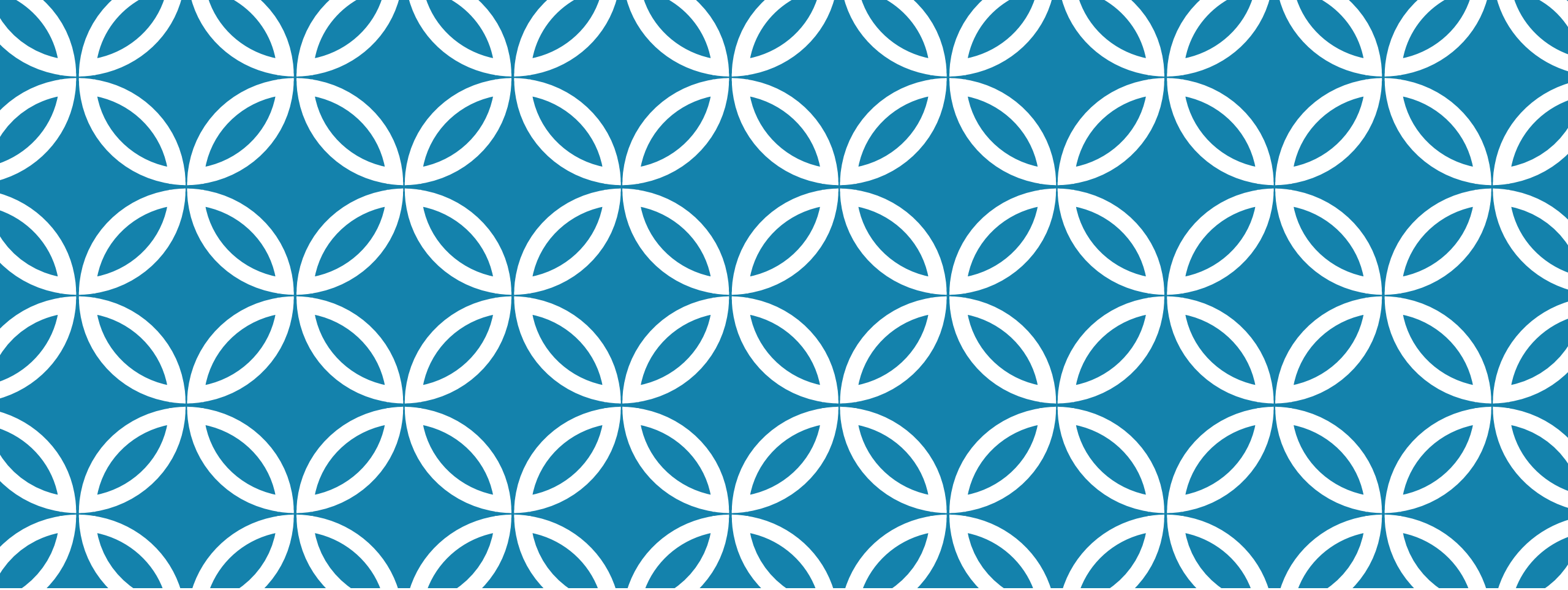


СОЗДАНИЕ ПЕРВОГО ПРОЕКТА

РАЗРАБОТКА ПРЕДСТАВЛЕНИЙ СПИСКА И ДЕТАЛЬНОЙ ИНФОРМАЦИИ

Теперь, когда вы понимаете, как использовать **ORM-преобразователь**, вы готовы к разработке представлений приложения **blog**.

Представление Django – это просто функция Python, которая получает веб-запрос и возвращает веб-ответ. Вся логика желаемого ответа находится внутри функции-представления. Сначала в своем приложении нужно создать функции-представления, затем по каждому представлению сформировать шаблон **URL-адреса** и, наконец, создать шаблоны **HTML**, чтобы прорисовывать сгенерированные представлениями данные.

Каждое представление будет прорисовывать шаблон, передавая ему переменные, и возвращать **HTTP-ответ** с прорисованным результатом.

СОЗДАНИЕ МОДЕЛЬНЫХ МЕНЕДЖЕРОВ

Теперь можно импортировать модель **Post** и извлечь все опубликованные посты, заголовки которых начинаются с **Who**, исполнив следующий ниже набор запросов **QuerySet**:

```
>>> from blog.models import Post
>>> Post.published.filter(title__startswith='Who')
```

Для того чтобы получить результаты этого набора запросов, проверьте, чтобы поле **status** было равным значению **PUBLISHED** в объекте **Post**, поле **title** которого начинается со слова **Who**.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ СПИСКА ПОСТОВ И ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Давайте начнем с создания представления списка постов на странице.

Отредактируйте файл **views.py** приложения **blog**, придав ему следующий вид.

Новые строки выделены жирным шрифтом:

```
from django.shortcuts import render
from .models import Post

def post_list(request):
    posts = Post.published.all()
    return render(request,
                  'blog/post/list.html',
                  {'posts': posts})
```

Это ваше самое первое представление Django. Представление **post_list** принимает объект **request** в качестве единственного параметра. Указанный параметр необходим для всех функций-представлений. В данном представлении извлекаются все посты со статусом **PUBLISHED**, используя менеджер **published**, который мы создали ранее

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ СПИСКА ПОСТОВ И ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Наконец, мы используем функцию сокращенного доступа **render()**, предоставляемую Django, чтобы прорисовать список постов заданным шаблоном. Указанная функция принимает объект **request**, путь к шаблону и контекстные переменные, чтобы прорисовать данный шаблон.

Она возвращает объект **HttpResponse** с прорисованным текстом (обычно исходным кодом HTML).

Функция сокращенного доступа **render()** учитывает контекст запроса, поэтому любая переменная, установленная процессорами контекста шаблона, доступна данному шаблону.

Процессоры контекста шаблона – это просто вызываемые объекты (функции, методы и классы), которые назначают контекст переменным.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ СПИСКА ПОСТОВ И ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Давайте создадим второе представление одиночного поста на странице. Добавьте следующую ниже функцию в файл **views.py**:

```
from django.http import Http404

def post_detail(request, id):
    try:
        post = Post.published.get(id=id)
    except Post.DoesNotExist:
        raise Http404("No Post found.")

    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ СПИСКА ПОСТОВ И ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

```
from django.http import Http404

def post_detail(request, id):
    try:
        post = Post.published.get(id=id)
    except Post.DoesNotExist:
        raise Http404("No Post found.")

    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

Это представление детальной информации о посте. Указанное представление принимает аргумент **id** поста. Здесь мы пытаемся извлечь объект **Post** с заданным **id**, вызвав метод **get()** стандартного менеджера **objects**. Мы создаем исключение **Http404**, чтобы вернуть ошибку **HTTP** с кодом состояния, равным **404**, если возникает исключение **DoesNotExist**, то есть модель не существует, поскольку результат не найден. Наконец, мы используем функцию сокращенного доступа **render()**, чтобы прорисовать извлеченный пост с использованием шаблона.

ПРИМЕНЕНИЕ ФУНКЦИИ СОКРАЩЕННОГО ДОСТУПА GET_OBJECT_OR_404()

Django предоставляет функцию сокращенного доступа для вызова метода **get()** в заданном модельном менеджере и вызова исключения **Http404** вместо исключения **DoesNotExist**, когда объект не найден.

Отредактируйте файл **views.py**, импортировав функцию сокращенного доступа **get_object_or_404** и изменив представление **post_detail**, как показано ниже.

Новый исходный код выделен жирным шрифтом:

```
from django.shortcuts import render, get_object_or_404

# ...

def post_detail(request, id):
    post = get_object_or_404(Post,
                             id=id,
                             status=Post.Status.PUBLISHED)
    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

ПРИМЕНЕНИЕ ФУНКЦИИ СОКРАЩЕННОГО ДОСТУПА GET_OBJECT_OR_404()

```
from django.shortcuts import render, get_object_or_404

# ...

def post_detail(request, id):
    post = get_object_or_404(Post,
                             id=id,
                             status=Post.Status.PUBLISHED)
    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

Теперь в представлении детальной информации о посте используется функция сокращенного доступа **get_object_or_404()**, чтобы извлекать желаемый пост.
Указанная функция извлекает объект, соответствующий переданным параметрам, либо исключение **HTTP** с кодом состояния, равным **404** (не найдено), если объект не найден

ДОБАВЛЕНИЕ ШАБЛОНОВ URL-АДРЕСОВ ПРЕДСТАВЛЕНИЙ

Шаблоны **URL-адресов** позволяют соотносить **URL-адреса** с представлениями. Шаблон **URL-адреса** состоит из строкового шаблона, представления и, опционально, имени, которое позволяет именовать **URL-адрес** в масштабе всего проекта.

Django просматривает каждый шаблон **URL-адреса** и останавливается на первом, который совпадает с запрошенным **URL-адресом**. Затем Django импортирует представление, совпадающее с шаблоном **URL-адреса**, и исполняет его, передавая экземпляр класса **HttpRequest** и именованные или позиционные аргументы.

Внутри каталога приложения **blog** создайте файл **urls.py** и добавьте в него следующие ниже строки:

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

app_name = 'blog'

urlpatterns = [
    # представления поста
    path('', views.post_list, name='post_list'),
    path('<int:id>/', views.post_detail, name='post_detail'),
]
```

ДОБАВЛЕНИЕ ШАБЛОНОВ URL-АДРЕСОВ ПРЕДСТАВЛЕНИЙ

В приведенном выше исходном коде определяется именное пространство приложения с помощью переменной **app_name**. Такой подход позволяет упорядочивать **URL-адреса** по приложениям и при обращении к ним использовать имя.

С помощью функции **path()** определяются два разных шаблона.

Первый шаблон **URL-адреса** не принимает никаких аргументов и соотносится с представлением **post_list**.

Второй шаблон соотносится с представлением **post_detail** и принимает только один аргумент **id**, который совпадает с целым числом, заданным целым числом конвертора путей **int**

ДОБАВЛЕНИЕ ШАБЛОНОВ URL-АДРЕСОВ ПРЕДСТАВЛЕНИЙ

Для захвата значений из **URL-адреса** используются угловые скобки. Любое значение, указанное в шаблоне **URL-адреса** как **<parameter>**, записывается в качестве строкового литерала. Для конкретного сопоставления и возврата целого числа используются конверторы путей, такие как **<int:year>**.

Например, **<slug:post>** будет, в частности, совпадать со слагом (строковым литералом, который может содержать только буквы, цифры, подчеркивания или дефисы).

Если функции **path()** и конверторов будет недостаточно, то вместо них можно использовать **re_path()**, чтобы определять сложные шаблоны **URL-адресов** с помощью регулярных выражений Python.

ДОБАВЛЕНИЕ ШАБЛОНОВ URL-АДРЕСОВ ПРЕДСТАВЛЕНИЙ

Далее необходимо вставить шаблоны **URL-адресов** приложения **blog** в главные шаблоны **URL-адресов** проекта.

Отредактируйте файл **urls.py**, расположенный внутри каталога **mysite** проекта, придав ему следующий вид. Новый исходный код выделен жирным шрифтом:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls', namespace='blog')),
]
```

ДОБАВЛЕНИЕ ШАБЛОНОВ URL-АДРЕСОВ ПРЕДСТАВЛЕНИЙ

Новый шаблон **URL-адреса**, определенный с помощью функции **include**, ссылается на шаблоны **URL-адресов**, определенные в приложении **blog**, чтобы они были включены в рамки пути **blog/**.

Указанные шаблоны вставляются в рамки именного пространства **blog**. Именные пространства должны быть уникальными для всего проекта.

Позже можно будет легко ссылаться на **URL-запросы** блога, используя именное пространство, за которым следует двоеточие, и имя **URL-запроса**, например **blog:post_list** и **blog:post_detail**

СОЗДАНИЕ ШАБЛОНОВ ПРЕДСТАВЛЕНИЙ

Вы создали представления и шаблоны **URL-адресов** для приложения **blog**. Шаблоны **URL-адресов** соотносят **URL-адреса** с представлениями, а те, в свою очередь, решают, какие данные будут возвращаться пользователю.

Шаблоны определяют способ отображения данных; обычно они пишутся на **HTML** в сочетании с языком шаблонов Django.

Давайте добавим в приложение шаблоны, чтобы отображать посты в удобном для пользователя виде. Внутри каталога приложения **blog** создайте следующие ниже каталоги и файлы:

```
templates/
    blog/
        base.html
        post/
            list.html
            detail.html
```

СОЗДАНИЕ ШАБЛОНОВ ПРЕДСТАВЛЕНИЙ

Приведенная выше структура будет файловой структурой ваших шаблонов. Файл **base.html** будет включать в себя главную **HTML-структуру** веб-сайта и разделит контент на главную область содержимого и боковую панель.

Файлы **list.html** и **detail.html** будут наследовать от файла **base.html**, чтобы прорисовывать представления соответственно списка постов блога и детальной информации о посте. Django обладает мощным языком шаблонов, который позволяет указывать внешний вид отображения данных.

Он основан на шаблонных тегах, шаблонных переменных и шаблонных фильтрах:

- шаблонные теги управляют прорисовкой шаблона и выглядят как **{% tag %}**;
- шаблонные переменные заменяются значениями при прорисовке шаблона и выглядят как **{{ variable }}**;
- шаблонные фильтры позволяют видоизменять отображаемые переменные и выглядят как **{{ variable|filter }}**

СОЗДАНИЕ БАЗОВОГО ШАБЛОНА

Отредактируйте файл **base.html**, добавив в него следующий ниже исходный код:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  <link href="{% static "css/blog.css" %}" rel="stylesheet">
</head>
<body>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
  <div id="sidebar">
    <h2>My blog</h2>
    <p>This is my blog.</p>
  </div>
</body>
</html>
```

СОЗДАНИЕ ШАБЛОНА СПИСКА ПОСТОВ

Давайте отредактируем файл **post/list.html** и придадим ему следующий вид:

```
{% extends "blog/base.html" %}

{% block title %}My Blog{% endblock %}

{% block content %}
  <h1>My Blog</h1>
  {% for post in posts %}
    <h2>
      <a href="{% url 'blog:post_detail' post.id %}">
        {{ post.title }}
      </a>
    </h2>
    <p class="date">
      Published {{ post.publish }} by {{ post.author }}
    </p>
    {{ post.body|truncatewords:30|linebreaks }}
  {% endfor %}
{% endblock %}
```

СОЗДАНИЕ ШАБЛОНА СПИСКА ПОСТОВ

Шаблонный тег **{% extends %}** сообщает Django, что надо наследовать от шаблона **blog/base.html**.
Затем заполняются блоки **title** и **content** базового шаблона. Посты прокручиваются в цикле, и их заголовок, дата, автор и тело отображаются на странице, включая ссылку в заголовке на подробный **URL-адрес** поста.
URL-адрес формируется с использованием предоставляемого веб-фреймворком Django шаблонного тега **{% url %}**.
Этот шаблонный тег позволяет формировать **URL-адреса** динамически по их имени. Мы используем **blog:post_detail**, чтобы ссылаться на **URL-адрес post_detail** в именном пространстве **blog**.
Мы передаем необходимый параметр **post.id**, чтобы сформировать **URL-адрес** для каждого поста.

Для формирования URL-адресов в своих шаблонах следует всегда использовать шаблонный тег `{% url %}`, а не писать жестко привязанные URL-адреса. Такой подход упростит техническое сопровождение URL-адресов в будущем.

СОЗДАНИЕ ШАБЛОНА СПИСКА ПОСТОВ

В теле поста применяются два шаблонных фильтра:
truncatewords усекает значение до указанного числа слов, а **linebreaks** конвертирует результат в разрывы строк в формате **HTML**.

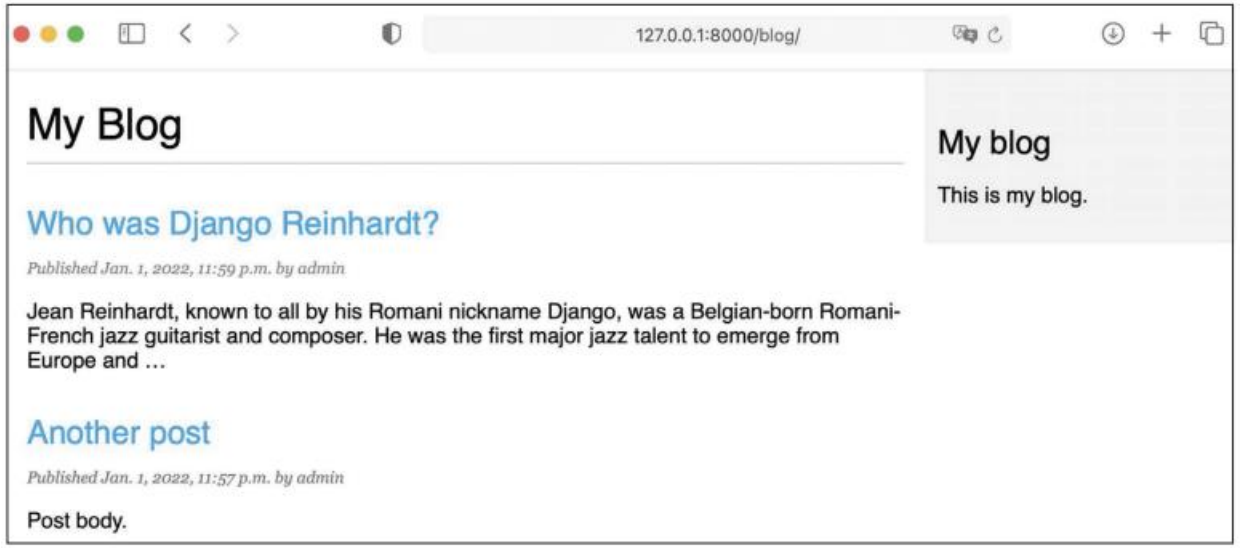
При этом можно конкатенировать столько шаблонных фильтров, сколько потребуется;
каждый из них будет применен к результату, сгенерированному предыдущим.

ДОСТУП К ПРИЛОЖЕНИЮ

Откройте командную оболочку и исполните следующую ниже команду, чтобы запустить сервер разработки:

```
python manage.py runserver
```

Пройдите по **URL-адресу http://127.0.0.1:8000/blog/** в своем браузере; вы увидите, что все работает. Обратите внимание, что для того чтобы можно было отобразить здесь посты, необходимо иметь несколько постов со статусом **PUBLISHED**. Вы должны увидеть что-то вроде этого:



Страница представления списка постов

СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Далее отредактируйте файл **post/detail.html**:

```
{% extends "blog/base.html" %}

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
  <h1>{{ post.title }}</h1>
  <p class="date">
    Published {{ post.publish }} by {{ post.author }}
  </p>
  {{ post.body|linebreaks }}
{% endblock %}
```

Затем можно вернуться в свой браузер и кликнуть по одному из заголовков постов, чтобы просмотреть детальную информацию о посте. Вы должны увидеть что-то вроде этого:

СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Страница представления детальной информации о посте

Взгляните на **URL-адрес** – он должен содержать автоматически генерируемый ИД поста. Например, **/blog/1/**.

СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Давайте рассмотрим цикл запроса/ответа Django, воспользуясь приложением, которое мы разработали.
Следующая ниже схема показывает упрощенный пример того, как Django обрабатывает **HTTP-запросы** и генерирует **HTTP-ответы** (рисунок ниже).

Рассмотрим процесс запроса/ответа Django.

1. Веб-браузер запрашивает страницу по ее **URL-адресу**, например **https://domain.com/blog/33/**. Веб-сервер получает **HTTP-запрос** и передает его Django.
2. Django пробегает по всем шаблонам **URL-адресов**, определенным в конфигурации шаблонов **URL-адресов**. Он проверяет каждый шаблон на соответствие заданному пути **URL-адреса** в порядке их появления и останавливается на первом, который совпадает с запрошенным **URL-адресом**. В данном случае шаблон /blog// соответствует пути /blog/33/

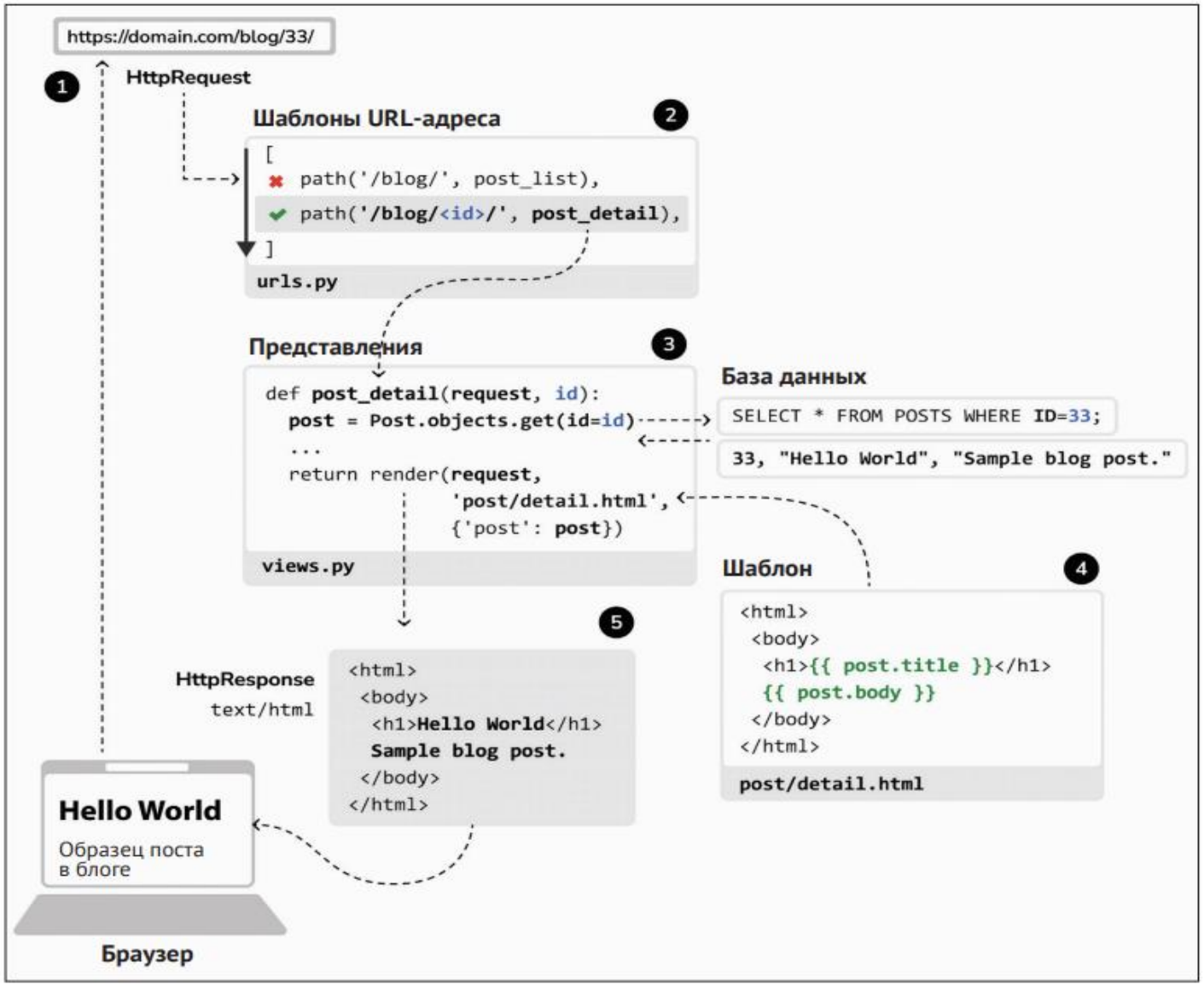
СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

3. Django импортирует представление совпавшего шаблона **URL-адрес**а и исполняет его, передавая экземпляр класса **HttpRequest** и именованные либо позиционные аргументы. Представление использует модели, чтобы извлечь информацию из базы данных. С помощью встроенного в Django **ORM-преобразователя** наборы запросов **QuerySets** транслируются в **SQL** и исполняются в базе данных.

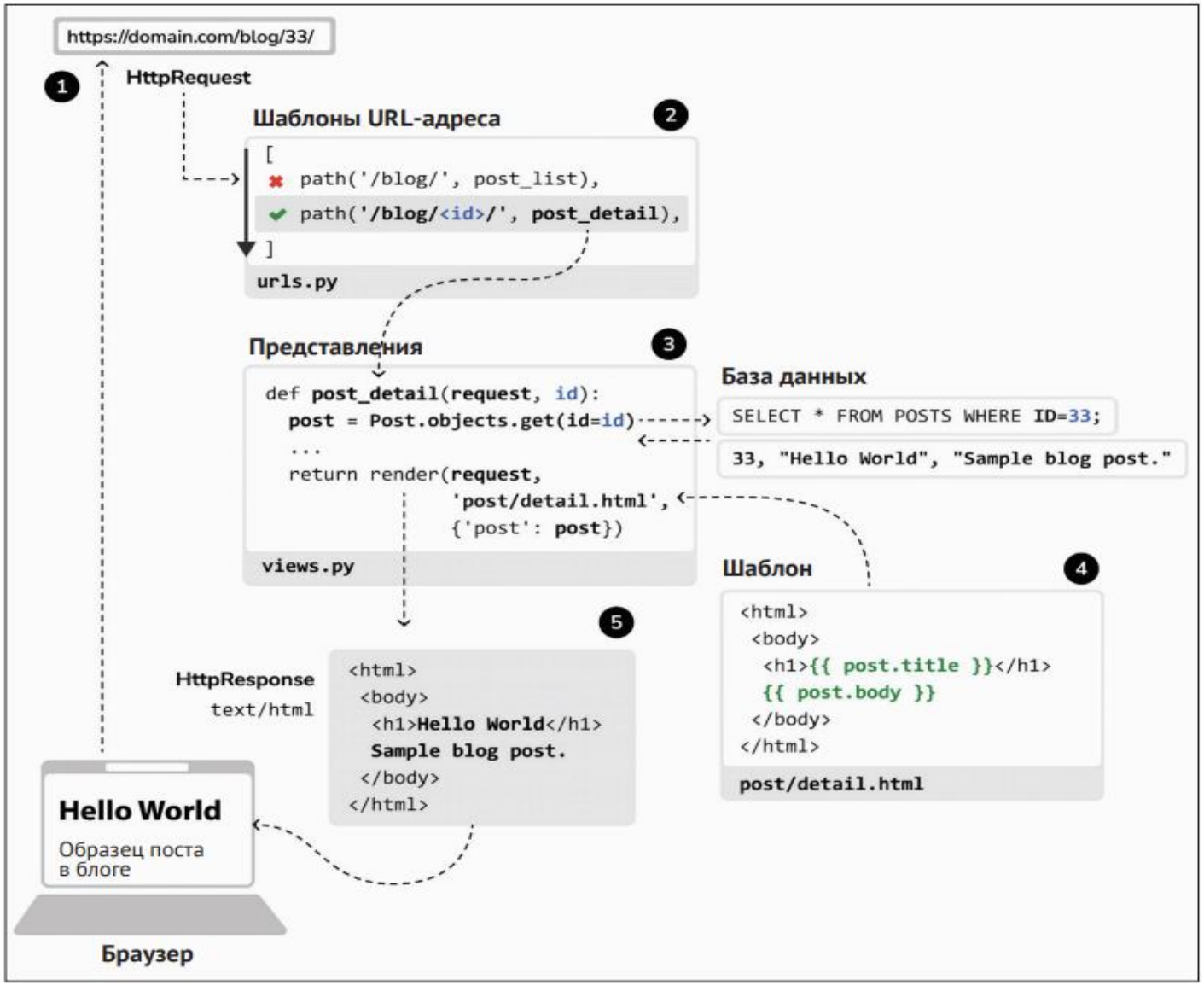
4. В представлении используется функция **render()**, которая прорисовывает шаблон **HTML**, передав в него объект **Post** в качестве контекстной переменной.

5. Прорисованный контент возвращается представлением в виде объекта **HttpResponse**, по умолчанию с типом контента **text/html**

СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ



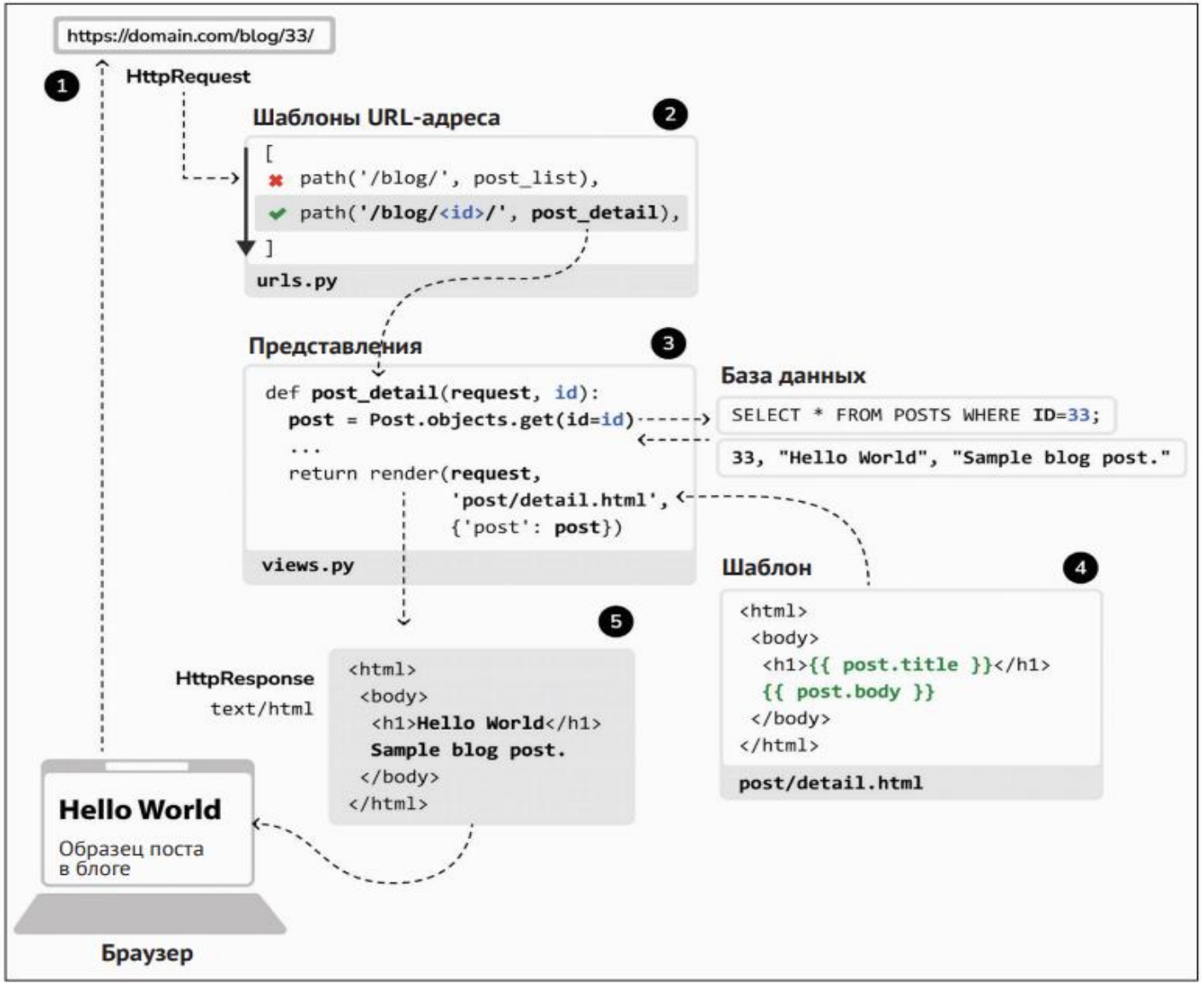
СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ



Цикл запроса/ответа Django

СОЗДАНИЕ ШАБЛОНА ДЕТАЛЬНОЙ ИНФОРМАЦИИ О ПОСТЕ

Цикл запроса/ответа Django



Указанную схему всегда можно использовать в качестве базового ориентира в отношении того, как Django обрабатывает запросы. В целях простоты данная схема не содержит промежуточные программные компоненты Django.