Agiliq

When and how to use Django FormView

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.

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.

Vanilla View

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

Agiliq

```
    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>
```

Agiliq

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.

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.

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:

Agiliq

```
        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
```

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
```

Agiliq

```
        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.

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.CAS
    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')
```

Agiliq

View would look like:

```
class OrderComplaintView(View):
    def get(self, request, *args, **kwargs):
        initial = {'complaint': 'I am unhappy with this order!', 'user_name': s
        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': s
        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
```

Adding form kwargs with FormView

Modify your FormView subclassed OrderComplaintView to look like:

```
ass OrderComplaintView(FormView):
    form_class = OrderComplaintForm
    template_name = 'orders/order-complaint.html'
```

Agiliq

```
        initial = super(OrderComplaintView, self).get_initial()
        initial.update({'user_name': self.request.user.get_full_name(), 'complain
        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().

Our other posts on generic class views

— TemplateView
— ListView
— DetailView
— CreateView

Thank you for reading the Agiliq blog. This article was written by Akshar on Jan 6, 2019 in django .

You can subscribe ⚛ to our blog.

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

Agiliq

Agiliq

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

Phone us: **+919949997612**