**СТиВПП 3 - Проект каталог. База данных**

**Выполненные работы**

В рамках задания была реализована основная модель SSD с полями, описывающими характеристики товара:

* Артикул (sku)
* Производитель (brand)
* Модель (model)
* Ёмкость (capacity\_gb)
* Интерфейс (interface)
* Форм-фактор (form\_factor)
* Скорость чтения и записи (read\_speed, write\_speed)
* Гарантия (warranty\_years)
* Цена (price)
* Наличие на складе (in\_stock)

Для полей interface и form\_factor реализованы отдельные модели (Interface, FormFactor) со связью **один-ко-многим** (ForeignKey с on\_delete=models.PROTECT), как требовалось по заданию.

Модель SSD содержит ограничения по типу данных и уникальности, например sku — уникальное поле, числовые характеристики реализованы через PositiveIntegerField и DecimalField.

**Настройка административной панели**

Для всех моделей была настроена админка:

* Отображение ключевых полей в списке
* Фильтрация по интерфейсу, форм-фактору и наличию
* Поиск по артикулу, бренду и модели
* Разбиение формы редактирования на логические блоки

Это упростило процесс добавления и редактирования записей через панель администратора.

**Изменение представлений**

Домашняя страница (HomeView) реализована на основе ListView, и отображает список SSD-дисков с фильтрацией по:

* Поисковому запросу
* Интерфейсу
* Диапазону цен

Представление для детального описания (SSDDetailView) отображает полную информацию о выбранном товаре.

В представлении реализована передача параметров запроса в шаблон для сохранения состояния фильтров при пагинации.

**Чек-лист требований заказчика**

|  |  |
| --- | --- |
| **Требование** | **Примечание** |
| Создать модели для хранения информации о товарах | Модель SSD содержит все необходимые поля |
| Поля типа «один из» вынесены в отдельные таблицы | Реализованы модели Interface и FormFactor |
| Настроена административная панель Django | Все модели добавлены в админку с удобным интерфейсом |
| Представления используют данные из базы | Используются ListView и DetailView, реализована фильтрация |
| Прописаны ограничения по типам и допустимым значениям полей | Все поля строго типизированы |
| Связи один-ко-многим реализованы | Поля interface и form\_factor — внешние ключи |
| Обновлены шаблоны, при необходимости | Шаблоны адаптированы под использование связанных моделей |

Изменение проекта

1. Модели (catalog/models.py)

Созданы SSD, Interface и FormFactor. Поля interface и form\_factor заменены на ForeignKey для соответствия требованию об отдельной таблице и связи один-ко-многим. Добавлены ограничения на поля (уникальность, типы данных).

|  |
| --- |
| from django.db import models   class Interface(models.Model):  name = models.CharField(max\_length=20, unique=True, verbose\_name="Интерфейс")   def \_\_str\_\_(self):  return self.name   class FormFactor(models.Model):  name = models.CharField(max\_length=20, unique=True, verbose\_name="Форм-фактор")   def \_\_str\_\_(self):  return self.name   class SSD(models.Model):  sku = models.CharField(max\_length=50, unique=True, verbose\_name="Артикул")  brand = models.CharField(max\_length=100, verbose\_name="Производитель")  model = models.CharField(max\_length=100, verbose\_name="Модель")  capacity\_gb = models.PositiveIntegerField(verbose\_name="Ёмкость (ГБ)")  interface = models.ForeignKey(Interface, on\_delete=models.PROTECT, verbose\_name="Интерфейс")  form\_factor = models.ForeignKey(FormFactor, on\_delete=models.PROTECT, verbose\_name="Форм-фактор")  read\_speed = models.PositiveIntegerField(verbose\_name="Скорость чтения (МБ/с)")  write\_speed = models.PositiveIntegerField(verbose\_name="Скорость записи (МБ/с)")  warranty\_years = models.PositiveIntegerField(null=True, blank=True, verbose\_name="Гарантия (лет)")  price = models.DecimalField(max\_digits=10, decimal\_places=2, verbose\_name="Цена")  in\_stock = models.BooleanField(default=True, verbose\_name="В наличии")   def \_\_str\_\_(self):  return f"{self.brand} {self.model}" |

Изменения:

- Созданы модели *Interface* и *FormFactor* для полей «один из».

- Поля *interfac* и *form\_factor* в модели SSD заменены на ForeignKey с on\_delete=models.PROTECT, чтобы предотвратить удаление связанных записей.

- Ограничения на поля сохранены: unique=True для sku, PositiveIntegerField для числовых полей, DecimalField для цены.

2. Админка (catalog/admin.py)

Все модели зарегистрированы с настройками для удобного управления данными.

|  |
| --- |
| from django.contrib import admin  from .models import SSD, Interface, FormFactor   @admin.register(Interface) class InterfaceAdmin(admin.ModelAdmin):  list\_display = ('name',)   @admin.register(FormFactor) class FormFactorAdmin(admin.ModelAdmin):  list\_display = ('name',)   @admin.register(SSD) class SSDAdmin(admin.ModelAdmin):  list\_display = ('brand', 'model', 'sku', 'price', 'in\_stock')  list\_filter = ('interface', 'form\_factor', 'in\_stock')  search\_fields = ('brand', 'model', 'sku')  fieldsets = (  ('Основное', {  'fields': ('sku', 'brand', 'model', 'price')  }),  ('Характеристики', {  'fields': ('capacity\_gb', 'interface', 'form\_factor')  }),  ('Дополнительно', {  'fields': ('warranty\_years', 'in\_stock')  }),  ) |

Изменения:

- Добавлены InterfaceAdmin и FormFactorAdmin для управления интерфейсами и форм-факторами.

3. Представления (catalog/views.py)

Представления: HomeView и SSDDetailView обновлены для работы с базой данных. В HomeView исправлена фильтрация по interface с учетом новой структуры моделей.

|  |
| --- |
| from django.db.models import Q from django.views.generic import ListView, DetailView, TemplateView  from .models import SSD   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')  min\_price = self.params.get('min\_price')  max\_price = self.params.get('max\_price')   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 min\_price:  queryset = queryset.filter(price\_\_gte=min\_price)  if max\_price:  queryset = queryset.filter(price\_\_lte=max\_price)   return queryset   def get\_context\_data(self, \*\*kwargs):  context = super().get\_context\_data(\*\*kwargs)  context['params'] = self.params  return context   class SSDDetailView(DetailView):  model = SSD  template\_name = "catalog/detail.html"   class AboutView(TemplateView):  template\_name = "catalog/about.html" |

Изменения:

- В HomeView фильтр по interface изменен на interface\_\_name, так как теперь это поле ссылается на модель Interface.

4. Генерация тестовых данных (catalog/management/commands/ seed\_catalog.py)

Команда seed\_catalog адаптирована для создания экземпляров Interface и FormFactor, а затем генерации SSD с ссылками на них.

|  |
| --- |
| from decimal import Decimal from django.core.management.base import BaseCommand from django\_seed import Seed from faker import Faker from catalog.models import SSD, Interface, FormFactor  class Command(BaseCommand):  help = "Генерирует тестовые данные для SSD"  fake = Faker(['en\_US'])   def add\_arguments(self, parser):  parser.add\_argument('--number', type=int, default=50, help='Количество SSD для генерации')  parser.add\_argument('--clear', action='store\_true', help='Очистить базу перед генерацией')   def handle(self, \*args, \*\*options):  if options['clear']:  SSD.objects.all().delete()  Interface.objects.all().delete()  FormFactor.objects.all().delete()  self.stdout.write(self.style.SUCCESS("База данных очищена"))   *# Создаем интерфейсы* interfaces = ['SATA', 'NVMe']  for interface in interfaces:  Interface.objects.get\_or\_create(name=interface)   *# Создаем форм-факторы* form\_factors = ['2.5"', 'M.2']  for form\_factor in form\_factors:  FormFactor.objects.get\_or\_create(name=form\_factor)   seeder = Seed.seeder()  seeder.faker = self.fake   brands = ["Samsung", "Western Digital", "Kingston", "Crucial", "SanDisk", "Intel", "Seagate"]  models = ["EVO", "Pro", "Blue", "Green", "Ultra", "Plus", "Extreme"]   seeder.add\_entity(SSD, options['number'], {  'sku': lambda x: f"SSD{seeder.faker.unique.random\_number(digits=8)}",  'brand': lambda x: seeder.faker.random\_element(brands),  'model': lambda x: f"{seeder.faker.random\_element(models)} {seeder.faker.random\_element(['Pro', 'Plus', 'Ultra', 'X', 'Lite'])}"[:50],  'capacity\_gb': lambda x: seeder.faker.random\_element([250, 500, 1000, 2000, 4000]),  'interface': lambda x: Interface.objects.order\_by('?').first(),  'form\_factor': lambda x: FormFactor.objects.order\_by('?').first(),  'read\_speed': lambda x: seeder.faker.random\_int(500, 7000),  'write\_speed': lambda x: seeder.faker.random\_int(400, 6000),  'warranty\_years': lambda x: seeder.faker.random\_int(1, 5),  'price': lambda x: Decimal(seeder.faker.random\_int(2000, 30000) + Decimal(seeder.faker.random\_int(0, 99)) / 100),  'in\_stock': lambda x: seeder.faker.boolean(chance\_of\_getting\_true=75),  })   seeder.execute()  self.stdout.write(self.style.SUCCESS(f"Успешно создано {options['number']} SSD-дисков")) |

Изменения:

- Добавлено создание объектов Interface и FormFactor перед генерацией SSD.

- Поля interface и form\_factor теперь ссылаются на случайные экземпляры из соответствующих моделей.

5. Шаблоны

Шаблоны home.html и detail.html остаются функциональными, так как interface и form\_factor автоматически преобразуются в строки благодаря методу \_\_str\_\_ в моделях Interface и FormFactor. Однако в home.html нужно обновить фильтр интерфейса, чтобы он соответствовал значениям name из модели Interface:

Остальные шаблоны (base.html, about.html) не требуют изменений.