

Geekbrains

Разработка интернет-магазина на фреймворке Django

ИТ-специалист:
Программист Python Цифровые профессии
Дружкова С. А.

Москва
2023

Содержание

| | |
|---|----|
| Введение..... | 3 |
| Назначение и цели создания сайта..... | 6 |
| Структура сайта: приложения и требования к функциональности | 7 |
| Связи между моделями и формы | 11 |
| Настройка административной панели | 14 |
| Главная страница..... | 16 |
| Галерея | 18 |
| Логика и реализация работы приложения..... | 18 |
| Работа с административной панелью | 22 |
| Каталог | 23 |
| Логика и реализация работы приложения..... | 23 |
| Работа с административной панелью | 29 |
| Корзина | 32 |
| Настройка работы корзины с помощью сессий | 32 |
| Создание формы для добавления товаров в корзину | 36 |
| Создание и настройка модели Заказов | 40 |
| Работа с административной панелью | 44 |
| Дальнейшее развитие проекта..... | 47 |
| Заключение | 48 |
| Использованная литература | 50 |

Введение

Несмотря на растущий интерес широкой аудитории к маркетплейсам и относительно доступный вход в онлайн-торговлю, бизнес не может рассчитывать на данный инструмент, как на единственный канал продаж. Штрафы, комиссии, высокая конкуренция на рынке – всё это несёт с собой большие издержки. Для того, чтобы не становится «заложником» маркетплейсов, бизнес должен иметь альтернативный канал сбыта в Интернете. Таким ресурсом служит корпоративный сайт.

Бытует мнение, что создание собственного ресурса затратный и трудоемкий процесс. Однако согласно маркетинговым исследованиям¹ 40% селлеров соглашались во мнении, что электронные площадки хороши для старта, в долгосрочной перспективе необходим корпоративный ресурс.

При планировании разработки сайта компании необходимо заранее определиться, какие цели мы ставим перед корпоративным ресурсом. Будет ли достаточно сайта-визитки с информационной справкой о компании или нам потребуется полноценный интернет-магазин или каталог со сложной структурой и встроенными сервисами.

Выбор CMS (система управления контентом) – это ответственный шаг, определяясь с которым стоит учитывать не только текущие цели, но и то, что нам может потребоваться в перспективе. В этом плане фреймворк Django выступает универсальным инструментом для разработки веб-приложения. Вне зависимости от того, что вам необходимо в настоящее время, в будущем уже существующий ресурс, можно изменять, дополнять, видоизменять.

Определимся с понятием, что такое фреймворк? Академия Яндекса говорит нам, что это готовый набор инструментов, который помогает разработчику быстро создать продукт: сайт, приложение, интернет-магазин, CMS-систему². Звучит неплохо, а что там с Django. Django считается один из самых популярных.

Нужен сайт-визитка, пожалуйста. К визитке необходимо добавить каталог? Django предоставляет нам такую возможность. Необходимо настроить работу с базой данных – пожалуйста. Нужен интернет-магазин, поддержка регистрации и личных кабинетов – такая возможность также есть.

¹ СЕЛЛЕРЫ НА РОССИЙСКИХ МАРКЕТПЛЕЙСАХ 2022. DATA INSIGHT

² Академия Яндекса. Что такое фреймворк

Благодаря такой гибкости фреймворк завоевал сердца многих компаний. В настоящее время на данной платформе функционируют такие гиганты, как: YouTube, Google, Pinterest, Reddit и другие.

Почему компании выбирают Django в качестве фреймворка для написания больших и сложных проектов. Отметим основные преимущества Django:

1. Всё включено – Django со старта предлагает разработчику большое разнообразие уже установленных инструментов и библиотек.
2. Развитая система – структура Django позволяет нам собирать ресурс, как конструктор;
3. Долгая история сервиса – Django на рынке разработки функционирует уже более 18 лет, за которые сервис успел развиваться в высокоуровневую платформу;
4. Расширяемость – большое количество библиотек и плагинов предоставляет даёт возможность быстро добавлять новые программные модули;
5. Библиотеки – с помощью библиотек мы получаем готовые решения реализации сложных задач. К популярным библиотекам Django можно отнести: Django REST Framework для работы с API, Pillow – для работы с изображениями, Django-rest-swagger – для автоматической генерации документации.
6. Административная панель – ресурс автоматически генерируется при создании приложений. Отсутствие необходимости создавать вручную административную панель – весомый аргумент при выборе инструмента разработки;
7. SEO-дружественность – в Django мы легко можем реализовать требующиеся функции для настройки поисковой оптимизации.
8. ORM - в Django представлено объектно-реляционное отображение, благодаря которому происходит взаимодействие приложение с базами данных.

Django написан на языке программирования Python, что полностью отражается в его структуре. В основе лежит архитектура MVT³ – Model-View-Template – модель-представление-шаблон.

³ Рейтинг ТОП 10 CMS сайтов: Какую лучше выбрать

В данной архитектуре в моделях мы описываем объекты, их свойства и функции. За счет моделей происходит создание и добавление объектов, их обновление и удаление.

В представлениях решаются задачи обработки HTTP-запросов, исполнения бизнес-логики с помощью прописанных методов и свойств, формирование ответов на запросы.

Если модели и представления – это бэкенд, то за фронтенд отвечают шаблоны. В Django продумана собственная система подключения HTML-шаблонов. Их главная функция в отличие от представлений не исполнение логики, а отображение данных. Стоит отметить, что шаблоны могут быть, как статическими, так и динамическими.

Возможность создавать интернет-ресурсы, написав всего несколько строк кода, очаровывает. Это обуславливает популярность Django в среде разработки и наш выбор в качестве дипломного проекта темы «Разработка интернет-магазина на фреймворке Django».

В своей дипломной работе мы ставим перед собой следующие задачи:

1. Изучение фреймворка Django и инструментов его работы
2. Разработка интернет-магазина
3. Выработка умений и навыков проектирования структуры базы данных

Отметим, что приоритетной целью для нас является реализация бизнес-логики, и вторичной – оформление страниц. Внешний дизайн должен быть интуитивно понятный, но простой.

Назначение и цели создания сайта

Своей целью мы ставим разработку интернет-магазина для компании DSL Wood. Компания занимается разработкой и изготовлением крафтовых изделий с гравировкой: ежедневников, альбомов, шкатулок. Ассортимент в настоящее время небольшой, но в будущем планируется расширение производства.

Сайт должен представлять компанию в Интернете. Быть визиткой компании -предоставляя Клиентам информацию о профиле компании, ассортименте, а также предлагать каналы коммуникации с продавцом.

На сайте должен быть представлен каталог компании с возможностью добавления новых разделов. Кроме того, должна быть реализована возможность добавления товаров в корзину и формирования заказов с сохранение информации о заказах в базе данных.

Сайт должен составлять единую структуру взаимосвязанных элементов.

Целевая аудитория ресурса – физические лица, а также корпоративные заказчики. Товары, изготавливаемые компаний, могут приобретаться в качестве именных подарков, брендированной имиджевой продукции.

От аудитории компании на маркетплейсах, аудитория интернет-ресурса отличается, тем, что в отличие от массового продукта, представленного на маркетплейсе, в интернет-магазине Клиенты могут, как заказать продукцию с индивидуальным дизайном, так и выбрать готовое решение.

Структура сайта: приложения и требования к функциональности

При запуске проекта на Django на старте автоматически создается директория с вложенными файлами. Структура проекта при первом запуске имеет следующий вид:

- myproject/
 - manage.py
 - myproject/
 - __init__.py
 - settings.py
 - urls.py
 - asgi.py
 - wsgi.py

С помощью файла manage.py мы будем осуществлять управление проектом – запускать сервер, создавать новые приложения, выполнять миграции информации в базу данных, создавать администратора, создавать и менять пароль и много другое.

Далее идет пакет Python одноименный с нашим проектом, в примере выше это пакет с названием «myproject». Как пакет его определяет наличие файла __init__.py. Данный пакет содержит стандартные файлы, которые используются для настройки проекта:

- settings.py – файл, в котором прописываются настройки проекта. Включает в себя описание путей к приложениям, настройки статики, важные параметры, связанные с работой подключаемых модулей и настроек безопасности.
- urls.py – в данном файле прописываются маршруты приложения. По адресам, прописанным в данном файле, будет строиться навигация на нашем ресурсе. На смотря на то, что данный файл может содержать все ассоциации url адресов с представлениями, принято разделять его на части и в каждом приложении создавать свой файл urls.py.
- asgi.py – модуль, описывающий связь проекта с веб-сервером через интерфейс WSGI.
- wsgi.py - модуль, описывающий связь проекта с веб-сервером посредством интерфейса ASGI.

Если стартовый пакет содержит файлы, отвечающие за общие настройки проекта, то за функциональная логика прописывается в приложениях.

Приложение в Django⁴ представляет собой отдельный фрагмент функциональности сайта. Оно может, как реализовать работу всего сайта, так и отдельной функциональности (раздела или подсистемы). При разработке структуры сайта рекомендуется выделять приложения по выполняемому спектру задач

В нашем проекте мы выделяем следующие приложения:

- Приложение «main» - данный пакет будет отвечать за работу главной страницы сайта и страницы с информацией о доставке. Также здесь мы создадим директорию templates, в которой будут храниться базовые шаблоны, содержащие общие для всего ресурса элементы: шапку сайта с меню, подвал с общей информацией.
- Приложение «gallery» - этот пакет будет отвечать за работу модуля Галерея. На сайте данный модуль будет представлен в разделе «Наши работы». Он представляет собой раздел с подгружаемыми фотографиями из базы данных. Первоочередная задача данного модуля – репрезентативная выдача фотографий с возможностью увеличения изображений.
- Приложение «catalog» - пакет, отвечающий за работу каталога товаров. Раздел на сайте, в котором реализуется логика данного приложения также называется каталог.
- Приложение «cart» - пакет, в котором будет прописана логика работы корзины: добавление товаров в корзину, процесс оформления заказов, сохранения информации о заказах в базе данных.

Каждое приложение после создания необходимо зарегистрировать в основном файле settings.py:

```
INSTALLED_APPS = [  
    'main',  
    'catalog',  
    'gallery',  
    'cart',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

⁴ Введение в Django & DevOps

Как мы видим, кроме прописанных нами приложений в `INSTALLED_APPS` есть и другие записи – стандартные приложения Django, в которых реализуются встроенные подсистемы. Мы также будем использовать их в своей работе. Например:

`django.contrib.auth` – описывает подсистему разграничений доступа;

`django.contrib.sessions` – обслуживает серверные сессии.

А также прописать маршруты в файле `urls.py` основного пакета:

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path("", include('main.urls')),  
    path('cart/', include('cart.urls')),  
    path('catalog/', include('catalog.urls')),  
    path('gallery/', include('gallery.urls')),  
]
```

Каждое приложение при создании будет иметь общий набор файлов:

- `__init__.py` – определяющий приложение, как пакет.
- `admin.py` – файл, в котором мы будем настраивать при необходимости отображение моделей приложения в административной панели.
- `apps.py` – файл, в котором подключаемое приложение определяют как подкласс `AppConfig`. Это необходимо для автоматического подключения данной конфигурации в Django.
- `models.py` – файл, в котором мы будем создавать модели (классы) для создания таблиц в базе данных.
- `tests.py` – файл, который используется для написания и проведения тестов.
- `views.py` – файл, в котором прописывают представления.

Нам в обязательном порядке необходимо будет создать файл `urls.py`, чтобы прописать маршруты url адресов. Мы бы назвали маршрутизацию в Django многослойной. У нас есть первый слой адресов, прописанный в `settings.py` основного приложения – здесь мы прописываем адреса наших основных модулей. Непосредственно в приложениях в файлах `urls.py` мы прописываем следующий слой адреса, которых мы будем добавлять к основному адресу модуля. Объясним на примере:

1. есть адрес сайта: <http://127.0.0.1:8000/>
2. добавляем к нему «cart/» из `settings.py` основного приложения:
<http://127.0.0.1:8000/cart/> - переходим в корзину

3. добавляем к полученному адресу «order_create/» из файла urls.py из приложения cart: http://127.0.0.1:8000/cart/order_create/ - и переходим на страницу оформления заказа.

Кроме файла urls.py, по необходимости мы можем добавлять в приложение любые файлы, описывающие требующуюся нам функциональность. Например, forms.py для создания новых форм.

Кроме созданных приложений в корневой директории проекта также будут:

- Папка images, в которую будут сохраняться изображения
- Директория static, в которой мы создадим статические файлы форма CSS для описания стилей, подключаемых к HTML-шаблонам нашего проекта.
- db.sqlite3 – база данных, в которой будут храниться наши модели и данные.

Структура сайта, которую будет видеть перед собой посетитель ресурса, должна выглядеть следующим образом:

- Главная страница
- Каталог
- Доставка
- Наши работы
- Корзина

Навигация на сайте осуществляется в соответствии с вышеописанной структурой и отображается в главном меню:

ГЛАВНАЯ | КАТАЛОГ | ДОСТАВКА | НАШИ РАБОТЫ | КОРЗИНА

Главное меню и ссылки в нём должны быть доступны пользователю с любой страницы ресурса. Это требование учитывается в базовом шаблоне. В нашем проекте данный шаблон будет носить название layout.html.

В подвале сайта мы размещаем информацию об авторе проекта.

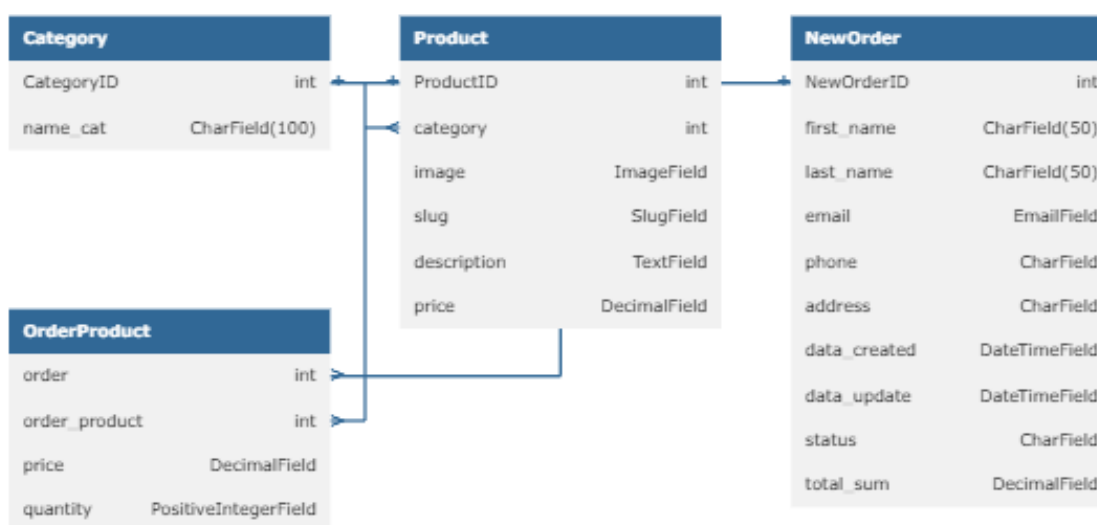
Связи между моделями и формы

Приложения проекта Django могут получать доступ, управлять и взаимодействовать с данными через Python модели. В моделях мы заранее определяем структуру данных, типы полей, а при желании можем указать какие значения по умолчанию будут иметь данные поля.

Модели принято определять в приложении `models.py`. Они реализуются как подклассы `django.db.models.Model`⁵, и могут включать поля, методы и метаданные.

Модели отражают в себе суть объектно-ориентированного программирования. В них мы заранее прописываем из каких объектов будет состоять наш ресурс, его свойства и базовое поведение.

Далее мы подробно рассмотрим каждую модель из используемых нашем в проекте. В данном же разделе опишем общую структуру базы данных, наглядно продемонстрируя её на схеме отношений таблиц:



На диаграмме выше представлена структура связей продуктов и заказов. Основой для нашей работы будет служить таблица продуктов **Product**. Все продукты подразделяются на категории – таблица **Category**, что отражено в связи:

```
category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

Для оформления заказов и хранения данных о них используется две таблицы **NewOrder** и **OrderProduct**. В первой храниться информация о клиенте, дата создания и обновления заказа, а также данные необходимые для доставки.

⁵ Django учебник Часть 3: Использование моделей

Вторая таблица OrderProduct отвечает за связь таблицы NewOrder и OrderProduct. Это отражено в следующих полях:

```
order = models.ForeignKey(NewOrder, on_delete=models.CASCADE)
order_product = models.ForeignKey(Product, on_delete=models.CASCADE)
```

В работе приложения Галерея задействована другая таблица Image. Она не имеет созависимостей с другими таблицами. Структура таблицы представлена ниже:

| Image | |
|-------------|---------------|
| ImageID | int |
| name | CharField(50) |
| description | TextField |
| image | ImageField |

Заполнение корзины и оформление заказов происходит со стороны Клиента. Когда информацию необходимо получать со стороны пользователя при разработке сайтов используются формы. Формы в Django прописываются внутри приложений в файле forms.py.

В нашем проекте будет использовано два типа форм:

1. Форма CartAddProductForm, которая наследует свои свойства от класса forms.Form. Данная форма будет применяться для добавления товаров в корзину.
2. Форма OrderCreate, которая наследует свои свойства от класса forms.ModelForm, и напрямую связана с таблицей NewOrder. Данная форма будет использоваться для оформления заказов.

Работы вышеупомянутых форм мы подробнее рассмотрим в соответствующих разделах.

Для работы с базой данных проекта проверим файл settings.py основного проекта, а именно эту строку:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

В настройках прописано, что для нашего проекта мы будем использовать базу данных SQLite, что подходит для нас, так как не предполагает большое количество запросов. Данная настройка идёт «с коробки» и менять мы её не будем.

SQLite представляет собой быструю и легкую базу данных, которая предоставляет возможность хранить данные прямо внутри нашего приложения⁶. Отметим ряд преимуществ использования SQLite:

1. Простота - SQLite не требует отдельного сервера или настройки. База данных работает, как часть приложения;
2. Надежность: Транзакции и ACID-свойства обеспечивают надежность и целостность данных;
3. Кроссплатформенность: SQLite поддерживается на множестве платформ, включая Windows, macOS и Linux;
4. Эффективность: SQLite требуются минимальные ресурсы системы, что отлично подходит, например, для мобильных устройств.

⁶ Работа с SQLite в Python

Настройка административной панели

Django дает разработчикам возможность использовать модели для автоматической генерации административной панели. Последняя предоставляет собой полноценный интерфейс и даёт нам возможность управлять данными приложений: создавать, редактировать и удалять записи в базе данных, а также производить множество иных действий, управляющих работой приложений.

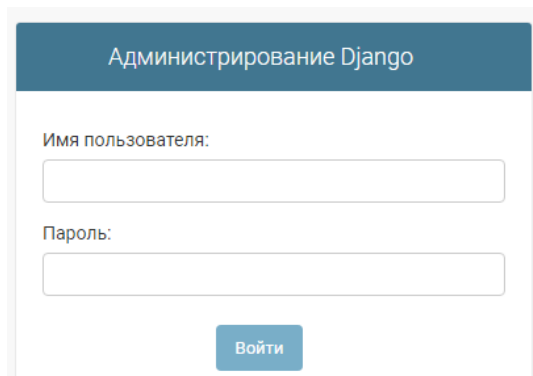
Административная панель даёт нам возможность быстро и удобно управлять данными и значительно упрощает жизнь разработчика в том числе и за счёт того, что является встроенным элементом и поставляется из «коробки».

С момента создания проекта административная панель уже доступна для использования по адресу <http://127.0.0.1:8000/admin/>. Но для работы с ней необходимо выполнить ряд настроек.

В первую очередь внесём изменения в файл `settings.py`⁷ основного приложения и изменим язык отображения административной панели:

```
LANGUAGE_CODE = 'ru'
```

После этого при обновлении страницы админ-панель станет русифицированной:



Для входа нам необходимо будет создать суперпользователя. При запуске проекта в базе данных зарегистрированных пользователей нет. Суперпользователь создается из консоли с помощью команды:

- `python manage.py createsuperuser`

Далее нам будет предложено системой ввести логин, email и пароль для нашего администратора. После это по введенным данным мы сможем зайти в административную панель.

⁷ Начинаем работу с Django — подключение админки

На старте админ-панель также будет пуста. Мы заполним её информацией в процессе разработки проекта.

Отметим, что разработчики фреймворка Django не просто с коробки добавили в него административную панель, но заложили возможность настраивать интерфейс под требования администратора сайта.

Кроме того, мы можем добавлять различные группы пользователей с разным уровнем доступа, в зависимости от стоящих перед ними задач. Например, менеджеру, обрабатывающему заказы, нет необходимости давать доступ к созданию и удалению пользователей, товаров или категорий. Мы можем открыть для него таблицу заказов на просмотр и редактирование. Это не только позволит избежать ошибок в работе сайта из-за неумелого обращения, но и облегчит процесс обучения персонала более ограниченному интерфейсу.

Главная страница

За отображение Главной страницы, а также страницы «Доставка» отвечает приложение `main`. В данном проекте эти страницы носят по большей степени информационный характер.

Представления, описанные в файле `views.py`, базовые в формате запрос-ответ. Главная роль в данном приложении отводится шаблону, расположенным в директории `templates`. Именно они отвечают за визуальное отображение страницы.

Первым мы бы отметили шаблон, играющий роль базового для всего проекта, - `layout.html`. В качестве общих элементов для всего ресурса мы описываем: шапку сайта, футер, а также подключаемые стили, общие для всех страниц сайта. Для работы базового шаблона мы используем шаблонизатор Jinja⁸. Для отображения переменных в наших HTML-шаблонах мы будем использовать двойные фигурные скобки `{{ }}`, инструкция `if`, циклы `for`.

Стили CSS для базового шаблона лежат в отдельной папке `static` в корневом каталоге проекта.

Для создания визуала используем в том числе стили Bootstrap. Для подключения прописываем соответствующую ссылку в блоке `head` файла `layout.html`

Кроме базового шаблона общие для всей стилистики сайта являются:

- Фиксированная шапка сайта.
- Фиксированный футер с информацией о разработчике.
- Шрифт - 'Ubuntu Condensed', sans-serif.
- Общая стилистика отображения фотографий: форма.
- Единая цветовая гамма: активные элементы выделены цветом `#fa8e47`.
- Основное поле контента располагается по центру страницы.

Главная страница встречает нас кратким описанием деятельности компании, визуальной заставкой, а также легкой навигацией. Клиенту предлагается открыть страницу с отзывами о компании или перейти в каталог:

⁸ Основы шаблонизатора Jinja

DSL Wood

Сувенирная продукция с лазерной гравировкой. Принимаем к изготовлению заказы от 1 шт, так и серийно.

Разработаем и согласуем дизайн проект за 3 рабочих дня

Срок изготовления продукции на заказ - 3-7 рабочих дней

При заказе продукции из наличия - отправка в день заказа

По всем вопросам обращайтесь по телефонам:

+7 (904) 022-5х-91, +7 (904) 022-3х-80

ОТЗЫВЫ О НАС

КАТАЛОГ



На странице «Доставка» расположен информационный блок об организации доставки продукции, вариантах оплаты.

Доставка и оплата

Условия доставки

Доставка по Центральным регионам России - бесплатно при заказе более 50 000 рублей.

Доставка в другие регионы - по согласованию с менеджером.

Наша компания заботится об удобстве покупателей, мы сотрудничаем со всеми ведущими транспортными компаниями, что позволяет отделу логистики оптимизировать скорость и качество доставки в любой регион в зависимости от объема и веса.

Все заказы Вы легко сможете отслеживать по трек номерам транспортных компаний. Также Вы можете самостоятельно забрать любой готовый заказ совершенно бесплатно с нашего склада по адресу:

г. Москва, Алтуфьевское шоссе, д.37, стр.1, БЦ "Аврора", 3 этаж, офис 307.

График работы склада ПН-ПТ с 10:00 до 17:00.

Условия оплаты

Мы отгружаем заказы на условиях 100% предоплаты, однако постоянным партнерам может быть предоставлена отсрочка платежа (кредитная линия), величина которой зависит от объема продаж компании-партнера.

Все условия Вы сможете согласовать непосредственно с нашим менеджером, отвечающим за Ваш заказ.

Галерея

Логика и реализация работы приложения

Приложение gallery отвечает за работу блока «Наши работы». Данный блок представляет собой галерею с изображениями. Цели, которые мы ставили перед собой, планируя разработку данного блока:

1. Разработка возможности добавления новых изображений в блок без правки кода.
2. Разработка шаблона с адаптивной сеткой для удобного отображения страницы на любом устройстве.
3. Создание интерактивных элементов на странице – возможность увеличивать изображения.

Для хранения и отображения изображений была создана модель Image в файле models.py:

```
class Image(models.Model):
    name = models.CharField('Название', max_length=50)
    description = models.TextField('Описание')
    image = models.ImageField('Изображение', upload_to='images/')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'Галерея'
        verbose_name_plural = 'Галерея'
```

В классе Meta мы прописываем комфортные для нас названия блока, которые будут отображаться в административной панели.

В файле admin.py регистрируем нашу модель для отображения в административной панели:

```
admin.site.register(Image)
```

Также для корректной работы с изображениями нам необходимо добавить следующие строки кода в файл settings.py основного приложения:

```
MEDIA_DIR = os.path.join(BASE_DIR, 'images')
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'
```

В данных строках кода мы прописываем путь к серверу для хранения файлов. MEDIA_URL⁹- это ссылочный URL-адрес для браузера для доступа к файлам через HTTP.

Представление в файле view.py также довольно простое:

```
def gallery(request):
    images = Image.objects.all()
    return render(request, 'gallery/gallery.html', {'images': images})
```

В images мы сохраняем изображения из базы данных.

Сложнее и интереснее код в шаблоне gallery.html:

```
<div class="container">
  <div class="grid-layout">
    {% for image in images %}
    <div class="grid-item grid-item-1">

    </div>
    {% endfor %}
  </div>
</div>
```

Мы формируем контейнер по тегу div. Далее создаем адаптивную сетку, описанную в классе css grid-layout. Далее в каждый отдельный элемент адаптивной сетки мы подгружаем изображение из нашей базы данных, перебирая весь список images.

Отзывчивая css сетка задается следующим стилям:

```
.grid-layout {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
  grid-gap: 10px;
  grid-auto-rows: minmax(180px, auto);
  grid-auto-flow: dense;
  padding: 10px;
}

.grid-item {
  border-radius: 6px;
}
```

⁹ Python - Загрузка изображений в Django

Стилистика изображений, поведение до и после наведения курсора на объект, а также увеличение изображения описываются стилями изображения, применяемыми к тегу `img`.

Отображение блока «Наши работы» до взаимодействия с объектами (наведения курсора):

[ГЛАВНАЯ](#) | [КАТАЛОГ](#) | [ДОСТАВКА](#) | [НАШИ РАБОТЫ](#) | [КОРЗИНА](#)

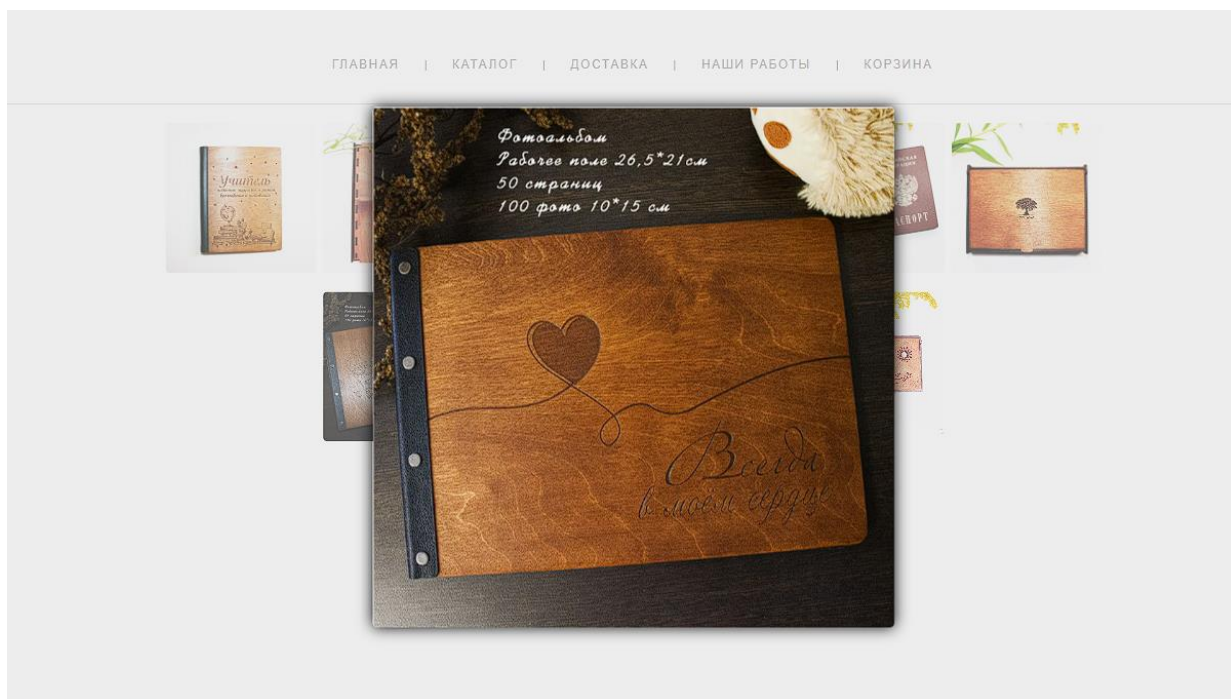


Светлана Дружкова * 2023 год

Каждое новое изображение будет органично встраиваться в следующую ячейку сетки Галереи:



При клике на изображение, фотография увеличивается. Если убрать курсор с фото, изображение принимает прежний размер:



Подводя итоги, скажем, что блок «Наши работы» полностью выполняет задачи, поставленные нами на этапе планирования данного раздела.

Работа с административной панелью

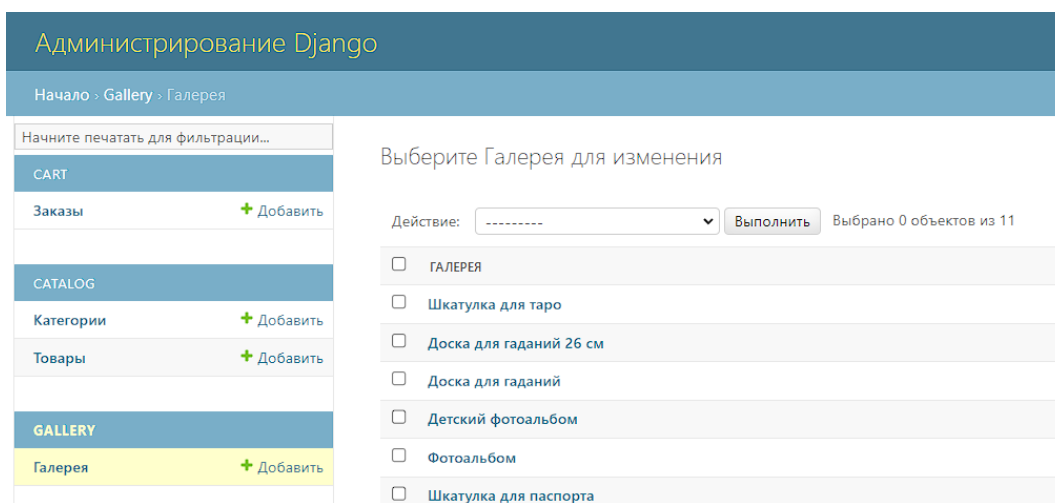
Настроим возможность подгружать изображения в нашу галерею через панель администратора для это в файле `admin.py` зарегистрируем нашу модель:

```
admin.site.register(Image)
```

А также добавим класс `Meta` в описание модели `Image`, в котором назначим имена для модели в админ-панели:

```
class Meta:
    verbose_name = 'Галерея'
    verbose_name_plural = 'Галерея'
```

Проведя данные изменения, мы получаем удобное отображение класса `Image` в административной панели:



Для удобства мы дали собственные названия каждому полю на страницы добавления/изменения объекта класса `Image`:

Изменить Галерея ИСТОРИЯ

Шкатулка для таро

Название:

Описание:

Изображение: На данный момент: images/Снимок.PNG
Изменить: Файл не выбран

Каталог

Логика и реализация работы приложения

Приложение catalog отвечает за одноименный блок «Каталог». Так как наш сайт носит презентационный характер, в настоящее время мы не ставим для себя задачу разработки сложной онлайн площадки. Ассортимент узкий и пока не требует сложных систем. Какие цели мы ставили перед собой разрабатывая данный блок:

1. Создать каталог товаров с удобной системой ранжирования по категориям.
2. Карточки товаров должны отображаться в виде адаптивной сетки.
3. Добавление новых товарных позиций должно легко осуществляться из административной панели сайта, не требуя от администратора внесения изменений в код.
4. Добавление новых категорий должно легко осуществляться из административной панели сайта, не требуя от администратора внесения изменений в код.

Для выполнения данных задач в файле models.py приложения мы создаем две взаимосвязанные модели Category и Product.

Модель Category будет использована для фильтрации продуктов по категориям, как в html-шаблонах при настройке визуального отображения блока, так и в административной панели. Описание класса будет выглядеть следующим образом:

```
class Category(models.Model):
    CHOICES_STATUS = (
        ('Опубликовано', 'Опубликовано'),
        ('Скрыто', 'Скрыто'),
    )

    name_cat = models.CharField(max_length=100)
    status = models.CharField(choices=CHOICES_STATUS, max_length=100,
                              default='Опубликовано')
```

Поле name_cat представляет собой текстовое поле для названия категории. В поле status подставляется одна из определенных констант из списка вариантов. В данном случае список представлен под именем CHOICES_STATUS.

Статус категории может быть «опубликована» или «скрыта» — это позволит нам легко регулировать отображение категорий на сайте и скрывать неактуальные группы товаров, не удаляя их. Например, такая логика будет

удобной в случае с сезонными товарами, выдачу которых необходимо регулировать в зависимости от времени года.

Модель Product несёт в себе основную информацию о товаре:

```
class Product(models.Model):
    title = models.CharField(verbose_name='Название', max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
    verbose_name='Категория')
    image = models.ImageField(verbose_name='Фото', upload_to='products/')
    slug = models.SlugField(max_length=50, unique=True, default=title,
    verbose_name='URL')
    description = models.TextField(verbose_name='Описание', default="")
    price = models.DecimalField(verbose_name='Стоимость', max_digits=10,
    decimal_places=2)
```

Рассмотрим подробнее поля и их типы:

- Поле title – название продукта.
- Поле category – данное поле связывает модель Product с таблицей категорий по внешнему ключу. При этом при описании поля стоит параметр on_delete со значением models.CASCADE. При такой настройке работы поля, при удалении соответствующей категории будут удалены все продукты, связанные с ней по внешнему ключу.
- В поле image будет храниться изображение. Папкой для хранения изображения товаров назначаем products.
- Поле slug – в данном поле будет прописываться часть адресной строки, по которой мы хотели бы отображать данный продукт.
- Поле description – текстовое поле, которое содержит информацию о товаре.
- Поле price – десятичное поле, количество цифр после запятой в котором ограничивается двумя.

Для реализации наших задач нам потребуются три представления. Рассмотрим каждое из них. Переходим к файлу views.py.

Первое представление будет отвечать за то, что мы видим, открывая страницу каталога без фильтрации по категориям. Рассмотрим код:

```
def category_products(request):
    """ Представление для списка категорий и продуктов """
    status = 'Опубликовано'
    categories = Category.objects.filter(status=status).all()
    category = None
    products = Product.objects.all()
    form = CartAddProductForm()
```



```

return render(request, 'catalog/product_detail.html', {'category': category,
'categories': categories,
'products': products,
"form": form})

```

Мы получаем категории со статусом «Опубликовано» из таблицы Category.

В products подгружаем все продукты из таблицы Product.

В данном представлении у нас также присутствует объект form. Его функция – добавление товара в корзину. Его принцип работы мы рассмотрим чуть позже при изучении работы приложения cart.

Фильтрация продуктов по категориям выполняется с помощью следующего приложения:

```

def cat_slug_products(request, category_slug=None):
    """ Представление для списка категорий и продуктов по категориям """
    status = 'Опубликовано'
    categories = Category.objects.filter(status=status).all()
    category = None
    products = None
    form = CartAddProductForm()
    if category_slug:
        category = Category.objects.get(id=category_slug)
        products = Product.objects.filter(category=category)
    else:
        products = Product.objects.all()
    return render(request, 'catalog/product_detail.html', {'category': category,
'categories': categories,
'products': products,
"form": form})

```

В файле urls.py приложения catalog прописываем следующие ссылки:

```

urlpatterns = [
    path("", views.category_products, name='category_products'),
    path('<slug:category_slug>/', views.cat_slug_products,
name='cat_slug_products'),
]

```

Представление cat_slug_products получает в запросе идентификатор категории, по которому происходит фильтрация выдачи продуктов. Оба представления работают на базе одного HTML-шаблона product_detail.html.

В данном HTML-шаблоне тело страницы мы делим на две части. В левой будут выводиться категории в формате интерактивного меню. При

добавлении, внесении изменений или удалении категории – информация будет автоматически подгружаться в боковое меню. Рассмотрим код:

```
<div class="vertical-menu">
  {% for category in categories %}
    <a href="{% url 'cat_slug_products' category_slug=category.id %}">
      {{category.name_cat}} </a>
    {% endfor %}
</div>
```

Как мы видим в представленном коде строки меню автоматически подгружаются из таблицы Category. Каждая строка несёт в себе ссылку на соответствующий раздел категории – данная ссылка и пробрасывается в представление cat_slug_products.

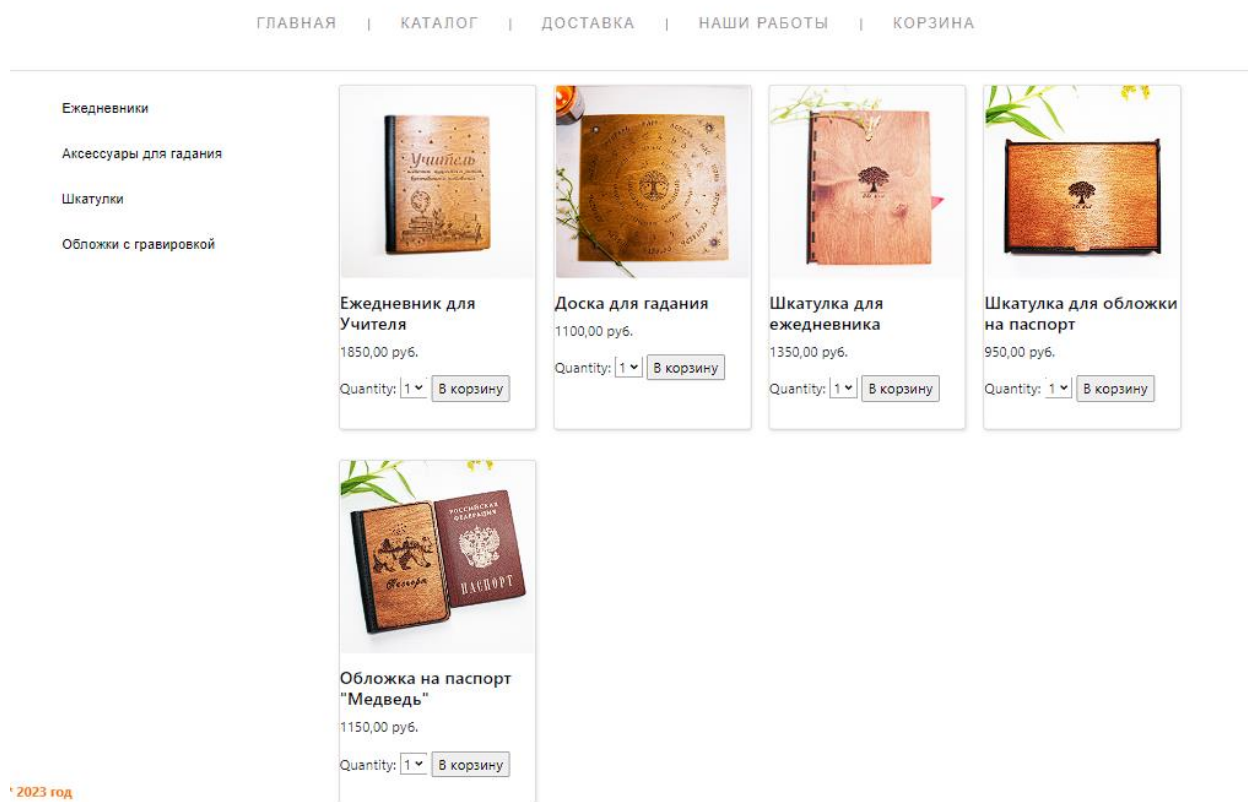
На правой части страницы в адаптивную сетку подгружаются продукты из таблицы Product:

```
<div class="cards">
  {% for product in products %}
    <div class="card">
      <div class="card__top">
        <a href="#" class="card__image">
          
        </a>
      </div>
      <div class="card__bottom">
        <h5 class="product_name">{{product.title}}</h5>
        <p class="product_price">{{product.price}} руб. </p>
        <form action="{% url 'cart_add' product_id=product.id %}"
method="post">
          {% csrf_token %}
          {{ form }}
          <input type="submit" value="В корзину">
        </form>
      </div>
    </div>
  {% endfor %}
</div>
```

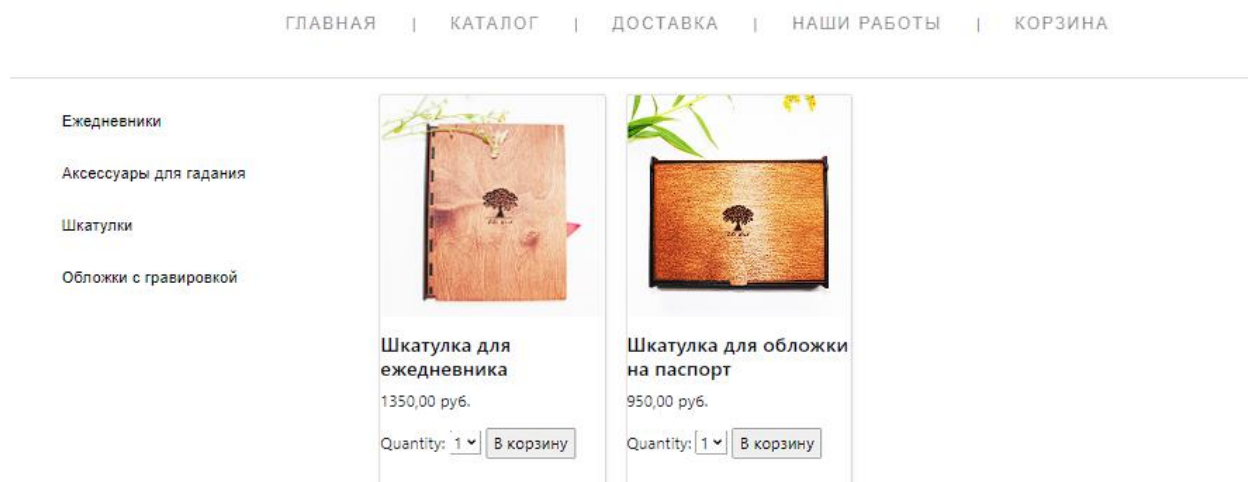
В шаблоне выводятся по каждому продукту: название, изображение, стоимость. Также здесь представлена форма добавления товара в корзину.

При клике на раздел «Каталог» в главном меню, срабатывает представление category_products, в котором выводится весь список товаров.

Отметим, что при добавлении или удалении товара из базы данных шаблон автоматически подстраивается под изменения. Визуально страница отображается так:



Пункт меню категорий, при наведении курсора, меняет цвет. После клика мы переходим в соответствующий раздел и видим товары только из указанной категории:



В HTML-шаблон мы также добавили блок `<style>`, в котором указали несколько стилей, работающих с данным модулем. За разделение страницы на блоки отвечают следующие стили:

```
.col-sm-3 {  
    width: 25%;  
}  
  
.col-sm-9 {  
    width: 75%;  
}
```

Работа вертикального меню, в которое подгружается список категорий, настраивается в CSS с помощью тега `vertical-menu`.

Подводя итоги данного блока, можем сказать, что нам удалось добиться решения поставленных задач:

- ✓ Все изменения в базе данных по товарам и категориям автоматически отражаются на сайте, не требуя правки кода.
- ✓ Товары в каталоге представлены в формате удобной адаптивной сетки.

Работа с административной панелью

Для добавления, внесения правок или удаления товаров или категорий нами планируется использовать функционал админ-панели. Для этого нам необходимо настроить удобное отображение блоков в админ-панели.

В модели Category прописываем класс Meta:

```
class Meta:
    verbose_name = 'Категория'
    verbose_name_plural = 'Категории'
```

Те же действия выполняем в модели Product:

```
class Meta:
    verbose_name = 'Товар'
    verbose_name_plural = 'Товары'
```

Регистрируем модели в файле admin.py:

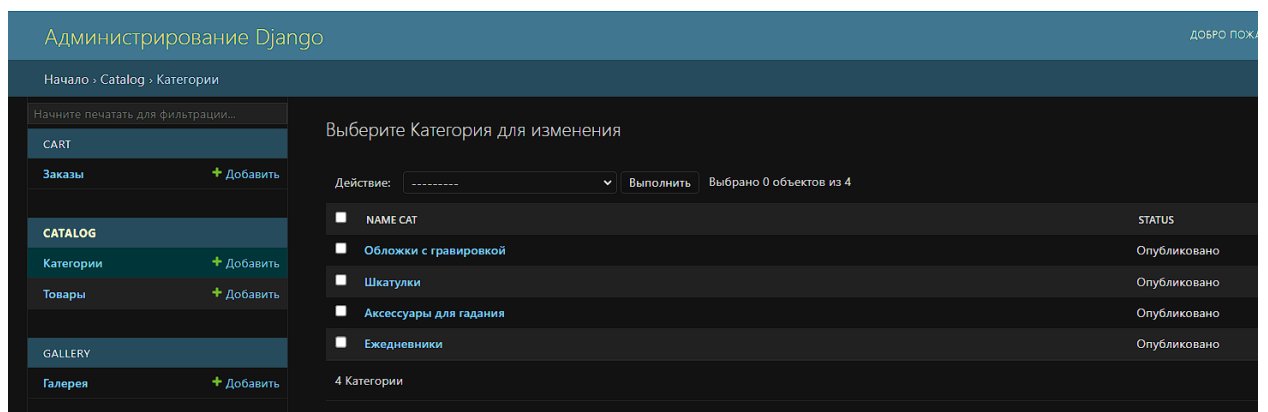
```
admin.site.register(Category, CategoryAdmin)
admin.site.register(Product, ProductAdmin)
```

Также для удобного отображения таблицы продуктов в административной панели добавляем следующую настройку:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = ['title', 'category', 'price']

class CategoryAdmin(admin.ModelAdmin):
    list_display = ['name_cat', 'status']
```

Если мы откроем админ-панель, то увидим, что в приложении CATALOG у нас отображается две модели:



В разделе Категории мы видим название и статус по каждой категории.

В разделе Товаров в общей выдаче представлены колонки название, категория, цена:

Выберите Товар для изменения

Действие: Выбрано 0 объектов из 5

| <input type="checkbox"/> | НАЗВАНИЕ | КАТЕГОРИЯ | СТОИМОСТЬ |
|--------------------------|---------------------------------|------------------------|-----------|
| <input type="checkbox"/> | Обложка на паспорт "Медведь" | Обложки с гравировкой | 1150,00 |
| <input type="checkbox"/> | Шкатулка для обложки на паспорт | Шкатулки | 950,00 |
| <input type="checkbox"/> | Шкатулка для ежедневника | Шкатулки | 1350,00 |
| <input type="checkbox"/> | Доска для гадания | Аксессуары для гадания | 1100,00 |
| <input type="checkbox"/> | Ежедневник для Учителя | Ежедневники | 1850,00 |

5 Товары

Взаимодействовать с каждым товаром в качестве админа мы будем посредством административной панели. При добавлении товара система предложит нам выбрать категорию из представленного списка или добавить новую:

Изменить Товар

Обложка на паспорт "Медведь"

ИСТОРИЯ

СМОТРЕТЬ НА САЙТЕ

Название:

Категория:

Фото: На данный момент: products/1_43j858.jpg
Изменить: Файл не выбран

URL:

Описание:

Стоимость:

СОХРАНИТЬ

Сохранить и добавить другой объект

Сохранить и продолжить редактирование

Удалить

Как мы видим страница изменения товаров полностью русифицирована. Ранее при описании модели Product нами был прописан у каждого свойства модели атрибут verbose_name.

Для удобной фильтрации товаров в файл admin.py пропишем настройку фильтрации:

```
class ProductAdmin(admin.ModelAdmin):
    list_display = ['title', 'category', 'price']
    list_filter = ['category', 'price']
```

Получаем более удобную для пользователя модель отображения списка товаров:

Выберите Товар для изменения

ДОБАВИТЬ ТОВАР +

Действие: ----- Выполнить Выбрано 0 объектов из 5

| <input type="checkbox"/> НАЗВАНИЕ | КАТЕГОРИЯ | СТОИМОСТЬ |
|--|------------------------|-----------|
| <input type="checkbox"/> Обложка на паспорт "Медведь" | Обложки с гравировкой | 1150.00 |
| <input type="checkbox"/> Шкатулка для обложки на паспорт | Шкатулки | 950.00 |
| <input type="checkbox"/> Шкатулка для ежедневника | Шкатулки | 1350.00 |
| <input type="checkbox"/> Доска для гадания | Аксессуары для гадания | 1100.00 |
| <input type="checkbox"/> Ежедневник для Учителя | Ежедневники | 1850.00 |

5 Товары

ФИЛЬТР

Категория

Все
Ежедневники
Аксессуары для гадания
Шкатулки
Обложки с гравировкой

Стоимость

Все
950.00
1100.00
1150.00
1350.00
1850.00

Полностью настроив работу административной панели по данным разделам, переходим к следующему блоку.

Корзина

Настройка работы корзины с помощью сессий

Приложение cart отвечает за обработку заказов и функционал корзины покупок. При разработке данного приложения мы ставим перед собой следующие задачи:

1. Обеспечить для пользователей возможность выбирать товары и добавлять необходимое количество в корзину.
2. Временно хранить данные в корзине во время посещения пользователем сайта.
3. Настроить сохранение корзины в сессии.
4. Настроить возможность оформления заказов.
5. Сохранять данные по заказу в базе данных.
6. Настройка удобного отображения таблицы заказов в админ-панели.

Для организации работы корзины мы будем использовать механизм Django's session framework. Он позволяет сохранять информацию на сервере во время взаимодействия с веб-ресурсом¹⁰.

Django – это session framework, поддерживающую анонимные и пользовательские сессии¹¹. Данные сессии юзера по умолчанию хранятся в базе данных. Сеансы осуществляются посредством файла и кэша.

Чтобы использовать возможности сессий, во-первых, проверим настройки, прописанные в файле settings.py основного приложения. Нас интересует параметр MIDDLEWARE_CLASSES, а именно строка:

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

Указанное программное обеспечение осуществляет управление сессиями. Строка добавляется по умолчанию при создании проекта. Объект сессии request.session предоставляет доступ к текущей сессии.

Словарь сессий принимает любой объект, который можно сериализовать в JSON.

Для указания места хранения сессии необходимо настроить параметр SESSION_ENGINE. Для нашего проекта мы будем использовать Database sessions. При выборе данного варианта хранения сессий данные записываются и хранятся в базе данных.

¹⁰ Сессия пользователя Django – установка сеанса

¹¹ Django в примерах

Для реализации приложения необходимо создать структуру, которую мы сможем сериализовать в JSON. Корзина будет включать следующие элементы:

- id Продукта;
- Количество товара;
- Стоимость единицы товара.

Для хранения и извлечения информации используется словарь Python.

Добавим ещё одну настройку в settings.py основного приложения следующую строку кода:

```
CART_SESSION_ID = 'cart'
```

Этот параметр представляет собой ключ, который будет использован для хранения корзины в сессии пользователя.

Описание работы сессий мы будем выполнять в файле cart.py в приложении cart. В этом файле создаем класс Cart, с помощью которого будем управлять корзиной:

```
class Cart(object):

    def __init__(self, request):
        """
        Инициализация корзины
        """
        self.session = request.session
        cart = self.session.get(settings.CART_SESSION_ID)
        if not cart:
            # сохраняем ПУСТУЮ корзину в сессии
            self.session[settings.CART_SESSION_ID] = dict()
            cart = self.session[settings.CART_SESSION_ID]
        self.cart = cart
```

В __init__ мы инициализируем корзину, получая возможность хранить текущую сессию с помощью self.session = request.session.

Для добавления товаров в корзину, обновления их количества и удаления добавим методы add(), save() и remove():

```
def add(self, product, quantity=1, update_quantity=False):
    """
    Добавляем товар в корзину или обновляем его количество.
    """
    product_id = str(product.id)
    if product_id not in self.cart:
        self.cart[product_id] = {'quantity': 0,
                                  'price': str(product.price)}
```

```

if update_quantity:
    self.cart[product_id]['quantity'] = quantity
else:
    self.cart[product_id]['quantity'] += quantity
self.save()

def save(self):
    # сохраняем товар
    self.session.modified = True

def remove(self, product):
    """
    Удаляем товар
    """
    product_id = str(product.id)
    if product_id in self.cart:
        del self.cart[product_id]
    self.save()

```

Методы add() и remove() обращаются к объекту product из базы данных товаров. С помощью метода save() мы сохраняем изменения в корзине, пометчая сессию session.modified = True.

Доступ к элементам в корзине мы получим, определив метод __iter__:

```

def __iter__(self):
    """
    Перебираем товары в корзине и получаем товары из базы данных.
    """
    product_ids = self.cart.keys()
    # получаем товары и добавляем их в корзину
    products = Product.objects.filter(id__in=product_ids)

    cart = self.cart.copy()
    for product in products:
        cart[str(product.id)]['product'] = product

    for item in cart.values():
        item['price'] = Decimal(item['price'])
        item['total_price'] = item['price'] * item['quantity']
    yield item

```

В данном методе мы извлекаем из нашего словаря продукты, подсчитываем количество товаров в корзине, а также подсчитываем значение 'total_price' – итоговой суммы.

Для расчёта общей стоимости добавляем метод get_total_price:

```
def get_total_price(self):  
    # получаем общую стоимость  
    return sum(Decimal(item['price']) * item['quantity'] for item in self.cart.values())
```

После оформления заказа нам потребуется автоматически очищать корзину. Мы будем делать это с помощью метода `clear`:

```
def clear(self):  
    # очищаем корзину в сессии  
    del self.session[settings.CART_SESSION_ID]  
    self.save()
```

Создание формы для добавления товаров в корзину

Для реализации функции добавления товаров в корзину мы будем использовать форму. Для создания форм в Django не требует больших усилий – генерация форм поддерживается фреймворком с момента старта проекта¹².

В файле forms.py приложения cart создаем класс CartAddProductForm:

```
PRODUCT_QUANTITY_CHOICES = [(i, str(i)) for i in range(1, 6)]

class CartAddProductForm(forms.Form):
    quantity =
forms.TypedChoiceField(choices=PRODUCT_QUANTITY_CHOICES,
coerce=int)
    update = forms.BooleanField(required=False, initial=False,
widget=forms.HiddenInput)
```

Параметр quantity отвечает за количество определённого товара в корзине. По умолчанию клиенту доступно количество товаров по одной позиции от 1 до 5. Мы используем тип данных TypedChoiceField с coerce=int для преобразования данных о количестве в целое число.

С помощью параметра update мы даём указание, как нам поступить при добавление товара в корзину: если товар уже существует в корзине, то сумма добавится к итоговому значению стоимости данного продукта. При создании параметра мы используем виджет HiddenInput – это значит, что элемент будет скрыт от пользователя.

За добавление продукта в корзину будет отвечать метод cart_add, который мы создадим в файле views.py приложения cart:

```
@require_POST
def cart_add(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    form = CartAddProductForm(request.POST)
    if form.is_valid():
        cd = form.cleaned_data
        cart.add(product=product,
                quantity=cd['quantity'],
                update_quantity=cd['update'])
    return redirect('cart_detail')
```

Данное представление отвечает за добавление товара в корзину и обновление его количества. Для него мы используем декоратор

¹² Python: Разработка на фреймворке Django

@require_POST. При использовании такого декоратора представление будет отвечать только на POST-запросы.

В качестве параметра представление получает id товара из таблицы Product. Далее мы иницилируем корзину. В product мы получаем товар по переданному id. В form мы передаем созданную нами форму добавления товаров.

Если форма валидна, то происходит добавление и обновление товара в корзине.

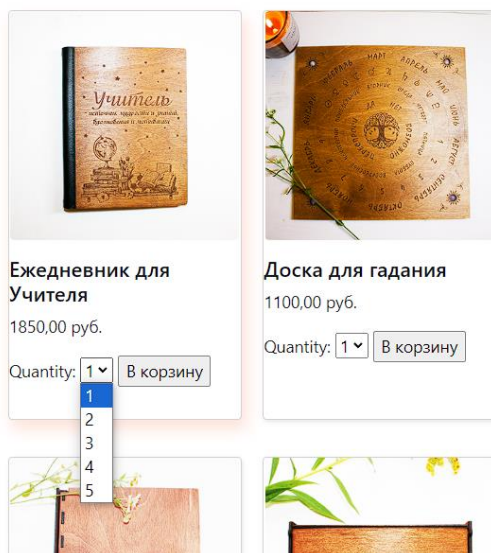
Представление перенаправляет пользователя по адресу корзины, прописанному нами в urls.py.

В HTML-шаблоне форма будет прописана в файле product_detail.html, который храниться в приложении catalog, так как выводится она именно на странице каталога:

```
<form action="{% url 'cart_add' product_id=product.id%}" method="post">
  {% csrf_token %}
  {{ form }}
  <input type="submit" value="В корзину">
</form>
```

Для защиты от подделки межсайтовых запросов используем в форме тег {% csrf_token %}¹³. У нас уже прописаны настройки MIDDLEWARE в файле settings.py и создан секретный ключ.

Посмотрим, как данная форма выглядит на фронденте:



При клике на кнопку «В корзину» происходит редирект в раздел корзина.

¹³ Защита от подделки межсайтовых запросов

Для реализации работы корзины создаем приложение `cart_detail`:

```
def cart_detail(request):
    cart = Cart(request)
    for item in cart:
        item['update_quantity_form'] = CartAddProductForm(initial={
            'quantity': item['quantity'],
            'override': True})
    return render(request, 'cart/detail.html', {'cart': cart})
```

Передаем в `cart` корзину. В строке `for item in cart` перебираем объекты в корзине. В качестве ответа представление возвращает страницу с HTML-шаблоном, в качестве контекста передаем `cart`.

Для удаления товаров из корзины нам также потребуется соответствующее представление `cart_remove`:

```
def cart_remove(request, product_id):
    cart = Cart(request)
    product = get_object_or_404(Product, id=product_id)
    cart.remove(product)
    return redirect('cart_detail')
```

Данное представление использует для удаление метод `remove`, созданный нами в файле `cart.py`. Выполняя удаление, мы остаемся в корзине.

В файле `urls.py` прописываем адреса, необходимые для работы корзины:

```
urlpatterns = [
    path("", views.cart_detail, name='cart_detail'),
    path('add/<slug:product_id>', views.cart_add, name='cart_add'),
    path('remote/<slug:product_id>', views.cart_remove, name='cart_remove'),
]
```

Рассмотрим работу HTML-шаблона `detail.html`, с помощью которого выводится содержимое корзины.

Рассмотрим первую ситуацию, если корзина пуста:

```
{% if not cart %}
<div class="container">
<h3>Корзина пуста</h3>
</div>
{% else %}
```

Если корзина пуста `cart` не существует и пользователю выводится соответствующий текст:



Корзина пуста

Если в корзине есть покупки, то в шаблоне мы перебираем каждое значение:

```
{% for item in cart %}
```

И в табличном формате выводим информацию о продуктах, итоговой сумме и количестве:

Корзина покупок

| Картинка | Товар | Обновить кол-во | Кол-во | Цена за шт | Общая стоимость | Удалить |
|---|------------------------|-----------------|--------|------------|-----------------|-------------------------|
|  | Ежедневник для Учителя | 1 ▾ Обновить | 1 | 1850,00Р | 1850,00Р | Удалить |
|  | Доска для гадания | 1 ▾ Обновить | 1 | 1100,00Р | 1100,00Р | Удалить |
| Всего | | | | | 2950,00Р | |

[В каталог](#) [Оформить заказ](#)

Непосредственно находясь в корзине, мы можем обновить количество товаров, удалить позицию. Также из корзины, кликнув на соответствующие ссылки, мы можем вернуться в каталог или оформить заказ. Про оформление заказов мы поговорим в следующем разделе.

Создание и настройка модели Заказов

Для реализации функции оформления заказов в файле `models.py` приложения `cart` создадим две модели:

- `NewOrder` – для хранения информации о пользователе и данных по заказу;
- `OrderProduct` – для хранения информации о списке и количестве товаров в заказе.

Рассмотрим каждую из моделей по очереди. Таблица `NewOrder`:

```
class NewOrder(models.Model):
    ORDER_STATUS = (
        ('Неоплачен', 'Неоплачен'),
        ('Оплачен', 'Оплачен'),
        ('Доставляется', 'Доставляется'),
        ('Доставлен', 'Доставлен'),
    )
    first_name = models.CharField(max_length=50, default=None)
    last_name = models.CharField(max_length=50, default=None)
    email = models.EmailField(default=None)
    phone = models.CharField(max_length=12, default=None)
    address = models.CharField(max_length=250, default=None)
    data_created = models.DateTimeField(auto_now_add=True)
    data_update = models.DateTimeField(auto_now_add=True)
    status = models.CharField(choices=ORDER_STATUS, max_length=100,
                              default='Неоплачен')
    total_sum = models.DecimalField(max_digits=8, decimal_places=2, default=0)
```

Первые пять полей в данной таблице несут в себе информацию о клиенте: имя и фамилия – текстовые поля, электронный адрес – поле соответствующего формата, телефон и адрес – текстовые поля с ограниченным количеством символов¹⁴.

Следующие поля дата создания и дата обновления – заполняются автоматически по текущей дате операции. Поле статус – проставляет администратором, занимающимся обработкой заказа, по умолчанию значение данного поля 'Неоплачен'. Поле `total_sum` будет заполняться через функцию в представлении.

Модель `OrderProduct` будет связывать таблицу `NewOrder` с таблицей `Product`:

```
class OrderProduct(models.Model):
    order = models.ForeignKey(NewOrder, on_delete=models.CASCADE)
```

¹⁴ Руководство Django Girls


```
order_product = models.ForeignKey(Product, on_delete=models.CASCADE)
price = models.DecimalField(max_digits=8, decimal_places=2, default=0)
quantity = models.PositiveIntegerField(default=1)
```

В поле `order` храниться внешний ключ – `id` заказа, `product` – `id` продукта, в десятичном поле `price` – данные о стоимости товара, `quantity` – количество схожих позиций в заказе. Одно строка таблицы `OrderProduct` содержит данные об одной товарной позиции, заказанной клиентом. Клиент может добавить в заказ несколько разных товаров. Информация о каждом из них будет расположена на новой строке.

К оформлению заказа клиент переходит после клика на соответствующую ссылку в корзине:

| | | | | | | |
|-------|--|--|--|--|----------|--|
| Всего | | | | | 3200,00Р | |
|-------|--|--|--|--|----------|--|

[В каталог](#) [Оформить заказ](#)

Для заполнения данных со стороны клиента нами была создана форма в файле `forms.py` приложения `cart`:

```
class NewOrder(models.Model):
    ORDER_STATUS = (
        ('Неоплачен', 'Неоплачен'),
        ('Оплачен', 'Оплачен'),
        ('Доставляется', 'Доставляется'),
        ('Доставлен', 'Доставлен'),
    )
    first_name = models.CharField(max_length=50, default=None,
    verbose_name='Имя')
    last_name = models.CharField(max_length=50, default=None,
    verbose_name='Фамилия')
    email = models.EmailField(default=None, verbose_name='Эл. почта')
    phone = models.CharField(max_length=12, default=None,
    verbose_name='Тел.')
    address = models.CharField(max_length=250, default=None,
    verbose_name='Адрес')
    data_created = models.DateTimeField(auto_now_add=True,
    verbose_name='Дата создания')
    data_update = models.DateTimeField(auto_now_add=True,
    verbose_name='Обновлен')
    status = models.CharField(choices=ORDER_STATUS, max_length=100,
    default='Неоплачен', verbose_name='Статус')
    total_sum = models.DecimalField(max_digits=8, decimal_places=2, default=0,
    verbose_name='Сумма')
```

Классы OrderCreate отличается от CartAddProductForm, описанного нами ранее, тем, что является наследником класса ModelForm. Последний позволяет нам создавать формы из полей базы данных¹⁵.

Внутри дочернего класса мы объявляем вложенный класс Meta с атрибутами model, fields, labels и widgets¹⁶. В model мы устанавливаем связь формы с моделью NewOrder. В fields определяем поля для отображения в форме, при этом мы не включаем в данный атрибут поля, информацию для заполнения которых мы получаем не от пользователя.

В атрибут labels в формате словаря полям присваиваются названия, которые будут отображаться для пользователя. В widgets по каждому полю мы заполняем тип данных, а также стили и плейсхолдер.

Логика отображение формы и сохранения информации в базу данных описана в представлении order_create в файле views.py приложения cart:

```
def order_create(request):
    cart = Cart(request)
    if request.method == 'POST':
        form = OrderCreate(request.POST)
        if form.is_valid():
            order = form.save()
            order.total_sum = cart.get_total_price()
            order.save()
            for item in cart:
                OrderProduct.objects.create(order=order,
                                             order_product=item['product'],
                                             price=item['price'],
                                             quantity=item['quantity'])

            cart.clear()
            return render(request, 'cart/created.html', {'order': order})
        else:
            form = OrderCreate()
    return render(request, 'cart/create_order.html', {'form': form})
```

Сначала мы инициализируем объект cart. Если к нам поступает POST-запрос, пользователь переходит на страницу заполнения формы:

```
path('order_create/', views.order_create, name='order_create'),
```

Адрес страницы в файле urls.py.

¹⁵ Класс ModelForms в Django

¹⁶ Формы, связанные с моделями. Пользовательские валидаторы

Заполните форму, чтобы оформить заказ

| | | | | | | | | | |
|---|------------------------------------|----------|---------------------------------------|-----------|--|-------|---|--------|--|
| Имя: | <input type="text" value="Елена"/> | Фамилия: | <input type="text" value="Дружкова"/> | Эл.почта: | <input type="text" value="3@mail.ru"/> | Тел.: | <input type="text" value="+79898794894"/> | Адрес: | <input type="text" value="г. Балашиха"/> |
| <input type="button" value="Оформить заказ"/> | | | | | | | | | |

Если форма была заполнена валидными значениями, значения сохраняются в таблицу `NewOrder`. Отметим, что благодаря использованию класса `ModelForm` при создании формы `OrderCreate`, все поступившие данные сохраняются нами в одну строку: `order = form.save()`

Значение `total_sum` генерируем с помощью метода корзины `get_total_price()`, описанного в файле `cart.py`.

Далее мы перебираем все элементы корзины и каждый из них сохраняем в таблице `OrderProduct`.

После этого мы выполняем очистку корзины методом `clear()`.

В случае успешного заполнения формы на экран пользователя выводится информация об успешном оформлении заказа:

Спасибо! Ваш заказ в обработке

Работа с административной панелью

Выполнив настройку функционала приложения, перейдем к настройке административной панели.

Для обработки заказов администратору магазина понадобится удобный интерфейс для отслеживания информации. Для этого в файле `models.py` приложения пропишем вложенный класс `Meta` для класса `NewOrder` с указанием наименований таблицы в админ-панели:

```
class Meta:
    verbose_name = 'Заказ'
    verbose_name_plural = 'Заказы'
```

А также зарегистрируем модель в файле `admin.py`. И пропишем столбцы, которые бы мы хотели видеть при просмотре списка заказов в админ-панели:

```
list_display = ['first_name', 'email', 'data_created', 'status']
```

Для удобства работы с заказами добавим также настройки фильтрации заказов по дате создания и статусу:

```
list_filter = ['data_created', 'status']
```

Проведя настройки, мы получаем следующий внешний вид отображения заказов в панели администратора.

Выберите Заказ для изменения

Действие: Выполнить Выбрано 0 объектов из 9

| <input type="checkbox"/> | ИМЯ | ЭЛ. ПОЧТА | ДАТА СОЗДАНИЯ | СТАТУС |
|--------------------------|----------|-----------|--------------------------|-----------|
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 15:17 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 15:16 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 15:16 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 15:15 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 15:14 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 14:54 | Неоплачен |
| <input type="checkbox"/> | Елена | 3@mail.ru | 4 ноября 2023 г. 14:53 | Неоплачен |
| <input type="checkbox"/> | Ольга | 2@mail.ru | 4 ноября 2023 г. 14:11 | Неоплачен |
| <input type="checkbox"/> | Светлана | 1@mail.ru | 30 октября 2023 г. 20:56 | Неоплачен |

9 Заказы

ДОБАВИТЬ ЗАКАЗ +

ФИЛЬТР

↓ Дата создания

- Любая дата
- Сегодня
- Последние 7 дней
- Этот месяц
- Этот год

↓ Статус

- Все
- Неоплачен
- Оплачен
- Доставляется
- Доставлен

При необходимости в будущем можно будет добавить ранжирование по сумме заказов.

Значение каждого столбца носит удобное название, благодаря тому, что ранее мы прописали его при создании модели `NewOrder` в атрибуте `verbose_name`.

Для отображения полной информации о заказе нам необходимо также выводить информацию о продуктах, добавленных в заказ. Для этого при настройке административной панели мы воспользуемся возможностью добавления связанных объектов в админ-панели Django.

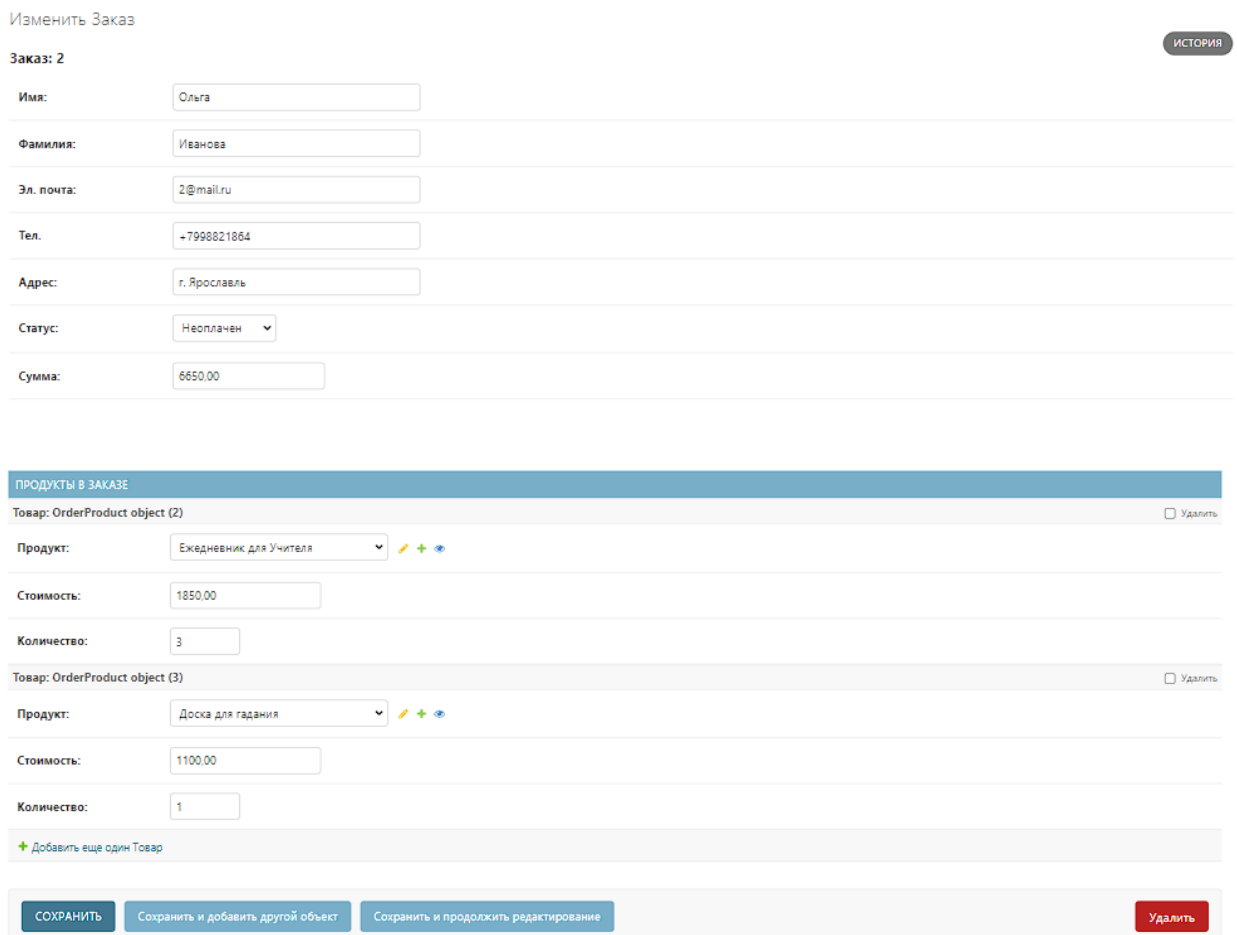
Для этого в файле `admin.py` приложения `cart` мы указываем следующий код:

```
class ItemInline(admin.StackedInline):
    model = OrderProduct

class NewOrderAdmin(admin.ModelAdmin):
    inlines = [ItemInline]
```

Класс `StackedInline`¹⁷ позволяет нам создать вложенный набор форм из других моделей, имеющих взаимосвязь с отображаемой. Таким образом в модель `NewOrder` мы включаем информацию о продуктах из модели `OrderProduct`, получая список всех продуктов, прикрепленных к заказу.

Откроем заказ в админ-панели и посмотрим, какая информация отображается внутри панели:



Изменить Заказ

Заказ: 2 ИСТОРИЯ

Имя:

Фамилия:

Эл. почта:

Тел.

Адрес:

Статус:

Сумма:

ПРОДУКТЫ В ЗАКАЗЕ

Товар: OrderProduct object (2) Удалить

Продукт: ✎ + 👁

Стоимость:

Количество:

Товар: OrderProduct object (3) Удалить

Продукт: ✎ + 👁

Стоимость:

Количество:

+ Добавить еще один Товар

СОХРАНИТЬ Сохранить и добавить другой объект Сохранить и продолжить редактирование Удалить

Как мы видим, информация о заказе успешно отображается в панели. В том числе, подгружается информация из таблицы `OrderProduct`, связанная с данным номером заказа.

¹⁷ Пример Django Admin Stacked Inline: отношения многие-к-одному и многие-ко-многим

Подведем итоги описания приложения cart. Нам удалось выполнить поставленные задачи:

- ✓ Реализована возможность добавления товаров в корзину;
- ✓ Корзина сохраняется в сессии пользователя;
- ✓ Информация о заказах сохраняется в базе данных;
- ✓ Настроено удобное отображение заказов в панели администратора.

Дальнейшее развитие проекта

Дальнейшее развития проекта мы видим в расширении функционала сайта. При росте продаж стоит развивать ресурс в сторону клиентоориентированности:

- Добавить на сайт возможность регистрации пользователя;
- В личном кабинете пользователя сохранять информацию по всем заказам и их статусам.

Для этого нам потребуется добавить модель User в базу данных. Заполнение данных и регистрация нового пользователя будет выполняться клиентами сайта посредством заполнения форм.

Для информирования пользователей о действиях, выполненных на сайте, или обновления информации о заказе можно воспользоваться встроенной в Django платформой messages framework¹⁸.

Также развитие проекта неразрывно связано с ростом ассортимента товаров, а значит в будущем нам потребуется не только выведение товаров в формате адаптивной сетки, но добавления возможности формирования страниц каталога. Для этого нам понадобится пагинация¹⁹.

Для реализация данной функциональности потребуется создать экземпляр Paginator в нашем view. Импортируя его из встроенной библиотеки django.core.paginator. Далее мы определим количество элементов на одной странице и внесем изменения в шаблон.

Звучит просто, на деле, как всегда, потребует максимального погружения. Django, как омут, но омут полезный, наполненный тоннами полезных фишек.

Подводя итоги данного раздела, отметим, что в какую бы сторону не стал развиваться наш проект, мы обязательно найдем решение для новой реализации.

¹⁸ Django template Using Message Message Framework

¹⁹ Постраничная навигация (пагинация)

Заключение

Подводя итоги проведенной работы, можно смело сказать, что фреймворк Django справедливо считается одним из самых популярных в среде разработки.

Чтобы изучить все возможности Django мало одного проекта, с ним хочется работать и дальше, изучая другие библиотеки и средства реализации различных функциональностей. Фреймворк обладает развитой экосистемой. Все более чем 18 лет существования он изменялся и совершенствовался. Какая бы задача не стояла перед нами в веб-разработке с большой вероятностью мы сможем найти ответ в уже готовом решении. Последнему немало способствует большое комьюнити разработчиков, использующих данную платформу, а также большое количество поддерживаемых библиотек.

В результате своей работы мы полностью реализовали задачи, которые ставили перед собой на старте работы над проектом:

- ✓ Изучили структуру фреймворка Django;
- ✓ Познакомились и применили на практике инструменты работы с изображениями, сессиями, формами, моделями, базами данных;
- ✓ Получили по итогу полноценный веб-ресурс, готовый к работе с пользователями.

Какие сложности возникли при разработке сайта? Тяжелее всего оказалось организовать работу корзины. Понять принципы работы с сессиями. Сейчас, когда всё настроено, всё кажется логичным и простым. Но достаточно долгое время было потрачено на то, чтобы вывести форму добавления товаров в каталог. Причина оказалась в наименовании формы в представлении: вместо стандартного `form` мы использовали другое наименование, и форма не отображалась.

В ходе проектами нами были реализованы следующие блоки веб-ресурса:

- ✓ Главная страница и страница Доставки;
- ✓ Раздел «Каталог» с возможностью сортировки товаров по категориям;
- ✓ Раздел «Наши работы» с удобным интерфейсом просмотра фотографий;
- ✓ Корзина со списком выбранных товаров, данные которой хранятся в сессии;
- ✓ Возможность оформления заказов с сохранением информации в базе данных;

- ✓ Настроили удобный интерфейс административной панели по каждому функциональному разделу сайта;
- ✓ Настроили стили на сайте;
- ✓ Сделали общий шаблон с единым меню для всех страниц сайта.

Благодаря гибкости, которую нам даёт фреймворк проект может расти и развиваться в любую сторону. Мы можем добавить на наш сайт раздел с блогом, подключить регистрацию через социальные сети и много многое другое.

Использованная литература

1. [СЕЛЛЕРЫ НА РОССИЙСКИХ МАРКЕТПЛЕЙСАХ 2022. DATA INSIGHT](#)
2. [Академия Яндекса. Что такое фреймворк](#)
3. [Рейтинг ТОП 10 CMS сайтов: Какую лучше выбрать](#)
4. [Введение в Django & DevOps](#)
5. [Django учебник Часть 3: Использование моделей](#)
6. [AndreiMalevanyi. Работа с SQLite в Python](#)
7. [Python - Загрузка изображений в Django](#)
8. [Основы шаблонизатора Jinja](#)
9. [Начинаем работу с Django — подключение админки](#)
10. [Сессия пользователя Django – установка сеанса](#)
11. [Django в примерах](#)
12. [Python: Разработка на фреймворке Django](#)
13. [Защита от подделки межсайтовых запросов](#)
14. [Руководство Django Girls](#)
15. [Класс ModelForms в Django](#)
16. [Формы, связанные с моделями. Пользовательские валидаторы](#)
17. [Пример Django Admin Stacked Inline: отношения многие-к-одному и многие-ко-многим](#)
18. [Django template Using Message Message Framework](#)
19. [Постраничная навигация \(пагинация\)](#)