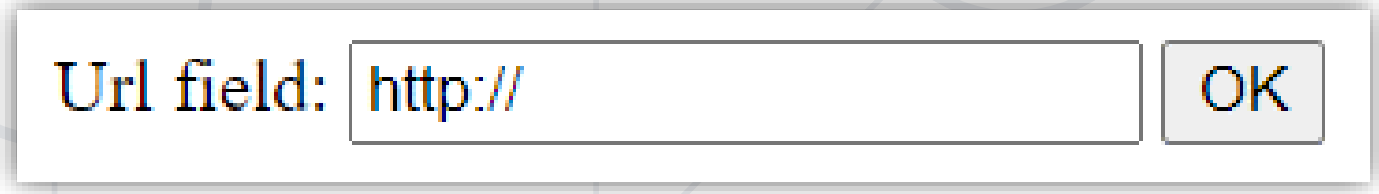
- You can specify a **"human-friendly" label** for a field
- It is used when the Field is **displayed** in a form

```
first_name = forms.CharField(label="Add First Name")
```

Form Field Arguments (3)

- You can display an "empty" form in which a field is **initialized** to a **particular value**
- The data will be **passed into the view** when submitted

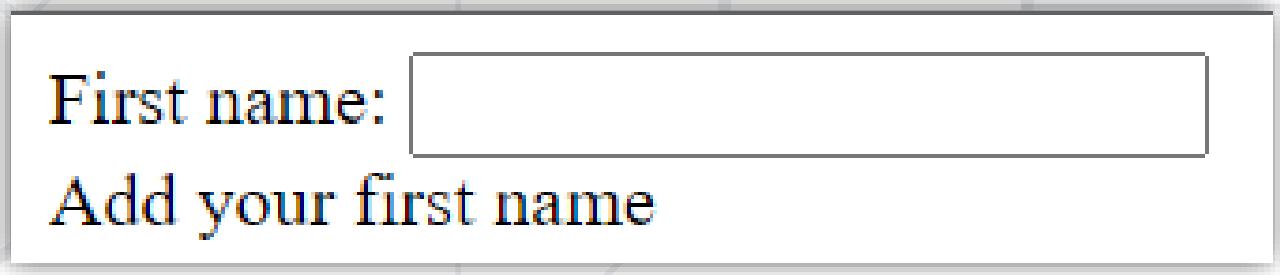
```
url_field = forms.URLField(initial='http://')
```



Form Field Arguments (4)

- You can specify a **help text** for a field
- It will be **displayed next to the field**

```
first_name = forms.CharField(help_text='Add your first name')
```



More Field Arguments: https://docs.djangoproject.com/en/4.1/ref/forms/fields/#core-field-arguments

Built-in Widgets

Django Widget

- Widgets handle:
  - **Rendering** of HTML form input elements
  - **Extraction** of raw submitted data
- Widgets are **assigned to form fields**
  - Each form field has a **corresponding** widget
  - To use a **different widget** for a field, add it as an argument

Built-in Widgets (1)

- **`CharField`** uses **`TextInput`** widget by **default**
  - Renders as: **`<input type="text" ...>`**

Comment:

- You can specify a form that uses a **larger `Textarea` widget**

```
comment = forms.CharField(
            widget=forms.Textarea
      )
```

Comment:

Built-in Widgets (2)

- **NumberInput**
  - HTML input type: "number"
- **EmailInput**
  - HTML input type: "email"
- **PasswordInput**
  - HTML input type: "password"

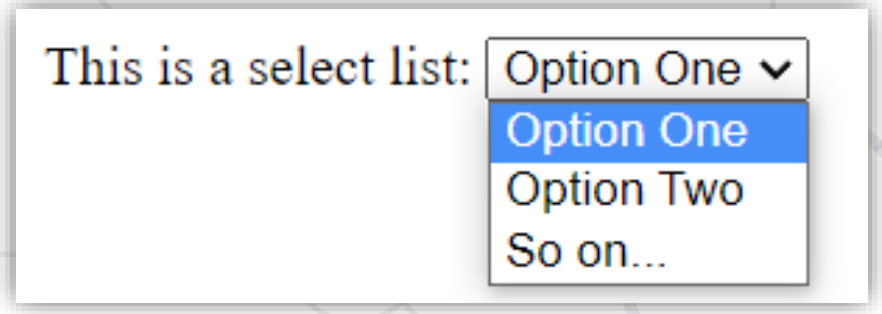
Built-in Widgets (3)

- **URLInput**
  - HTML input type: "url"
- **DateInput**
  - HTML input type: "text"
- **DateTimeInput**
  - HTML input type: "text"

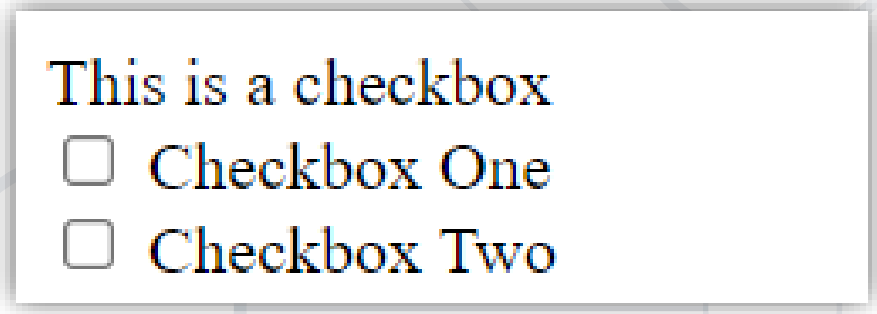
More built-in widgets: https://docs.djangoproject.com/en/4.1/ref/forms/widgets/#built-in-widgets

Select, Checkbox and Radio Button (1)

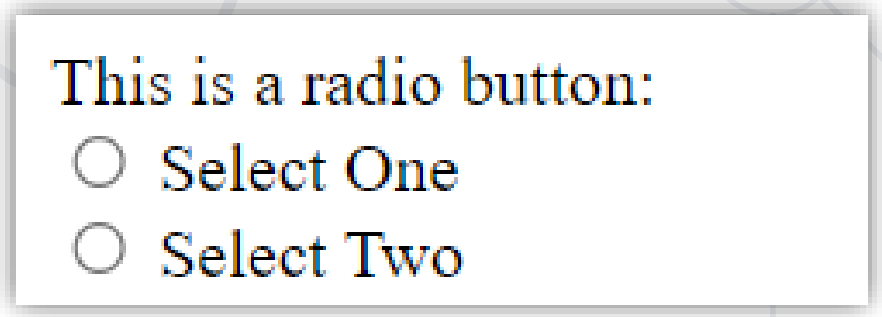
- A **select list** allows you to choose options from a drop-down menu



- A **checkbox** allows you to select options from a list of options



- A **radio button** allows you to select only one option from a list of options



Select List Widget

- **Select** is the default widget for **ChoiceField**
- But can be **used in other fields**

```
class SelectOptionForm(forms.Form):
    CHOICES = (
        ('1', 'Option One'),
        ('2', 'Option Two'),
    )

    choice_field = forms.ChoiceField(choices=CHOICES)
    char_field = forms.CharField(widget=forms.Select(choices=CHOICES))
```

The same as in the model field

Checkbox Widget

- **`CheckboxInput`** is the default widget for **BooleanField**
- Returns **True**, if it is checked

```
class CheckboxForm(forms.Form):
    checkbox_field = forms.BooleanField(required=False)
```

- Note: to create a checkbox that can be **either checked or unchecked**, set the attribute **`required`** to **`False`**

Radio Button Widget

- `RadioSelect` is similar to the Django **Select** widget
  - It can be **used on a ChoiceField** instead of Select

```
class RadioButtonForm(forms.Form):
    CHOICES = (...)

    choices_field = forms.ChoiceField(
        choices=CHOICES,
        widget=forms.RadioSelect(),
    )

    char_field = forms.CharField(
        widget=forms.RadioSelect(choices=CHOICES),
    )
```

Django Widget Attributes

- Widgets give you the opportunity to **set HTML attributes** using Python code

```
comment = forms.CharField(
    widget=forms.Textarea(
        attrs={'cols': 80, 'rows': 20,
               'class': 'special',
               'title': 'Add a comment'}))
```

- Note: It is not always a good idea to **narrow the distinction** between the main code logic and the front end

Django ModelForm Class

The ModelForm Class

- In a case of a **database-driven** app, the forms might **overlap** with the models
  - The field types are already defined in the **model**
  - Using the **`ModelForm`** help to avoid duplicating your model description
- This way, the form is automatically created based on a particular model

Form vs ModelForm

- **Form**:
  - Independent of a model
  - Does not directly interact with models
  - E.g., search form, contact form, subscription form
- **ModelForm**:
  - Convert a model into a form
  - Directly add or edit a model
  - E.g., registration form, newsletter article form, blog post form

Create a Django Model Form (1)

- First, create a **model** with fields

models.py

```
from django.db import models

class Name(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models. CharField(max_length=50)
```

Create a Django Model Form (2)

- To create a form:
  - **Inherit** from the **ModelForm** class
  - Specify the **model** you want to create a form for
  - **Add** the created **model fields**

forms.py

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        fields = '__all__'
```

Create a Django Model Form (3)

- Create a **view** with a corresponding URL path

views.py

```
from .forms import NameForm

def add_new_name(request):
    if request.method == "GET":
        form = NameForm()

    if request.method == "POST":
        form = NameForm(request.POST)
        if form.is_valid():
            form.save()
            #  redirect to the desired page

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

Create a Django Model Form (4)

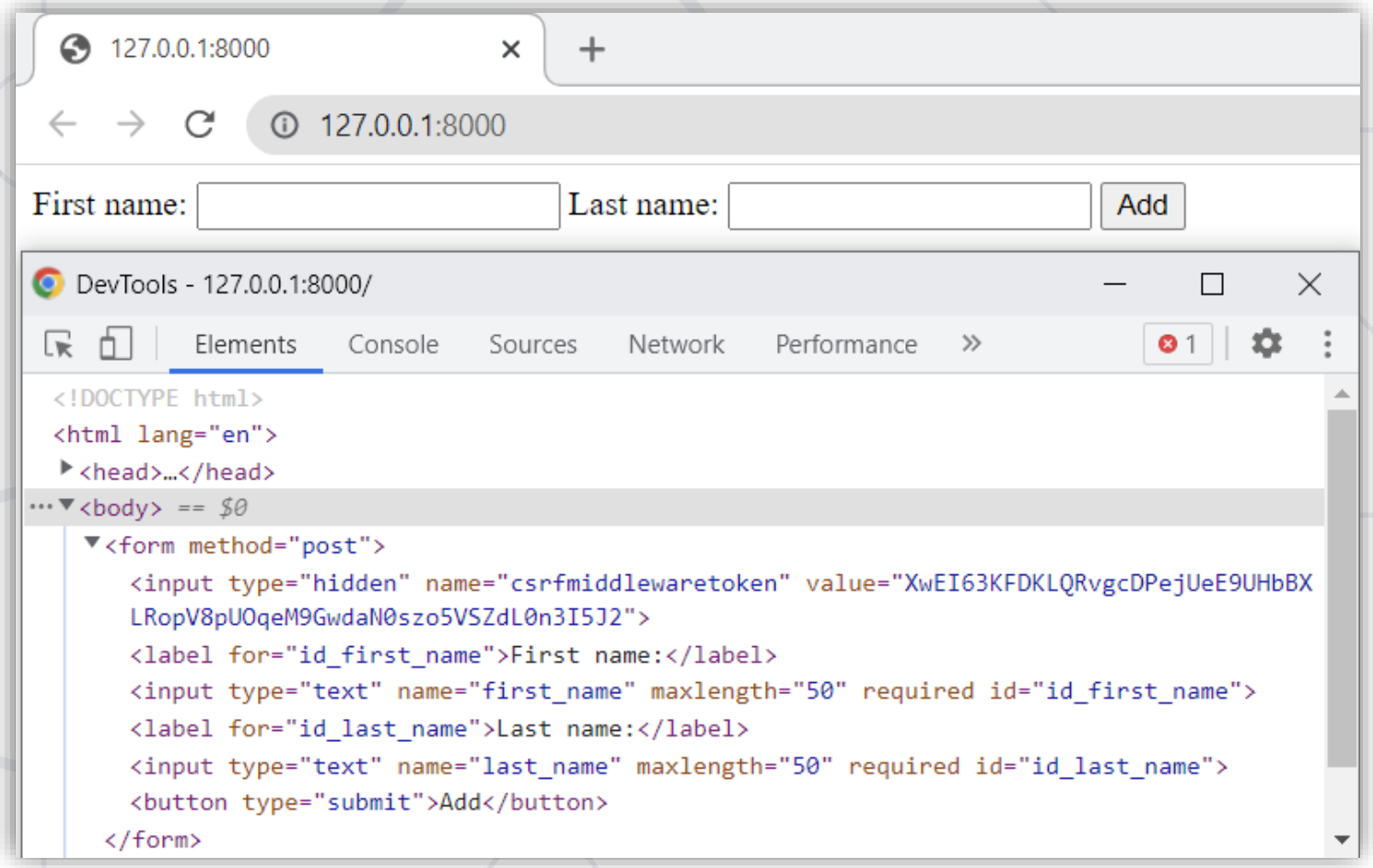
- Create a **template** with the form

index.html

```
<body>
  <form method="post">
    {% csrf_token %}
    {{ form }}
    <button type="submit">Add</button>
  </form>
</body>
```

Create a Django Model Form (5)

- Start the development server



Update a Model using Form (1)

- Create an update **view** with a corresponding URL path

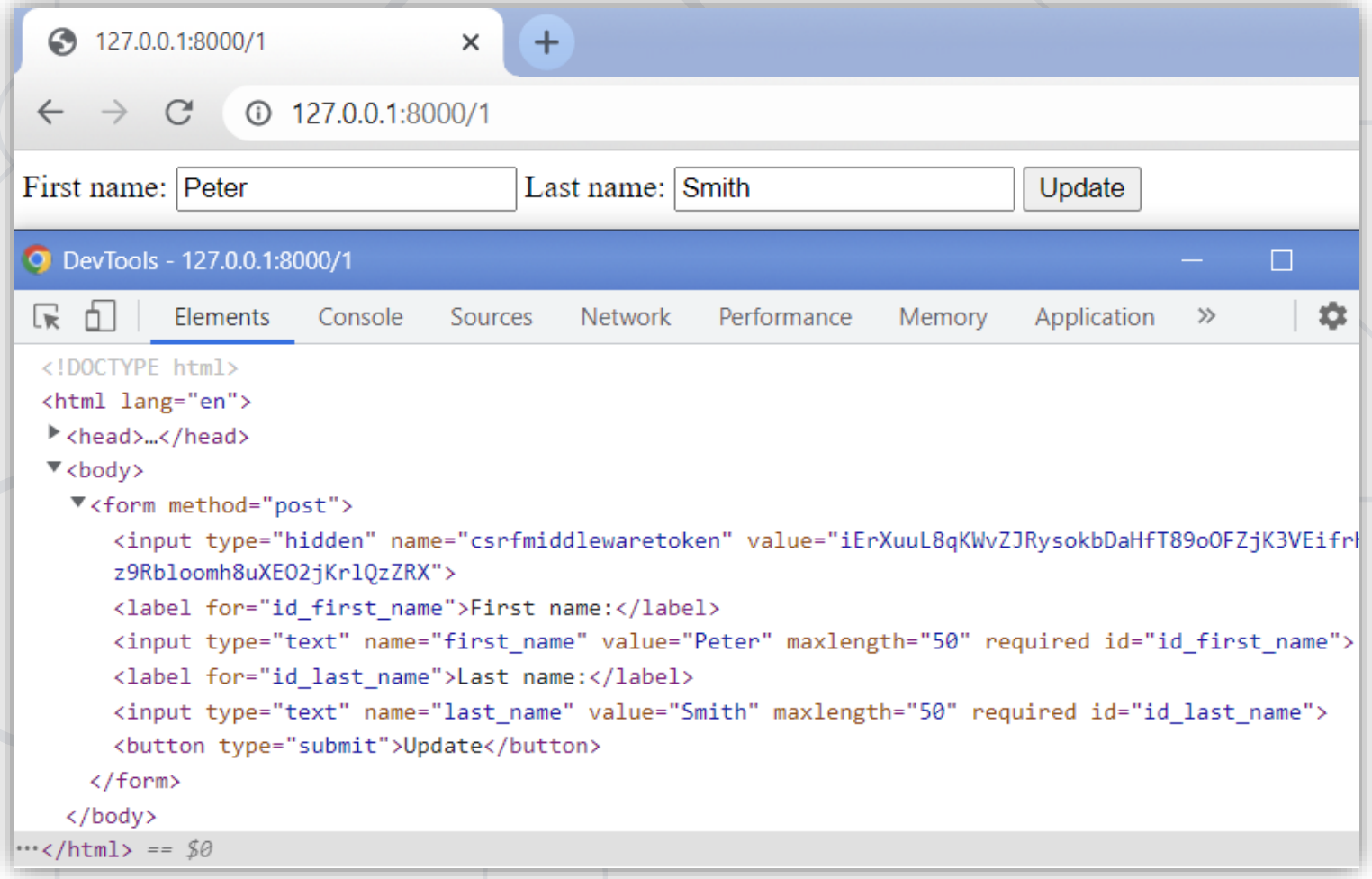
views.py

```
from .forms import NameForm
from .models import Name

def update_name(request, pk):
    name = Name.objects.get_object_or_404(pk=pk)
    form = NameForm(request.POST or None, instance=name)
    if form.is_valid():
        form.save()
        # redirect to the desired page
    return render(request, 'update.html', {'form': form})
```

Update a Model using Form (2)

- Start the development server



ModelForm Options

Class Meta

- Like the **Model**, a **ModelForm** class has an **inner `Meta` class** with predefined options
- The **Meta options** indicate how the form works and appears
- A **full list of the options** can be found in the Django class **`ModelFormOptions`**, in its **`__init__()`** method

Model Option

- Each time you configure **`ModelForm`**, you must indicate **which model to use** to generate the form
- Its value should be set to the Model class (not instance)

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
```

Fields Option

- You should **set the fields** that will be edited in the form
  - Failure to do so can easily lead to security problems
- Set to **`__all__`** to use all model fields

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        fields = ['first_name', 'last_name']
```

Exclude Option

- Sometimes, it is easier to set which **fields to be excluded** from the form

```
from django import forms
from .models import Name

class NameForm(forms.ModelForm):
    class Meta:
        model = Name
        exclude = ['last_name']
```

ModelForm Field Types

- Each model field has a corresponding default form field

Model Field	Form Field
CharField	CharField with max_length set
IntegerField	IntegerField
FloatField	FloatField
BooleanField	BooleanField, or NullBooleanField if null=True
ForeignKey	ModelChoiceField
ManyToManyField	ModelMultipleChoiceField

Full table: https://docs.djangoproject.com/en/4.1/topics/forms/modelforms/#field-types

Overriding the Default Fields (1)

- You have the flexibility of **changing the field type** for the model
- Use the **`widgets`** option
  - A dictionary mapping **field names** to **widget classes/instances**

```
class CommentForm(forms.ModelForm):
    class Meta:
        ...
        widgets = {
            'comment': Textarea(),
        }
```

Overriding the Default Fields (2)

- You can **specify a different label** for a field
- Use the **`labels`** option
  - A dictionary mapping **field names** to **strings**

```
class NameForm(forms.ModelForm):
    class Meta:
        ...
        labels = {
            'first_name': 'Add Your First Name',
        }
```

Overriding the Default Fields (3)

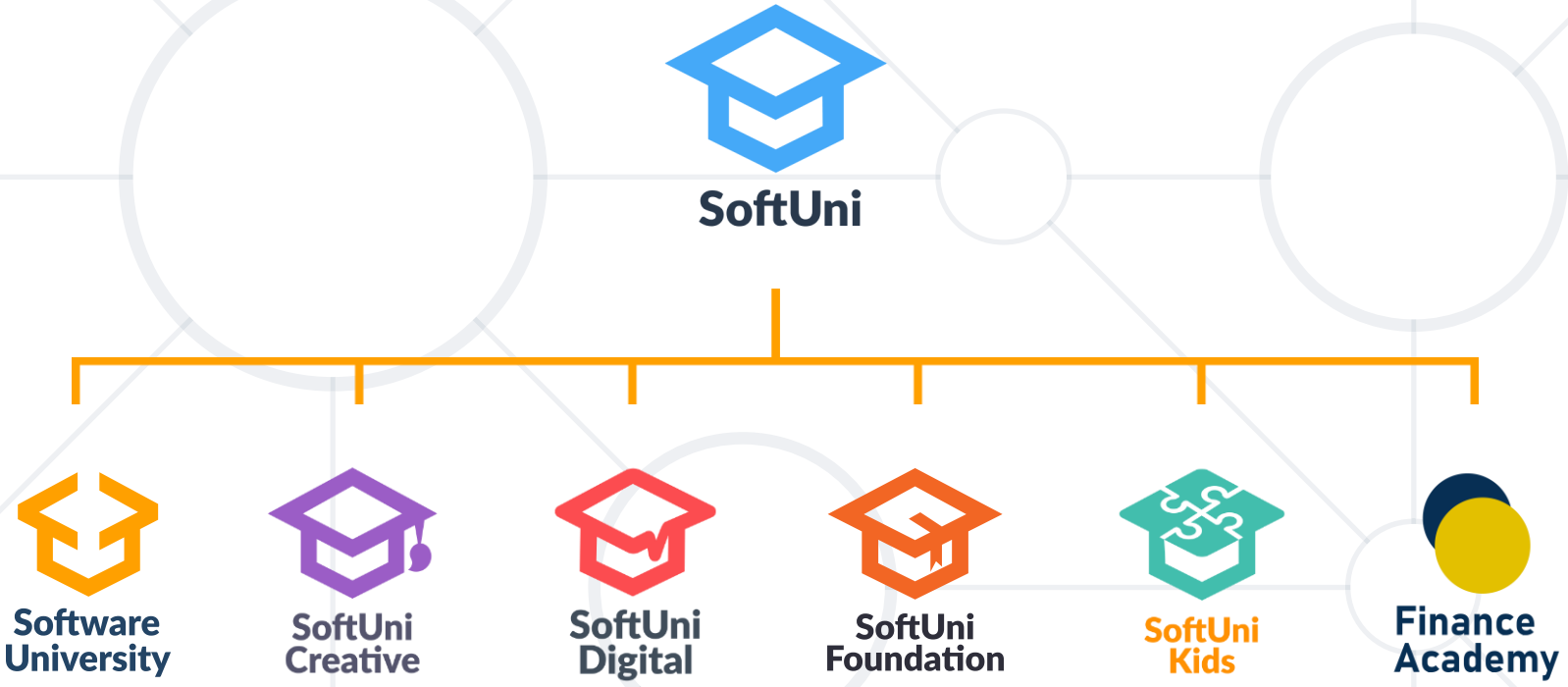
- You can **add a help text** for a field
- Use the **`help_texts`** option
  - A dictionary mapping **field names** to **strings**

```
class EmailForm(forms.ModelForm):
    class Meta:
        ...
        help_texts = {
            'email': 'You must supply a valid email address',
        }
```

Summary

- Use forms when building apps that **accept input** from their visitors
- Each form field has a corresponding **Widget**
- When we want to **skip** defining the **field types** of our **forms**, we use **ModelForm**

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

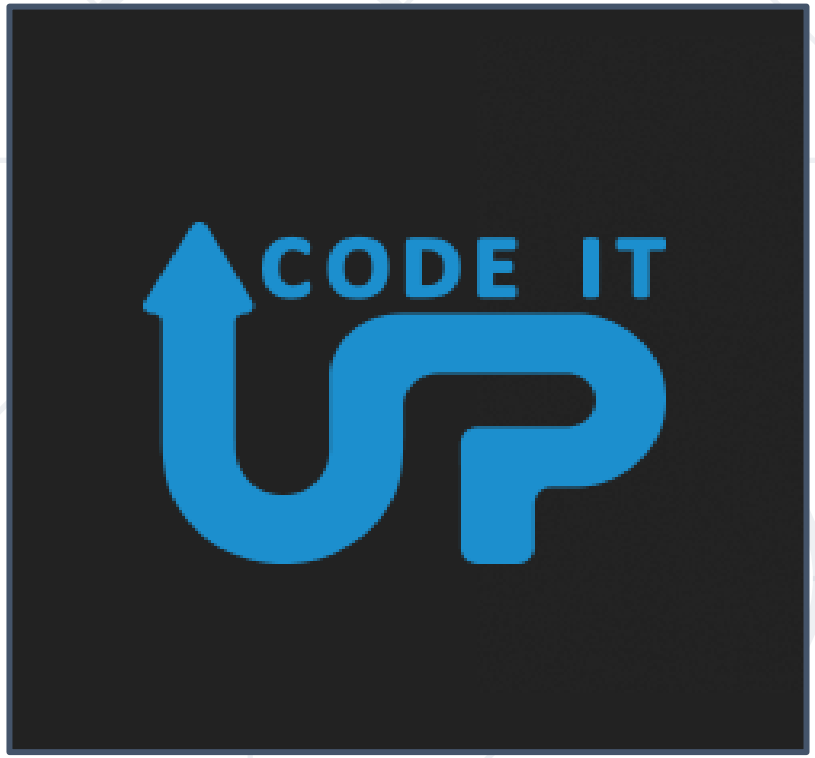
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg