ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ПИ им. Фельдмана

Факультет ИСП

Отчет по лабораторной работе №2

Тема: «Работа с ветками»

Курс: «Профессиональная практика программной инженерии»

Выполнил:

ст. гр. ПИ-18а

Дробоцкий В.А.

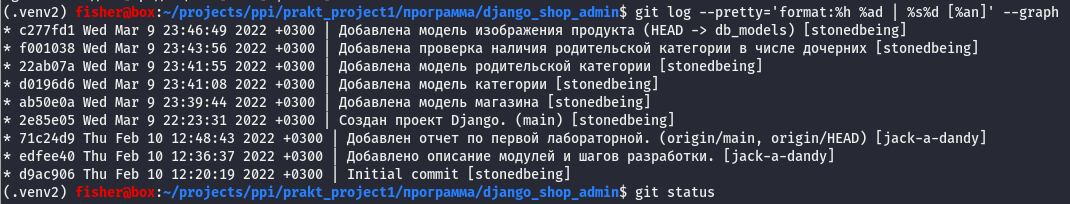
Проверили:

Филипишин Д. А.

Донецк – 2022

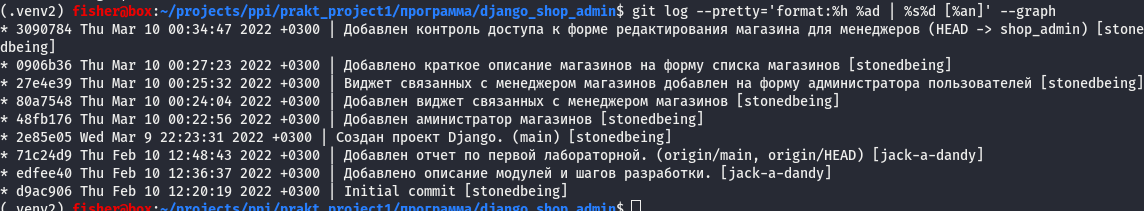
Модуль моделей ORM — ветка db\_models

Результат вывода git log --pretty='format:%h %ad | %s%d [%an]' —graph



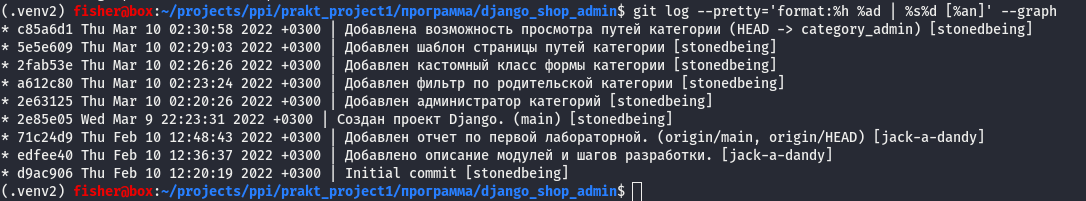
Модуль администратора магазинов — shop\_admin

Результат вывода git log --pretty='format:%h %ad | %s%d [%an]' —graph



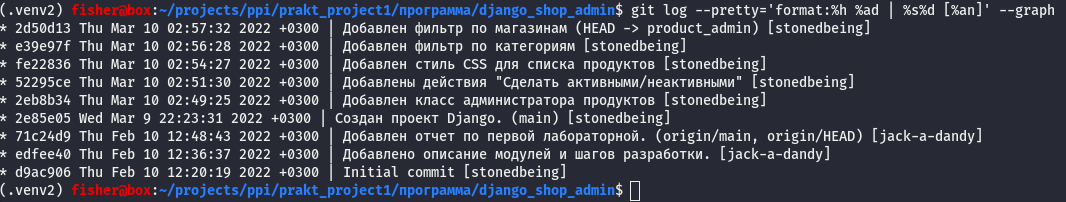
Модуль администратора категорий — category\_admin

Результат вывода git log --pretty='format:%h %ad | %s%d [%an]' —graph



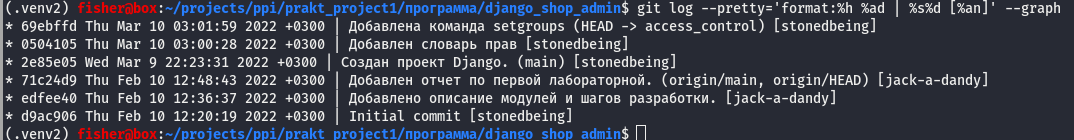
Модуль администратора продуктов — product\_admin

Результат вывода git log --pretty='format:%h %ad | %s%d [%an]' —graph

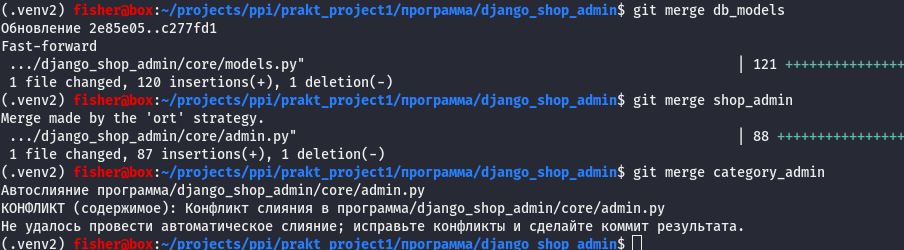


Модуль контроля доступа — access\_control

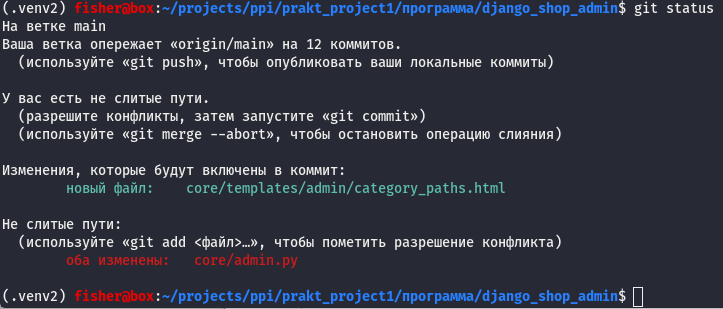
Результат вывода git log --pretty='format:%h %ad | %s%d [%an]' —graph



Ветки были слиты в порядке создания.



При слиянии ветки category\_admin получили конфликт в файле admin.py.



Конфликт в добавленных новых классах.

=======

class ParentCategoryFilter(admin.SimpleListFilter):

title = 'Род. категория'

parameter\_name = 'parents\_\_id'

def lookups(self, request, model\_admin):

objs = Category.objects.filter(from\_category\_\_isnull=False).only('title').distinct().order\_by('title')

return [(o.pk, o.title) for o in objs]

def queryset(self, request, queryset):

value = self.value()

if value is not None:

return queryset.filter(parents\_\_id=self.value())

return queryset

class CategoryAdminForm(forms.ModelForm):

parents = forms.ModelMultipleChoiceField(label='Родительские категории',

queryset = Category.objects.only('title').order\_by('title'),

required=False,

widget=FilteredSelectMultiple(

verbose\_name='Родительские категории',

is\_stacked=False

))

children = forms.ModelMultipleChoiceField(label='Дочерние категории',

queryset=Category.objects.only('title').order\_by('title'),

required=False,

widget=FilteredSelectMultipleWithReadonlyMode(

verbose\_name='Дочерние категории',

is\_stacked=False

)

)

class Meta:

model = Category

fields = '\_\_all\_\_'

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

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

instance = kwargs.get("instance")

if instance and instance.pk:

self.fields['parents'].queryset=Category.objects.filter(

~(Q(pk=instance.pk)|Q(pk\_\_in=instance.category\_set.values('id')))

).only('title').order\_by('title')

self.fields['parents'].initial=instance.parents.all()

self.fields['children'].queryset=Category.objects.filter(

~(Q(pk=instance.pk)|Q(pk\_\_in=instance.parents.values('id')))

).only('title').order\_by('title')

self.fields['children'].initial=instance.category\_set.all()

self.fields['children'].widget.attrs['readonly']=True

def save(self, commit=True):

category = super(CategoryAdminForm, self).save(commit=False)

if commit:

error=1

try:

with transaction.atomic():

category.save()

data = self.cleaned\_data['parents']

if self.fields['parents'].has\_changed(self.fields['parents'].initial, data):

category.parents.set(data)

error=2

data = self.cleaned\_data['children']

if self.fields['children'].has\_changed(self.fields['children'].initial, data):

category.category\_set.set(data)

except forms.ValidationError as e:

self.add\_error('parents' if error==1 else 'children', e)

except Exception as e:

self.add\_error(None, e)

return category

@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin)

list\_display = ('title','id', 'description', 'category\_actions')

search\_fields = ('products\_\_id', 'title')

list\_filter = (ParentCategoryFilter,)

ordering = ('title',)

readonly\_fields = ('id',)

form = CategoryAdminForm

def get\_fields(self, request, obj=None):

return ('id', 'title', 'description', 'parents', 'children')

def get\_urls(self):

urls = super().get\_urls()

custom\_urls = [

path(

'<int:category\_id>/paths/',

self.admin\_site.admin\_view(self.process\_paths),

name='category-paths',

),

]

return custom\_urls + urls

def category\_actions(self, obj):

return format\_html(

'<a class="button" href="{}">Показать пути</a>',

reverse('admin:category-paths', args=[obj.pk])

)

category\_actions.short\_description = 'Действия'

category\_actions.allow\_tags = True

def process\_paths(self, request, category\_id, \*args, \*\*kwargs):

obj = self.get\_object(request, category\_id)

context = self.admin\_site.each\_context(request)

context['opts'] = self.model.\_meta

paths = list(obj.get\_all\_paths())

context['paths'] = paths

context['title'] = f'Пути к категории {obj.title} ({len(paths)})'

return TemplateResponse(

request,

'admin/category\_paths.html',

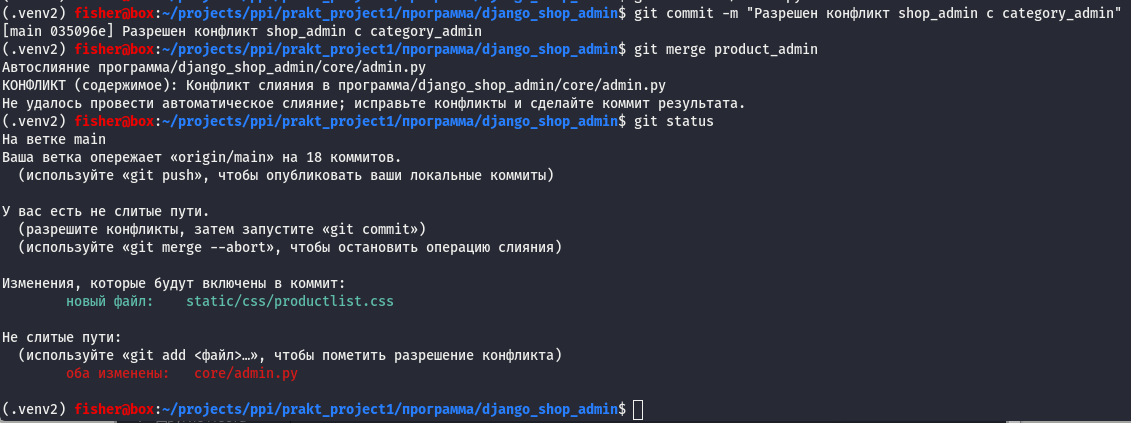
context,

)

>>>>>>> category\_admin

Просто улаляем разделители конфликта в файле для его разрешения.

Затем получили новый конфликт при слиянии с product\_admin



=======

class ShopFilter(admin.SimpleListFilter):

title = 'Магазин'

parameter\_name = 'shop\_\_id'

def lookups(self, request, model\_admin):

objs = Shop.objects if request.user.is\_superuser else request.user.managed\_shops

objs = objs.filter(products\_\_isnull=False).only('title').distinct().order\_by('title')

>>>>>>> product\_admin

return [(o.pk, o.title) for o in objs]

def queryset(self, request, queryset):

value = self.value()

if value is not None:

<<<<<<< HEAD

return queryset.filter(parents\_\_id=self.value())

return queryset

class CategoryAdminForm(forms.ModelForm):

parents = forms.ModelMultipleChoiceField(label='Родительские категории',

queryset = Category.objects.only('title').order\_by('title'),

required=False,

widget=FilteredSelectMultiple(

verbose\_name='Родительские категории',

is\_stacked=False

))

children = forms.ModelMultipleChoiceField(label='Дочерние категории',

queryset=Category.objects.only('title').order\_by('title'),

required=False,

widget=FilteredSelectMultipleWithReadonlyMode(

verbose\_name='Дочерние категории',

is\_stacked=False

)

)

class Meta:

model = Category

fields = '\_\_all\_\_'

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

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

instance = kwargs.get("instance")

if instance and instance.pk:

self.fields['parents'].queryset=Category.objects.filter(

~(Q(pk=instance.pk)|Q(pk\_\_in=instance.category\_set.values('id')))

).only('title').order\_by('title')

self.fields['parents'].initial=instance.parents.all()

self.fields['children'].queryset=Category.objects.filter(

~(Q(pk=instance.pk)|Q(pk\_\_in=instance.parents.values('id')))

).only('title').order\_by('title')

self.fields['children'].initial=instance.category\_set.all()

self.fields['children'].widget.attrs['readonly']=True

def save(self, commit=True):

category = super(CategoryAdminForm, self).save(commit=False)

if commit:

error=1

try:

with transaction.atomic():

category.save()

data = self.cleaned\_data['parents']

if self.fields['parents'].has\_changed(self.fields['parents'].initial, data):

category.parents.set(data)

error=2

data = self.cleaned\_data['children']

if self.fields['children'].has\_changed(self.fields['children'].initial, data):

category.category\_set.set(data)

except forms.ValidationError as e:

self.add\_error('parents' if error==1 else 'children', e)

except Exception as e:

self.add\_error(None, e)

return category

@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin)

list\_display = ('title','id', 'description', 'category\_actions')

search\_fields = ('products\_\_id', 'title')

list\_filter = (ParentCategoryFilter,)

ordering = ('title',)

readonly\_fields = ('id',)

form = CategoryAdminForm

def get\_fields(self, request, obj=None):

return ('id', 'title', 'description', 'parents', 'children')

def get\_urls(self):

urls = super().get\_urls()

custom\_urls = [

path(

'<int:category\_id>/paths/',

self.admin\_site.admin\_view(self.process\_paths),

name='category-paths',

),

]

return custom\_urls + urls

def category\_actions(self, obj):

return format\_html(

'<a class="button" href="{}">Показать пути</a>',

reverse('admin:category-paths', args=[obj.pk])

)

category\_actions.short\_description = 'Действия'

category\_actions.allow\_tags = True

def process\_paths(self, request, category\_id, \*args, \*\*kwargs):

obj = self.get\_object(request, category\_id)

context = self.admin\_site.each\_context(request)

context['opts'] = self.model.\_meta

paths = list(obj.get\_all\_paths())

context['paths'] = paths

context['title'] = f'Пути к категории {obj.title} ({len(paths)})'

return TemplateResponse(

request,

'admin/category\_paths.html',

context,

)

=======

return queryset.filter(shop\_\_id=self.value())

return queryset

class CategoryFilter(admin.SimpleListFilter):

title = 'Категория'

parameter\_name = 'categories\_\_id'

def lookups(self, request, model\_admin):

filters = {'products\_\_isnull': False}

if not request.user.is\_superuser:

filters['products\_\_shop\_\_id\_\_in']=request.user.managed\_shops.values\_list('id', flat=True)

objs = Category.objects.filter(\*\*filters).only('title').distinct().order\_by('title')

return [(o.pk, o.title) for o in objs]

def queryset(self, request, queryset):

value = self.value()

if value is not None:

return queryset.filter(categories\_\_id=self.value())

return queryset

@admin.register(Product)

class ProductAdmin(NumericFilterModelAdmin):

list\_display = ('title','main\_image', 'id', 'amount', 'price', 'active', 'shop\_id')

fieldsets = ((None, {'fields':('id', 'shop', 'title', 'description', 'active', 'amount', 'price')}),

('КАТЕГОРИИ', {'fields': ('categories',), 'classes': ('collapse',)}),

('ОСНОВНОЕ ФОТО', {'fields': ('main\_image',)}),

)

search\_fields = ('id', 'title')

list\_filter = ('active',CategoryFilter,ShopFilter)

readonly\_fields = ('id',)

filter\_horizontal = ('categories',)

actions = ('make\_active', 'make\_inactive')

list\_per\_page = 50

class Media:

css = {'all': ('css/productlist.css',)}

def main\_image(self, instance):

url = instance.images.only('image').first()

if url:

return format\_html("<img src='{}{}' width=100 height=100 style='object-fit:contain' />",

settings.MEDIA\_URL, url.image)

else:

return format\_html("<img alt='—' />")

main\_image.short\_description = 'Фото'

def formfield\_for\_manytomany(self, db\_field, request, \*\*kwargs):

if db\_field.name == "categories":

kwargs["queryset"] = Category.objects.only('title').order\_by('title')

return super().formfield\_for\_manytomany(db\_field, request, \*\*kwargs)

def formfield\_for\_foreignkey(self, db\_field, request, \*\*kwargs):

if db\_field.name == 'shop':

qs = None

if not request.user.is\_superuser:

qs = request.user.managed\_shops

else:

qs = Shop.objects

kwargs['queryset']=qs.only('title').order\_by('title')

return super().formfield\_for\_foreignkey(db\_field, request, \*\*kwargs)

def get\_queryset(self, request):

qs = super().get\_queryset(request)

if request.user.is\_superuser:

return qs

else:

return qs.filter(shop\_\_id\_\_in=request.user.managed\_shops.values\_list('id', flat=True))

@admin.action(description='Сделать активными')

def make\_active(self, request, queryset):

queryset.update(active=True)

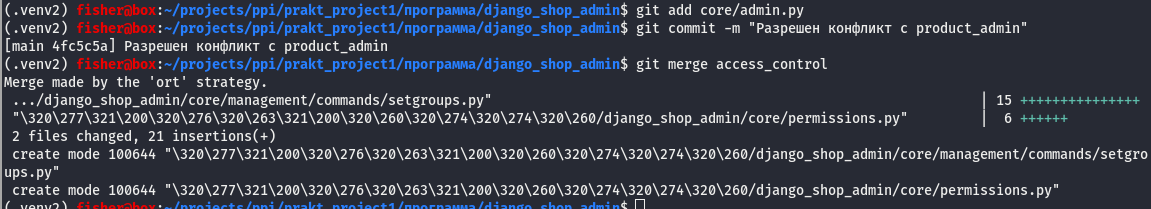
@admin.action(description='Сделать неактивными')

def make\_inactive(self, request, queryset):

queryset.update(active=False)

>>>>>>> product\_admin

Тут ситуация немного запутанная из-за идентичных кусочков кода в разных классах. Для решения конфликта нужно переместатить часть кода, а так же скопировать и вставить другую часть.



Результат

