Learning Objectives: Viewing Database Information

Learners will be able to...

- **Display all records in a database with a `ListView` generic class-based view**
- **Differentiate between `ListView` and `TemplateView`**
- **Iterate over the list of records from the database**
- **Access specific fields from a record in a template**
- **Identify the types of data compatible with the `for` template tag**

info

Make Sure You Know

You should be familiar with views, templates, and URL patterns in Django. You are also comfortable with basic interactions in the terminal.

Limitations

The features of this website are basic. The focus is not on style, but on the mechanics of iterating over information from the database.

Views and Templates

List View

Start by activating the `django` virtual environment.

```
conda activate django
```

In a previous project, we saw how the `TemplateView` generic class-based view can be used to render content to the screen. This worked great when the content came from an HTML file with all of the information hard coded into the files.

This project is different in that we want to display is stored in a database. In fact, we want to list all of the records in the database. Because of this, `TemplateView` is inadequate for our needs. Django, being ever so helpful, has a `ListView` that creates a list of every record in the database.

Open the `views.py` file from the Django app. Import the `ListView` generic class-based view. We also need to import the `Review` model used in our database. Our project is only going to have a single page with all of the movie reviews, so we only need one view. Name the view `HomePageView` and inherit from `ListView`.

```
from django.views.generic import ListView
from .models import Review

class HomePageView(ListView):
```

Just as with the `TemplateView` class-based view, we are going to create the variable `template_name` and assign it an HTML file. In this case, the file is called `home.html`. In addition, we need to create the variable `model` and assign it our model `Review`.

```
from django.views.generic import ListView
from .models import Review

class HomePageView(ListView):
    template_name = 'home.html'
    model = Review
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black reviews/views.py
```

Templates

We are going to store all our templates in a single `templates` directory. In the terminal, enter the following command to create this directory.

```
mkdir templates
```

We need to update the `settings.py` file since we want to use our own directory for the templates. Look for the variable `TEMPLATES` which should be around line 65. Look for the key `"DIRS"`, whose value is an empty list. Update this value to the one below.

```
        "DIRS": [BASE_DIR / 'templates'],
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black movie_review/settings.py
```

Create the `home.html` file inside of the `templates` directory with the command below.

```
touch templates/home.html
```

Open `home.html`. For now, we are going to have an `<h1>` tag with the title of our website. Then we want to display just the movie title for each film in the database. Django has a `for` template tag that iterates over a list. We are going to use `review_list` as the list. This is a list of every record in our database. The name of the list is derived from the name of the model (all lowercase) followed by `_list`. We are able to do this thanks to the `ListView` generic class-based view from above. The for loop function almost identically to a traditional for look in Python. When using the loop variable, be sure to surround with `{{ }}` to differentiate it from HTML. We also need to end the `for` template tag.

```
<h1>Movie Reviews</h1>
{% for review in review_list %}
<h3>{{ review.title }}</h3>
{% endfor %}
```

Since we did not edit any Python code, we do not need to run the Black formatter. Unfortunately, we cannot yet look at our website until we address the URL patterns.

Deactivate the virtual environment.

```
conda deactivate
```

URL Patterns

Creating the Patterns

Start by activating the `django` virtual environment.

```
conda activate django
```

The first URL pattern will be for the Django project. We want to include the `urls.py` file in the `reviews` app. Add this to the list after the URL pattern for the Django Admin.

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

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

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black movie_review/urls.py
```

Currently, the `urls.py` file for the `reviews` app does not exist. Enter the following command to create it. Then click the link to open the file.

```
touch reviews/urls.py
```

Import `path` from `django.urls` and import our `HomePageView` from the `views.py` file. Create a pattern for the root of our website which renders `HomePageView`. Because this is a class-based view, we need to use the `as_view()` method. Give this pattern the name `home`.

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

urlpatterns = [
    path('', HomePageView.as_view(), name='home'),
]
```

Open the terminal and run the Black formatter to make sure our code has a consistent style.

```
black reviews/urls.py
```

Checking our Work

We are finally ready to display information from the database to our Django website. Run the dev server and open the website. We should only see the titles of the four films — "The Big Lebowski", "Snatch", "Watchmen", and "Morbius". The text should have a little styling from the `<h3>` tag.

```
python manage.py runserver 0.0.0.0:8000
```

The image depicts the page title "Movie Reviews" and the four movie titles — "The Big Lebowski", "Snatch", "Watchmen", and "Morbius".

Let's expand the amount of information shown to the user. Leave the dev server running and open up `home.html`. Create a separate `<p>` tag for all of the other fields in the database. Preface the stars, actors, year, and director

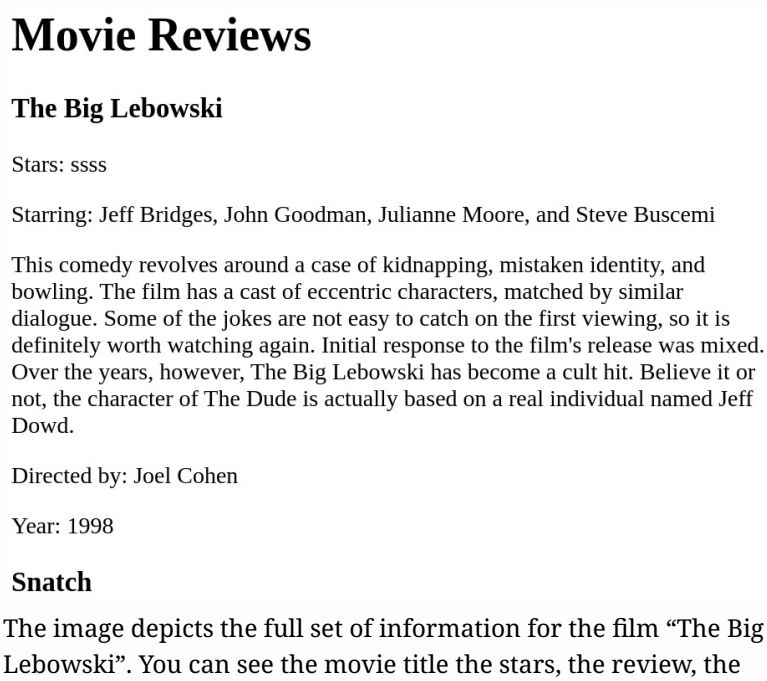
with a label so the user better understands the information on the screen. The review does not need a label as it is pretty clear what that is. All of these changes must go inside the `for` template tag.

```
<h1>Movie Reviews</h1>
{% for review in review_list %}
<h3>{{ review.title }}</h3>
<p>Stars: {{ review.stars }}</p>
<p>{{ review.review }}</p>
<p>Starring: {{ review.actors }}</p>
<p>Directed by: {{ review.director }}</p>
<p>Year: {{ review.year }}</p>
{% endfor %}
```

The dev server should still be running. Open the website and refresh it with the blue arrows in the shape of a circle.

The image depicts two blue icons. The one on the left is in the shape of a triangle and the one on the right is two arrows in the shape of a circle.

You should see the full set of information for each film in the database. Scroll down to make sure that all of the films appear as expected.



Movie Reviews

The Big Lebowski

Stars: ssss

Starring: Jeff Bridges, John Goodman, Julianne Moore, and Steve Buscemi

This comedy revolves around a case of kidnapping, mistaken identity, and bowling. The film has a cast of eccentric characters, matched by similar dialogue. Some of the jokes are not easy to catch on the first viewing, so it is definitely worth watching again. Initial response to the film's release was mixed. Over the years, however, The Big Lebowski has become a cult hit. Believe it or not, the character of The Dude is actually based on a real individual named Jeff Dowd.

Directed by: Joel Cohen

Year: 1998

Snatch

The image depicts the full set of information for the film “The Big Lebowski”. You can see the movie title the stars, the review, the actors, the director, and the year.

▼ **Did you notice?**

> The stars rating looks bad because a series of “s” characters on the the screen instead of a number or images of stars. We will address this problem when we style the website.

Switch back to the terminal and stop the dev server with `Ctrl+C` on the keyboard. Then deactivate the virtual environment.

```
conda deactivate
```

Styling the Site

Base Template

Start by activating the `django` virtual environment.

```
conda activate django
```

We are going to use Bootstrap to style our website. This means we need to create a base template which imports Bootstrap. The `home.html` template will then inherit and extend the base template. Create `base.html` in the `templates` directory.

```
touch templates/base.html
```

Open the file and give it the basic structure of an HTML page.

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
    </head>

    <body>

    </body>

</html>
```

Import Bootstrap by providing a link to its hosted files in the `<head>`. Then put a `<script>` tag in the `<body>` of the website.

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
        <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap
        .min.css" rel="stylesheet" integrity="sha384-
        1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
        crossorigin="anonymous">


    </head>


    <body>
        <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle
        js" integrity="sha384-
        ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
        crossorigin="anonymous"></script>


    </body>

</html>
```

Create a `<div>` with the Bootstrap class `container`. This will provide left and right margins for the movie reviews. Use the `p-3` class to give the container a little bit of padding. Inside the container, add the `<h1>` tag with the webiste title. Use the `display-1` Bootstrap class to style the title. Finally, create a `block` template tag named `content`. This serves as a placeholder for the child content.

```
    <body>
        <div class="container p-3">
            <h1 class="display-1">Movie Reviews</h1>
            {% block content %} {% endblock content %}
            </div>
        </div>
    <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle
        js" integrity="sha384-
        ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
        crossorigin="anonymous"></script>


    </body>
```

Extending the Parent Template

Open `home.html` and add a template to inherit from the base template. Add a `block` template tag named `content` so that the code in this file overwrites the `content` template tag in the base template. Remove the `<h1>` tag with the title as it is now in `base.html`. Close the block template tag at the very end of the file.

```
{% extends "base.html" %}
{% block content %}
{% for review in review_list %}
<h3>{{ review.title }}</h3>
<p>Stars: {{ review.stars }}</p>
<p>{{ review.review }}</p>
<p>Starring: {{ review.actors }}</p>
<p>Directed by: {{ review.director }}</p>
<p>Year: {{ review.year }}</p>
{% endfor %}
{% endblock content %}
```

Open the terminal, launch the dev server, and load our Django website. We should now see some basic styling on our site. It now has margins, the main title is taller and thinner due to the `display` class, and the `<h3>` tags look slightly different since Bootstrap overrides these default settings.

```
python manage.py runserver 0.0.0.0:8000
```

The image depicts the reviews with basic Bootstrap styling. There are margins on the left and right. The main title movie titles have different font due to Bootstrap.

Switch back to the terminal and stop the dev server with `Ctrl+C` on the keyboard. Then deactivate the virtual environment.

```
conda deactivate
```

Final Touches

Adding Visual Cues

Start by activating the `django` virtual environment.

```
conda activate django
```

Even though this movie review site is supposed to be simple, it looks a little too plain. We are going to tweak a few things to add visual cues to emphasize important information and introduce some visual separation between reviews.

Open `home.html`. For now, leave the movie title and stars alone. We will adjust them later. Use the `lead` class on the `<p>` tag for the review. For every label, add a `<strong>` tag. Add the `pb-3` class to the last `<p>` tag to provide a small amount of padding on the bottom.

```
{% extends "base.html" %}
{% block content %}
{% for review in review_list %}
<h3>{{ review.title }}</h3>
<p>Stars: {{ review.stars }}</p>
<p class="lead">{{ review.review }}</p>
<p><strong>Starring:</strong> {{ review.actors }}</p>
<p><strong>Directed by:</strong> {{ review.director }}</p>
<p class="pb-3"><strong>Year:</strong> {{ review.year }}</p>
{% endfor %}
{% endblock content %}
```

Open the terminal, launch the dev server, and load our Django website. The `lead` class makes the text taller and skinnier. It draws your eye to the review, which is an important piece of information. You can also locate other information (director, actors, year) easier as well.

```
python manage.py runserver 0.0.0.0:8000
```

Stars

The last thing to address is the movie title and star rating. We would like for both of them to appear on the same line. We also want to replace the `s` characters with images of a star.

Remember, Bootstrap uses the grid system for layout. Create a `<span>` with the class `row`. In addition, add the class `text-primary` so that the movie title and stars are blue. Inside the row, create `<h3>` tags for the title and stars. If we do not specify the number of columns, Bootstrap will use all 12. Use the `col-4` class for the movie title and `col-8` for the stars.

```
{% extends "base.html" %}
{% block content %}
{% for review in review_list %}
<span class="row text-primary">
    <h3 class="col-4">{{ review.title }}</h3>
    <h3 class="col-8">{{ review.stars }}</h3>
</span>
<p class="lead">{{ review.review }}</p>
<p><strong>Starring:</strong> {{ review.actors }}</p>
<p><strong>Directed by:</strong> {{ review.director }}</p>
<p><strong>Year:</strong> {{ review.year }}</p>
<p class="pb-3"><strong>Review by:</strong> {{ review.author }}
        </p>
{% endfor %}
{% endblock content %}
```

These changes, however, do not address the issue of star images. If you remember back to the very beginning of this project, the field type for the stars was `CharField` and not `PositiveSmallIntegerField`. We did this because Django does not have a template tag that allows you to iterate over a range. For example:

```
{% for star in range(review.stars) %}
    # display a star
{% endfor %}
```

Django does let you iterate over a string. Loop over `review.stars` just as you would iterate over a string in Python. Place the template tag inside the `<h3>` tags for the second column. Inside the for loop use `&#9733;` which is the unicode for a star. In addition, because it is unicode and not an actual image, the stars will inherit the styling from the `<span>` and match the styling of the movie title.

```
<span class="row text-primary">
    <h3 class="col-4">{{ review.title }}</h3>
    <h3 class="col-8">{% for star in review.stars %}
        &#9733;
        {% endfor %}
    </h3>
</span>
```

Open the terminal, launch the dev server, and load our Django website. The eye is drawn to the three most important pieces of information for each review — the title of the film, the number of stars, and the review itself. The extra padding (as well as the color blue) help separate the reviews from one another.

```
python manage.py runserver 0.0.0.0:8000
```

The image depicts the final version of our website. The movie title and stars are on the same line and are blue. The “s” characters have been replaced with a star. The text for the movie review is taller and thinner than the other text in the review. Finally, there is some extra white space between each review as well.

Switch back to the terminal and stop the dev server with `Ctrl+C` on the keyboard. Then deactivate the virtual environment.

```
conda deactivate
```