views_v4.py

```
from django.views.generic import CreateView, ListView, UpdateView, DetailView, DeleteView
from django.urls import reverse, reverse_lazy
from django.contrib import messages
from django.utils.translation import gettext_lazy as _
from django.shortcuts import get_object_or_404, redirect
from django.http import HttpResponseRedirect
from django.template.loader import render_to_string

from oscar.core.loading import get_classes
from oscar.apps.dashboard.catalogue.views import \
    ProductListView as CoreProductListView, \
    ProductCreateRedirectView as CoreProductCreateRedirectView, \
    ProductCreateUpdateView as CoreProductCreateUpdateView
from oscar.apps.catalogue.models import ProductClass
from oscar.apps.dashboard.catalogue.mixins import PartnerProductFilterMixin
from oscar.apps.dashboard.catalogue.formsets import \
    ProductCategoryFormSet, \
    ProductImageFormSet, \
    ProductRecommendationFormSet, \
    StockRecordFormSet

from myapps.catalogue.models import Metal, Gemstone, ProductAttribute, Product
from myapps.dashboard.catalogue.forms import MetalSelectForm, GemstoneSelectForm, ProductForm
#from myapps.dashboard.catalogue.formsets import \
#    ProductCategoryFormSet, \
#    ProductImageFormSet, \
#    ProductRecommendationFormSet, \
#    StockRecordFormSet


class ProductCreateUpdateView(CoreProductCreateUpdateView):

    def get_object(self, queryset=None):
        """
        This parts allows generic.UpdateView to handle creating products as
        well. The only distinction between an UpdateView and a CreateView
        is that self.object is None. We emulate this behavior.

        This method is also responsible for setting self.product_class, self.metal
        and self.parent.
        """
        self.creating = 'pk' not in self.kwargs
        if self.creating:
            # Specifying a parent product is only done when creating a child
            # product.
            parent_pk = self.kwargs.get('parent_pk')
            if parent_pk is None:
                self.parent = None
                # A product class needs to be specified when creating a
                # standalone product.
                product_class_slug = self.kwargs.get('product_class_slug')
                metal_slug = self.kwargs.get('metal_slug')
                self.product_class = get_object_or_404(
                    ProductClass, slug=product_class_slug)
                self.metal = get_object_or_404(
                    Metal, slug=metal_slug)
            else:
                self.parent = get_object_or_404(Product, pk=parent_pk)
                self.product_class = self.parent.product_class
                self.metal = self.parent.metal

            return None  # success
        else:
            product = super().get_object(queryset)
            self.product_class = product.get_product_class()
            self.parent = product.parent
            self.metal = product.get_metal_type()
```

```
        return product

    def get_context_data(self, **kwargs):
        ctx = super().get_context_data(**kwargs)
        #ctx = {}
        ctx['product_class'] = self.product_class
        ctx['metal'] = self.metal
        ctx['parent'] = self.parent
        ctx['title'] = self.get_page_title()

        for ctx_name, formset_class in self.formsets.items():
            if ctx_name not in ctx:
                ctx[ctx_name] = formset_class(self.product_class,
                                              self.metal,
                                              self.request.user,
                                              instance=self.object)
        return ctx

    def get_page_title(self):
        if self.creating:
            if self.parent is None:
                return _('Create new %(product_class)s product') % {
                    'product_class': self.product_class.name}
            else:
                return _('Create new variant of %(parent_product)s') % {
                    'parent_product': self.parent.title}
        else:
            if self.object.title or not self.parent:
                return self.object.title
            else:
                return _('Editing variant of %(parent_product)s') % {
                    'parent_product': self.parent.title}

    def get_form_kwargs(self):
        kwargs = super().get_form_kwargs()
        #kwargs = {}
        kwargs['product_class'] = self.product_class
        kwargs['parent'] = self.parent
        kwargs['metal'] = self.metal
        return kwargs
```