STEP ACADEMY

УРОК №9

Django Модели и валидаторы

Ключевые темы

- Абсолютный адрес записи
- Валидаторы
- Валидация полей
- Валидация моделей

Абсолютный адрес

get_absolute_url() возвращает URL, привязанный к записи в БД.

К примеру, в приложении имеется БД с таблицей, в которой каждая запись содержит информацию об авторах книг. Требуется сделать так, чтобы для каждого автора существовала html-страница с информацией о нём.

Обычно, для перехода на другие страницы сайта используются теги **<a>**, у которых в атрибут **href** вставляют URL для перехода. С помощью **DTL** можно использовать функцию **get_absolute_url()** для передачи ссылки на нужную запись.

Абсолютный адрес

Метод **get_absolute_url()** используется для получения канонического URL-адреса объекта модели. Этот метод позволяет получить URL-адрес конкретного объекта модели, чтобы использовать его в представлениях, шаблонах или других местах, где требуется ссылка на этот объект.

```
def get_absolute_url(self):
    return f"/authors/{self.id}"
```

Оформление в шаблоне

Стартовым шаблоном для примера будет служить страница с перечислением авторов книг, где каждая строка будет заключена в тег **<a>** для перехода на страницу, связанной с определённым автором в БД.

```
{% for author in authors %}
    <div>
        <a href="{{ author.get_absolute_url }}">
            {{ author }}
        </a>
    </div>
{% endfor %}
```

Адрес записи

В **urls.py** нужно построить маршрут для функции, которая будет отвечать за вывод индивидуальной для автора страницы:

```
path('authors/<int:author>', show_author),
```

Функции-представления могут принимать параметры, через которые могут передаваться различные данные. Подобные параметры передаются в адресе URL. Пример запроса на информацию об авторе (ориентируемся на **primary_key** в модели):

```
"http://127.0.0.1:8000/authors/4"
```

Параметры представлений

```
"http://127.0.0.1:8000/authors/4"
```

127.0.0.1:8000/ – базовый адрес сайта

/authors/ – сегмент сайта, отвечающий за вывод авторов

/4 – параметр представления (**id** автора в базе данных)

Последний сегмент может представлять параметры URL, которые могут быть связанны с параметрами **view-функций** через систему маршрутизации. Подобные параметры еще можно назвать параметрами маршрута.

Параметры представлений

Параметры заключаются в угловые скобки в формате **<спецификатор:название_параметра>**.

Например, здесь параметр **author_name** имеет спецификатор **str**:

```
path('authors/<str:author_name>', show_author)
```

Количество и название параметров в шаблонах адресов URL соответствуют количеству и названиям параметров соответствующих функций, которые обрабатывают запросы по данным адресам.

Параметры представлений

Самые распространённые спецификаторы:

- **str**: соответствует любой строке за исключеченем символа /. Если спецификатор не указан, то используется по умолчанию;
- **int**: соответствует любому положительному числу;
- **slug**: соответствует последовательности буквенных символов ASCII, цифр, дефиса и символа подчеркивания, например, building-your-1st-django-site;
- **uuid**: соответствует идентификатору UUID, например, 075194d3-6885-417e-a8a8-6c931e272f00.

Организация view

View-функции помимо **request** способны принимать также и передаваемые в URL параметры:

```
def show_author(request, author):
    return render(request, "author_info.html",
                  context={"author": Author.objects.get(id=author)})
```

В данном случае в качестве передаваемого параметра идёт значение **id**, которое сокрыто в переданном параметре **author**. Через него происходит поиск конкретного автора и вывод информации о нём через **context**.

Переопределяемые методы

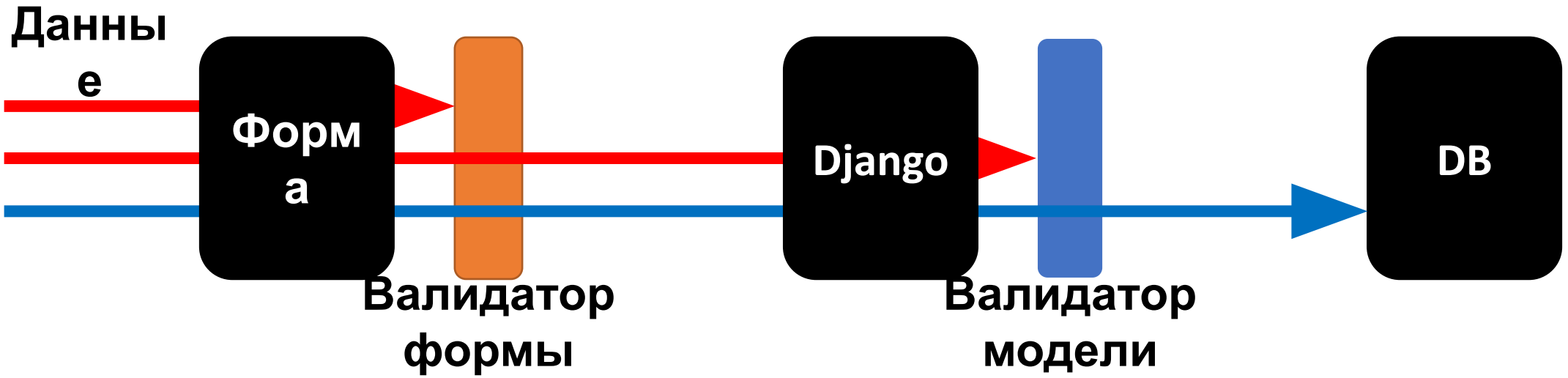
Помимо атрибутов класса, представляющих поля модели, и вложенного класса Meta, где объявляются параметры модели, в классе модели можно объявить (или переопределить) методы:

- **__str__(self)**
- **save (self, *args, **kwargs)**
- **delete (seif, *args, **kwargs)**

Валидация

Валидацией называется проверка на корректность данных, занесенных в поля модели.

Валидацию можно реализовать непосредственно в модели или же в форме, которая используется для занесения в нее данных



Валидаторы

Валидацию значений, заносимых в отдельные поля модели, выполняют **валидаторы**, реализованные в виде функций или классов.

Некоторые типы полей уже используют определенные валидаторы. Так, строковое поле **charfield** задействует валидатор **maxlengthvaiidator**, проверяющий, не превышает ли длина заносимого строкового значения указанную максимальную длину.

Валидаторы

Используемые валидаторы можно явно указывать в свойствах полей модели через аргумент **validators**:

```
page_count = models.IntegerField(validators=[MinValueValidator(1),
                                             MaxValueValidator(1000)])
```

Встроенные в Django валидаторы находятся в модуле django.core.validators. Самые популярные валидаторы:

EmailValidator	**MaxLengthValidator**	**UniqueValidator**
URLValidator	**MinLengthValidator**	**RegexValidator**

RegexValidator

RegexValidator – валидатор, который проверяет, соответствует ли значение поля указанному регулярному выражению.

```
surname = models.CharField(max_length=30,
                           validators=[RegexValidator(r'^[A-Za-z]+$')])
```

В приведённом примере **RegexValidator** с регулярным выражением **r'^[A-Za-z]+$'**, которое проверяет, что значение поля содержит только буквы латинского алфавита (в верхнем или нижнем регистре).

Сообщения об ошибках

Во многих случаях стандартные сообщения об ошибках, выводимые валидаторами, вполне понятны. Но временами возникает необходимость вывести сообщение, более подходящее ситуации.

Собственные сообщения об ошибках указываются в параметре **error_messages** конструктора класса поля. Значением этого параметра должен быть словарь Python, у которого **ключи** элементов должны совпадать с **кодами ошибок**, а **значения** задавать сами **тексты сообщений**.

Сообщения об ошибках

```
surname = models.CharField(max_length=30,
                           validators=[RegexValidator(r'^[A-Za-z]+$')],
                           error_messages={'invalid': 'Wrong surname!'})
```

Примеры кодов ошибок:

- **null** – поле таблицы не может хранить значение null
- **blank** – в элемент управления должно быть занесено значение;
- **invalid** – неверный формат значения;
- **invalid choice** – в поле-список заносится значение, не указанное в списке;
- **unique** – в поле заносится неуникальное значение, что недопустимо;

Персональный валидатор

Если нужный валидатор отсутствует в стандартном наборе, его можно написать самостоятельно, реализовав его в виде функции или класса.

```
def validate_even(value):
    if value % 2 != 0:
        raise ValidationError("Число нечётное!",
                              code='odd',
                              params={'value': value})
```

```
page_count = models.IntegerField(validators=[validate_even])
```

Валидация модели

Метод **clean()** в моделях используется для **валидации** и очистки данных перед их сохранением. Он выполняется автоматически при вызове метода **save()** на экземпляре модели.

Метод **clean()** позволяет проверить и модифицировать значен

```python
def clean(self):
    super().clean()
    if self.publish_year:
        raise ValidationError('Не указан год публикации!')
    if self.price < 0:
        raise ValidationError('Цена не может быть отрицательной.')
```

Конец


ДОКТОР, А МОЖНО Я СВОИ СИМПТОМЫ ЗАГУГЛЮ?
НЕ ЗАНИМАЙТЕСЬ САМОЛЕЧЕНИЕМ!
Я САМ ЗАГУГЛЮ
prg_memes