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])
```