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 ● urls.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 × urls.py

reviews > views.py > ...

```
38  #         if form.is_valid():
39  #             form.save()
40  #             return HttpResponseRedirect("/th
41  #     else:
42  #         form = ReviewForm()
43  #     return render(request, "reviews/review.h
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)

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

    # If we need to interpulate data in the template
    # we retutn a dictionary - keys -  are value names , vlaues 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.h

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

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 > ...

```python
from multiprocessing import context
from re import template
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ReviewForm
from django.views import View
from django.views.generic.base import Template
from .models import Review
from django.views.generic import ListView
# Create your views here.
```

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

views.py × single_review.html urls.py

reviews > views.py > ReviewsListView

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

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

```
{% extends 'reviews/base.html' %}

{% block title %}
All Reviews
{% endblock title %}
{% block content %}
    <ul>
        {% for review in object_list %}
        <li>{{review.user_name}} (Rating: {{review.rating}})</li>
        {% endfor %}
    </ul>
{% 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

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

    def get_queryset(self):
        base_query = super().get_queryset()
        data = base_query.filter(rating__gt=1)
        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 on e 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.

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 × views.py urls

reviews > templates > reviews > single_review.htm

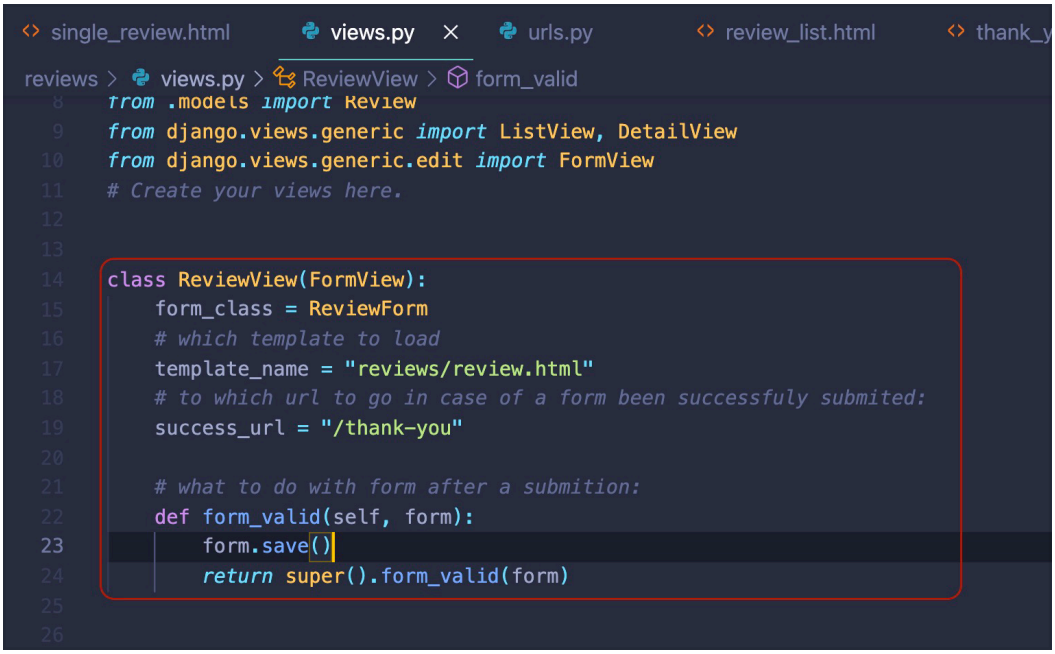
```
{% extends 'reviews/base.html' %}
{% block title %}
Review Detail
{% endblock title %}

{% block content %}
    <h1>{{review.user_name}}</h1>
    <p>Rating: {{review.rating}}</p>
    <p>{{review.review_text}}</p>
{% endblock  %}
```

FormView:

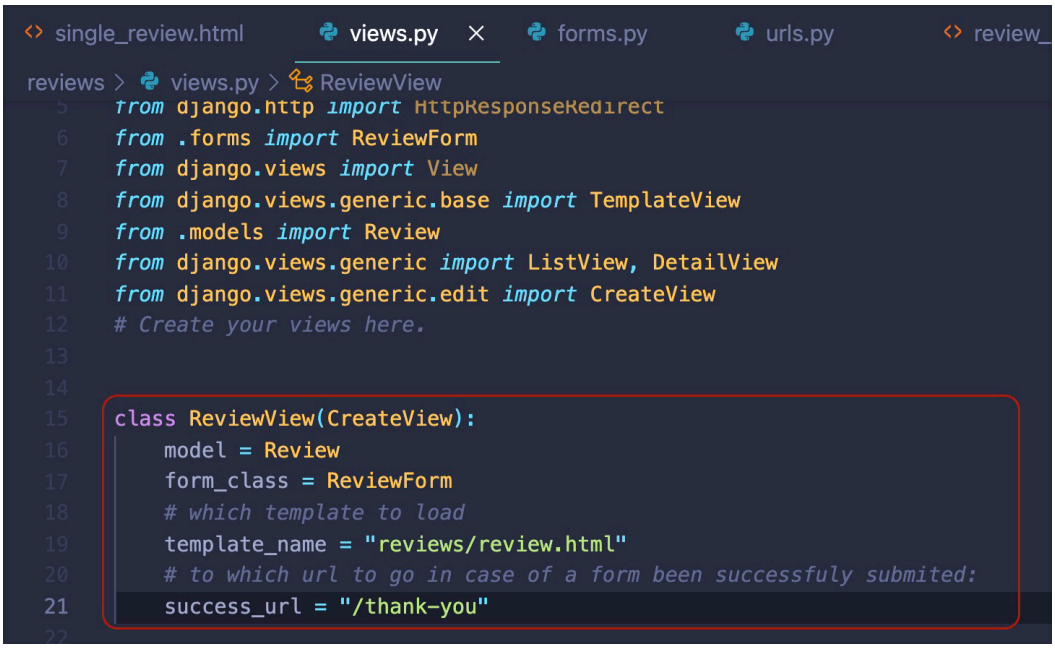
single_review.html views.py ● urls.py

reviews > views.py > ...

```
from multiprocessing import context
from re import template
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ReviewForm
from django.views import View
from django.views.generic.base import TemplateView
from .models import Review
from django.views.generic import ListView, DetailVie
from django.views.generic.edit import FormView
# Create your views here.
```



CreateView:



File Uploads:

Making the File Upload Work:

New Field named “FILES”

create_profile.html views.py

profiles > views.py > ...

```python
from django.shortcuts import render
from django.views import View
from django.http import HttpResponseRedirect

# Create your views here.


class CreateProfileView(View):
    def get(self, request):
        return render(request, "profiles/create_profile.html")

    def post(self, request):
        # Special property populated by Django which gives us access to uploaded files.
        print(request.FILES["image"])
        return HttpResponseRedirect("/profiles")

```

create_profile.html views.py

profiles > templates > profiles > create_profile.html

```python
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Create a Profile</title>
  <link rel="stylesheet" href="{% static "profiles/styles.css" %}">
</head>
<body>
  <form action="/profiles/" method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <input type="file" name="image"/>
    <button>Upload!</button>
  </form>
</body>
</html>
```

Using Models for File Storage:

create_profile.html views.py settings.py models.py forms.py

profiles > forms.py > ...

```python
from django.forms import forms


class ProfileForm(forms.Form):
    user_imgage = forms.FileField()

```

create_profile.html × views.py settings.py ● models.py

profiles > templates > profiles > create_profile.html

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Create a Profile</title>
  <link rel="stylesheet" href="{% static "profiles/styles.css" %}">
</head>
<body>
  <form action="/profiles/" method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    {{form}}
    <button>Upload!</button>
  </form>
</body>
</html>
```

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

create_profile.html views.py settings.py models.py ×

profiles > models.py > UserProfile

```
from django.db import models

# Create your models here.


class UserProfile(models.Model):
    """
    This file will not be stored on the database.
    FileFiled will (under the hood) (once we save a model with a file) is it will
    take the file and move it somewhere in our disk and only store the path to the file.
    """
    image = models.FileField(upload_to="images")
```

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

```python
from django.db import models

# Create your models here.


class UserProfile(models.Model):
    """
    This file will not be stored on the database.
    FileFiled will (under the hood) (once we save a model with a file) is it will
    take the file and move it somewhere in our disk and only store the path to the file.
    """
    image = models.ImageField(upload_to="images")

```

To use this we need to install a package “Pillow”

create_profile.html views.py models.py forms.py ×

profiles > forms.py > ProfileForm

```python
from django.forms import forms


class ProfileForm(forms.Form):
    user_image = forms.ImageField()

```

create_profile.html views.py × models.py forms.py

profiles > views.py > CreateProfileView > post

```python
from django.views import View
from django.http import HttpResponseRedirect
from .forms import ProfileForm
from .models import UserProfile
# Create your views here.


class CreateProfileView(View):
    def get(self, request):
        form = ProfileForm()
        return render(request, "profiles/create_profile.html", {
            "form": form,
        })

    def post(self, request):
        submites_form = ProfileForm(request.POST, request.FILES)
        if submites_form.is_valid():
            # forms.py - UserProfile has a field name "user_image"
            # the form is rendered by form class
            profile = UserProfile(image=request.FILES["user_image"])
            profile.save()
            return HttpResponseRedirect("/profiles")
        return render(request, "profiles/create_profile.html", {
            "form": submites_form,
        })

```

Using a CreateView:

```python
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 | forms.py

profiles > models.py > UserProfile

```python
from django.db import models

# Create your models here.


class UserProfile(models.Model):
    """
    This file will not be stored on the database.
    FileFiled will (under the hood) (once we save a model with a file) is it will
    take the file and move it somewhere in our disk and only store the path to the file.
    """
    image = models.ImageField(upload_to="images")
```

Working with the File Field:

create_profile.html | views.py | user_profiles.html

profiles > views.py > ProfilesView

```python
    fields = __all__
    success_url = "/profiles"


class ProfilesView(ListView):
    model = UserProfile
    template_name = "profiles/user_profiles.html"
    context_object_name = "profiles"
```

create_profile.html views.py user_profiles.html × models.py

profiles > templates > profiles > user_profiles.html

```
<!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>
            <img src="{{ profile.image.url }}" alt="">
        </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Serving Uploaded Files:

views.py settings.py × urls.py

feedback > settings.py > ...

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

views.py urls.py × user_profiles.html models.py

feedback > urls.py > ...

```
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from turtle import st
from django.contrib import admin
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 genrally 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',
40      'django.contrib.messages',
41      'django.contrib.staticfiles',
42  ]
43
44  MIDDLEWARE = [
45      'django.middleware.security.SecurityMiddleware',
46      'django.contrib.sessions.middleware.SessionMiddleware',
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
```

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 seesion 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 settings.py

reviews > templates > reviews > single_review.html

```
{% extends 'reviews/base.html' %}
{% block title %}
Review Detail
{% endblock title %}

{% block content %}
    <h1>{{review.user_name}}</h1>
    <p>Rating: {{review.rating}}</p>
    <p>{{review.review_text}}</p>
    <form action="/reviews/favorite" method="POST">
        <input type="hidden" name="review_id" value="{{ review.id }}">
        <button>Favorite</button>
    </form>
{% endblock  %}
```

single_review.html urls.py views.py

reviews > urls.py > ...

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

urlpatterns = [
    path("", views.ReviewView.as_view()),
    path("thank-you", views.ThankYouView.as_view()),
    path("reviews", views.ReviewsListView.as_view()),
    path("reviews/favorite", views.AddFavoriteView.as_view()),
    path("reviews/<int:pk>", views.SingleReviewView.as_view())
]
```

single_review.html urls.py views.py

reviews > views.py > AddFavoriteView > post

```
    # to which url to go in case of a form been successfuly submited:
    success_url = "/thank-you"


class AddFavoriteView(View):
    def post(self, request):
        review_id = request.POST['review_id']
        fav_review = Review.objects.get(pk=review_id)
```

Storing Data in Sessions:

single_review.html urls.py views.py

reviews > views.py > AddFavoriteView > post

```python
    # to which url to go in case of a form been successfuly submi
    success_url = "/thank-you"


class AddFavoriteView(View):
    def post(self, request):
        review_id = request.POST["review_id"]
        fav_review = Review.objects.get(pk=review_id)
        # access this property to add data or to get data.
        request.session["favorite_review"] = fav_review
        return HttpResponseRedirect("/reviews/" + review_id)
```

Which Kind of Data Should be Stored:

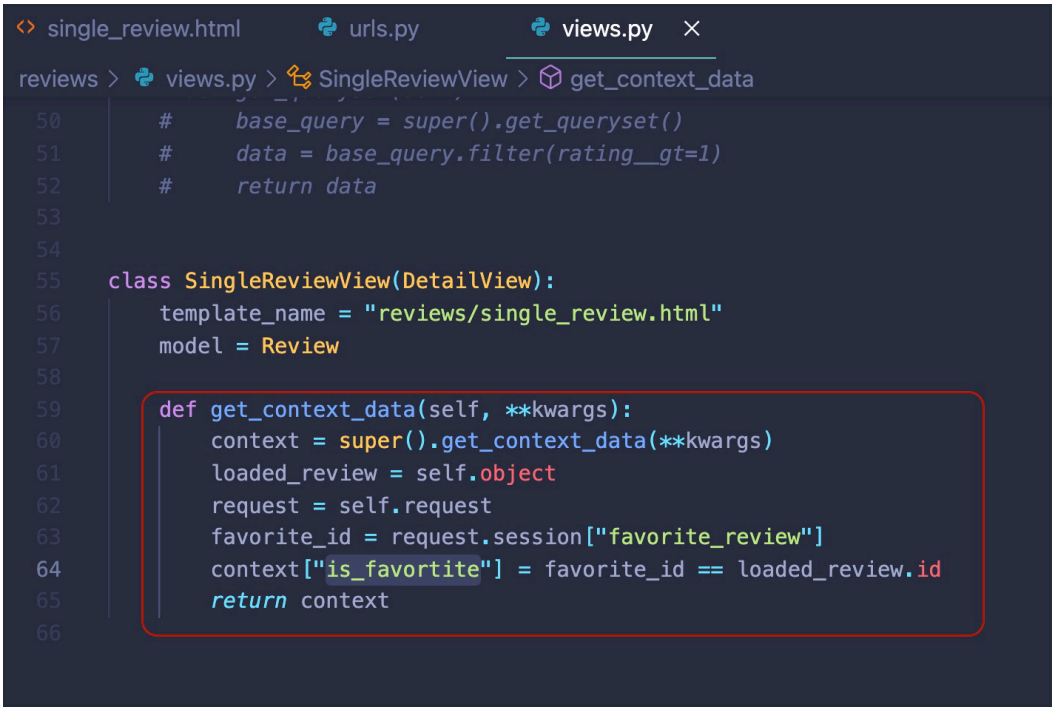
We do not want to store objects in sessions (like the above example) instead we would like to store strings (or number or dic ...primitive).

Using Session Data:

single_review.html urls.py views.py

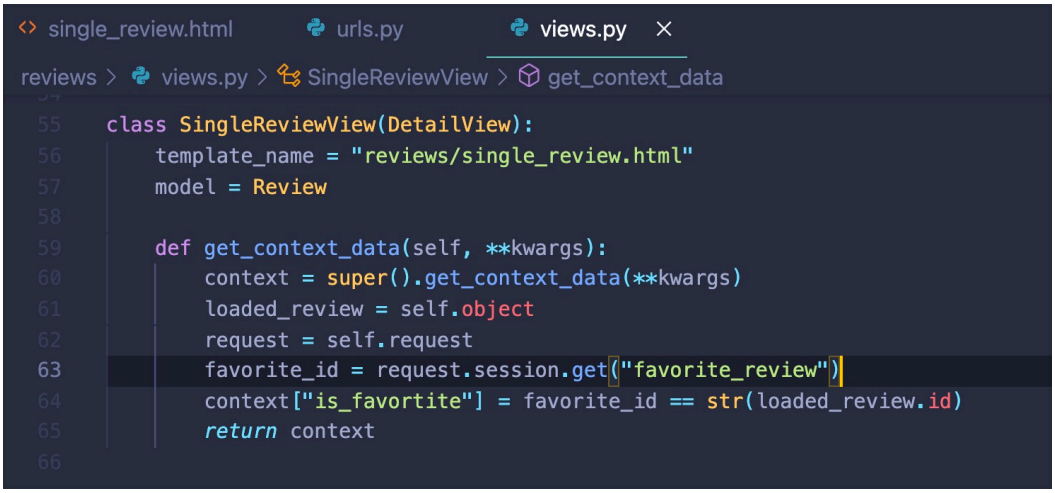
reviews > templates > reviews > single_review.html

```
{% extends 'reviews/base.html' %}
{% block title %}
Review Detail
{% endblock title %}

{% block content %}
    <h1>{{review.user_name}}</h1>
    <p>Rating: {{review.rating}}</p>
    <p>{{review.review_text}}</p>
    {% if is_favortite %}
    <p>This is my favorite!</p>
    {% else %}
    <form action="/reviews/favorite" method="POST">
        {% csrf_token %}
        <input type="hidden" name="review_id" value="{{ review.id }}">
        <button>Favorite</button>
    </form>
    {% endif %}
{% endblock  %}
```



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:

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**



Section 15 - Deployment: