```
from django import forms
from django.core import exceptions
from django.utils.translation import gettext_lazy as _

from oscar.apps.dashboard.catalogue.forms import ProductForm as CoreProductForm
from oscar.apps.dashboard.catalogue.forms import SEOFormMixin
from oscar.apps.dashboard.catalogue.forms import \
    _attr_text_field, _attr_textarea_field, _attr_integer_field, _attr_boolean_field, _attr_float_field, \
    _attr_date_field, _attr_datetime_field, _attr_option_field, _attr_multi_option_field, _attr_entity_field, \
    _attr_numeric_field, _attr_file_field, _attr_image_field

from myapps.catalogue.models import Metal, Gemstone, ProductAttribute, Product


class ProductForm(SEOFormMixin, forms.ModelForm):
    """
    Unable to override and customize CoreProductForm. Get the following traceback:
    ProductForm.set_initial() missing 1 required positional argument: 'kwargs'

    For now, just copied all the code from Oscar and introduced my customizations and this
    is working fine. But in future, we need to change this approach.
    """
    FIELD_FACTORIES = {
        "text": _attr_text_field,
        "richtext": _attr_textarea_field,
        "integer": _attr_integer_field,
        "boolean": _attr_boolean_field,
        "float": _attr_float_field,
        "date": _attr_date_field,
        "datetime": _attr_datetime_field,
        "option": _attr_option_field,
        "multi_option": _attr_multi_option_field,
        "entity": _attr_entity_field,
        "numeric": _attr_numeric_field,
        "file": _attr_file_field,
        "image": _attr_image_field,
    }

    class Meta:
        model = Product
        fields = [
            'title', 'upc', 'description', 'is_public', 'is_discountable', 'structure', 'slug', 'meta_title',
            'meta_description']
        widgets = {
            'structure': forms.HiddenInput(),
            'meta_description': forms.Textarea(attrs={'class': 'no-widget-init'})
        }

    def __init__(self, product_class, metal, data=None, parent=None, *args, **kwargs):
        self.set_initial(product_class, metal, parent, kwargs)
        super().__init__(data, *args, **kwargs)
        if parent:
            self.instance.parent = parent
            # We need to set the correct product structures explicitly to pass
            # attribute validation and child product validation. Note that
            # those changes are not persisted.
            self.instance.structure = Product.CHILD
            self.instance.parent.structure = Product.PARENT

            self.delete_non_child_fields()
        else:
            # Only set product class for non-child products
            self.instance.product_class = product_class
            self.instance.metal = metal
        self.add_attribute_fields(product_class, metal, self.instance.is_parent)

        if 'slug' in self.fields:
            self.fields['slug'].required = False
            self.fields['slug'].help_text = _('Leave blank to generate from product title')
        if 'title' in self.fields:
            self.fields['title'].widget = forms.TextInput(
                attrs={'autocomplete': 'off'})

    def set_initial(self, product_class, metal, parent, kwargs):
        """
        Set initial data for the form. Sets the correct product structure
        and fetches initial values for the dynamically constructed attribute
        fields.
        """
        if 'initial' not in kwargs:
            kwargs['initial'] = {}
```

```
        self.set_initial_attribute_values(product_class, metal, kwargs)
        if parent:
            kwargs['initial']['structure'] = Product.CHILD

    def set_initial_attribute_values(self, product_class, metal, kwargs):
        """
        Update the kwargs['initial'] value to have the initial values based on
        the product instance's attributes
        """
        instance = kwargs.get('instance')
        if instance is None:
            return
        for attribute in product_class.attributes.all():
            try:
                value = instance.attribute_values.get(
                    attribute=attribute).value
            except exceptions.ObjectDoesNotExist:
                pass
            else:
                kwargs['initial']['attr_%s' % attribute.code] = value

        for attribute in metal.product_attribute.all():
            try:
                value = instance.attribute_values.get(
                    attribute=attribute).value
            except exceptions.ObjectDoesNotExist:
                pass
            else:
                kwargs['initial']['attr_%s' % attribute.code] = value

    def add_attribute_fields(self, product_class, metal, is_parent=False):
        """
        For each attribute specified by the product class, this method
        dynamically adds form fields to the product form.
        """
        for attribute in product_class.attributes.all():
            field = self.get_attribute_field(attribute)
            if field:
                self.fields['attr_%s' % attribute.code] = field
                # Attributes are not required for a parent product
                if is_parent:
                    self.fields['attr_%s' % attribute.code].required = False

        for attribute in metal.product_attribute.all():
            field = self.get_attribute_field(attribute)
            if field:
                self.fields['attr_%s' % attribute.code] = field
                # Attributes are not required for a parent product
                if is_parent:
                    self.fields['attr_%s' % attribute.code].required = False

    def get_attribute_field(self, attribute):
        """
        Gets the correct form field for a given attribute type.
        """
        return self.FIELD_FACTORIES[attribute.type](attribute)

    def delete_non_child_fields(self):
        """
        Deletes any fields not needed for child products. Override this if
        you want to e.g. keep the description field.
        """
        for field_name in ['description', 'is_discountable']:
            if field_name in self.fields:
                del self.fields[field_name]

    def _post_clean(self):
        """
        Set attributes before ModelForm calls the product's clean method
        (which it does in _post_clean), which in turn validates attributes.
        """
        for attribute in self.instance.attr.get_all_attributes():
            field_name = 'attr_%s' % attribute.code
            # An empty text field won't show up in cleaned_data.
            if field_name in self.cleaned_data:
                value = self.cleaned_data[field_name]
                setattr(self.instance.attr, attribute.code, value)
        super()._post_clean()
```