

Python Django - Part B.

Class Views:

TemplateView:

If we just need to return template we can use the built-in class that handles that (the "Thank you" page use case):



```
views.py
reviews > views.py > ...
1   from django.shortcuts import render
2   from django.http import HttpResponseRedirect
3   from .forms import ReviewForm
4   from django.views import View
5   from django.views.generic.base import TemplateView # |
6   # Create your views here.
7 
```



```
views.py
reviews > views.py > ...
38  #             if form.is_valid():
39  #                 form.save()
40  #                 return HttpResponseRedirect("/thank-you")
41  #             else:
42  #                 form = ReviewForm()
43  #                 return render(request, "reviews/review.html",
44  #                               {"form": form,
45  #                                })
46
47 class ThankYouView(TemplateView):
48     template_name = "reviews/thank_you.html"
49 
```

If we need to inject data into the template we do it by the "context" var. (we add keys and values to this dic)

```
class ThankYouView(TemplateView):
    template_name = "reviews/thank_you.html"

    # If we need to interpolate data in the template
    # we return a dictionary - keys - are value names , values are the data.
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["message"] = "This Works!"
        return context
```

views.py thank_you.html

reviews > templates > reviews > thank_you.html

```
1  {% extends 'reviews/base.html' %}           1
2  {% block title %}                         2
3  Thank You                                3
4  {% endblock title %}                      4
5
6  {% block content %}                       5
7  <h1>Thank You</h1>                     6
8  <p>{{ message }}</p>                   7
9  {% endblock content %}                    8
```

The ListView:

A built in Django class to send a list of data from a model to a template.

```
views.py    ●   single_review.html    urls.py
reviews > views.py > ...
1  from multiprocessing import context
2  from re import template
3  from django.shortcuts import render
4  from django.http import HttpResponseRedirect
5  from .forms import ReviewForm
6  from django.views import View
7  from django.views.generic.base import TemplateView
8  from .models import Review
9  from django.views.generic import ListView
10 # Create your views here.
11
```

We just need to point to which model we want to fetch the data from:

```
views.py    X   single_review.html    urls.py
reviews > views.py > ReviewsListView
59
60
61 class ReviewsListView(ListView):
62     template_name = "reviews/review_list.html"
63     # get all the data relatef to this model.
64     model = Review
65
```

The name of the list in the template the we pass the data to will be called "object_list":

```
views.py      single_review.html      urls.py      review_list.html
reviews > templates > reviews > review_list.html
1  {% extends 'reviews/base.html' %}

2
3  {% block title %}
4  All Reviews
5  {% endblock title %}
6  {% block content %}
7      <ul>
8          {% for review in object_list %}
9              <li>{{review.user_name}} (Rating: {{review.rating}})</li>
10             {% endfor %}
11         </ul>
12     {% endblock content %}
```

We can change that name to what ever we want by:

```
class ReviewsListView(ListView):
    template_name = "reviews/review_list.html"
    # get all the data relatef to this model.
    model = Review
    context_object_name = "reviews"
```

We can also filter for data we want to pass by:

```
views.py      single_review.html      urls.py
reviews > views.py > ReviewsListView > get_queryset
59
60
61 class ReviewsListView(ListView):
62     template_name = "reviews/review_list.html"
63     # get all the data relatef to this model.
64     model = Review
65     context_object_name = "reviews"

66
67     def get_queryset(self):
68         base_query = super().get_queryset()
69         data = base_query.filter(rating__gt=1)
70         return data
```

DetailView:

When ever we want to return a template from a "GET" request with data about a single piece of data using a "DetailView" might be the one we want.

Django knows which record he need to get (from a model) by the "primary key" or the "slug" we pass in the view.



The screenshot shows a code editor with four tabs: single_review.html, views.py, urls.py, and review_list.html. The urls.py tab is active. The code defines a URL pattern for a single review:

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path("", views.ReviewView.as_view()),
6     path("thank-you", views.ThankYouView.as_view()),
7     path("reviews", views.ReviewsListView.as_view()),
8     path("reviews/<int:pk>", views.SingleReviewView.as_view())
9 ]
```



The screenshot shows a code editor with three tabs: single_review.html, views.py, and urls.py. The views.py tab is active. It contains a class-based view for a single review:

```
70 #     return data
71
72
73 class SingleReviewView(DetailView):
74     template_name = "reviews/single_review.html"
75     model = Review
76
```

In the template that we use this data, we can use the name "object" or the model name that we got the data from all lower case.

In this example we get the data from the "Review" model therefor we use the name "review" in the template in order to get access to the data.

```
<> single_review.html X     ⚡ views.py     ⚡ urls.py

reviews > templates > reviews > <> single_review.htm
1   {% extends 'reviews/base.html' %} 
2   {% block title %} 
3   Review Detail 
4   {% endblock title %} 
5 
6   {% block content %} 
7       <h1>{{review.user_name}}</h1> 
8       <p>Rating: {{review.rating}}</p> 
9       <p>{{review.review_text}}</p> 
10  {% endblock %}
```

FormView:

```
<> single_review.html     ⚡ views.py ●     ⚡ urls.py     <>

reviews > ⚡ views.py > ...
1   from multiprocessing import context 
2   from re import template 
3   from django.shortcuts import render 
4   from django.http import HttpResponseRedirect 
5   from .forms import ReviewForm 
6   from django.views import View 
7   from django.views.generic.base import TemplateView 
8   from .models import Review 
9   from django.views.generic import ListView, DetailView 
10  from django.views.generic.edit import FormView 
11  # Create your views here. 
12
```

```
< single_review.html      views.py  X  urls.py      < review_list.html      < thank_y
reviews > views.py > ReviewView > form_valid
8   from .models import Review
9   from django.views.generic import ListView, DetailView
10  from django.views.generic.edit import FormView
11  # Create your views here.
12
13
14 class ReviewView(FormView):
15     form_class = ReviewForm
16     # which template to load
17     template_name = "reviews/review.html"
18     # to which url to go in case of a form been successfully submitted:
19     success_url = "/thank-you"
20
21     # what to do with form after a submition:
22     def form_valid(self, form):
23         form.save()
24         return super().form_valid(form)
25
26
```

CreateView:

```
< single_review.html      views.py  X  forms.py      urls.py      < review_
reviews > views.py > ReviewView
5   from django.http import HttpResponseRedirect
6   from .forms import ReviewForm
7   from django.views import View
8   from django.views.generic.base import TemplateView
9   from .models import Review
10  from django.views.generic import ListView, DetailView
11  from django.views.generic.edit import CreateView
12  # Create your views here.
13
14
15 class ReviewView(CreateView):
16     model = Review
17     form_class = ReviewForm
18     # which template to load
19     template_name = "reviews/review.html"
20     # to which url to go in case of a form been successfully submitted:
21     success_url = "/thank-you"
22
```

File Uploads:

Making the File Upload Work:

New Field named "FILES"

```
create_profile.html      views.py  X
profiles > views.py > ...
1  from django.shortcuts import render
2  from django.views import View
3  from django.http import HttpResponseRedirect
4
5  # Create your views here.
6
7
8  class CreateProfileView(View):
9      def get(self, request):
10         return render(request, "profiles/create_profile.html")
11
12     def post(self, request):
13         # Special property populated by Django which gives us access to uploaded files.
14         print(request.FILES["image"])
15         return HttpResponseRedirect("/profiles")
16
```

```
create_profile.html  views.py
profiles > templates > profiles > create_profile.html
1  {% load static %}
2
3  




4  <html lang="en">
5  <head>
6      <meta charset="UTF-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Create a Profile</title>
9      <link rel="stylesheet" href="{% static "profiles/styles.css" %}">
10     </head>
11     <body>
12         <form action="/profiles/" method="POST" enctype="multipart/form-data">
13             {% csrf_token %}
14             <input type="file" name="image"/>
15             <button>Upload!</button>
16         </form>
17     </body>
18 </html>
```

Using Models for File Storage:

```
create_profile.html      views.py      settings.py  ●  models.py  forms.py  X
profiles > forms.py > ...
1  from django.forms import forms
2
3
4  class ProfileForm(forms.Form):
5      user_image = forms.FileField()
6
```

```

profiles > templates > profiles > <> create_profile.html
1  {% load static %}
2
3  <!DOCTYPE html>
4  <html lang="en">
5  <head>
6      <meta charset="UTF-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Create a Profile</title>
9      <link rel="stylesheet" href="{% static "profiles/styles.css" %}">
10 </head>
11 <body>
12     <form action="/profiles/" method="POST" enctype="multipart/form-data">
13         {% csrf_token %}
14         {{form}}
15         <button>Upload!</button>
16     </form>
17 </body>
18 </html>

```

This “MEDIA_ROOT” setting will automatically be taken into account when a file is uploaded and move because of upload Filed:

The screenshot shows the VS Code interface with the settings.py file open. The line `MEDIA_ROOT = BASE_DIR / "uploads"` is highlighted with a red rectangle.

```

EXPLORER      ...
OPEN EDITORS   <> create_profile.html    views.py    settings.py X  models.py
feedback > settings.py > ...
112 USE_I18N = True
113
114 USE_TZ = True
115
116
117 # Static files (CSS, JavaScript, Images)
118 # https://docs.djangoproject.com/en/4.0/howto/static-files/
119
120 STATIC_URL = 'static/'
121
122 MEDIA_ROOT = BASE_DIR / "uploads"
123 # Default primary key field type
124 # https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field
125
126 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
127

```

The screenshot shows the VS Code interface with the models.py file open. The line `image = models.FileField(upload_to="images")` is highlighted with a red rectangle.

```

<> create_profile.html    views.py    settings.py    models.py X
profiles > models.py > Userprofile
1 from django.db import models
2
3 # Create your models here.
4
5
6 class UserProfile(models.Model):
7     """
8         This file will not be stored on the database.
9         FileFiled will (under the hood) (once we save a model with a file) is it will
10        take the file and move it somewhere in our disk and only store the path to the file.
11
12     image = models.FileField(upload_to="images")
13

```

Using an Imagefield:

This will make sure that the file that has been uploaded is a type of “image” -> “jpg”, “png” and more.

```

create_profile.html views.py models.py
profiles > models.py > UserProfile
1   from django.db import models
2
3   # Create your models here.
4
5
6   class UserProfile(models.Model):
7       """
8           This file will not be stored on the database.
9           FileField will (under the hood) (once we save a model with a file) is it will
10          take the file and move it somewhere in our disk and only store the path to the file.
11      """
12      image = models.ImageField(upload_to="images")
13

```

To use this we need to install a package "Pillow"

```

create_profile.html views.py models.py forms.py
profiles > forms.py > ProfileForm
1   from django.forms import forms
2
3
4   class ProfileForm(forms.Form):
5       user_image = forms.ImageField()
6

```

```

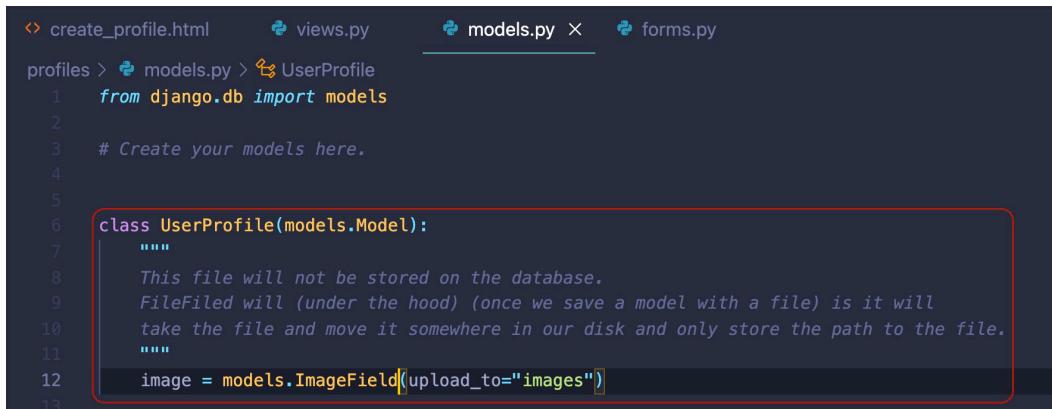
create_profile.html views.py models.py forms.py
profiles > views.py > CreateProfileView > post
1   from django.views import View
2   from django.http import HttpResponseRedirect
3   from .forms import ProfileForm
4   from .models import UserProfile
5   # Create your views here.
6
7
8
9   class CreateProfileView(View):
10      def get(self, request):
11          form = ProfileForm()
12          return render(request, "profiles/create_profile.html", {
13              "form": form,
14          })
15
16      def post(self, request):
17          submited_form = ProfileForm(request.POST, request.FILES)
18          if submited_form.is_valid():
19              # forms.py - UserProfile has a field name "user_image"
20              # the form is rendered by form class
21              profile = UserProfile(image=request.FILES["user_image"])
22              profile.save()
23              return HttpResponseRedirect("/profiles")
24          return render(request, "profiles/create_profile.html", {
25              "form": submited_form,
26          })
27

```

Using a CreateView:

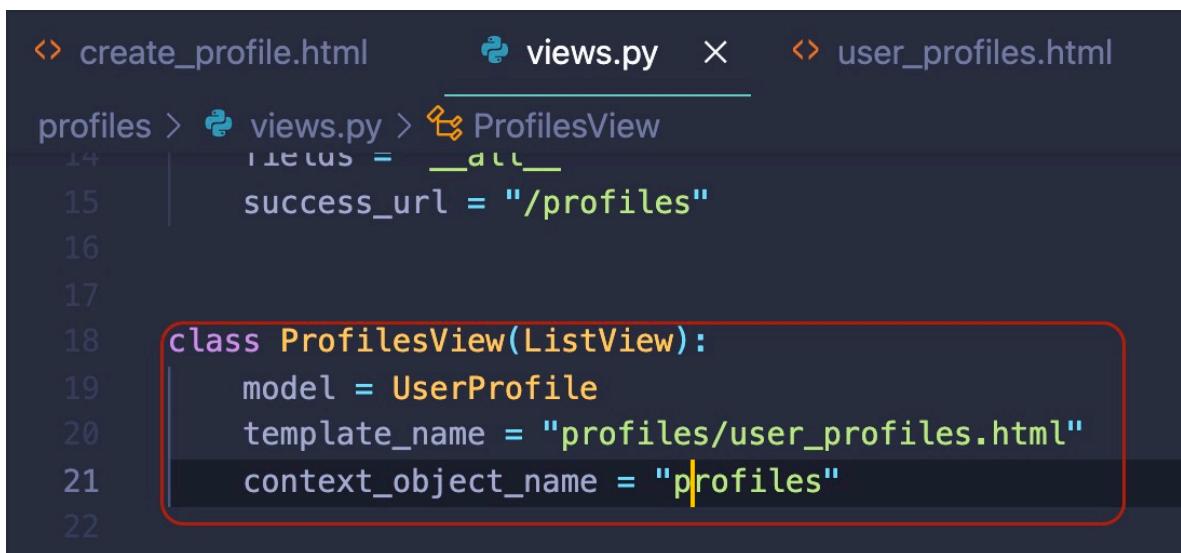
```
from django.shortcuts import render
from django.views import View
from django.http import HttpResponseRedirect
from .forms import ProfileForm
from .models import UserProfile
from django.views.generic.edit import CreateView
# Create your views here.
```

```
class CreateProfileView(CreateView):
    template_name = "profiles/create_profile.html"
    model = UserProfile
    fields = "__all__"
    success_url = "/profiles"
```



```
create_profile.html  views.py  models.py X  forms.py
profiles > models.py > UserProfile
1 from django.db import models
2
3 # Create your models here.
4
5
6 class UserProfile(models.Model):
7     """
8         This file will not be stored on the database.
9         FileField will (under the hood) (once we save a model with a file) is it will
10            take the file and move it somewhere in our disk and only store the path to the file.
11        """
12    image = models.ImageField(upload_to="images")
```

Working with the File Field:



```
create_profile.html  views.py X  user_profiles.html
profiles > views.py > ProfilesView
14    title = "__all__"
15    success_url = "/profiles"
16
17
18 class ProfilesView(ListView):
19     model = UserProfile
20     template_name = "profiles/user_profiles.html"
21     context_object_name = "profiles"
22
```

```

    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>User Profiles</title>
    </head>
    <body>
        <ul>
            {% for profile in profiles %}
            <li>
                
            </li>
            {% endfor %}
        </ul>
    </body>
</html>

```

Serving Uploaded Files:

```

MEDIA_ROOT = BASE_DIR / "uploads"
MEDIA_URL = "/user-media/"
# Default primary key field type

```

```

from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include("reviews.urls")),
    path("profiles/", include("profiles.urls")),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Sessions:

We generally use it to store a user-specific data.

A session is a “ongoing connection” between client (browser) and a server.

Enabling & Configuring Sessions:

```
single_review.html      settings.py ×
feedback > settings.py > ...
32
33 INSTALLED_APPS = [
34     'reviews',
35     'profiles',
36     'django.contrib.admin',
37     'django.contrib.auth',
38     'django.contrib.contenttypes',
39     'django.contrib.sessions', highlighted
40     'django.contrib.messages',
41     'django.contrib.staticfiles',
42 ]
43
44 MIDDLEWARE = [
45     'django.middleware.security.SecurityMiddleware',
46     'django.contrib.sessions.middleware.SessionMiddleware', highlighted
47     'django.middleware.common.CommonMiddleware',
48     'django.middleware.csrf.CsrfViewMiddleware',
49     'django.contrib.auth.middleware.AuthenticationMiddleware',
50     'django.contrib.messages.middleware.MessageMiddleware',
51     'django.middleware.clickjacking.XFrameOptionsMiddleware',
52 ]
53
54 ROOT_URLCONF = 'feedback.urls'
```

```
single_review.html ×  settings.py ×
feedback > settings.py > ...
118 # https://docs.djangoproject.com/en/4.0/howto/static-files/
119
120 STATIC_URL = 'static/'
121
122 MEDIA_ROOT = BASE_DIR / "uploads"
123 MEDIA_URL = "/user-media/"
124 # Default primary key field type
125 # https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field
126
127 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
128
129 # how long a session should survive
130 # the default is two weeks but, we set another value. the value we set is in seconds
131 # SESSION_COOKIE_AGE = 120
132
```

How we can utilize sessions:

```
single_review.html X settings.py
reviews > templates > reviews > single_review.html
1  {% extends 'reviews/base.html' %} 
2  {% block title %} 
3  Review Detail
4  {% endblock title %} 
5
6  {% block content %} 
7      <h1>{{review.user_name}}</h1>
8      <p>Rating: {{review.rating}}</p>
9      <p>{{review.review_text}}</p>
10     <form action="/reviews/favorite" method="POST">
11         <input type="hidden" name="review_id" value="{{ review.id }}">
12         <button>Favorite</button>
13     </form>
14  {% endblock %}
```

```
single_review.html urls.py X views.py
reviews > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path("", views.ReviewView.as_view()),
6      path("thank-you", views.ThankYouView.as_view()),
7      path("reviews", views.ReviewsListView.as_view()),
8      path("reviews/favorite", views.AddFavoriteView.as_view()),
9      path("reviews/<int:pk>", views.SingleReviewView.as_view())
10 ]
11
```

```
single_review.html urls.py X views.py
reviews > views.py > AddFavoriteView > post
20  # to which url to go in case of a form been successfully submitted:
21  success_url = "/thank-you"
22
23
24  class AddFavoriteView(View):
25      def post(self, request):
26          review_id = request.POST['review_id']
27          fav_review = Review.objects.get(pk=review_id)
```

Storing Data in Sessions:

```

single_review.html      urls.py      views.py ×
reviews > views.py > AddFavoriteView > post
20     # to which url to go in case of a form been successfully submitted
21     success_url = "/thank-you"
22
23
24 class AddFavoriteView(View):
25     def post(self, request):
26         review_id = request.POST["review_id"]
27         fav_review = Review.objects.get(pk=review_id)
28         # access this property to add data or to get data.
29         request.session["favorite_review"] = fav_review
30         return HttpResponseRedirect("/reviews/" + review_id)
31

```

Which Kind of Data Should be Stored:

We do not want to store objects in sessions (like the above example) instead we would like to store strings (or number or dic ... primitive).

Using Session Data:

```

single_review.html ×      urls.py      views.py
reviews > templates > reviews > single_review.html
1  {% extends 'reviews/base.html' %} 
2  {% block title %} 
3  Review Detail
4  {% endblock title %} 
5
6  {% block content %} 
7      <h1>{{review.user_name}}</h1>
8      <p>Rating: {{review.rating}}</p>
9      <p>{{review.review_text}}</p>
10     {% if is_favorite %} 
11         <p>This is my favorite!</p>
12     {% else %} 
13         <form action="/reviews/favorite" method="POST">
14             {% csrf_token %} 
15             <input type="hidden" name="review_id" value="{{ review.id }}">
16             <button>Favorite</button>
17         </form>
18     {% endif %} 
19  {% endblock %} 

```

```
single_review.html      urls.py      views.py  X
reviews > views.py > SingleReviewView > get_context_data
50     #     base_query = super().get_queryset()
51     #     data = base_query.filter(rating__gt=1)
52     #     return data
53
54
55 class SingleReviewView(DetailView):
56     template_name = "reviews/single_review.html"
57     model = Review
58
59     def get_context_data(self, **kwargs):
60         context = super().get_context_data(**kwargs)
61         loaded_review = self.object
62         request = self.request
63         favorite_id = request.session["favorite_review"]
64         context["is_favorite"] = favorite_id == loaded_review.id
65
66         return context
```

When we store "favorite_review" we store it as string therefore the comparison we do - "favorite_id == loaded_review.id" will always Fail.

To fix this we need to convert the type as in the example below:

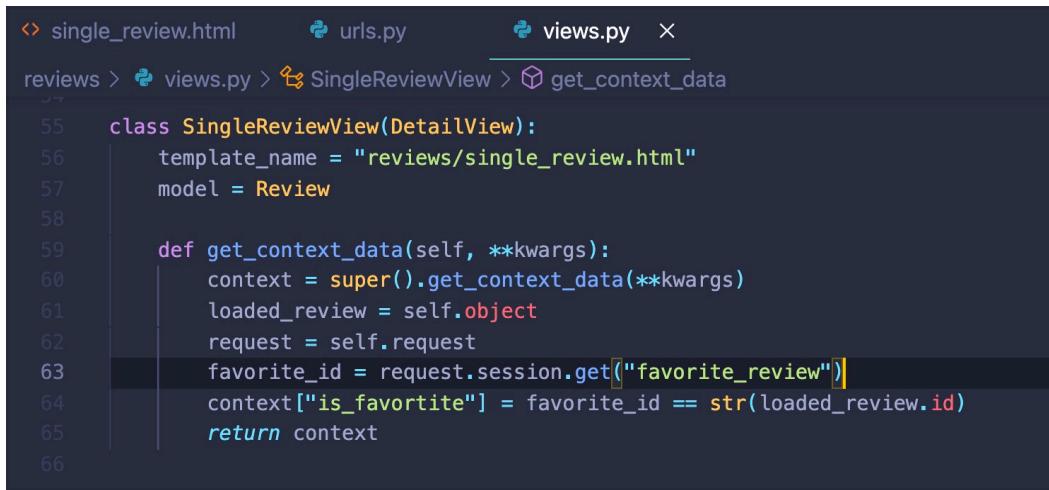
```
class SingleReviewView(DetailView):
    template_name = "reviews/single_review.html"
    model = Review

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        loaded_review = self.object
        request = self.request
        favorite_id = request.session["favorite_review"]
        context["is_favorite"] = favorite_id == str(loaded_review.id)
        return context
```

Safely Accessing Session Data:

A safer way to get session data (if we entered the browser for the first time and didn't add the 'favorite_review' in this case, than we can face an error trying to get

This data which doesn't exist). **We use the get() method**



A screenshot of a code editor showing a Python file named `views.py`. The file contains code for a `SingleReviewView` class, which is a `DetailView`. It sets the template name to `"reviews/single_review.html"` and the model to `Review`. The `get_context_data` method is overridden to add context data. Specifically, it calls `super().get_context_data(**kwargs)`, retrieves the loaded review from `self.object`, gets the favorite review ID from the session, and adds a key `"is_favorite"` to the context with a value of `True` if the loaded review's ID matches the session's favorite review ID. The code editor interface shows tabs for `single_review.html`, `urls.py`, and `views.py`, with `views.py` being the active tab.

```
55  class SingleReviewView(DetailView):
56      template_name = "reviews/single_review.html"
57      model = Review
58
59      def get_context_data(self, **kwargs):
60          context = super().get_context_data(**kwargs)
61          loaded_review = self.object
62          request = self.request
63          favorite_id = request.session.get("favorite_review")
64          context["is_favorite"] = favorite_id == str(loaded_review.id)
65
66          return context
```

Section 15 - Deployment: