Маршрутизация

Маршрутизация

1. Django определяет корневой URLconf модуль по умолчанию. Обычно, это — значение параметра ROOT_URLCONF.
2. Django загружает указанный модуль и ищет в нём urlpatterns, которая должна быть последовательностью django.urls.path() и (или) django.urls.re_path().
3. Django последовательно ищет первое соответствие URL шаблону, сопоставляя его с path_info.
4. Как только один из шаблонов URL совпадает, Django импортирует и вызывает заданное представление, передавая представлению следующие параметры:
   - Экземпляр HttpRequest
   - Если сопоставленный шаблон URL не содержит именованных групп, то совпадения из регулярного выражения предоставляются как позиционные аргументы.
   - Ключевые аргументы состоящие из любых именованных частей, совпадающих с указанным выражением пути.
5. Если совпадение не найдено, возникает исключение.

Пример маршрутов

```
from django.urls import path

from . import views

urlpatterns = [
    path("articles/2026/", views.special_case_2026),
    path("articles/<int:year>/", views.year_archive),
    path("articles/<int:year>/<int:month>/", views.month_archive),
    path("articles/<int:year>/<int:month>/<slug:slug>/",
views.article_detail),
]

/articles/2026/03/: views.month_archive(request, year=2026, month=3)
/articles/2026/: views.special_case_2026(request)
/articles/2026:
/articles/2025/06/building-a-django-site/: views.article_detail(request,
year=2026, month=3, slug="building-a-django-site")
```

Конверторы маршрутов

- str — любая непустая строка за исключением / (по умолчанию)
- int — целое неотрицательное число
- slug — буквы, цифры, подчёркивание, тире
- uuid — форматированный UUID (например, 75194d3-6885-417e-a8a8-6c931e272f01)
- path — любая непустая строка

Регулярные выражения в маршрутах

```
from django.urls import path, re_path

  from . import views

  urlpatterns = [
     path("articles/2026/", views.special_case_2026),
     re_path(r"^articles/(?P<year>[0-9]{4})/$", views.year_archive),
     re_path(r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$", views.month_archive),
     re_path(
         r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<slug>[\w-]+)/$",
         views.article_detail,
     ),
]
```

При переходе с path() на re_path(), или наоборот, тип аргументов представления может измениться!

Включение других URLConf

```
from django.urls import include, path

urlpatterns = [
    …
    path("community/", include("aggregator.urls")),
    path("contact/", include("contact.urls")),
    …
]
```

Реверсирование URL-адресов

Django предоставляет инструменты для реверсирования URL-адресов, соответствующие различным уровням, на которых требуются URL-адреса:

- В шаблонах: использование тега шаблона url.
- В коде Python: использование функции reverse() и reverse_lazy().
- В коде более высокого уровня (например, моделях): метод get_absolute_url().

Пространства имен URL

```
from django.urls import path
from . import views

app_name = "polls"
urlpatterns = [
    path("", views.IndexView.as_view(), name="index"),
    path("<int:pk>/", views.DetailView.as_view(), name="detail"),
    ...,
]
```

```
{% url 'polls:index' %}
reverse("polls:index")
```

Представления

Представление (view)

Представления или views — главные составляющие веб-приложений на основе Django. Приложение может не взаимодействовать с базой данных, может не использовать шаблоны, но views в нем будут обязательно — приложение должно как-то отвечать на запросы.

В Django используются два вида представлений:

Представления-функции — functions based vies (FBV)
Представления-классы — class based views (CBV)

В одном проекте могут одновременно использоваться оба вида. И нельзя сказать, что один вид лучше другого по всем признакам. У обоих есть свои сильные и слабые стороны.

Представления-функции и маршруты

polls/views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello World!")
```

polls/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

http://localhost:8000/polls

Представления-функции и маршруты

polls/views.py

Представления могут получать данные из-вне

```
def detail(request, question_id):
    return HttpResponse(f"Вопрос {question_id}.")

def results(request, question_id):
    return HttpResponse(f"Ответы на вопрос {question_id}.")

def vote(request, question_id):
    return HttpResponse(f"Голосование по вопросу {question_id}.")
```

Добавление маршрутов к представлениями

polls/urls.py

```
from django.urls import path
from . import views

app_name = "polls"
urlpatterns = [
    path('', views.index, name='index'),
    path("<int:question_id>/", views.detail, name="detail"),
    path("<int:question_id>/results/", views.results, name="results"),
    path("<int:question_id>/vote/", views.vote, name="vote"),
]
```

http://localhost:8000/polls/1/
http://localhost:8000/polls/1/results/
http://localhost:8000/polls/1/vote/

Представление index. Корректировка 1

polls/views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by("-pub_date")[:5]
    output = ", ".join([q.question_text for q in latest_question_list])
    return HttpResponse(output)
```

Представление index. Корректировка 2

polls/views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by("-pub_date")[:5]
    context = {"latest_question_list": latest_question_list}
    return render(request, "polls/index.html", context)
```

Шаблоны. Settings.py

Параметр `TEMPLATES` описывает как Django будет загружать и отображать шаблоны.

Значение `APP_DIRS` равное `True` реализует поиск шаблонов в каталоге **`templates`** каждого из приложений.

`mysite/settings.py`

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Первый шаблон

Создадим директории и файл для шаблона

polls/templates/polls/index.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Опросник</title>
</head>
<body>
    {% if latest_question_list%}
        <h1>Вопросы</h1>
        <ul>
            {% for question in latest_question_list %}
                <li><a href="/polls/{{ question.id }}">{{ question.question_text }}</a></li>
            {% endfor%}
        </ul>
    {% else %}
        <h1>Вопросы не найдены</h1>
    {% endif %}
</body>
</html>
```

Язык шаблонов Django

Шаблон Django - это текстовый документ или строка Python, размеченная с использованием языка шаблонов Django. Некоторые конструкции распознаются и интерпретируются механизмом шаблонов. Основные из них - переменные и теги.

Синтаксис языка шаблонов Django включает четыре конструкции:

- Переменные
- Теги
- Фильтры
- Комментарии

Язык шаблонов Django. Переменные

Переменная выводит значение из контекста, которое представляет собой объект, похожий на словарь (ассоциативный массив, хэш).

Переменные окружаются знаками {{ и }}:

```
Моё имя — {{ first_name }}. Моя фамилия — {{ last_name }}.
```

в контексте {'first_name': 'Наталья', 'last_name': 'Самойленко'} будет преобразовано в

Моё имя — Наталья. Моя фамилия — Самойленко.

Технически, когда система шаблонов встречает точку, она пытается выполнить следующие поиски в следующем порядке:

- Поиск по словарю
- Поиск атрибута или метода
- Поиск по числовому индексу

Если результирующее значение можно вызвать, оно вызывается без аргументов. Результат вызова становится значением шаблона.

```
{{ my_dict.key }}

{{ my_object.attribute }}

{{ my_list.0 }}
```

Язык шаблонов Django. Теги

Теги обеспечивают произвольную логику в процессе рендеринга.

Теги окружаются {% и %}:

Примеры:

```
{% csrf_token %}
{% if user.is_authenticated %}Привет, {{ user.username }}.{% endif %}
<tr>
    <td class="{% cycle 'row1' 'row2' as rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
<tr>
    <td class="{% cycle rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
```

https://docs.djangoproject.com/en/6.0/ref/templates/builtins/#ref-templates-builtins-tags

Язык шаблонов Django. Фильтры

Вы можете изменять переменные для отображения с помощью фильтров .

Фильтры выглядеть следующим образом: `{{ name|lower }}`

Здесь отображается значение переменной после фильтрации через фильтр, который преобразует текст в нижний регистр. Используйте вертикальную черту, чтобы применить фильтр.

Некоторые фильтры принимают аргументы. `{{ my_date|date:"Y-m-d" }}`

https://docs.djangoproject.com/en/6.0/ref/templates/builtins/#ref-templates-builtins-filters

https://docs.djangoproject.com/en/6.0/howto/custom-template-tags/#howto-writing-custom-template-filters

Язык шаблонов Django. Комментарии

```
{# Комментарий #}
{% comment "Необязательный комментарий" %}
    <p>Commented out text with {{ create_date|date:"c" }}</p>
{% endcomment %}
```

Встроенные теги

Встроенные теги. If

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

При написании условий можно использовать and, or и not, но нельзя использовать скобки! Можно использовать операторы ==, !=, <, >, <=, >=, in, not in, is, и is not.

Встроенные теги. for

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>

{% for x, y in points reversed %}
    There is a point at {{ x }},{{ y }}
{% endfor %}


{% for key, value in data.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

Встроенные теги

Встроенные теги. for. Переменные
Цикл for устанавливает ряд переменных, доступных в цикле:
`forloop.counter` — текущая итерация цикла (начинается с 1)
`forloop.counter0` — текущая итерация цикла (начинается с 0)
`forloop.revcounter` — количество итераций с конца цикла (начинается с 1)
`forloop.revcounter0` — количество итераций с конца цикла (начинается с 0)
`forloop.first` — True при первой итерации
`forloop.last` — True при последней итерации
`forloop.parentloop` — родительский итератор для вложенных циклов

Встроенные теги. for...empty

```
<ul>
  {% if athlete_list %}
    {% for athlete in athlete_list %}
      <li>{{ athlete.name }}</li>
    {% endfor %}
  {% else %}
    <li>Sorry, no athletes in this list.</li>
  {% endif %}
</ul>
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>Sorry, no athletes in this list.</li>
{% endfor %}
</ul>
```

Встроенные теги

Встроенные теги. include

Загружает шаблон и отображает его с текущим контекстом

```
{% include "foo/bar.html" %}
{% include template_name %}
{% include "name_snippet.html" with person="Jane" greeting="Hello" %}
{% include "name_snippet.html" with greeting="Hi" only %}
```

Встроенные теги. cycle

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

Встроенные теги. extends

Сигнализирует, что этот шаблон расширяет родительский шаблон.

```
{% extends "base.html" %}
{% extends variable %}
```

Встроенные фильтры

Встроенные фильтры. date

Форматирует дату в соответствии с заданным форматом.

```
{{ value|date:"D d M Y" }}
{{ value|date:"DATE_FORMAT" }}
{{ value|date:"DATETIME_FORMAT" }}
{{ value|date:"SHORT_DATE_FORMAT" }}
{{ value|date:"SHORT_DATETIME_FORMAT" }}
```

Встроенные фильтры. time

Форматирует время в соответствии с заданным форматом.

```
{{ value|time:"H:i" }}
{{ value|time:"H\h i\m" }}
{{ value|time }}
{{ value|time:"TIME_FORMAT" }}
{{ value|date:"D d M Y" }} {{ value|time:"H:i" }}
```

Встроенные фильтры

Встроенные фильтры. default и default_if_none

```
{{ value|default:"nothing" }}
{{ value|default_if_none:"nothing" }}
```

Встроенные фильтры. dictsort

Принимает список словарей и возвращает этот список, отсортированный по ключу, указанному в аргументе.

```
{{ value|dictsort:"name" }}

{% for book in books|dictsort:"author.age" %}
    * {{ book.title }} ({{ book.author.name }})
{% endfor %}

{{ value|dictsort:0 }}
```

Встроенные фильтры

Встроенные фильтры. join
Объединяет список со строкой аналогично `str.join(list)`

```
{{ value|join:" // " }}
```

Встроенные фильтры. length
Возвращает длину значения.

```
{{ value|length }}
```

Встроенные фильтры. linebreaks

```
{{ value|linebreaks }}
{{ value|linebreaksbr }}
```

Если value: "Joel\nis a slug"

```
<p>Joel<br>is a slug</p>
Joel<br>is a slug
```

Встроенные фильтры. slice
Возвращает фрагмент списка.
Используется такой же формат, что и в вырезках в Python.

```
{{ some_list|slice:":2" }}
```

Встроенные теги. url

```
<a href="/polls/{{ question.id }}/">{{ question.question_text }}</a>
```

Но что если представление `detail` будет в нескольких приложениях? В маршрутах `polls/urls.py` у нас есть определение маршрутов:

```
app_name = "polls"
urlpatterns = [
    path('', views.index, name='index'),
    path("<int:question_id>/", views.detail, name="detail"),
    path("<int:question_id>/results/", views.results, name="results"),
    path("<int:question_id>/vote/", views.vote, name="vote"),
]
```

На их основании мы можем более правильно формировать ссылки:

```
<a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a>
```

И исправим polls/urls.py

```
app_name = "polls"
urlpatterns = [
    path('', views.index, name='index'),
    path("question/<int:question_id>/", views.detail, name="detail"),
    path("question/<int:question_id>/results/", views.results, name="results"),
    path("question/<int:question_id>/vote/", views.vote, name="vote"),
]
```

Представление и шаблон detail

polls/views.py

```
def detail(request, question_id):
    question = Question.objects.get(pk=question_id)
    return render(request, "polls/detail.html", {"question": question})
```

polls/templates/polls/detail.html

Здесь и далее на слайдах будет размещаться только содержательная часть шаблонов без тегов `<html>`, `<head>`, `<body>`.

```
    {{question}}
```

Представление и шаблон detail

polls/views.py

```
from django.http import HttpResponse
from django.shortcuts import get_object_or_404, render
from .models import Question

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, "polls/detail.html", {"question": question})
```

polls/templates/polls/detail.html

Исправим представление.

```
    <h1>{{ question.question_text }}</h1>
    <ul>
    {% for choice in question.choices.all %}
        <li>{{ choice.choice_text }}</li>
    {% endfor %}
    </ul>
```

choice_set заменено на choices в модели

```
    question = models.ForeignKey(Question, on_delete=models.CASCADE,
 verbose_name='Вопрос', related_name='choices')
```

Добавим голосование

polls/templates/polls/detail.html
Исправим представление. **choice_set заменено на choices в модели**

```
<form action="{% url 'polls:vote' question.id %}" method="post">
    {% csrf_token %}
    <fieldset>
        <legend>
            <h1>{{question.question_text}}</h1>
        </legend>
        {% for choice in question.choices.all %}
            <input type="radio" name="choice"
                    id="choice{{forloop.counter}}" value="{{choice.id}}">
            <label for="choice{{forloop.counter}}">
                    {{ choice.choice_text}}</label><br>
        {% endfor %}
    </fieldset>
    <input type="submit" value="Голосовать">
</form>
```

Добавим голосование

polls/views.py

Исправим представление. **choice_set заменено на choices в модели**

```
from django.http import HttpResponse, HttpResponseRedirect
from django.urls import reverse
from django.shortcuts import get_object_or_404, render
from .models import Question, Choice

def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choices.get(pk=request.POST["choice"])
    except (KeyError, Choice.DoesNotExist):
        return render(request,
                      "polls/detail.html",
                      {
                          "question": question,
                          "error_message": "Выберите один из вариантов.",
                      },
                      )
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return HttpResponseRedirect(reverse("polls:results", args=(question.id,)))
```

Представление и шаблон results

polls/views.py

```
def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, "polls/results.html", {"question": question})
```

polls/templates/polls/results.html choice_set заменено на choices в модели

```
<h1>{{question.question_text}}</h1>
<ul>
    {% for choice in question.choices.all %}
        <li>{{choice.choice_text}} - {{choice.votes}} </li>
    {% endfor %}
</ul>
<a href="{% url 'polls:detail'  question.id %}">Голосовать снова? </a>  |
<a href="{% url 'polls:index' %}">Все вопросы</a>
```