Geekbrains

Разработка интернет-магазина на фреймворке Django

|  |
| --- |
| Специальность:  Python – программист  «Цифровые профессии»  Студент: Яковлев И.Д. |
| Санкт-Петербург  2024 |

Содержание

[Введение 3](#_bookmark0)

[Назначение и цели создания сайта 9](#_bookmark1)

[Структура сайта: приложения и требования к функциональности 10](#_bookmark2)

[Связи между моделями и формы 14](#_bookmark3)

[Настройка административной панели 17](#_bookmark4)

[Главная страница 19](#_bookmark5)

[Галерея 21](#_bookmark6)

[Логика и реализация работы приложения 21](#_bookmark7)

[Работа с административной панелью 22](#_bookmark8)

[Каталог 24](#_bookmark9)

[Логика и реализация работы приложения 24](#_bookmark10)

[Работа с административной панелью 31](#_bookmark11)

[Корзина 34](#_bookmark12)

[Настройка работы корзины с помощью сессий 32](#_bookmark13)

[Создание формы для добавления товаров в корзину 39](#_bookmark14)

[Создание и настройка модели Заказов 43](#_bookmark15)

[Работа с административной панелью 45](#_bookmark16)

[Дальнейшее развитие проекта 48](#_bookmark17)

[Заключение 49](#_bookmark18)

[Использованная литература 50](#_bookmark19)

**Введение**

В наше время наблюдается тенденция неизменного роста количества различных маркетплейсов, соответственно интерес к различным маркетплейсам растет, это обусловлено доступным для большинства входом в онлайн торговлю. Но тем не менее бизнес не всегда может рассчитывать на такие площадки, тем более как единственный способ реализации товаров. Проблемы с доставкой, комиссии агрегаторов, большая конкуренция, частые отказы от товаров из-за повреждений – все это является большим толчком к созданию своего способа реализовывать товар. Способ должен быть таким, чтобы он позволял описать и рассказать все о товаре в различных уголках мира. Такой способ нашелся – это создание своего сайта.

Согласно маркетинговым исследованиям, 40% продавцов признают, что электронные площадки хороши для начала, но в долгосрочной перспективе требуется корпоративный ресурс. Перед началом разработки сайта компании необходимо определить цели, которые ставятся перед корпоративным ресурсом. Это может быть сайт-визитка с информационной справкой о компании или полноценный интернет-магазин с каталогом сложной структуры и встроенными сервисами.

При выборе CMS (системы управления контентом) необходимо учитывать не только текущие потребности, но и будущие возможности роста ресурса. Фреймворк Django является универсальным инструментом для разработки веб-приложений, который позволяет изменять, дополнять и пересматривать уже существующий ресурс в любое время.

Важно понимать, что такое фреймворк. По словам Академии Яндекса, это готовый набор инструментов, ускоряющих процесс создания продукта, такого как сайт, приложение, интернет-магазин или CMS-система. Django является одним из самых популярных фреймворков на сегодняшний день.

Django предоставляет широкий набор возможностей, начиная от создания сайта-визитки и каталога, и заканчивая разработкой интернет-магазина с поддержкой регистрации и личных кабинетов. Он обеспечивает удобную настройку работы с базой данных и позволяет расширять функционал сайта по необходимости.

Django: мощный и масштабируемый фреймворк для создания веб-приложений Django - это полнофункциональный веб-фреймворк для разработки высокопроизводительных и сложных веб-приложений на Python. Его гибкость сделала его популярным выбором среди крупных компаний, включая YouTube, Google, Pinterest и Reddit. Почему компании выбирают Django?

1. Богатая коллекция встроенных инструментов и библиотек: Django предоставляет широкий спектр предварительно установленных инструментов, упрощающих начало работы. Эти инструменты охватывают различные аспекты разработки веб-приложений, включая аутентификацию, администрирование, шаблонирование и обработку форм.
2. Модульная структура: Структура Django позволяет разработчикам создавать веб-приложения модульно, как конструктор. Это упрощает обслуживание и масштабирование приложений со временем.
3. Долговечность: Django существует на рынке разработки более 18 лет. За это время фреймворк приобрел надежную репутацию и превратился в зрелую и стабильную платформу для создания долгосрочных проектов.
4. Расширяемость: Широкий спектр библиотек и плагинов, доступных для Django, позволяет разработчикам легко добавлять новые программные модули, расширяя функциональность приложений.
5. Популярные библиотеки: Django поддерживает множество популярных библиотек, которые предоставляют готовые решения для реализации сложных задач. Среди них: Django REST Framework для разработки полнофункциональных API, Pillow для эффективной работы с изображениями Django-rest-swagger для автоматической генерации документации
6. Интегрированная административная панель: Django автоматически генерирует административную панель для каждого созданного приложения. Это избавляет разработчиков от необходимости создавать ее вручную, экономя время и усилия.
7. SEO-дружественность: Django позволяет легко реализовать необходимые функции для настройки поисковой оптимизации, помогая веб-приложениям добиться более высоких позиций в поисковых системах.
8. Объектно-реляционное отображение (ORM): Django предоставляет ORM, который облегчает взаимодействие приложений с базами данных. ORM выполняет автоматическое преобразование между объектами Python и записями в базе данных.
9. Архитектура Django Django построен на архитектуре MVT (Model-View-Template), которая разделяет логику приложения (модель), представление (представление) и пользовательский интерфейс (шаблон) на отдельные компоненты. Такой подход делает код более организованным и удобочитаемым.
10. Интеграция с Python Будучи написанным на языке программирования Python, Django полностью отражает его структуру. Это делает его знакомым и удобным для разработчиков, работающих с Python. Интеграция с Python также предоставляет доступ к обширному набору библиотек и инструментов этого языка. Дополнительные преимущества использования Django Высокая безопасность: Django обеспечивает надежную основу для построения безопасных приложений с использованием встроенных функций защиты от атак. Автоматизированное тестирование: Фреймворк поддерживает автоматическое тестирование, обеспечивая удобный способ проверки правильности работы веб-приложений.
11. Активное сообщество: Django имеет активное сообщество разработчиков и пользователей, предоставляющее поддержку и ресурсы в виде документации, форумов и групп пользователей. В заключение, Django представляет собой мощный и масштабируемый фреймворк для разработки надежных и сложных веб-приложений. Его богатая коллекция встроенных инструментов, модульная структура, долговечность и расширяемость делают его идеальным выбором для проектов любого масштаба. Кроме того, интеграция с Python и активное сообщество обеспечивают поддержку и возможности для роста.

В данной архитектуре в моделях описываются 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 отличии 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 я ставлю 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

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

**Цель создания сайта**

Своей целью я поставил разработку интернет-магазина VegeFoods. Этот магазин должен упростить жизнь обычных граждан, помочь им иметь возможность купить свежие фрукты, овощи и ягоды. А также помочь продавцу поднять продажи, сделать прибыль и свое имя.  
  
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.

Известно, что 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**

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

• vegefoods/

• manage.py

• project/

• \_\_init\_\_.py

• settings.py

• urls.py

• asgi.py

• wsgi.py

С помощью файла manage.py будет 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 в примере выше это пакет с названием «project». 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.

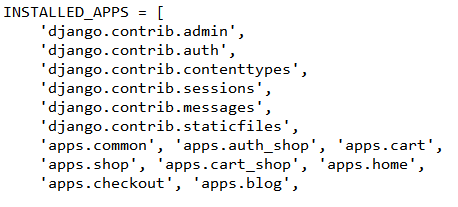
Приложение Django представляет собой отдельный фрагмент функциональности сайта. Оно может, как реализовать работу всего сайта, так и отдельной функциональности (радела или подсистемы). При разработке структуры сайта рекомендуется выделять приложения по выполняемому спектру задач

В нашем проекте выделены следующие приложения:

• Приложение «app.home» - данный пакет будет отвечать за работу главной страницы сайта и страницы с информацией о доставке. Также здесь необходимо создать директорию «templates». B которой будут хранится базовые шаблоны, содержащие общие для всего ресурса элементы: шапку сайта с меню, подвал с общей информацией.

• Приложение «app.shop» - этот пакет будет отвечать за работу модуля магазин. На сайте данный модуль будет представлен в разделе «Shop». Там

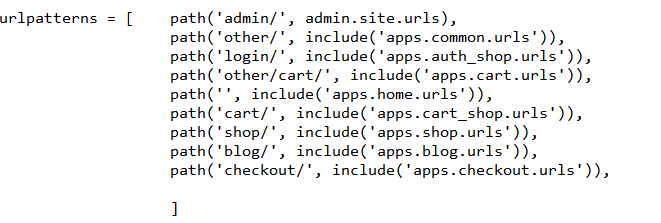
Каждое приложение после создания необходимо зарегистрировать в основном файле settings.py:



Как видно, кроме прописанных нами приложений в INSTALLED\_APPS есть и другие записи – стандартные приложения Django, в которых реализуются встроенные подсистемы. Также они будут использоваться в 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:



Каждое приложение при создании будет иметь общий набор файлов:

• \_\_init\_\_.py – опредлеяющий приложение, как пакет.

• admin.py – файл, в котором будет настраиваться при необходимости отображение моделей приложения в административной панели.

• apps.py – файл, в котором подключаемое приложение определяется как подкласс AppConfig. Это необходимо для автоматического подключения данной конфигурации в Django.

• models.py – файл, в котором будут создаваться 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. Маршрутизацию в Django можно назвать многослойной. Есть первый слой адресов, прописанный в settings.py основного приложения – здесь прописываются адреса основных модулей. Непосредственно в приложениях в файлах urls.py прописывается следующий слой адреса, который добавляется к основному адресу модуля. Это можно объяснить на примере:

1. есть адрес сайта: http://127.0.0.1:8000/

2. добавляется к нему «cart/» из settings.py основного приложения: http://127.0.0.1:8000/cart/ - переход в корзину

3. добавляется к полученному адресу «order\_create/» из файла urls.py из приложения cart: http://127.0.0.1:8000/cart/order\_create/ - и переход на страницу оформления заказа.

Кроме файла urls.py, по необходимости мы можно 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 создаются статические файлы форма CSS для описания стилей, подключаемых к HTML- шаблонам нашего проекта.

• db.sqlite3 – база данных, в которой будут хранится модели и данные.

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

• Главная страница

• Каталог

• Контакты

• Корзина

• Логин

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



Главное меню и ссылки в нём должны быть доступны пользователю с любой страницы ресурса. Это требование учитывается в базовом шаблоне.

В подвале сайта размещена следующая информация: контакты владельца магазина, виш лист, активные ссылки на главную страницу и т.д.

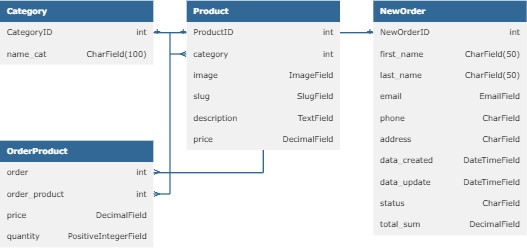
Связи между моделями

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, и могут включать поля, методы и метаданные.

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

Далее подробно будет рассмотрена каждая модель из используемых в проекте. В данном же разделе будет описана общая структура базы данных, наглядно продемонстрирована на схеме отношений таблиц:



На диаграмме выше представлена структура связей продуктов и заказов. Основой для 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.

Вторая таблица OrderProduct отвечает за связь таблицы

NewOrder и OrderProduct. Это отражено в следующих полях:

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



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.

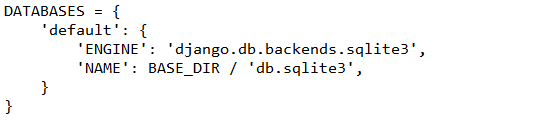
В нашем проекте будет использовано два типа форм:

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 подробнее будут рассмотрены в соответствующих разделах.

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



В настройках прописано, что для проекта будут использована база данных SQLite, что подходит, так как не предполагает большое количество запросов. Данная настройка идёт «с коробки» и меняться не будет.

SQLite представляет собой быструю и легкую базу данных, которая предоставляет возможность хранить данные прямо внутри нашего приложения. 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.

**Настройка административной панели**

Django дает разработчикам возможность использовать модели для автоматической генерации административной панели. Последняя предоставляет представляет 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.

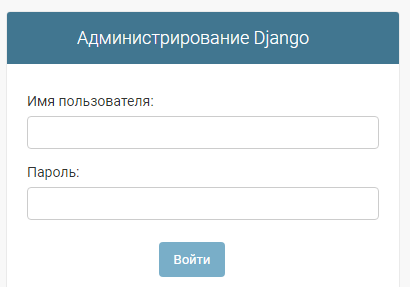
Административная панель даёт нам возможность быстро и удобно управлять данными и значительно упрощает жизнь разработчика в том числе и за счёт того, что является встроенным элементом и поставляет из «коробки».

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 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 этого по введенным данным можно зайти в административную панель.На старте админ-панель также будет пуста. Она заполняется 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, можно 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. Можно 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 «Home» 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 шаблоном, расположенным в директории templates. Именно они отвечают за визуальное отображение страницы.

Первым можно отметить шаблон, играющий роль базового для всего проекта, - layout.html. В качестве общих элементов для всего ресурса описываются: шапка сайта, футер, а также подключаемые стили, общие для всех страниц сайта. Для работы базового шаблоны используется шаблонизатор Jinja8. Для отображения переменных в наших HTML-шаблонах будут использоваться 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 файлa layout.html

Кроме базового шаблона общие для всей стилистики сайта являются:

• Фиксированная шапка сайта.

• Фиксированный футер с информацией о разработчике.

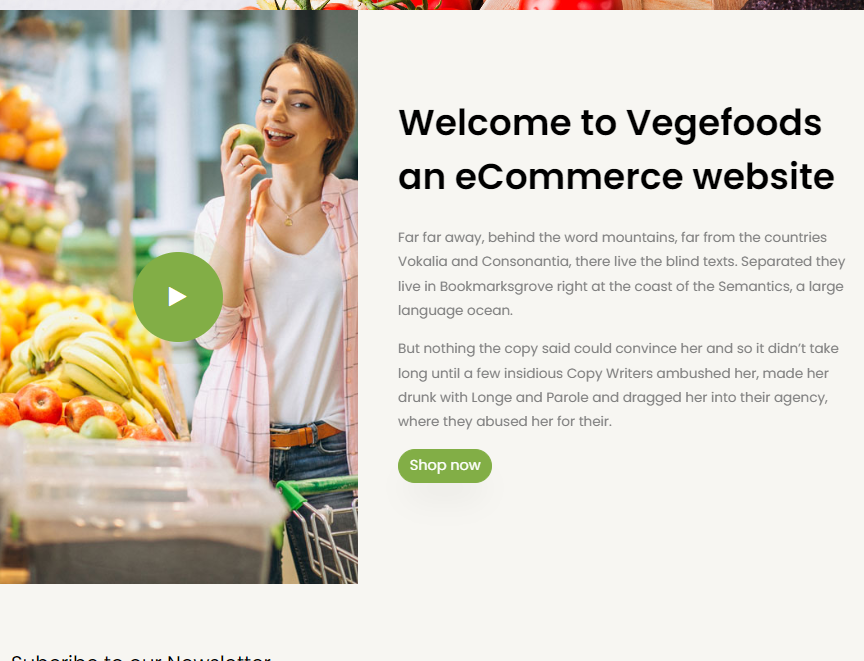
• Шрифт - 'Ubuntu Condensed', sans-serif.

• 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 перейти в каталог:



**Галерея**

**Логика и реализация работы приложения**

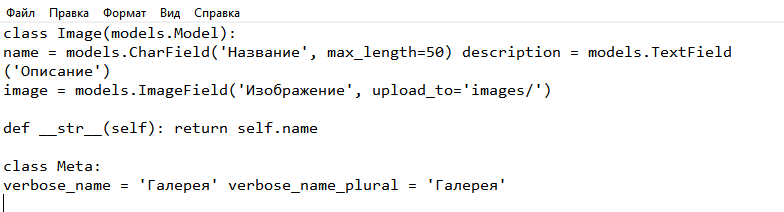
Приложение gallery отвечает за работу блока «Наши работы». Данный блок представляет собой галерею с изображениями. Цели, которые поставлены перед собой, планируя разработку данного блока:

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:



В классе Meta прописываются комфортные названия блока, которые будут отображаться в административной панели.

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

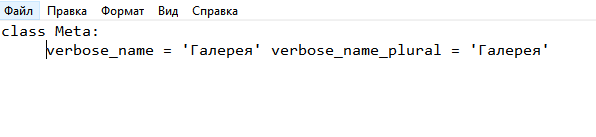
admin.site.register(Image)

**Работа с административной панелью**

Настроена возможность подгружать изображения в галерею через панель администратора для это в файле admin.py зарегистрирована модель:

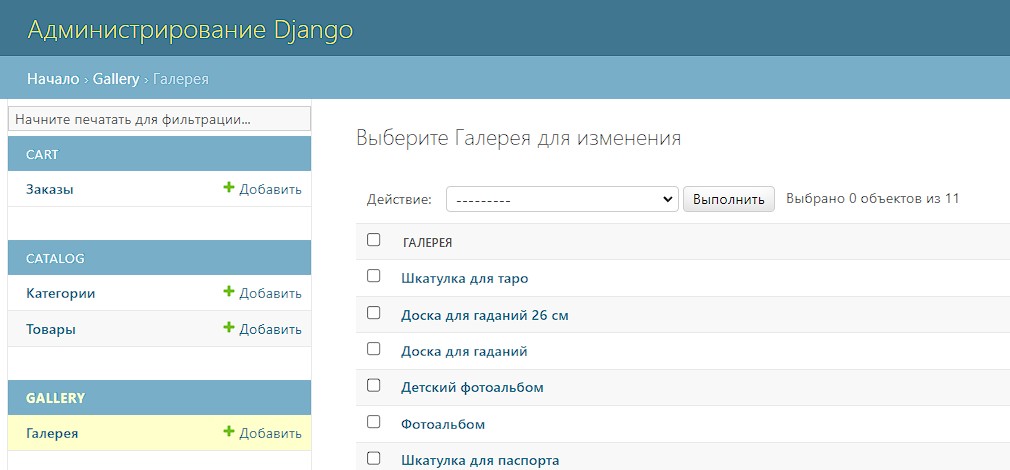
admin.site.register(Image)

А также добавлены класс Meta в описание модели Image, в котором назначены имена для модели в админ-панели:

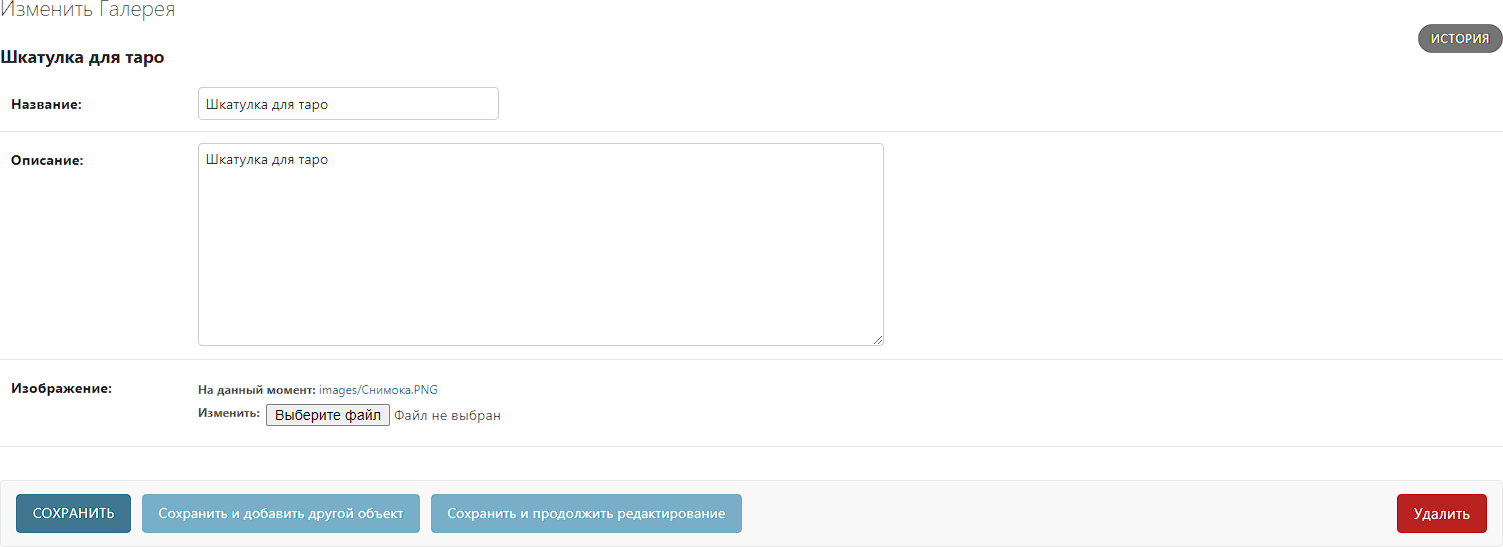


А также добавим класс Meta в описание модели Image, в котором назначим имена для модели в админ-панели:

Проведя данные изменения, получено удобное отображение класса Image в административной панели:



Для удобства даны собственные названия каждому полю на страницы добавления/изменения объекта класса Image:



**Каталог**

**Логика и реализация работы приложения**

Приложение catalog отвечает за одноименный блок «Каталог». Так как наш сайт носит презентационный характер, в настоящее время не ставится задача разработки сложной онлайн площадки. Ассортимент узкий и пока не требует сложных систем. Какие цели поставлены перед при разработке данного блока:

1. Создать каталог товаров с удобной системой ранжирования по категориям.

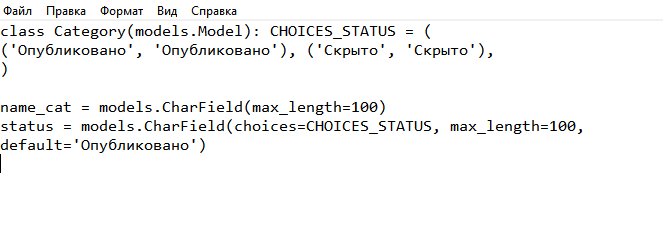
2. Карточки товаров должны отображаться в виде адаптивной сетки.

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 созданы 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. Описания класса будет выглядеть следующим образом:

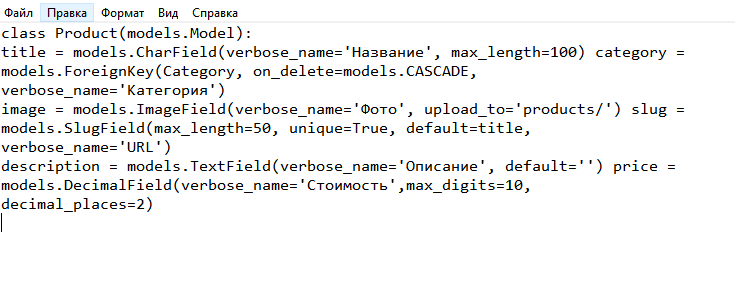


Поле name\_cat представляет собой текстовое поля для названия категории. В поле status подставляется одна из определенных констант из списка вариантов. В данном случае список представлен под именем CHOICES\_STATUS.

Статус категории может быть «опубликована» или «скрыто» — 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:



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 необходимо отображать данный продукт.

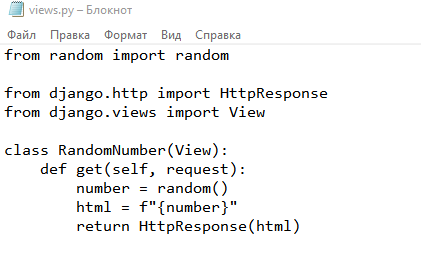
• Поле description – текстовое поле, которое содержит информацию о товаре.

• Поле price – десятичное поле, количество цифр после запятой в котором ограничивается двумя.

Для реализации задач потребуются три представления.

Рассмотрим каждое из них. Переходим к файлу views.py.

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



Получается категории со статусом «Опубликовано» из таблицы

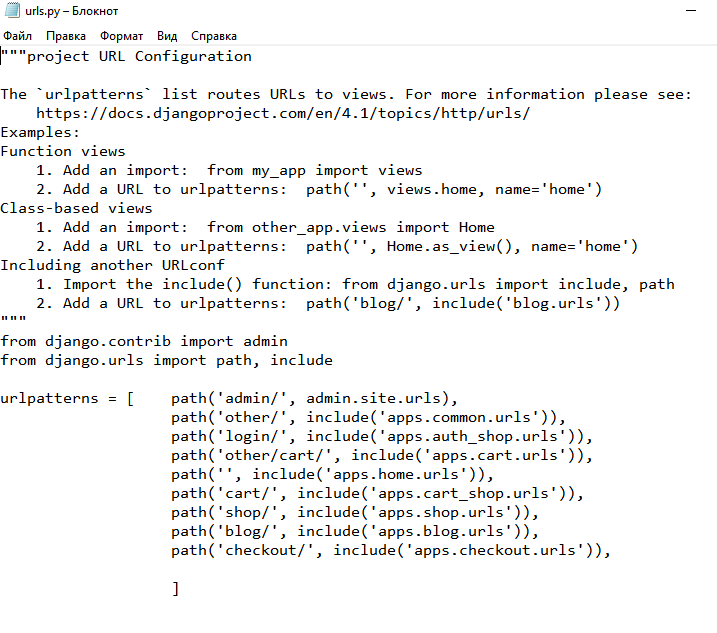
Category.

В products подгружаются все продукты из таблицы Product.

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

Фильтрация продуктов по категориям выполняется с помощью следующего приложения:

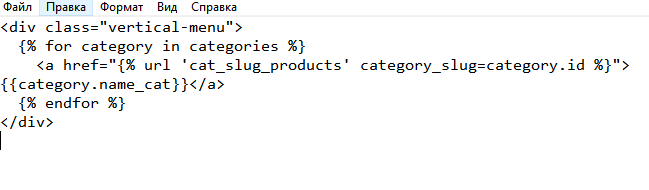
В файле urls.py приложения catalog прописываем следующие ссылки:



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.

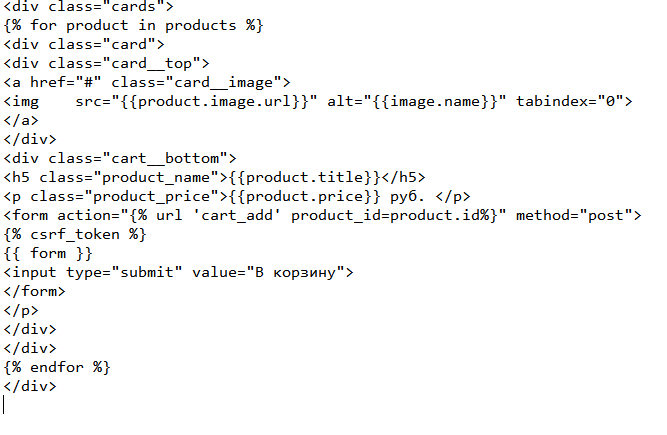
В данном HTML-шаблоне тело страницы будет поделено на две части. В левой будут выводиться категории в формате интерактивного меню.

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



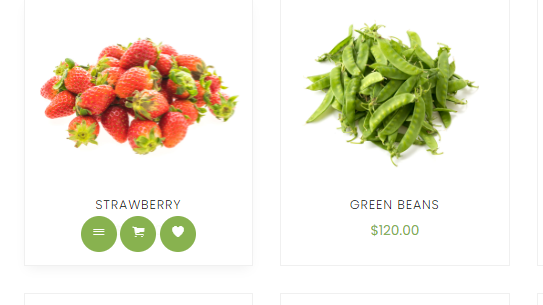
Как видно в представленном коде строки меню автоматически подгружаются из таблицы Category. Каждая строка несёт в себе ссылку на соответствующий раздел категории – данная ссылка и пробрасывается в представление cat\_slug\_products.

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

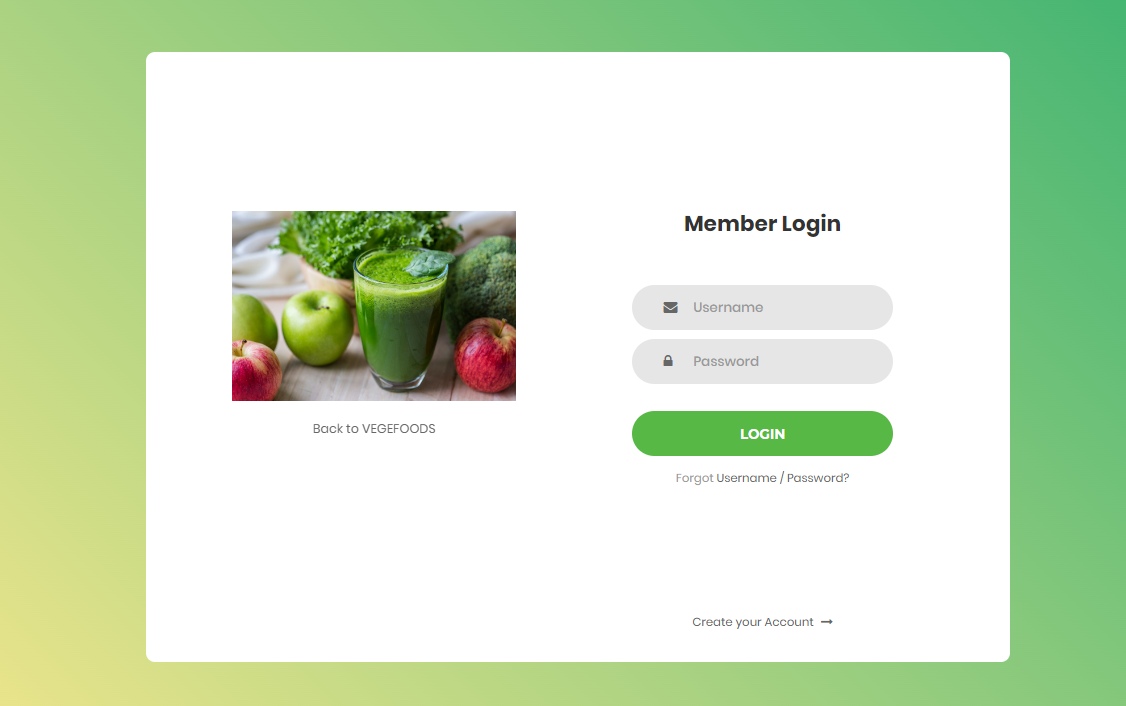


В шаблоне выводятся по каждому продукту: название, изображение, стоимость. Также здесь представлена форма добавления товара в корзину.

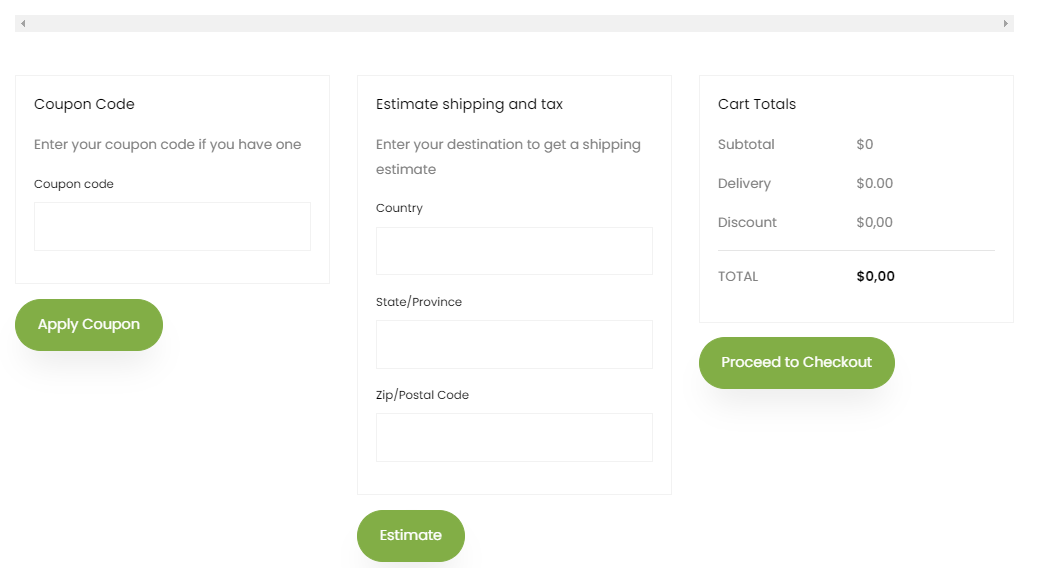
Есть возможность добавить товар в виш лист (избранное), и/или в корзину



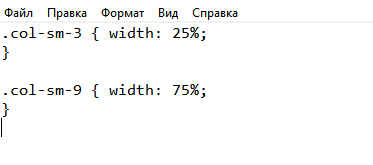
Для того, чтобы добавить в виш лист, необходимо зарегистрироваться и быть авторизованным.



После добавления товара в корзину появляется автоматически считается сумма:



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



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:

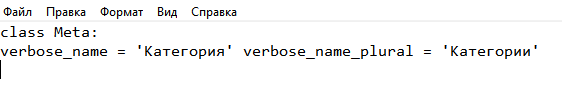
 Все изменения в базе данных по товарам и категориям автоматически отражаются на сайте, не требуя правки кода.

 Товары в каталоге представлены в формате удобной адаптивной сетки.

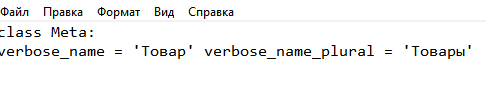
**Работа с административной панелью**

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.

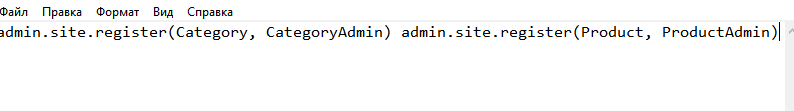
В модели Category прописываем класс Meta:



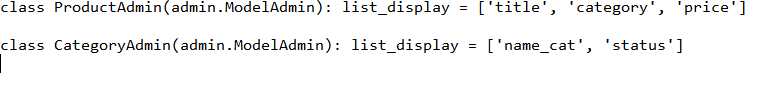
Те же действия выполняем в модели Product:



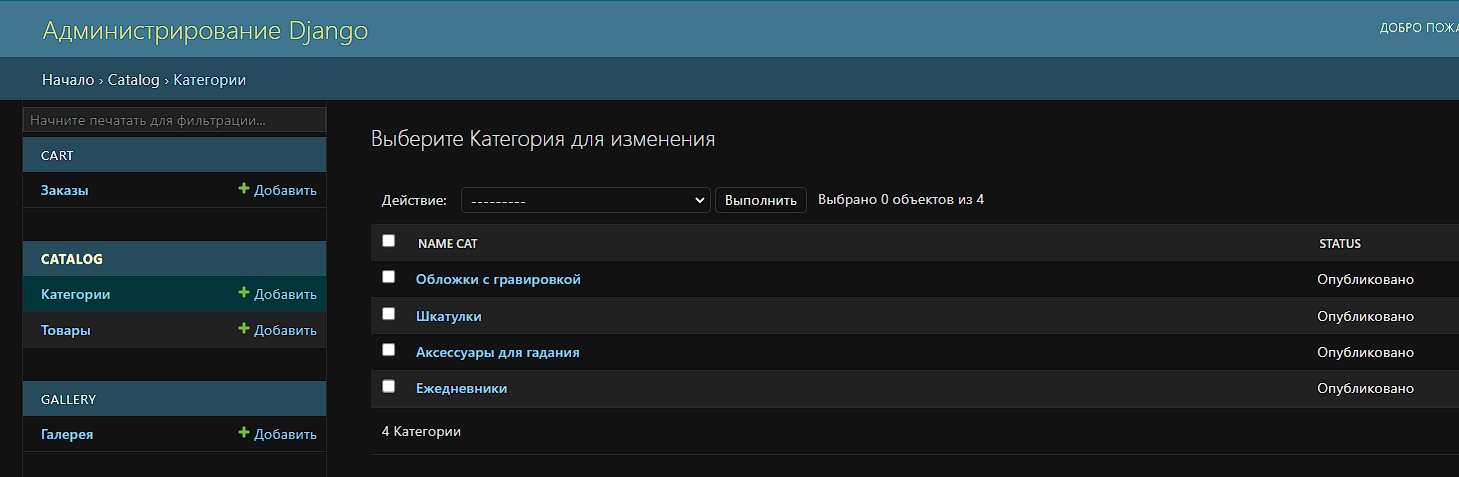
Регистрируем модели в файле admin.py:



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

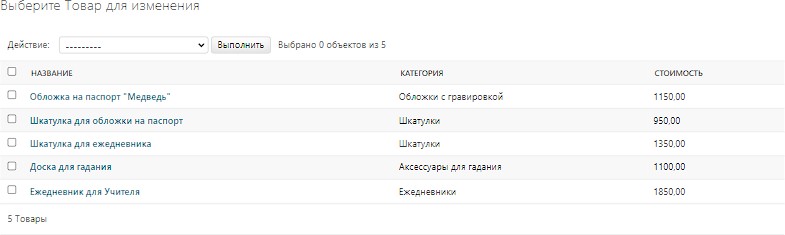


Если открыть админ-панель, то можно увидеть, что в приложении CATALOG у нас отображается две модели:

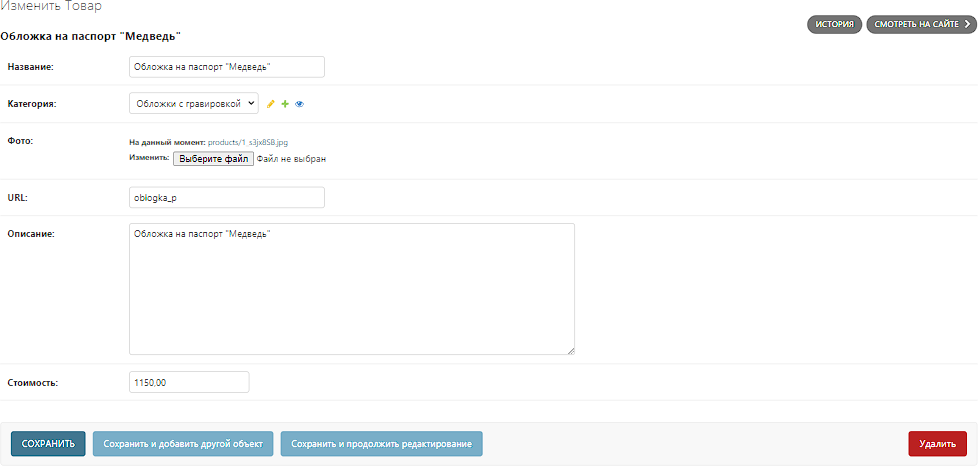


В разделе Категории видно название и статус по каждой категории. В разделе Товаров в общей выдаче представлены колонки название,

категория, цена:



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

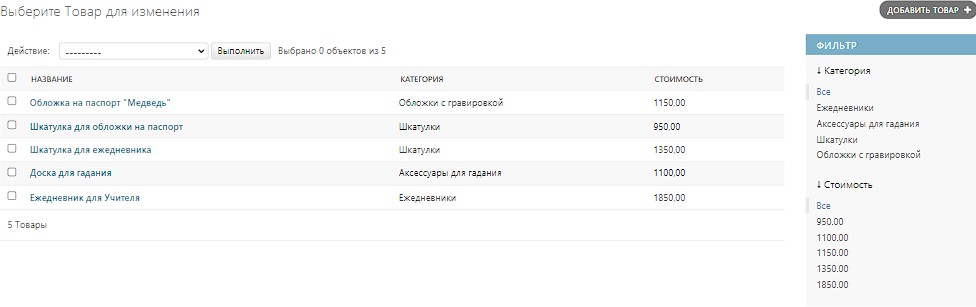


Как видно страница изменения товаров полностью русифицирована. Ранее при описании модели Product нами был прописан у каждого свойства модели атрибут verbose\_name.

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



Получаем более удобную для пользователя модель отображения списка товаров:



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 поставлены 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 будет использоваться 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 хранится 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 сериализировать в JSON.

Для указания места хранения сессии необходимо настроить параметр SESSION\_ENGINE. Для нашего проекта будет использоваться Database sessions. При выборе данного варианта хранения сессий данные записываются и хранятся в базе данных.

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

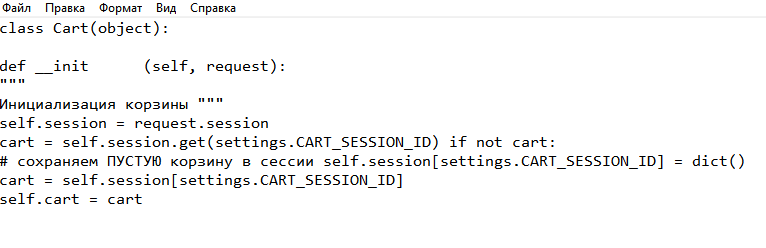
* id Продукта;
* Количество товара;
* Стоимость единицы товара.

Для хранения и извлечения информации используется словарь Python.

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

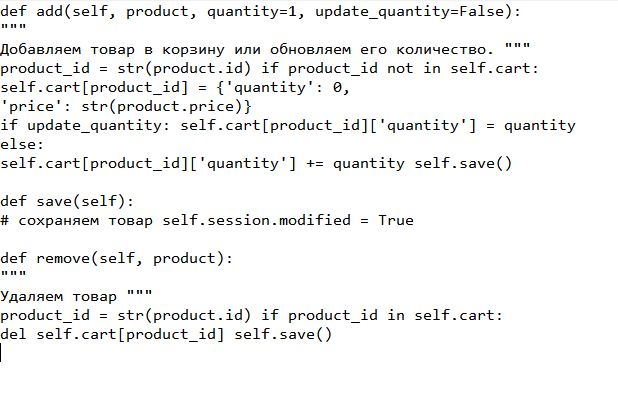
CART\_SESSION\_ID = 'cart'

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

Описание работы сессий будет выполняться в файле cart.py в приложении cart. В этом файле создаем класс Cart, с помощью которого можно управлять корзиной:

В \_\_init инициализируется корзина, получается возможность хранить текущую сессию с помощью self.session = request.session.

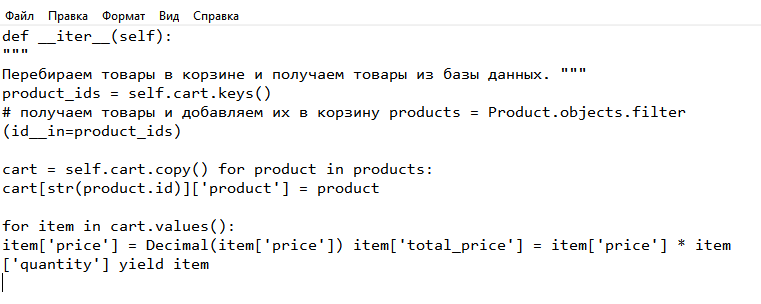
Для добавления товаров в корзину, обновления их количества и удаления добавим методы add(), save() и remove():



Методы add() и remove() обращаются к объекту product из базы данных

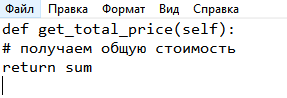
товаров. С помощью метода save() сохраняются изменения в корзине, помечая сессию session.modified = True.

Доступ к элементам в корзине получен, определив метод \_\_iter\_\_:

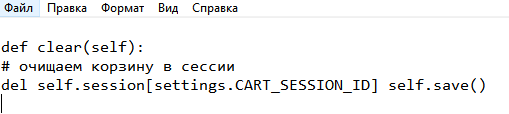


В данном методе извлекаются из словаря продукты, подсчитывается количество товаров в корзине, а также подсчитывается значение 'total\_price' – итоговой суммы.

Для расчёта общей стоимости добавляен метод get\_total\_price:



После оформления заказа потребуется автоматически очищать корзину. Это можно делать с помощью метода clear:

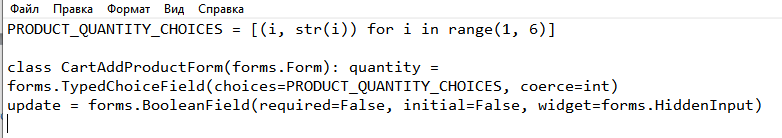


**Создание формы для добавления товаров в корзину**

Для реализации функции добавления товаров в корзину будет использоваться форма. Для создания форм в Django не требует больших усилий

– генерация форм поддерживается фреймворком с момента старта проекта12.

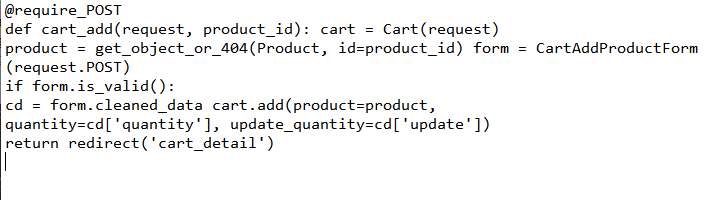
В файле forms.py приложения cart создаем класс CartAddProductForm:



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

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

За добавление продукта в корзину отвечает метод cart\_add, который создан в файле views.py приложения cart:



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

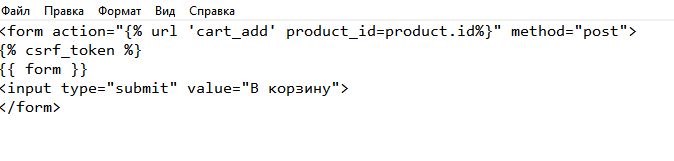
@require\_POST. При использовании такого декоратора представление будет отвечать только на POST-запросы.

В качестве параметра представление получает id товара из таблицы Product. Далее инициируется корзина. В product получается товар по переданному id. В form передается созданная форма добавления товаров.

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

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

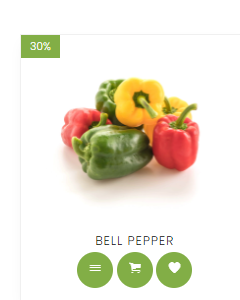
В HTML-шаблоне форма будет прописана в файле product\_detail.html, который храниться в приложении catalog, так как выводится она именно на странице каталога:



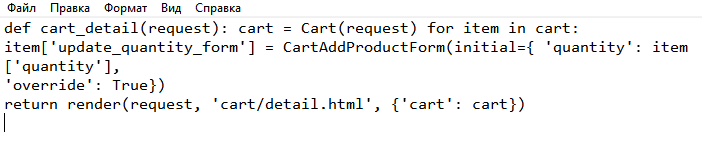
Для защиты от подделки межсайтовых запросов используем в форме тег

{% csrf\_token %}13. У нас уже прописаны настройки MIDDLEWARE в файле settings.py и создан секретный ключ.

Посмотрим, как данная форма выглядит на фронденте:

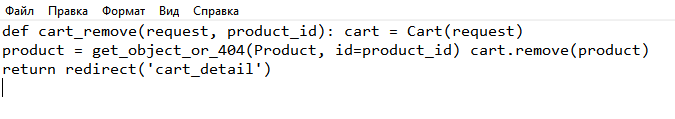


Для реализации работы корзины создаем приложение cart\_detail:



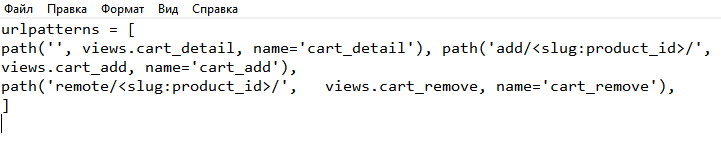
Передаем в cart корзину. В строке for item in cart перебираем объекты в корзине. В качестве ответа представление возвращает страницу с HTML- шаблоном, в качестве контекста передаем cart.

Для удаления товаров из корзины нам также потребуется соответствующее представление cart\_remove:



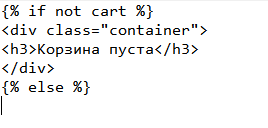
Данное представление использует для удаление метод remove, созданный нами в файле cart.py.

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

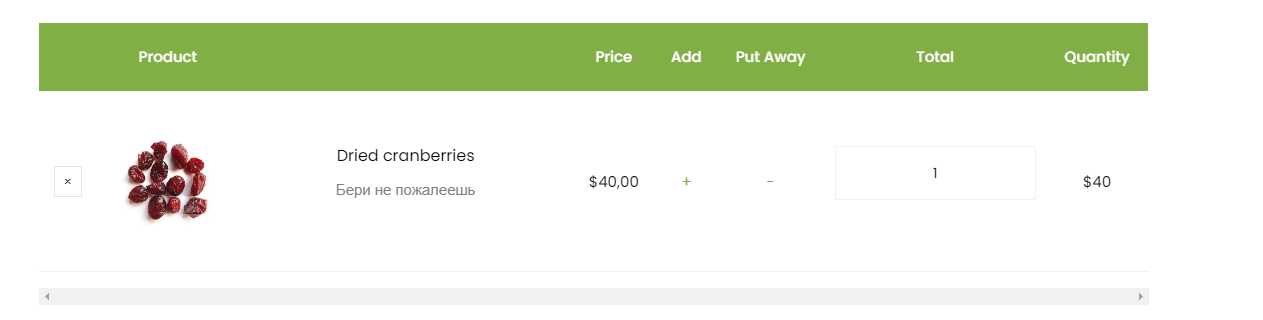


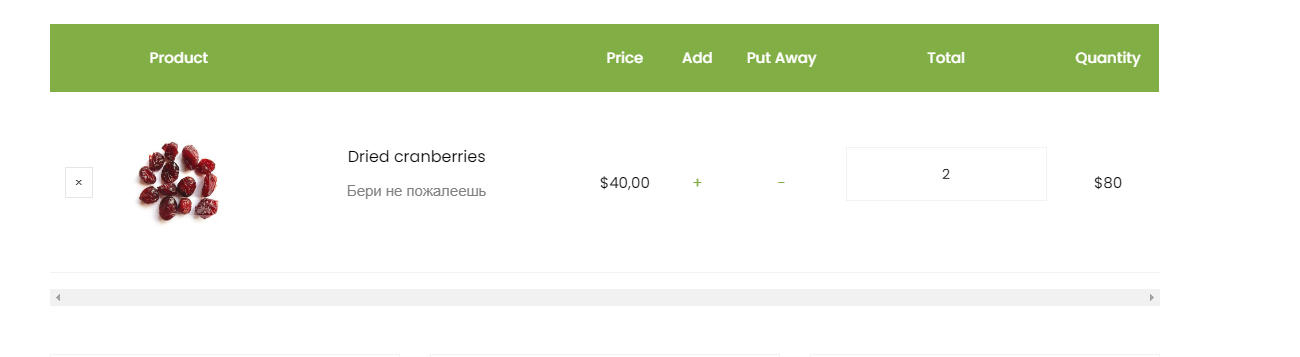
Рассмотрим работу HTML-шаблона detail.html, с помощью которого выводится содержимое корзины.

Если корзина пуста:



Если корзина пуста cart не существует и пользователю выводится соответствующее изображение:





Если в корзине есть покупки, то в шаблоне перебирается каждое значение:

{% for item in cart %}

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

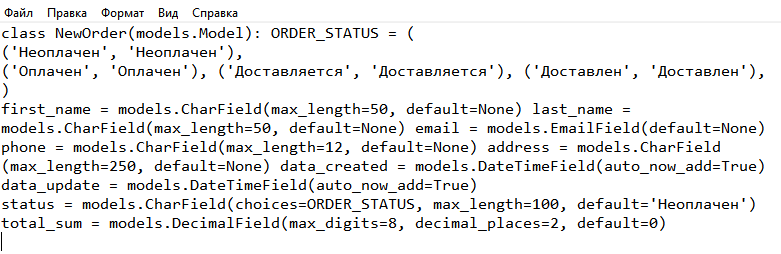
**Создание и настройка модели Заказов**

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.

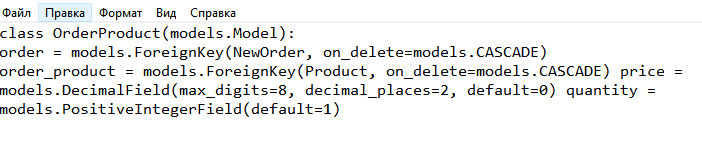
Рассмотрим каждую из моделей по очереди. Таблица NewOrder:



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.

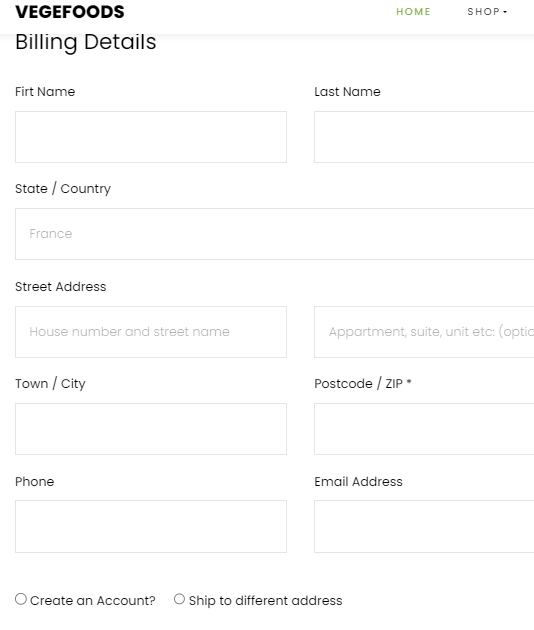
Модель OrderProduct будет связывать таблицу NewOrder с таблицей Product:



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.

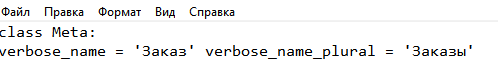
К оформлению заказа клиент переходит после клика на соответствующую ссылку в корзине:



**Работа с административной панелью**

Выполнив настройку функционала приложения, перейдем к настройке административной панели.

Для обработки заказов администратору магазина понадобиться 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:



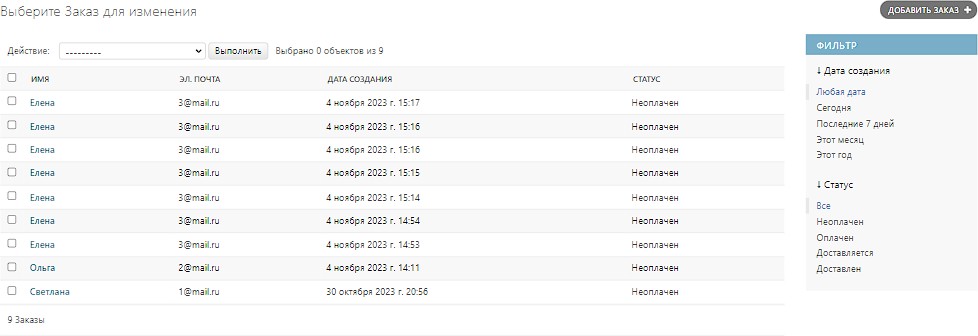
А также зарегистрируем модель в файле admin.py. И пропишем столбцы, которые необходимо видеть при просмотре списка заказов в админ-панели:

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, получаен следующий внешний вид отображения заказов в панели администратора.



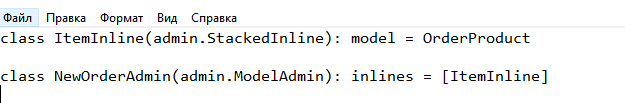
При необходимости в будущем можно будет добавить ранжирование по сумме заказов.

Значение каждого столбца носит удобное название, благодаря тому, что ранее прописано при создании модели NewOrder в атрибуте verbose\_name.

Для отображения полной информации о заказе необходимо также вывести информацию о продуктах, добавленных в заказ. Для этого при настройке административной панели можно пользоваться возможностью добавления связанных объектов в админ-панели Django.

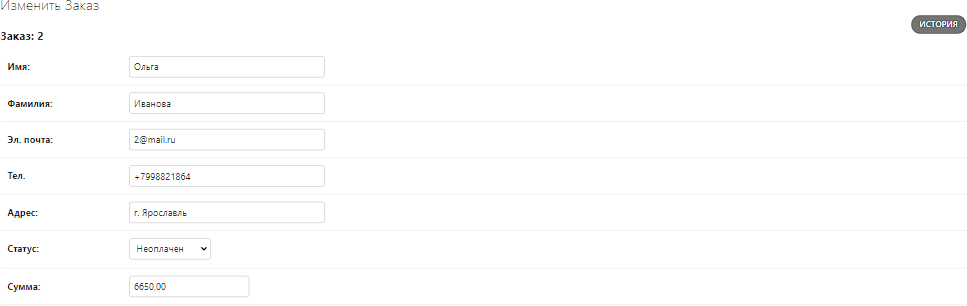
Для этого в файле admin.py приложения cart указывается следующий

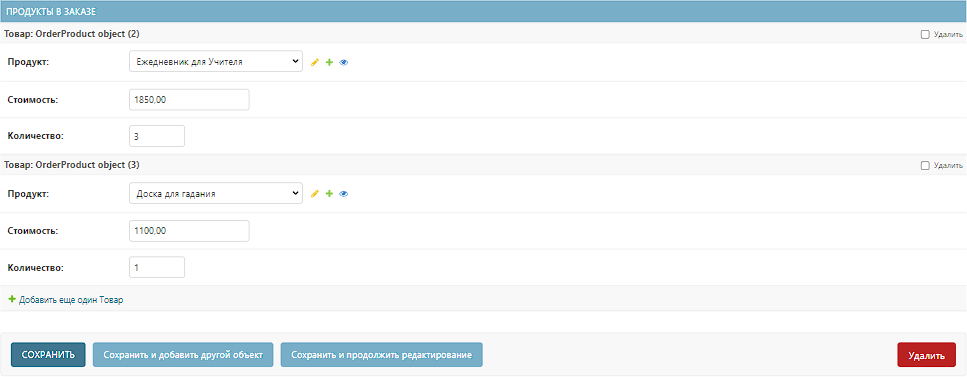
код:



Класс StackedInline17 позволяет нам создать вложенный набор форм из других моделей, имеющих взаимосвязь с отображаемой. Таким образом в модель NewOrder включается информация о продуктах из модели OrderProduct, получая список всех продуктов, прикрепленных к заказу.

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





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

Подводя итоги описания приложения cart. Поставленные задачи выполнены:

 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 виднеется 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 framework.

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 пагинация.

Для реализация данной функциональности потребуется создать экземпляр Paginator в нашем view. Импортируя его из встроенной библиотеки django.core.paginator. Далее можно определить 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, можно отметить, что в какую бы сторону не стал развиваться наш проект, мы обязательно найдем решение для новой реализации.

**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 можно найти ответ в уже готовом решении. Последнему немало способствует большое комьюнити разработчиков, использующих данную платформу, а также большое количество поддерживаемых библиотек.

В результате работы были полностью реализованы задачи, которые были поставлены на старте работы над проектом:

 Изучена структура фреймворка Django;

 Применены на практике инструменты работы с изображениями, сессиями, формами, моделями, базами данных;

 Получен 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 использовано 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 храняться в сессии;

 Возможность оформления заказов с сохранением информации в базе данных;

 Настроен удобный интерфейс административной панели по каждому функциональному разделу сайта;

 Настроены стили на сайте;

 Сделан общий шаблон с единым меню для всех страниц сайта.

Благодаря гибкости, которую даёт фреймворк проект может расти и развиваться в любую сторону.

**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)