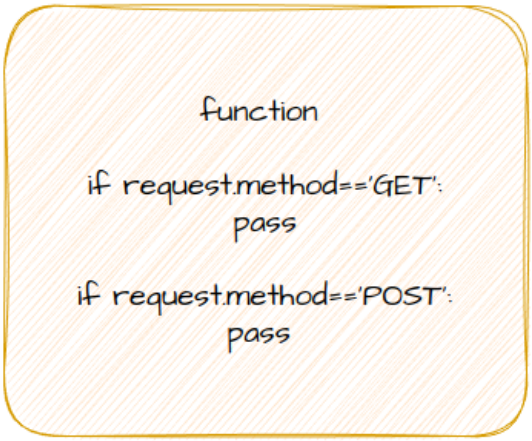
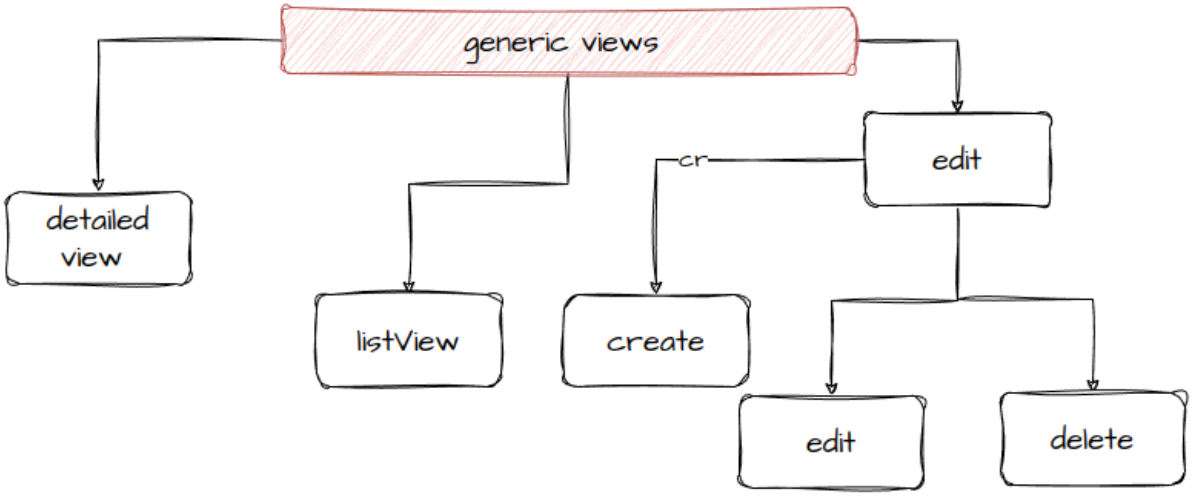
Class Based Views (Create, Update, and Delete)

Created	@October 15, 2023 12:01 AM





List View

-youtube listinng your subscription videos

-Web page listing all blogs / posts / products

Detailed View

when clicking on a product from the list, a detailed page about the product is displayed

Update View & Delete View

ability to update blogs // videos

note : **update view can also be called edit view**

Steps to create a List View :

1) Add a class based list view in Views.py

---

1) Add a class based List View :

First : import it from the views.generic

Second : Add a class that inherits from ListView

- **give it the model name**
- **the template name you're gonna use as your new home page bc django server expects a template by default and its naming convention is app/model_viewtype.html**

  for ex : pages/Product_list.html
- **the context object : the variable name that we're gonna loop over**
- **the ordering (optional) : if you wanna load your items in a certain order .. the minus (-) means from newest to oldest aka newest at the top & oldest at the bottom**

```
def home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html', context)


class PostListView(ListView):
    model = Post
    template_name = 'blog/home.html' # <app>/<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']
```

Steps to create a detailed view

1) import it from the django.controb.views

in views.py

```
from django.shortcuts import render
from django.views.generic import ListView, DetailView
from .models import Post
```

```
class PostDetailView(DetailView):
    model = Post
```

2) add the link in the app urls.py

but don't forget to first import it

```python
from django.urls import path
from .views import PostListView, PostDetailView
from . import views

urlpatterns = [
```

```python
urlpatterns = [
    path('', PostListView.as_view(), name='blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('about/', views.about, name='blog-about'),
]
```

3) add template post_detail.html

with the naming convention : app/model_viewtype.html

and every object like post,title or whateeveris changed to object.title and so on

```
{% extends "blog/base.html" %}
{% block content %}
  <article class="media content-section">
    <img class="rounded-circle article-img" src="{{ object.author.profile
    <div class="media-body">
      <div class="article-metadata">
        <a class="mr-2" href="#">{{ object.author }}</a>
        <small class="text-muted">{{ object.date_posted|date:"F d, Y" }}
      </div>
      <h2 class="article-title">{{ object.title }}</h2>
      <p class="article-content">{{ object.content }}</p>
    </div>
  </article>
```

and now you can change the link of your old details.html page

```
href="{% url 'post-detail' post.id %}">{{ post.title }}</a></h
{ post.content }}</p>
```

Steps to use create view

1) Views

2) App Urls

3) Template holding both create & update

4) Models

1) Views

- import them
- create a class that inherits from CreateView
- give it the fields that you wanna have in your form (title, content)

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView
)
```

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']
```

2) App Urls

import postdetailview and add its link

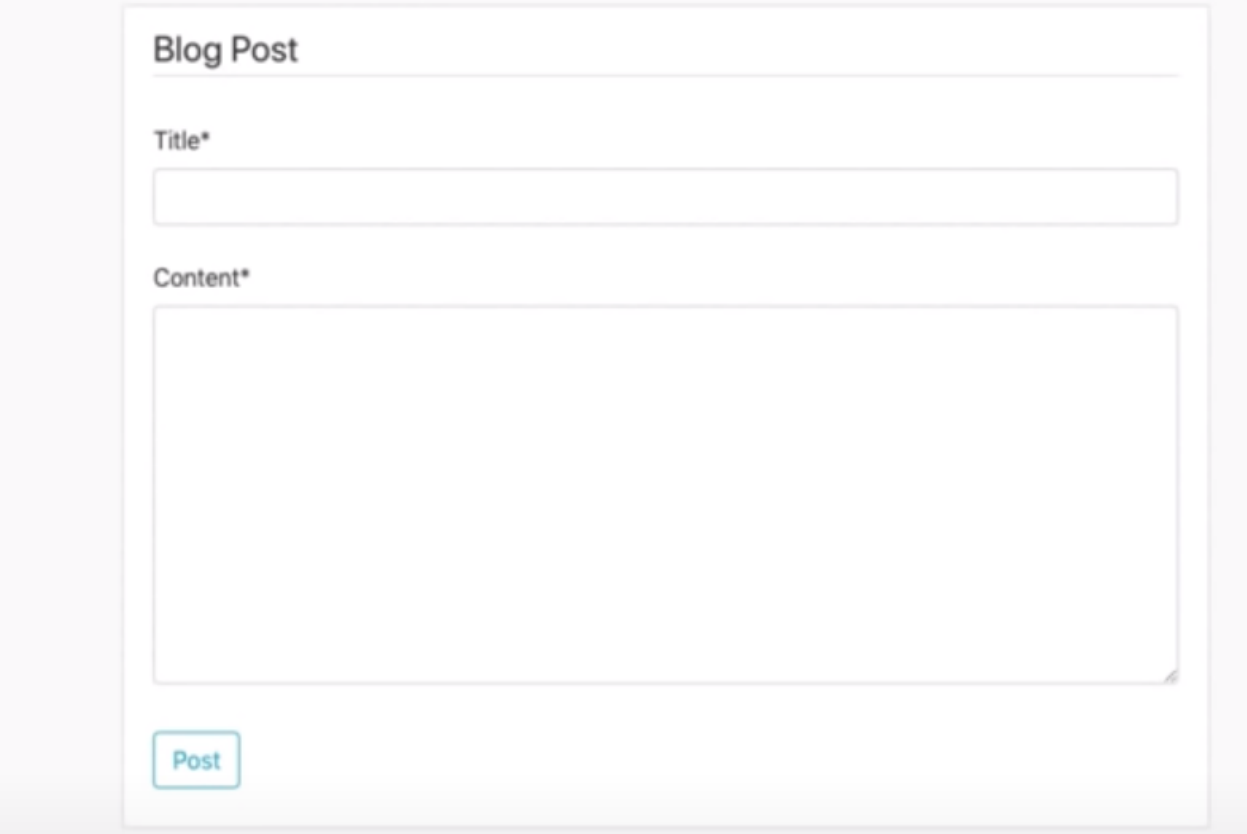
```
from django.urls import path
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView
)
from . import views

urlpatterns = [
    path('', PostListView.as_view(), name='blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-create'),
    path('about/', views.about, name='blog-about'),
]
```

3) Template holding both create & update

the same form we used a lot before but change the names to blog post or whatever the name you wanna be displayed

```
{% load crispy_forms_tags %}
{% block content %}
    <div class="content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Blog Post</legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Post
            </div>
        </form>
    </div>
{% endblock content %}
```

the result :



4) Updating Views by setting the author as the current user

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

still you'll get an error because the server can't find the url to the model object you're referring to, so you'll have to edit it in the model by creating a get absolute url method that returns the path to any specific instance

5) Models

-First : get url of a particular route , so you'll need to use the reverse function

---

Note : Difference between redirect and reverse :

redirect —> Actually redirects you to a specific route

reverse —> Simply returns the full URL to that route as a string

---

-And in our situation, we just want the URL as a string, and the views is gonna handle the redirect for us.

so ..

- **first import reverse**

- **create the get_absolute_url method to tell django to return the full path as a string**

  (remember it needs a specific post with a primary key, so set

  kwargs={'pk': self.pk}

  so the instance of a post is a primary key

and it will direct you to the details page you just created

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    date_posted = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('post-detail', kwargs={'pk': self.pk})
```

A login mixin

In your views.py :

1. **import it**

from django.contrib.auth.mixins import LoginRequiredMixin

so user is directed to login page whenever he opens the site

```python
from django.shortcuts import render
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import (
    ListView,
    DetailView,
    CreateView
)
```

2. **inherit it in the begging on the left in your create view**

```python
class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

Steps to use update view :

1) Views

2) App Urls

3) Same Create View Form Template

4) Check if user updating is the actual author of the post

1) Views

- **import it**

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView
)
```

- **create the class (similar to the create view)**

```
class PostUpdateView(LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)
```

2) App urls

- import it

```
from django.urls import path
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView,

)
from . import views
```

- **include its path**

(here, it’s similar to the detail view with the int:pk ..)

include(’post/<int:pk>/update/’, PostUpdateView.as_view(), name=’post-update’),

```
erns = [
h('', PostListView.as_view(), name='blog-home'),
h('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
h('post/new/', PostCreateView.as_view(), name='post-create'),
h('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
h('about/', views.about, name='blog-about'),
```

3) Same Create View Form Template

4) Check if user updating is the actual author

- **using another mixin called UserPassesTestMixin**

```
django.shortcuts import render
django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
django.views.generic import (
istView,
etailView,
reateView,
```

- **make your update view class inherit form it (and it MUST BE ON THE LEFT of the UpdateView)**

```
ass PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
  model = Post
  fields = ['title', 'content']

  def form_valid(self, form):
      form.instance.author = self.request.user
```

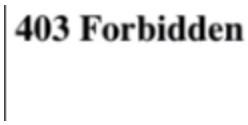
- **create a test function to test the mixin**

  -to get the post you're currently trying to update

  -check if the current user is the author of the post

```python
class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

    def form_valid(self, form):
        form.instance.author = self.request.user
        return super().form_valid(form)

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

and now, if you try to update another user's post, you'll get a 403 Forbidden message



Steps to use delete view

(similar to detail view)

1) Views

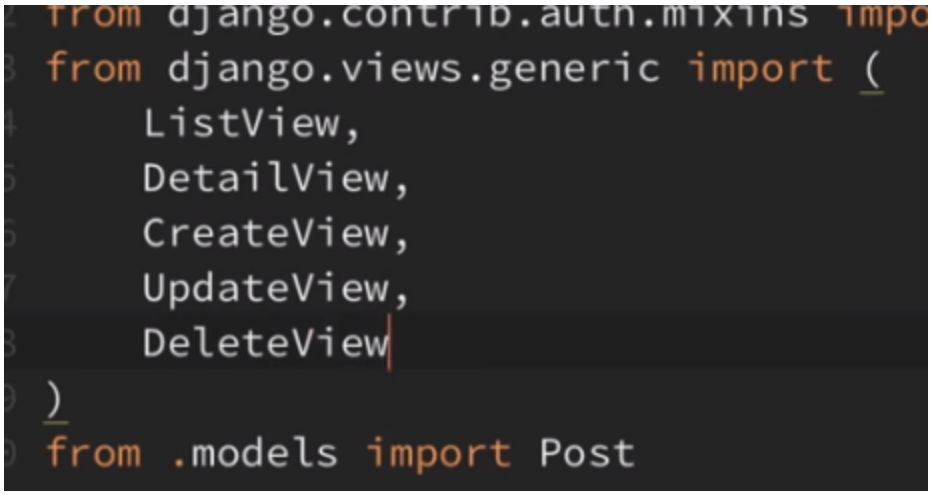
2) App Urls

3) Template : post_confirm_delete.html

4) Update Navbar

1) Views

- import



- **class delete view**

(like detail view in the model only , but takes the same mixin as the update view and the test_func)

```
class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

- Provide success url for after deleting a post

success_url = '/' .. the slash is the url of home page

```python
class PostDeleteView(LoginRequiredMixin, UserPassesTestMixin, DeleteView):
    model = Post
    success_url = '/'

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False
```

2) App Urls

import it above and write it path with the same int:pk ..

```python
PostDeleteView

. import views

atterns = [
path('', PostListView.as_view(), name='blog-home'),
path('post/<int:pk>/', PostDetailView.as_view(), name='post-detail'),
path('post/new/', PostCreateView.as_view(), name='post-create'),
path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete'),
path('about/', views.about, name='blog-about'),
```

3) Template : post_confirm_delete.html

-copy paste our normal forms and change the desired values

-Here we don't need the form, so we can delete the crispy forms tags .. it's just a page to confirm our delete post and put the title of the post : {{ object.title }}

- **create an href that takes you back to the detailed view of this exact post,**
- **we can put href = "{% url ' post-detail' object.id %}**

and pass in the primary key so we can create a URL of the post detail page for this post ID

---

4) Update Navbar & post_detail.html

In Navbar, add a **link to create a new post**

```
-- Navbar Right Side -->
iv class="navbar-nav">
{% if user.is_authenticated %}
  <a class="nav-item nav-link" href="{% url 'post-create' %}">New Post</a>
  <a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>
  <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
{% else %}
  <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>
  <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>
{% endif %}
```

In the post_Details.html template, add **links to update & delete the posts in the post_detail.html**

for update :

```
ounded-circle article-img" src="{{ object.author.profile.image.url }}">
edia-body">
'article-metadata">
'mr-2" href="#">{{ object.author }}</a>
ss="text-muted">{{ object.date_posted|date:"F d, Y" }}</small>
ct.author == user %}
="btn btn-secondary btn-" href="{% url 'post-update' object.id %}">Update</a
}
                    btn-secondary

rticle-title">{{ object.title }}</h2>
ticle-content">{{ object.content }}</p>
```

for delete :

```
class="mr-2" href="#">{{ object.author }}</a>
all class="text-muted">{{ object.date_posted|date:"F d, Y" }}</small>
if object.author == user %}
a class="btn btn-secondary btn-sm mt-1 mb-1" href="{% url 'post-update' obje
a class="btn btn-danger btn-sm mt-1 mb-1" href="{% url 'post-delete' object.
endif %}
>
lass="article-title">{{ object.title }}</h2>
```