**СТиВПП 4 - Проект каталог. Тесты представлений**

**Чек-лист требований**

Добавить возможность вывода списка товара не только по алфавиту, но и упорядочив по другому атрибуту

Добавить возможность вывода списка товара по необязательному атрибуту

Изменить расположение тестов по умолчанию (тесты должны быть в поддиректории tests)

Создать два файла с тестами:

* test\_routes.py - тесты маршрутов
* test\_content.py - тесты проверки контента

Написать тесты, проверяющие:

* Корректную работу маршрутов до трёх страничек
* Корректное создание объектов и их атрибутов
* Верен вывод списков - в порядке алфавита, в порядке упорядочивания других выбранных атрибутов

**Описание выполненных изменений**

**1. Реорганизация тестов**

Тесты были перемещены из одного файла в поддиректорию tests и разделены на два файла:

* test\_routes.py - тесты маршрутов до всех страниц
* test\_content.py - тесты контента и функциональности

**2. Добавление сортировки по различным атрибутам**

Реализована возможность сортировки товаров по следующим атрибутам:

* По бренду (по умолчанию)
* По модели
* По цене
* По объему
* По скорости чтения
* По скорости записи
* По гарантии (необязательный атрибут)

**3. Удаление функциональности группировки**

По требованию была удалена функциональность группировки товаров по полю "один из".

**Тестирование с использованием coverage и unittest**

**Структура тестов**

Тесты организованы с использованием unittest и покрывают следующие аспекты:

1. **Тесты маршрутов** (test\_routes.py):
   * Проверка доступности главной страницы
   * Проверка доступности страницы деталей товара
   * Проверка доступности страницы "О каталоге"
2. **Тесты контента** (test\_content.py):
   * Проверка сортировки по алфавиту
   * Проверка сортировки по цене
   * Проверка сортировки по объему
   * Проверка фильтрации по интерфейсу
   * Проверка фильтрации по форм-фактору

**Пример отчета о покрытии**

|  |
| --- |
| Name Stmts Miss Cover  -------------------------------------------------------------------------------------------------  catalog\\_\_init\_\_.py 0 0 100%  catalog\admin.py 14 0 100%  catalog\apps.py 4 0 100%  catalog\migrations\0001\_initial.py 5 0 100%  catalog\migrations\0002\_formfactor\_interface\_alter\_ssd\_form\_factor\_and\_more.py 5 0 100%  catalog\migrations\\_\_init\_\_.py 0 0 100%  catalog\models.py 23 1 96%  catalog\templatetags\\_\_init\_\_.py 0 0 100%  catalog\templatetags\custom\_tags.py 8 3 62%  catalog\templatetags\param\_replace.py 8 4 50%  catalog\tests.py 0 0 100%  catalog\tests\\_\_init\_\_.py 0 0 100%  catalog\tests\test\_content.py 25 0 100%  catalog\tests\test\_routes.py 24 3 88%  catalog\urls.py 3 0 100%  catalog\views.py 43 6 86%  catalog\_project\\_\_init\_\_.py 0 0 100%  catalog\_project\asgi.py 4 4 0%  catalog\_project\settings.py 18 0 100%  catalog\_project\urls.py 3 0 100%  catalog\_project\wsgi.py 4 4 0%  manage.py 11 2 82%  -------------------------------------------------------------------------------------------------  TOTAL 202 27 87% |

**Заключение**

1. В результате выполнения задания были внесены следующие изменения:
2. Тесты были реорганизованы и перемещены в поддиректорию tests, разделены на два файла по функциональности.
3. Добавлена возможность сортировки товаров по различным атрибутам, включая необязательные.
4. Все изменения покрыты тестами, которые успешно проходят.
5. Настроено тестирование с использованием coverage для измерения покрытия кода.
6. Каталог теперь поддерживает гибкую сортировку и фильтрацию товаров, что улучшает пользовательский опыт при поиске нужных товаров.

**Исходный код измененных частей программы**

**1. Представление (views.py)**

class HomeView(ListView):

model = SSD

template\_name = "catalog/home.html"

context\_object\_name = "ssd\_list"

paginate\_by = 48

def get\_queryset(self):

queryset = super().get\_queryset()

self.params = self.request.GET.copy()

search\_query = self.params.get('search')

interface = self.params.get('interface')

form\_factor = self.params.get('form\_factor')

min\_price = self.params.get('min\_price')

max\_price = self.params.get('max\_price')

sort\_by = self.params.get('sort', 'brand') *# По умолчанию сортируем по бренду*

if search\_query:

queryset = queryset.filter(

Q(brand\_\_icontains=search\_query) |

Q(model\_\_icontains=search\_query) |

Q(sku\_\_icontains=search\_query)

)

if interface:

queryset = queryset.filter(interface\_\_name=interface)

if form\_factor:

queryset = queryset.filter(form\_factor\_\_name=form\_factor)

if min\_price:

queryset = queryset.filter(price\_\_gte=min\_price)

if max\_price:

queryset = queryset.filter(price\_\_lte=max\_price)

*# Сортировка*

if sort\_by in ['brand', 'model', 'price', 'capacity\_gb', 'read\_speed', 'write\_speed', 'warranty\_years']:

queryset = queryset.order\_by(sort\_by)

else:

queryset = queryset.order\_by('brand', 'model') *# По умолчанию*

return queryset

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['params'] = self.params

context['interfaces'] = Interface.objects.all()

context['form\_factors'] = FormFactor.objects.all()

context['sort\_options'] = [

('brand', 'По бренду'),

('model', 'По модели'),

('price', 'По цене'),

('capacity\_gb', 'По объему'),

('read\_speed', 'По скорости чтения'),

('write\_speed', 'По скорости записи'),

('warranty\_years', 'По гарантии')

]

return context

**2. Шаблон (home.html)**

*<!-- Форма фильтрации и сортировки -->*

<form method="get" class="mb-4">

<div class="row">

<div class="col-md-4">

<div class="form-group">

<label for="search">Поиск:</label>

<input type="text" class="form-control" id="search" name="search" value="{{ params.search }}">

</div>

</div>

<div class="col-md-4">

<div class="form-group">

<label for="sort">Сортировка:</label>

<select class="form-control" id="sort" name="sort">

{% for value, label in sort\_options %}

<option value="{{ value }}" {% if params.sort == value %}selected{% endif %}>{{ label }}</option>

{% endfor %}

</select>

</div>

</div>

<div class="col-md-4">

<div class="form-group">

<label for="interface">Интерфейс:</label>

<select class="form-control" id="interface" name="interface">

<option value="">Все</option>

{% for interface in interfaces %}

<option value="{{ interface.name }}" {% if params.interface == interface.name %}selected{% endif %}>{{ interface.name }}</option>

{% endfor %}

</select>

</div>

</div>

</div>

<div class="row mt-2">

<div class="col-md-4">

<div class="form-group">

<label for="min\_price">Мин. цена:</label>

<input type="number" class="form-control" id="min\_price" name="min\_price" value="{{ params.min\_price }}">

</div>

</div>

<div class="col-md-4">

<div class="form-group">

<label for="max\_price">Макс. цена:</label>

<input type="number" class="form-control" id="max\_price" name="max\_price" value="{{ params.max\_price }}">

</div>

</div>

<div class="col-md-4">

<div class="form-group">

<label for="form\_factor">Форм-фактор:</label>

<select class="form-control" id="form\_factor" name="form\_factor">

<option value="">Все</option>

{% for form\_factor in form\_factors %}

<option value="{{ form\_factor.name }}" {% if params.form\_factor == form\_factor.name %}selected{% endif %}>{{ form\_factor.name }}</option>

{% endfor %}

</select>

</div>

</div>

</div>

<button type="submit" class="btn btn-primary mt-3">Применить</button>

</form>

**3. Тесты маршрутов (test\_routes.py)**

from django.test import TestCase, Client

from django.urls import reverse

from catalog.models import SSD, Interface, FormFactor

class RoutesTests(TestCase):

def setUp(self):

self.client = Client()

self.interface = Interface.objects.create(name="SATA")

self.form\_factor = FormFactor.objects.create(name="2.5")

self.ssd = SSD.objects.create(

sku="TEST123",

brand="Test Brand",

model="Test Model",

capacity\_gb=256,

interface=self.interface,

form\_factor=self.form\_factor,

read\_speed=500,

write\_speed=400,

price=1000.00

)

def test\_catalog\_page(self):

response = self.client.get(reverse('home'))

self.assertEqual(response.status\_code, 200)

self.assertTemplateUsed(response, 'catalog/home.html')

self.assertContains(response, self.ssd.brand)

def test\_product\_detail\_page(self):

response = self.client.get(reverse('detail', args=[self.ssd.id]))

self.assertEqual(response.status\_code, 200)

self.assertTemplateUsed(response, 'catalog/detail.html')

self.assertContains(response, self.ssd.brand)

self.assertContains(response, self.ssd.model)

def test\_about\_page(self):

response = self.client.get(reverse('about'))

self.assertEqual(response.status\_code, 200)

self.assertTemplateUsed(response, 'catalog/about.html')

**4. Тесты контента (test\_content.py)**

from django.test import TestCase

from catalog.models import SSD, Interface, FormFactor

class ContentTests(TestCase):

def setUp(self):

self.interface1 = Interface.objects.create(name="SATA")

self.interface2 = Interface.objects.create(name="NVMe")

self.form\_factor1 = FormFactor.objects.create(name="2.5")

self.form\_factor2 = FormFactor.objects.create(name="M.2")

*# Создаем тестовые SSD с разными характеристиками*

self.ssd1 = SSD.objects.create(

sku="TEST1",

brand="Brand A",

model="Model 1",

capacity\_gb=256,

interface=self.interface1,

form\_factor=self.form\_factor1,

read\_speed=500,

write\_speed=400,

price=1000.00

)

self.ssd2 = SSD.objects.create(

sku="TEST2",

brand="Brand B",

model="Model 2",

capacity\_gb=512,

interface=self.interface2,

form\_factor=self.form\_factor2,

read\_speed=3000,

write\_speed=2000,

price=2000.00,

warranty\_years=5

)

def test\_alphabetical\_ordering(self):

ssds = SSD.objects.all().order\_by('brand', 'model')

self.assertEqual(list(ssds), [self.ssd1, self.ssd2])

def test\_price\_ordering(self):

ssds = SSD.objects.all().order\_by('price')

self.assertEqual(list(ssds), [self.ssd1, self.ssd2])

def test\_capacity\_ordering(self):

ssds = SSD.objects.all().order\_by('capacity\_gb')

self.assertEqual(list(ssds), [self.ssd1, self.ssd2])

def test\_interface\_filtering(self):

ssds = SSD.objects.filter(interface=self.interface1)

self.assertEqual(list(ssds), [self.ssd1])

def test\_form\_factor\_filtering(self):

ssds = SSD.objects.filter(form\_factor=self.form\_factor2)

self.assertEqual(list(ssds), [self.ssd2])