

# Django Templates & Forms

@pvavilin

7 мая 2022 г.

# Outline

# HTML

```
<html>
  <head>
    <title>Мир! Труд! Май!</title>
  </head>
  <body>
    <center
      onclick="alert('привет!')"
      style="color:red;font-size:40px"
    >
      <!-- тут нужно вставить своё имя -->
      <i>Привет, &lt;ИМЯ&gt;!</i>
    </center>
  </body>
</html>
```

# Django templates

```
<html>
<head>
  <title>Мир! Труд! Май!</title>
</head>
<body>
  <center
    onclick="alert('привет!')"
    style="color:red;font-size:40px"
  >
    <i>Привет, {{ name }}!</i>
  </center>
</body>
</html>
```

# наследование

- extends** взять за основу *расширяемый шаблон* и *переопределить* в нём нужные блоки. Остальное содержимое шаблона останется прежним.
- include** вставить на место `{% include ... %}` содержимое подключаемого шаблона.

# передача переменных

- В шаблоне доступны объекты из middleware, например user
- Нужные вам переменные вы передаёте из вьюхи во время рендеринга

## Дополнительная литература

- <https://docs.djangoproject.com/en/2.2/ref/templates/builtins/>
- <https://docs.djangoproject.com/en/2.2/topics/templates/>
- <https://habr.com/ru/post/23132/?ysclid=l2hw076rb1>

# HTML формы

**HTML-формы** это просто текст. И вообще-то мы можем писать его вручную или, например, с использованием f-string в Python.

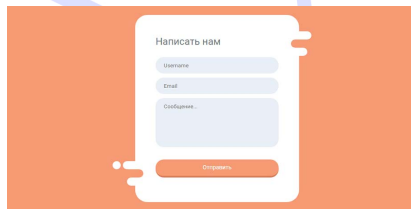
```
<form action="." method="post">
  <label for="message">Послание</label>
  <input type="text" name="message"
        value="сообщение"
        maxlength="100" />
  <input type="submit"
        value="Отправить" />
</form>
```

Послание



# HTML формы

Формы могут быть более красивыми



Написать нам

Username

Email

Сообщение...

Отправить

# HTML формы

И очень разнообразными. Но писать такие разнообразные формы вручную это очень утомительно. На помощь нам приходят **Django Forms**.

Name

Email

Password

Gender ☒ Male ☐ Female

Birthdate

Contacts

Skype  [Delete](#)

Other   [Delete](#)

or

Photo  Файл не выбран

Hobbies

Movies

Music

Cooking

Photography

Some words

☐ Agreement

# Django формы

```
from django import forms

class MessageForm(forms.Form):
    message = forms.CharField(
        label='Послание',
        max_length=100
    )

form = MessageForm(
    initial={'message': 'сообщение'}
)
```

*initial* заполнит форму какими-то данными. При рендеринге это будет значение атрибута *value* в HTML. Обычно мы передаём туда *request.POST*

# Django формы

```
<tr>
  <th>
    <label for="id_message">
      Послание:
    </label>
  </th>
  <td>
    <input type="text"
      name="message"
      value="сообщение"
      maxlength="100"
      required
      id="id_message" />
  </td>
</tr>
```

# Django формы

## Form Fields

Поля формы в Django описываются классами `Field`, каждый из которых имеет своё представление в виде `Widget`-а.

### Built-in **Field** classes

Naturally, the **forms** library comes with a set of **Field** classes that represent common validation needs. This section documents each built-in field.

For each field, we describe the default widget used if you don't specify **widget**. We also specify the value returned when you provide an empty value (see the section on **required** above to understand what that means).

# Bound / Unbound forms

Формы в *Django* **могут быть в двух состояниях**

`unbound` — форма пустая

`bound` — форма заполнена данными

# Unbound

Форма не связана ни с какими данными

```
|form = MessageForm()  
|form.is_bound      # -> False
```

# Bound

Форма *частично* или *полностью* заполнена

```
# обычно мы передаём request.POST
form = MessageForm({
    'message': 'foobar'
})
form.is_bound      # -> True
```



# Валидация форм

## Документация

```
form.is_valid()    # -> True / False
# в случае когда is_valid -> True,
# тогда у формы появляется атрибут
# cleaned_data, который содержит
# словарь со значениями полей
form.cleaned_data['field_name']
# если is_valid -> False
# то заполняется переменная
form.errors
```

# Валидаторы

Пример написания своего валидатора

```
from django.core.exceptions import (
    ValidationError

def validate_even(value):
    if value % 2 != 0:
        raise ValidationError(
            '%(value) нечётно',
            params={'value': value}
        )
```

# Валидаторы

```
from django import forms

class EvenNumbersForm(forms.Form):
    number = forms.IntegerField(
        validators=[validate_even]
    )
```

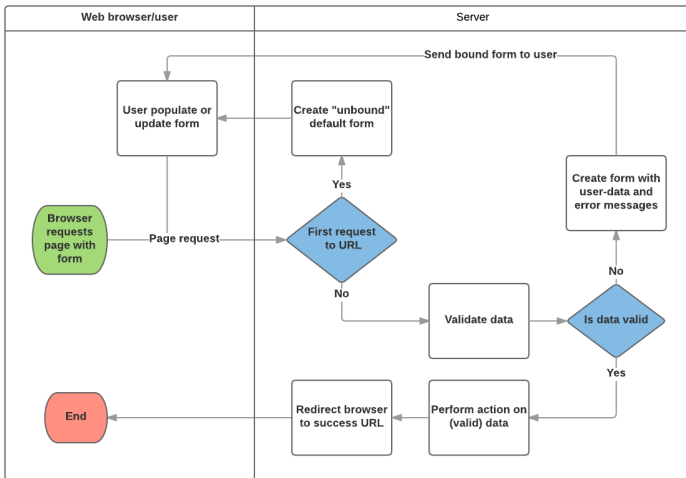
*validators* добавит валидаторы к уже существующему базовому валидатору *IntegerField*

# Валидаторы

Готовых валидаторов очень много!

```
◦ Built-in validators
  ▪ RegexValidator
  ▪ EmailValidator
  ▪ URLValidator
  ▪ validate\_email
  ▪ validate\_slug
  ▪ validate\_unicode\_slug
  ▪ validate\_ipv4\_address
  ▪ validate\_ipv6\_address
  ▪ validate\_ipv46\_address
  ▪ validate\_comma\_separated\_integer\_list
  ▪ int\_list\_validator
  ▪ MaxValueValidator
  ▪ MinValueValidator
  ▪ MaxLengthValidator
  ▪ MinLengthValidator
  ▪ DecimalValidator
  ▪ FileExtensionValidator
  ▪ validate\_image\_file\_extension
  ▪ ProhibitNullCharactersValidator
```

# Forms Workflow



# Рендеринг форм вручную

## ■ Статья

## ■ оф. документация

```
{{ form.non_field_errors }}  
<div class="fieldWrapper">  
    {{ form.subject.errors }}  
    <label  
        for="{{ form.subject.id_for_label }}"  
    >  
        Email subject:  
    </label>  
    {{ form.subject }}  
</div>
```

# Безопасность

Настоятельно рекомендую ознакомиться с этой документацией

`https://docs.djangoproject.com/en/3.2/topics/security/`

# CSRF

На сайте может быть обычная кнопка, предлагающая вам посмотреть фотографии.

View my pictures!



# CSRF

А на самом деле там будет отправляться форма перевода денег с вашего аккаунта на аккаунт злоумышленника.

```
<form
  action="bank.com/transfer.do"
  method="POST">
  <input type="hidden"
    name="acct" value="воришка" />
  <input type="hidden"
    name="amount" value="$1kk" />
  <input type="submit"
    value="View my pictures!" />
</form>
```

# CSRF

Но если на стороне банка используются csrf-токены в формах, то ничего страшного не случится. Запрос злоумышленника не может содержать нужное значение (случайное в рамках сессии) csrf-токена.

```
<form method="post"  
        action="{% url signup %}"  
        {% csrf_token %}  
</form>
```

```
<form method="post" action="/auth/signup/" >  
  <input type="hidden" name="csrfmiddlewaretoken"  
        value="72o06FKz7VBiTjui34o0HIE0s2l8IQp2UT5WTnLd4QW0uqqmCp0YFvW9M9J3SrRj" >
```

# ModelForm

Прекрасная документация

# ModelForm

У ModelForm появляется метод **.save()**

```
class NameForm(models.ModelForm):  
    class Meta:  
        model = Name  
  
form = NameForm(request.POST)  
# сохранить запись в базу данных  
form.save()
```

# ModelForm

```
save(commit=False)
```

```
class NameForm(models.ModelForm):  
    class Meta:  
        model = Name  
  
form = NameForm(request.POST)  
# создаёт объект модели Name  
# но не записываем его в базу  
model = form.save(commit=False)
```

# ModelForm

```
class YaForm(models.ModelForm):  
    class Meta:  
        # содержит поля X, Y, Z  
        model = YaModel  
        fields = ['X', 'Y']  
  
form = YaForm(request.POST)  
# не передаст в модель Z,  
# а значит в базу запишется  
# пустое значение поля Z  
form.save()
```

# ModelForm

Один из вариантов решения — определить модель заранее

```
model = YaModel(Z='foobar')  
form = YaForm(  
    request.POST,  
    instance=model  
)  
# форма будет содержать все  
# поля заполненными  
form.save()
```

# ModelForm

Или использовать *commit=False* чтобы доопределить модель перед записью в БД.

```
form = YaForm(request.POST)
model = form.save(commit=False)
model.z = 'foobar'
model.save()
```



# ModelForm

Допустим, мы определили модель

```
class Article(models.Model):  
    headline = models.CharField(  
        max_length=200,  
        null=True,  
        blank=True,  
    )  
    content = models.TextField()
```

# ModelForm

Если поле не перечислено в *fields* или добавлено в *excludes* в Meta-классе, то это поле будет исключено из данных передаваемых в модель.

```
class ArticleForm(ModelForm):  
    slug = CharField(  
        validators=[validate_slug]  
    )  
  
    class Meta:  
        model = Article  
        # slug не попадёт в save()  
        fields = ['headline', 'content']
```

# Widgets

Виджеты это то как формы будут представлены на web-страницы, то есть виджеты отвечают за генерацию HTML-кода для полей форм.

Документация

```
• Built-in widgets
  • Widgets handling input of text
    • TextInput
    • NumberInput
    • EmailInput
    • URLInput
    • PasswordInput
    • HiddenInput
    • DateInput
    • DateTimeInput
    • TimeInput
    • Textarea
  • Selector and checkbox widgets
    • CheckboxInput
    • Select
    • MultipleChoiceSelect
    • SelectMultiple
    • RadioSelect
    • CheckboxesSelectMultiple
  • File upload widgets
    • FileInput
    • ClearableFileInput
  • Composite widgets
    • MultiValueHiddenInput
    • SplitDateTimeWidget
    • SplitMultiDateFieldWidget
    • SelectDateWidget
```

# Widgets

Можно добавлять стили и другие атрибуты виджетам

```
class CommentForm(forms.Form):  
    name = forms.CharField(  
        widget=forms.TextInput(  
            attrs={'class': 'special'}  
        )  
    )  
    url = forms.URLField()  
    comment = forms.CharField(  
        widget=forms.TextInput(  
            attrs={'size': '40'}  
        )  
    )
```

# Widgets

```
class CommentForm(ModelForm):  
    class Meta:  
        model = Comment  
        fields = (  
            'name', 'url', 'comment'  
        )  
        widgets = {  
            'name': forms.TextInput(  
                attrs={'class': 'special', 'rows': 10}  
            ),  
            'comment': forms.TextInput(  
                attrs={'size': '40'}  
            )  
        }  
    }
```

# Widgets

```
<input type="text" name="name"  
       class="special" required>
```

```
<input type="url" name="url"  
       required>
```

```
<input type="text" name="comment"  
       size="40" required>
```

# Вопросы-ответы

