Exercise 13: Creating a DetailView

25 to 40 minutes

In this exercise, you will create the page to show information about an individual joke. To do this, you will inherit from `DetailView`.

1. Open `jokes/views.py` in your editor.
2. Import `DetailView` from `django.views.generic`:

```
from django.views.generic import DetailView, ListView
```

3. Create a `JokeDetailView` view that inherits from `DetailView`:

```
class JokeDetailView(DetailView):
    model = Joke
```

   Just like a `ListView`, a minimal `DetailView` only requires the model to query.

The `jokes/views.py` file should now look like this:

Exercise Code 13.1: jokes/views.py

```
1.   from django.views.generic import DetailView, ListView
2.
3.   from .models import Joke
4.
5.   class JokeDetailView(DetailView):
6.       model = Joke
7.
8.
9.   class JokeListView(ListView):
10.      model = Joke
```

URLConf

You must now configure a path to the new view. Because each joke will have its own detail page, you must identify the specific joke to show. For now, you will do this using the `joke` object's primary key.

1. Open `jokes/urls.py` in your editor.

2. Import the `JokeDetailView` view from `views.py`, which is in the same directory:

```
from .views import JokeDetailView, JokeListView
```

3. Add a new path to `urlpatterns` to `JokeDetailView.as_view()` at `'joke/<int:pk>/'`:

```
path('joke/<int:pk>/', JokeDetailView.as_view(), name='detail'),
```

The path is constructed as "`joke/<int:pk>/`". `int:pk` indicates that the value entered for this part of the path must be an integer. The view will have access to this value via `self.kwargs.get('pk')` and will use it to get the specific joke object to use.[11]

The `jokes/urls.py` file should now look like this:

Exercise Code 13.2: jokes/urls.py

```
from django.urls import path

from .views import JokeDetailView, JokeListView

app_name = 'jokes'
urlpatterns = [
    path('joke/<int:pk>/', JokeDetailView.as_view(), name='detail'),
    path('', JokeListView.as_view(), name='jokes'),
]
```

The Template

Can you guess the default name of the template used by the `JokeDetailView`? If you guessed `jokes/joke_detail.html`, you're right. Create that template:

Within the `templates/jokes` folder, add a `joke_detail.html` file with the following content:[12]

11. Look in `.venv\Lib\site-packages\django\views\generic\detail.py` at the `get_object()` method of the `SingleObjectMixin` class to see how this magic works behind the scenes.
12. **Don't want to type?** Copy from `starter-code/app-with-model/joke_detail.html`.

Exercise Code 13.3: templates/jokes/joke_detail.html

```
{% extends "_base.html" %}

{% block title %}Joke{% endblock %}
{% block main %}
  <div class="card border-primary m-auto mb-3 text-center"
    style="max-width: 30rem">
    <div class="card-header">{{ joke.question }}</div>
    <div class="card-body text-primary">
      <h5 class="card-title">{{ joke.answer }}</h5>
    </div>
    <div class="card-footer">
      <small class="text-muted">
        Created on: {{ joke.created }}
        Last updated: {{ joke.updated }}
      </small>
    </div>
  </div>
{% endblock %}
```

Things to notice:

1. The template extends `_base.html`:

   ```
   {% extends "_base.html" %}
   ```

2. The template has access to an auto-created variable called `joke` (the lowercase model name) containing the `joke` object.

Open `templates/jokes/joke_list.html` and make `joke.question` a link:

Exercise Code 13.4: templates/jokes/joke_list.html

```
-------Lines 1 through 6 Omitted-------
      {% for joke in joke_list %}
        <li class="list-group-item">
          <a href="{% url 'jokes:detail' joke.pk %}">{{ joke.question }}</a>
        </li>
      {% endfor %}
-------Lines 12 through 13 Omitted-------
```

In the `url` tag, `joke.pk` holds the primary key for the joke, which it will use to construct the URL. Think of a tag as a function with each subsequent item being an argument passed to the function. So, the following `url` tag:

```
{% url 'joke' joke.pk %}
```

…becomes:

```
url('joke', joke.pk)
```

The function constructs and returns the URL based on the arguments it receives.

Try It Out

Return to the browser and refresh the **jokes** page. The joke question should now be a link. Click it. The resulting page should look like this:

If you have followed along with everything exactly, the URL should read:

```
http://127.0.0.1:8000/jokes/joke/2/
```

However, if you added more jokes, it may show a different primary key.

Avoid Using Primary Keys in URLs

It is not a great idea to use primary keys in URLs. We will show a better way of creating URLs for jokes in a later lesson (see page 195).

get_absolute_url()

An alternative to constructing URLs with the `url` tag as we did in `joke_list.html` is to use the `get_absolute_url()` method, like this:

```
joke.get_absolute_url
```

However, before you do that, you need to add the `get_absolute_url()` method to the model:

1. Open `jokes/models.py` in your editor.
2. Import the `reverse()` function from `django.urls`:

   ```
   from django.urls import reverse
   ```

   The `reverse()` function gets and returns the URL based on the passed-in URL pattern name.
3. Add the `get_absolute_url()` function above the `__str()__` function in the `Joke` model:[13]

   ```
   def get_absolute_url(self):
       return reverse('jokes:detail', args=[str(self.pk)])
   ```

   The arguments passed to the preceding `reverse()` function are:

   A. The URL pattern name preceded by the namespace as defined in the URLConf:

      ```
      app_name = 'jokes'
      urlpatterns = [
          path('joke/<int:pk>/', JokeDetailView.as_view(), name='detail'),…
      ```

13. See `https://docs.djangoproject.com/en/3.1/ref/models/instances/#django.db.models.Model.get_absolute_url` for documentation on `get_absolute_url()`.

B. The arguments required by the URL pattern. In this case, it is just the `pk` of the joke converted to a string.[14]

The `jokes/models.py` file should now look like this:

Exercise Code 13.5: jokes/models.py

```
1.   from django.db import models
2.   from django.urls import reverse
3.
4.   class Joke(models.Model):
5.       question = models.TextField(max_length=200)
6.       answer = models.TextField(max_length=100, blank=True)
7.       created = models.DateTimeField(auto_now_add=True)
8.       updated = models.DateTimeField(auto_now=True)
9.
10.      def get_absolute_url(self):
11.          return reverse('jokes:detail', args=[str(self.pk)])
12.
13.      def __str__(self):
14.          return self.question
```

You can now update the template to use `get_absolute_url`:

Exercise Code 13.6: templates/jokes/joke_list.html

```
     -------Lines 1 through 6 Omitted-------
7.       {% for joke in joke_list %}
8.         <li class="list-group-item">
9.           <a href="{{ joke.get_absolute_url }}">{{ joke.question }}</a>
10.        </li>
11.      {% endfor %}
     -------Lines 12 through 13 Omitted-------
```

A big advantage of using `get_absolute_url` is that you don't have to change the template if you change the way that URLs are constructed for jokes, which you will eventually do.

14. See `https://docs.djangoproject.com/en/3/ref/urlresolvers/#django.urls.reverse` for documentation on `reverse()`.

Git Commit

Commit your code to Git.

2.5. GET and POST Requests

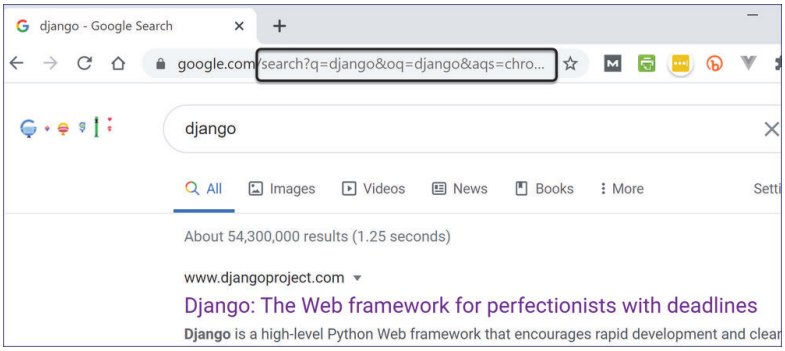
The value of a form's `method` attribute determines how the form data will be passed to the server.

get

When using the `get` method, which is the default, form data is sent to the server in the URL as a *query string*. The query string is appended to the website address starting with a question mark (`?`) and followed by name-value pairs delimited (separated) by an ampersand (`&`). A URL with a query string might look like this:

```
https://www.example.com?firstname=Nat&lastname=Dunn
```

The `get` method is commonly used by search engines, because it allows the resulting page to be bookmarked. For example, Google uses the `get` method. You can tell by looking at the location bar after doing a search:



post

When `post` is used, the name-value pairs are not sent as part of the query string. Instead, they are sent behind the scenes. This has the advantage of keeping the values hidden from anyone looking over the user's shoulder. Two other advantages of the `post` method are:

1. It allows for much more data to be submitted (i.e., larger forms).
2. It allows for files to be uploaded to the server.[15]

You should use the `post` method whenever the form submission will (or might) do anything more than request a web page. For example, if the submission will log a user in, modify the database or the file system, or send an email, you should use `post`.

15. Files can be uploaded to the server via the `file` input type. The tag syntax is: `<input type="file" name="filename">`.