

When and how to use Django CreateView

When to use CreateView?

Django provides several class based generic views to accomplish common tasks. One among them is CreateView.

CreateView should be used when you need a form on the page and need to do a db insertion on submission of a valid form.

CreateView is better than vanilla View

We will first write a vanilla view by subclassing **View**, and then modify the view to subclass **CreateView** instead of **View**.

CreateView is better than vanilla View in following ways:

- Avoid boilerplate code
- Succinct and more maintainable code.

Vanilla View

We want to create a page with a book creation form.

```
# books/models.py
class Book(models.Model):
    title = models.CharField(max_length=100)
    isbn = models.CharField(max_length=100, unique=True)
```

Agiliq

```
        return self.title

# books/forms.py
class BookCreateForm(forms.ModelForm):
    class Meta:
        model = Book
```

Vanilla view looks like:

```
books/views.py
class BookCreateView(CreateView):
    def get(self, request, *args, **kwargs):
        context = {'form': BookCreateForm()}
        return render(request, 'books/book-create.html', context)

    def post(self, request, *args, **kwargs):
        form = BookCreateForm(request.POST)
        if form.is_valid():
            book = form.save()
            book.save()
            return HttpResponseRedirect(reverse_lazy('books:detail', args=[book.id]))
        return render(request, 'books/book-create.html', {'form': form})
```

Template code looks like:

```
<!--books/templates/books/book-create.html-->

<form action="." method="POST">

{% csrf_token %}

<table>

</table>
```

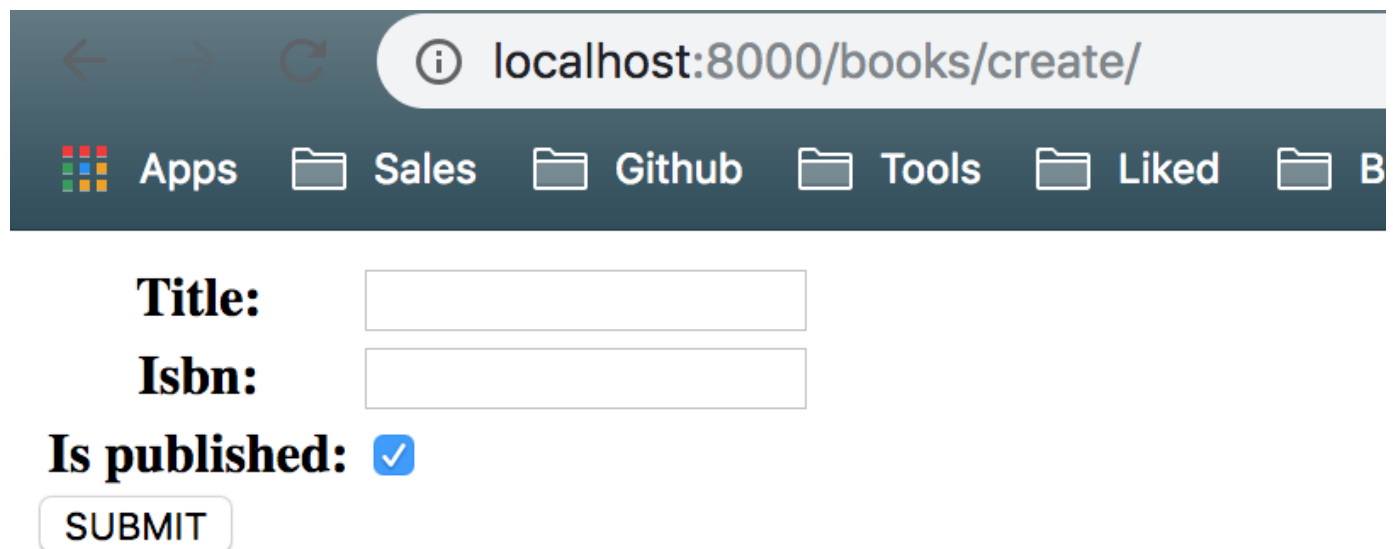
Agiliq

With proper urlpattern, you should be able to see the book creation form.

```
from django.urls import path

from . import views

app_name = 'books'
urlpatterns = [
    path('create/', views.BookCreateView.as_view(), name='create'),
    path('<int:pk>/', views.BookDetailView.as_view(), name='detail'),
]
```



← → ↻ ⓘ localhost:8000/books/create/

Apps Sales Github Tools Liked B

Title:

Isbn:

Is published: ☒

Using CreateView

Vanilla view has a lot of boilerplate code.

Any object creation view will have a `get()` implementation for creating context and rendering the response. Similarly object creation view will have a `post()` implementation to do `.save()`. CreateView, which is a generic class based view, can avoid this boilerplate code.

```
class BookCreateView(CreateView):
    template_name = 'books/book-create.html'
```

Agiliq

This change also needs that a `get_absolute_url()` be defined on the object which is being created.

So we need to provide a `get_absolute_url()` on model Book.

```
class Book(models.Model):  
    # More code  
    def get_absolute_url(self):  
        return reverse('books:detail', args=[self.id])
```

Refresh the page and you should still be able to achieve everything that was possible with vanilla view.

As you would have noticed, using a CreateView helped us avoid boilerplate `get()` and `post()` implementation. The code looks much more succinct as it only has few class attributes and there isn't any function implementation.

Adding initial data to CreateView

Assume we want to populate form's `title` field with some initial data.

Modify BookCreateView to look like:

```
class BookCreateView(CreateView):  
    template_name = 'books/book-create.html'  
    form_class = BookCreateForm  
  
    def get_initial(self, *args, **kwargs):  
        initial = super(BookCreateView, self).get_initial(**kwargs)  
        initial['title'] = 'My Title'  
        return initial
```

This code has better separation of concern. There is a separate method for dealing with initial data.

Had we used a vanilla view, initial data code would have been part of `get()`.

Adding form kwargs to CreateView

Let's add a `user` field to Book to track the user who creates a Book.

Agiliq

```
isbn = models.CharField(max_length=100, unique=True)
is_published = models.BooleanField(default=True)
user = models.ForeignKey(User, on_delete=models.CASCADE, **NULL_AND_BLANK)

def __str__(self):
    return self.title

def get_absolute_url(self):
    return reverse('books:detail', args=[self.id])
```

Assume you don't want to allow a user to create two books with same title. The title should be unique per user.

This validation needs writing a `clean_title()` method which would look like:

```
class BookCreateForm(forms.ModelForm):
    class Meta:
        model = Book
        exclude = ('user',)

    def __init__(self, *args, **kwargs):
        self.user = kwargs.pop('user')
        super(BookCreateForm, self).__init__(*args, **kwargs)

    def clean_title(self):
        title = self.cleaned_data['title']
        if Book.objects.filter(user=self.user, title=title).exists():
            raise forms.ValidationError("You have already written a book with :")
        return title
```

This needs that a `user` be supplied from view during form creation. This is where `CreateView.get_form_kwargs()` come into picture. Modify the view to look like:

```
class BookCreateView(CreateView):
    template_name = 'books/book-create.html'
```

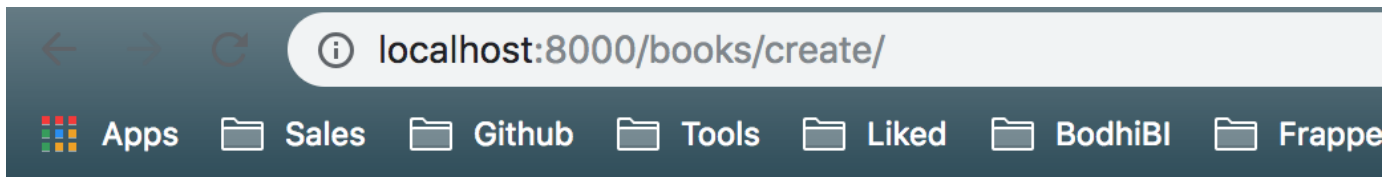
Agiliq

```
self.object = form.save(commit=False)
self.object.user = self.request.user
self.object.save()
return HttpResponseRedirect(self.get_success_url())

def get_initial(self, *args, **kwargs):
    initial = super(BookCreateView, self).get_initial(**kwargs)
    initial['title'] = 'My Title'
    return initial

def get_form_kwargs(self, *args, **kwargs):
    kwargs = super(BookCreateView, self).get_form_kwargs(*args, **kwargs)
    kwargs['user'] = self.request.user
    return kwargs
```

After this any logged in user wouldn't be able to create two Books with same title.



- You have already written a book with same title.

Title:

Isbn:

Is published: ☒

Our other posts on generic class views

- [TemplateView](#)
- [ListView](#)
- [DetailView](#)
- [FormView](#)

Agiliq

We love building amazing apps for web and mobile for our clients. If you are looking for development help, [contact us today](#) ☒.

Would you like to download 10+ free Django and Python books? [Get them here](#)



Agiliq

 Upvote

 Funny

 Love

 Surprised

 Angry

 Sad

0

0

0

0

0

0

Comments and reactions for this thread are now closed.

×

0 Comments

The Agiliq Blog

1

Login ▾

 Recommend

 Tweet

 Share


Sort by Oldest ▾

This discussion has been closed.

ALSO ON THE AGILIQ BLOG

Getting started with Celery and Redis - Agiliq Blog | Django web app ...


4 years ago



kamal kumar jeldi — Hi Akshar,This is an excellent tut for beginners. but I am facing some issue in the output of celery worker

Tastypie with ForeignKey - Agiliq Blog | Django web app development


5 years ago



ihmobileapps — This is great coding App Developers Long Island, NY and NYC . <http://www.ihmobileapps.com/>

AngularJS injectors internals - Agiliq Blog | Django web app development


2 years ago




Palani Suresh — Thanks for the valuable information to be share with us.This is very helpful to me,understanding the concepts of


Understanding Django Middlewares - Agiliq Blog | Django web app ...


4 years ago



Ashish Verma — you're right! The output is like: Middleware executed

 Subscribe

 Add Disqus to your siteAdd DisqusAdd

 Disqus' Privacy PolicyPrivacy PolicyPrivacy Policy

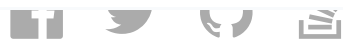
Agiliq

Building Amazing Apps. © 2010-2018, Agiliq
All rights reserved.

About Us

Blog

Agiliq



Email us : **hello@agiliq.com**

Phone us: **+919949997612**