When to use FormView?

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

FormView **should be used** when you need a form on the page and want to perform certain action when a valid form is submitted. eg: Having a contact us form and sending an email on form submission.

CreateView would probably be a better choice if you want to insert a model instance in database on form submission.

[#](https://www.agiliq.com/blog/2019/01/django-formview/#formview-is-better-than-vanilla-view)FormView is better than vanilla View

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

FormView is better than vanilla View in following ways:

* Avoid boilerplate code
* Better separation of concern
* Succinct and more maintainable code.

[#](https://www.agiliq.com/blog/2019/01/django-formview/#vanilla-view)Vanilla View

We want to create a page with a Contact Us form and send an email when a valid form is submitted.

# <app>/forms.py

class ContactForm(forms.Form):

name = forms.CharField()

message = forms.CharField(widget=forms.Textarea)

Vanilla view looks like:

# <app>/views.py

class ContactView(View):

def get(self, request, \*args, \*\*kwargs):

form = ContactForm()

context = {'form': form}

return render(request, 'contact-us.html', context)

def post(self, request, \*args, \*\*kwargs):

form = ContactForm(data=request.POST)

if form.is\_valid():

self.send\_mail(form.cleaned\_data)

form = ContactForm()

return render(request, 'contact-us.html', {'form': form})

return render(request, 'contact-us.html', {'form': form})

def send\_mail(self, valid\_data):

# Send mail logic

print(valid\_data)

pass

Template code looks like:

<!--<app>/templates/contact-us.html-->

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

{% csrf\_token %}

<table>

</table>

<button type="submit">SUBMIT</button>

</form>

With proper urlpattern, you should be able to see the contact us form. Submitting a valid form should be printing the valid data. Submitting invalid form should be redisplaying the form with invalid data.

[#](https://www.agiliq.com/blog/2019/01/django-formview/#using-formview)Using FormView

Modify ContactView to look like:

class ContactView(FormView):

form\_class = ContactForm

template\_name = 'contact-us.html'

success\_url = reverse\_lazy('<app-name>:contact-us')

def form\_valid(self, form):

self.send\_mail(form.cleaned\_data)

return super(ContactView, self).form\_valid(form)

def send\_mail(self, valid\_data):

# Send mail logic

print(valid\_data)

pass

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 FormView helped us avoid boilerplate get() and post() implementation.

[#](https://www.agiliq.com/blog/2019/01/django-formview/#adding-initial-data)Adding initial data

Say we wanted to add initial data to contact form while using vanilla view approach.

If user is logged in, then form’s name field should be populated with user’s full name.

This would require modifying ContactView as follows:

class ContactView(View):

def get(self, request, \*args, \*\*kwargs):

initial = None

if request.user.is\_authenticated:

initial = {'name': request.user.get\_full\_name()}

form = ContactForm(initial=initial)

context = {'form': form}

return render(request, 'books/contact-us.html', context)

def post(self, request, \*args, \*\*kwargs):

initial = None

if request.user.is\_authenticated:

initial = {'name': request.user.get\_full\_name()}

form = ContactForm(initial=initial, data=request.POST)

if form.is\_valid():

self.send\_mail(form.cleaned\_data)

form = ContactForm(initial=initial)

return render(request, 'books/contact-us.html', {'form': form})

return render(request, 'books/contact-us.html', {'form': form})

def send\_mail(self, valid\_data):

# Send mail logic

print(valid\_data)

pass

[#](https://www.agiliq.com/blog/2019/01/django-formview/#adding-initial-data-with-formview)Adding initial data with FormView

Modify FormView subclassed ContactView to look like:

class ContactView(FormView):

form\_class = ContactForm

template\_name = 'contact-us.html'

success\_url = reverse\_lazy('<app\_name>:contact-us')

def get\_initial(self):

initial = super(ContactView, self).get\_initial()

if self.request.user.is\_authenticated:

initial.update({'name': self.request.user.get\_full\_name()})

return initial

def form\_valid(self, form):

self.send\_mail(form.cleaned\_data)

return super(ContactView, self).form\_valid(form)

def send\_mail(self, valid\_data):

# Send mail logic

print(valid\_data)

This code has better separation of concern. There is a separate method for dealing with initial data and a separate method for dealing with what to do in case of a valid form.

[#](https://www.agiliq.com/blog/2019/01/django-formview/#adding-form-kwargs)Adding form kwargs

Say you have an ecommerce app which allows people to place order. Users can lodge complaints for orders placed by them. When they lodge a complaint, an email is sent to admin with order number and name of user.

# orders/models.py

class Order(models.Model):

user = models.ForeignKey(User, related\_name='orders', on\_delete=models.CASCADE)

name = models.CharField(max\_length=100)

order\_id = models.UUIDField()

def \_\_unicode\_\_(self):

return self.name

When a user comes to the page which shows complaint form, there should be a dropdown called orders and it should only display user’s orders.

# orders/forms.py

class OrderComplaintForm(forms.Form):

user\_name = forms.CharField()

complaint = forms.CharField()

order = forms.ChoiceField(choices=())

def \_\_init\_\_(self, \*args, \*\*kwargs):

user = kwargs.pop('user')

user\_orders = [(k, v) for k,v in user.orders.values('order\_id', 'name')]

super(OrderComplaintForm, self).\_\_init\_\_(\*args, \*\*kwargs)

self.fields['order'].choices = user\_orders

View would look like:

class OrderComplaintView(View):

def get(self, request, \*args, \*\*kwargs):

initial = {'complaint': 'I am unhappy with this order!', 'user\_name': self.request.user.get\_full\_name()}

kwargs = {'user': request.user}

form = OrderComplaintForm(initial=initial, \*\*kwargs)

context = {'form': form}

return render(request, 'orders/order-complaint.html', context)

def post(self, request, \*args, \*\*kwargs):

initial = {'complaint': 'I am unhappy with this order!', 'user\_name': self.request.user.get\_full\_name()}

kwargs = {'user': request.user}

form = OrderComplaintForm(initial=initial, data=request.POST, \*\*kwargs)

if form.is\_valid():

self.send\_mail(form.cleaned\_data)

form = OrderComplaintForm(initial=initial, \*\*kwargs)

context = {'form': form}

return render(request, 'orders/order-complaint.html', context)

context = {'form': form}

return render(request, 'orders/order-complaint.html', context)

def send\_mail(self, valid\_data):

# Send mail to admin with valid\_data['order'] and valid\_data['name']

print valid\_data

[#](https://www.agiliq.com/blog/2019/01/django-formview/#adding-form-kwargs-with-formview)Adding form kwargs with FormView

Modify your FormView subclassed OrderComplaintView to look like:

class OrderComplaintView(FormView):

form\_class = OrderComplaintForm

template\_name = 'orders/order-complaint.html'

success\_url = reverse\_lazy('order:order-complaint')

def get\_initial(self):

initial = super(OrderComplaintView, self).get\_initial()

initial.update({'user\_name': self.request.user.get\_full\_name(), 'complaint': 'I am unhappy with this order!'})

return initial

def get\_form\_kwargs(self):

kwargs = super(OrderComplaintView, self).get\_form\_kwargs()

kwargs.update({'user': self.request.user})

def form\_valid(self, form):

self.send\_mail(form.cleaned\_data)

return super(OrderComplaintView, self).form\_valid(form)

def send\_mail(self, valid\_data):

# Send mail to admin with valid\_data['order'] and valid\_data['user\_name']

print(valid\_data)

This code has better separation of concern with smaller methods instead of huge get() and post().

[#](https://www.agiliq.com/blog/2019/01/django-formview/#our-other-posts-on-generic-class-views)Our other posts on generic class views

* [TemplateView](https://www.agiliq.com/blog/2017/12/when-and-how-use-django-templateview/)
* [ListView](https://www.agiliq.com/blog/2017/12/when-and-how-use-django-listview/)
* [DetailView](https://www.agiliq.com/blog/2019/01/django-when-and-how-use-detailview/)
* [CreateView](https://www.agiliq.com/blog/2019/01/django-createview/)

Thank you for reading the Agiliq blog. This article was written by Akshar on Jan 6, 2019 in [django](https://www.agiliq.com/categories/#django) .

You can [subscribe ⚛ to our blog](https://www.agiliq.com/feed.xml).

We love building amazing apps for web and mobile for our clients. If you are looking for development help, [contact us today ✉](https://www.agiliq.com/contact/).