Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Южно-Уральский государственный университет» (НИУ)

Высшая школа экономики и управления

Кафедра «Информационные технологии в экономике»

Разработка интернет-магазина на фреймворке Django

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ККУРСОВОЙ

по дисциплине «Технологии разработки Web систем»

Руководитель работы, старший преподаватель кафедры

\_\_\_\_\_\_\_\_\_\_\_ (Костерин В.В)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Автор работы

Студент группы ЭУ-342

Тушин И.Е.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Работа защищена

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

с оценкой \_\_\_\_\_\_\_\_\_\_\_

Челябинск 2020

**АННОТАЦИЯ**

Тушин И.Е. Разработка интернет-магазина

на фреймворке Django

– Челябинск: ЮУрГУ,

ЭУ-342,2020. 37с., 14 рисунков,

Библиографический

список – 1 наим.

Данная работа посвящена разработке интернет-магазина мужской одежды на фреймворке Django. Данный магазин включает в себя главную страницу с основной информацией, витриной товаров с сортировкой, страницей товаров с возможностью добавления в корзину, сама корзина с возможностью удаления оттуда товаров, страница оформления заказа, и регистрация и авторизация пользователей.

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc59113321)

[**ГЛАВА 1 QUICK START DJANGO** 5](#_Toc59113322)

[**ГЛАВА 2 НАЧАЛО РАБОТЫ** 8](#_Toc59113323)

[**2.1 Первые шаги** 8](#_Toc59113324)

[**2.2 Создание супер пользователя для Django-Admin Panel и создание в ней первой БД** 10](#_Toc59113325)

[**ГЛАВА 3 СОЗДАНИЕ СОРТИРОВКИ И СТРАНИЦЫ ТОВАРОВ** 14](#_Toc59113326)

[**3.1 Сортировка товаров** 14](#_Toc59113327)

[**3.2 Создание страницы товара** 15](#_Toc59113328)

[**ГЛАВА 4 СОЗДАНИЕ КОРЗИНЫ** 18](#_Toc59113329)

[**ГЛАВА 5 СОЗДАНИЕ ОФОРМЛЕНИЯ ЗАКАЗА** 24](#_Toc59113330)

[**ГЛАВА 6 РЕГИСТРАЦИЯ И АВТОРИЗАЦИЯ ПОЛЬЗОВАТЕЛЕЙ** 31](#_Toc59113331)

[**ВЫВОД** 36](#_Toc59113332)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК** 37](#_Toc59113333)

# **ВВЕДЕНИЕ**

Современный мир развивается огромными темпами, поэтому сейчас все больше и больше появляется огромное количество интернет-магазинов, которые занимаются продажей различных товаров и услуг. И все они написаны на различных фреймворках. Что такое вообще фреймворк? Фреймворк – это набор компонентов, которые помогают разрабатывать веб-сайты быстро и просто.

Ведь каждый раз при разработке веб-сайтов требуются похожие компоненты. Например, способ аутентифицировать пользователей, а именно: вход, выход, регистрация, панель управления сайтом, инструменты для загрузки файлов, витрина товаров и услуг, страницы товаров, отзывы к ним, процесс оформления заказа и так далее. К счастью для нас, другие люди уже обратили внимание на возникновение однотипных проблем при веб-разработке, так что они объединились и создали фреймворки.

И одним из самых популярных фреймворков на данный момент является Django. И именно ему и посвящена данная работа, так как был написан интернет-магазин по продаже мужской одежды на его основе.

# **ГЛАВА 1 QUICK START DJANGO**

Прежде чем начинать писать проект на фреймворке Django, его, первым делом, нужно установить на свой компьютер. Именно этому и посвящена данная глава.

Для выполнения установки мы переходим в командную строку, с помощью комбинации «Windows + R» и в появившемся окне пишем «cmd», после чего у нас открывается терминал. В нем мы прописываем команду "pip install Django” и у нас автоматический происходит его установка. Далее, мы создаем папку в любом удобном для нас месте и под любым названием, в моем случае это рабочий стол и папка под названием "Site\_Django". После этого, мы кликаем на нашу папку один раз и после ее выделения зажимаем клавишу "Shift", нажимаем правой кнопкой мыши и выбираем пункт «открыть окно PowerShell». После чего, у нас открывается окно, похожее на уже использованную ранее командную строку, где мы будем прописывать определенные команды, представленные на рисунке ниже.

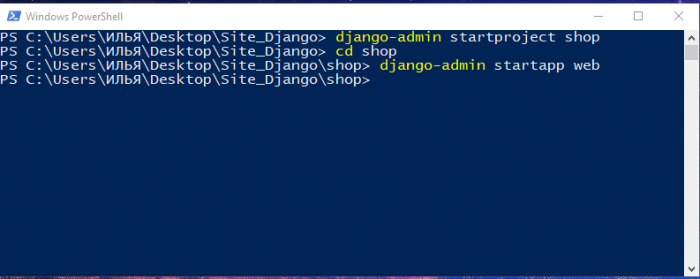


Рисунок 1 – Windows PowerShell

В данном окне мы запустили наш проект под название "shop" и создали приложение "web". Это необходимо для нашей работы, при этом, название папок могут быть абсолютно любыми. Далее, мы запускаем любой текстовые редактор, в моем случае это "Sublime Text 3" и переносим туда нашу созданную папку - "Site\_Django", после чего у нас создастся архитектура нашего проекта, изображенная на рисунке 2.

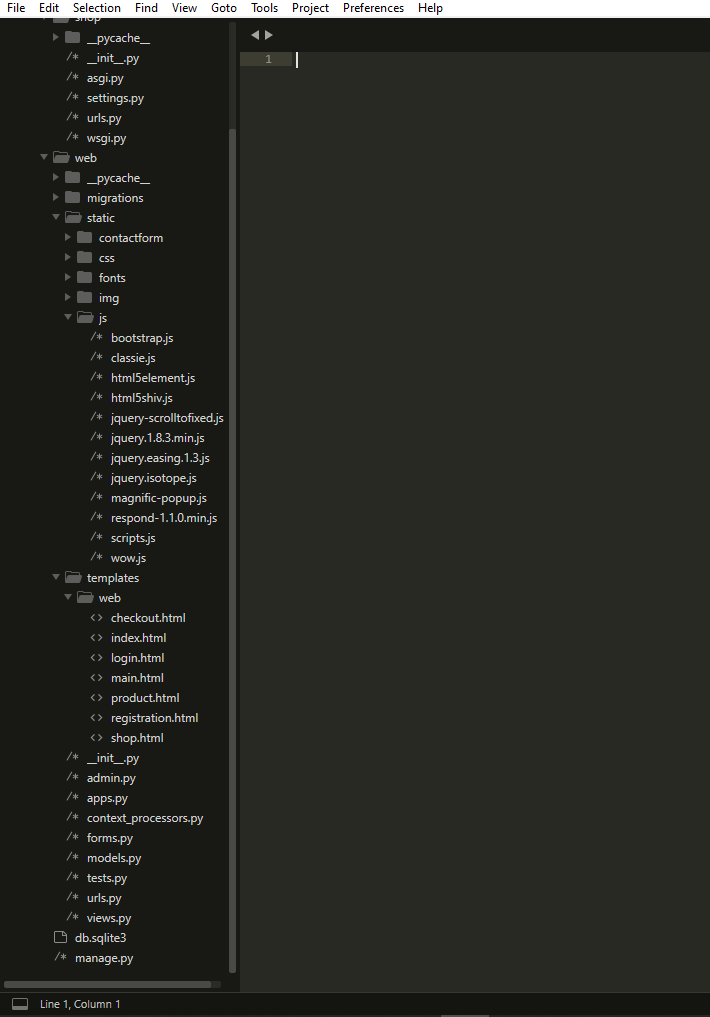


Рисунок 2 – Архитектура приложения

Далее мы подключаем наше приложение к основному проекту, для этого, мы переходим в папку "shop", файл "settings.py" и прописываем в "INSTALLED\_APPS" наше приложение. Результат представлен ниже.

INSTALLED\_APPS = [

'crispy\_forms',

'web.apps.WebConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

Проделав все это, мы уже можем запускать наше приложение. Для этого переходим во все тот же Windows PowerShell и прописываем команду "python manage.py runserver", которая отвечает за запуск нашего проекта. После чего, программа даст нам ссылку "http://127.0.0.1:8000/". Заходим в браузер и смотрим.

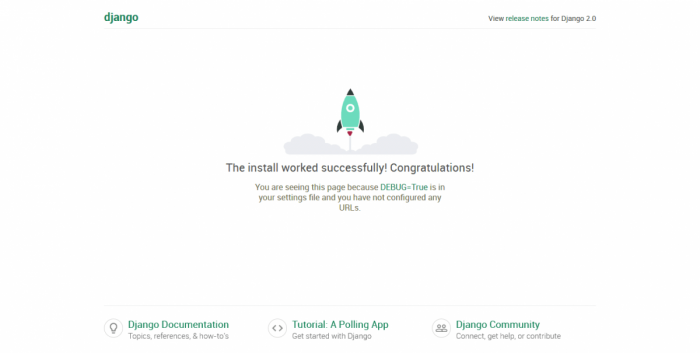


Рисунок 3 – Удачный запуск проекта

Если вы получили результат, изображенный на рисунке 3, то тогда поздравляю, вы все сделали правильно и ваш проект готов к работе!

# **ГЛАВА 2 НАЧАЛО РАБОТЫ**

## **2.1 Первые шаги**

После того, как я проделал все шаги из первой главы и мой проект был готов к работе, я решил найти шаблон интернет-магазина и подключить его к своему проекту, потому что если его писать вручную, то это займет огромное количество времени. Найдя и скачав понравившийся мне шаблон, я скопировал из него все "css" файлы и вставил в свой проект, а именно: static/css. То же самое, я сделал и с js, contactform, fonts и img. Далее, я создал файл main.html, где подключил скопированные мною файлы с помощью команды **{% load static %} в разделе "head".** Я сделал это для того, чтобы все файлы, которые я подключил, были статичны на всех страницах, то есть будут фиксированными. А также была добавлена в этот файл шапка моего сайта, с нужными мне разделами, а именно: главная и магазин. Кроме этого, был создан файл index.html, в котором я с помощью команды **{% extends 'web/main.html' %}** взял уже подключенные мною настройки, то есть этот файл будет иметь те же настройки, что и main.html, а также будет статический зафиксирована шапка сайта, так как она тоже находится в этом файле. И это, в дальнейшем, я делал абсолютно с каждой страницей. Ну и также я скопировал из моего ранее скаченного шаблона весь тег body и вставил в этот файл. Далее, я просто занимался небольшой версткой и изменением информацией с картинками, подходящими для интернет-магазина. Единственный, но очень важный момент заключался в том, что у меня не работали кнопки в шапке, поэтому я создал ссылку на переход в нужные мне разделы, а именно в разделе urls.py прописал необходимые мне модули path с определенными названиями и создал сами методы в файле views.py с одноименными названиями. Например, я в urls.py создал path с названием "shop" и в файле views.py создал метод с названием "shop", который выполнял у меня открытие этой страницы **(return render (request,'web/shop.html', context))** и уже в файле main.html я указал url ссылку на 'shop'. То есть, помимо этого, нам нужно еще создать html страницы, которые она будет открывать (в моем случае, я создал shop.html на которой у меня будет отображаться весь товар) Благодаря этому, при нажатии на раздел магазин, у меня открывался сам магазин.



Рисунок 4 – Результат подключения шаблона

С разделом "Главная" я сделал тоже самое, только при нажатии у меня открывался файл index.html (страница с основной информацией магазина). Получившийся результат представлен на рисунке 4.

## **2.2 Создание супер пользователя для Django-Admin Panel и создание в ней первой БД**

Чтобы создать супер пользователя, то есть пользователя, который будет являться, так сказать "Администратором", и иметь свой логин и пароль в Django-Admin Panel и иметь доступ ко всем созданным БД в проекте и всем записям, которые находятся в этих БД, необходимо зайти в Windows PowerShell и прописать команды python manage.py migrate и python manage.py createsuperuser. После чего, появится форма для его создания, в которой мы вводим Username и Password, то есть логин и пароль. Причем огромным плюсом является то, что таких "Администраторов" может быть несколько, а не только один. Далее запускаем сервер и в командной строке пишем **localhost:8000/admin/**, где у нас появится наш Django-Admin Panel, в который мы вводим наши логины и пароли и входим. И в открывшемся окне, будут отображаться все наши дальнейшие созданные БД.

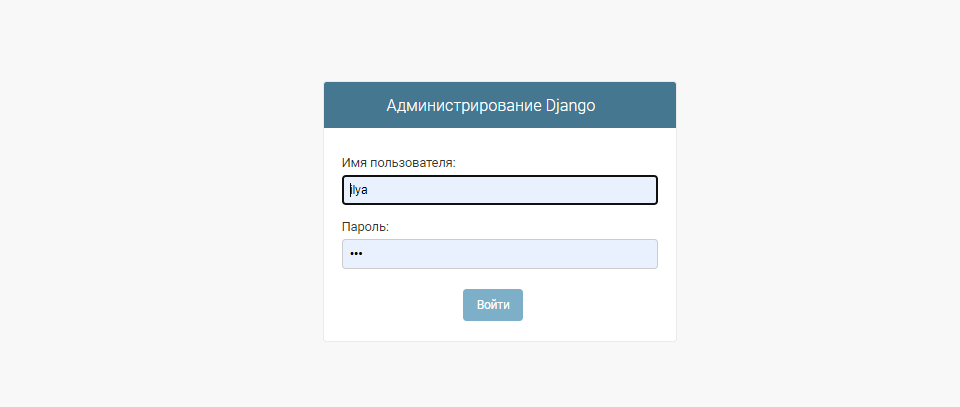


Рисунок 5 – Итог создания супер пользователя

Проделав эту манипуляцию, мы уже можем создавать наши БД. Что такое вообще БД? **База данных** — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов. Для этого, мы переходим в файл model.py, где создаем модель, включающая в себя определенные поля и настройки для каждого поля. В моем случае, создается БД для добавления и удаления товара на сайте, поэтому я создал следующую модель.

class Product(models.Model):

OUTERWEAR = 'Outerwear'

JEANS = 'Jeans'

SHOES = 'Shoes'

GLASSES = 'Glasses'

CHOISE\_GROUP = {

(OUTERWEAR,'Outerwear'),

(JEANS,'Jeans'),

(SHOES,'Shoes'),

(GLASSES,'Glasses'),

}

name = models.CharField(max\_length=100)

description = models.TextField()

price = models.IntegerField()

availability = models.BooleanField()

group = models.CharField(max\_length=20,choices=CHOISE\_GROUP,default=OUTERWEAR)

img = models.ImageField(default='no\_image.jpg', upload\_to='product\_image')

def \_str\_(self):

return f'{self.name}'

После создания модели, мы переходим в раздел admin.py и регистрируем созданную модель с помощью команды **admin.site.register(Product)**, после чего делаем миграцию созданной таблицы. Для этого, заходим в Windows PowerShell и прописываем следующие команды: **python manage.py makemigrations и python manage.py migrate.** После чего, созданная мною модель товаров появится в Django-Admin Panel, и я полноценно могу с ней работать, а именно: добавить товары на сайт. Результат представлен ниже.

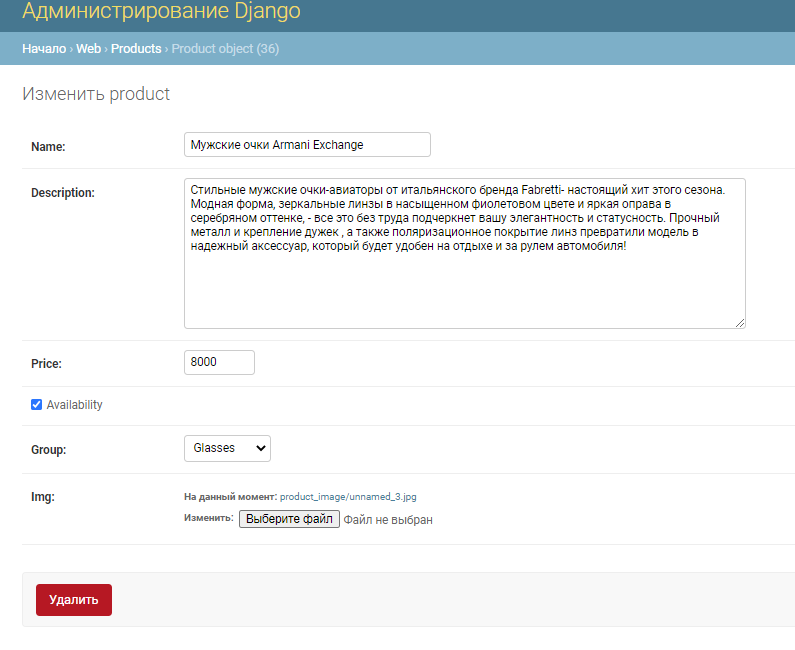


Рисунок 6 – БД для добавления и удаления товаров

Но помимо того, чтобы добавить товар, необходимо еще его и вывести, чтобы он отображался не только в нашей БД, но и в самом магазине (в моем случае, я хочу, чтобы товары отображались в разделе "Магазин"). Для этого, переходим в файл views.py и прописываем следующую команду:**"from web.models import Product"**, то есть мы импортируем нашу созданную таблицу Product из models.py. Далее, в созданном методе на открытие раздела магазин, пишем объект собирающий все строки нашей таблицы. Далее, создаем класс, в которую пишем название строки и присваиваем переменную product. После этого, в методе render, где указано название страницы, передаем сoзданный класс.

def shop (request):

product = Product.objects.all()

context = {

'pr' : product

}

return render (request,'web/shop.html', context)

После этого, настраиваем сам вывод, на странице shop.html. Например, в разделе "Верхняя одежда" и пишем:

<div class=" Portfolio-box branding">

<a href="img/Portfolio-pic3.jpg"><img src="{{ post.img.url }}" alt=""></a>

<h3>

<a href="{% url 'product' post.id %}">

{{ post.name }}

</a>

</h3>

<p>Цена: {{ post.price }}</p>

</div>

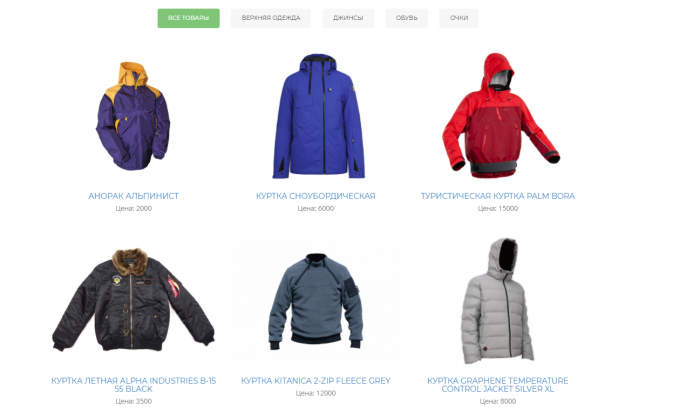


Рисунок 7 – Итоговый результат витрины товаров

В данном коде мы делаем вывод изображения, названия товара и его цены. И сам итоговый результат создания витрины товаров представлен на рисунке 7.

# **ГЛАВА 3 СОЗДАНИЕ СОРТИРОВКИ И СТРАНИЦЫ ТОВАРОВ**

## **3.1 Сортировка товаров**

Сделав витрину товаров на странице shop.html, которая открывалась сразу после нажатия на раздел магазин, я решил сделать сортировку всех этих товаров по категориям. Подключив скачанный шаблон, там были реализованы переключатели на странице, их я переименовал в нужные для меня названия: все товары, верхняя одежда, джинсы, обувь и очки. И саму сортировку я решил реализовать на этих переключателях, то есть при нажатии на верхнюю одежду у меня показывалась только верхняя одежда и так далее. Чтобы это реализовать, мы переходим в файл shop.html и прописываем метод **{% for post in pr %}** поверх всех наших категорий, который необходим для вывода всех вообще товаров. После этого, делаем сортировку по Availability с помощью команды **{% if post.availability %}.** Эта команда говорит нам о том, что если галочка стоит, то есть товар есть в наличии, то выводим этот товар, если нет, то нет. Далее, делаем проверку по указанной категории с помощью команды **{% if post.group == "Outerwear" %}.** Этот метод if говорит о том, что если товар соответствует категории верхней одежды, то он выводит этот товар.

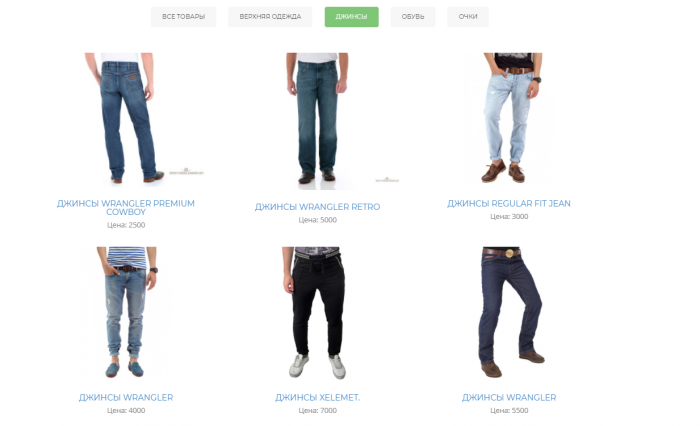


Рисунок 8 – Результат сортировки по категории «Джинсы»

Но всего в работе у нас четыре категории товара, поэтому после блока верхней одежды у нас идет блок обуви. Чтобы была сортировка по обуви, перед блоком мы уже пишем команду **{% elif post.group == "Shoes" %}.** После обуви у нас идет джинсы, поэтому прописываем **{% elif post.group == "Jeans" %}** и самый последний блок - очки. Перед этим блоком мы просто указываем команду **{% else %}**, потому что он является последним. После чего мы прописываем команды: {% endif %} (завершает проверку по блоку), {% endif %} (завершает проверку по наличию товара) и {% endfor %} (завершает цикл for). В результате выполнения этих действий, мы получаем сортировку товара. Результат сортировки представлен на рисунке 8.

## **3.2 Создание страницы товара**

Сделав вывод всех товаров на страницу, следующим этапом стало созданием страницы товаров, то есть, при нажатии на какой-нибудь товар, у нас открывается отдельная его страница, где будет его название, изображение, описание и цена. Для этого в файле urls.py я прописал следующую команду: **path('shop/', views.product, name='product').** Она говорит о том, что ее название product и что она будет открывать товар со страницы shop в соответствии с его id. То есть, у меня в магазине у каждого товара есть номер и все информация будет выводиться в соответствии с этим номером. После этого, в файле views.py я прописал сам метод, который назывался product. Его код изображен внизу.

def product (request, id):

product = Product.objects.get(pk=id)

context = {

'product': product

}

session\_key = request.session.session\_key

if not session\_key:

request.session["session\_key"]=123

request.session.cycle\_key()

print(request.session.session\_key)

return render (request,'web/product.html', context)

Ну и последним этапом является создание самой html страницы (в моем случае - product.html), код которой представлен также ниже:

{% extends 'web/main.html' %}

{% load static %}

{% block title\_page %}

<title>{{ product.name }}</title>

{% endblock title\_page %}

{% block content %}

<section class="main-section" id="service">

<div class="container">

<h2>{{ product.name }}</h2>

</div>

</section>

<div class="container">

<figure class="col-lg-7 col-sm-6 text-right wow fadeInUp delay-02s">

<a href="img/Portfolio-pic1.jpg"><img src="{{ product.img.url }}" alt="Изображение"></a>

</figure>

<div class="featured-box-col2 wow fadeInRight delay-02s">

<h3><strong>Цена товара:</strong> {{ product.price }} рублей</h3>

<h3><strong>Описание товара:</h3></strong>

<p>{{ product.description }}</p>

<div>

<form id="form\_buying\_product" class="form-inline" action="{% url 'basket\_adding' %}">{% csrf\_token %}

<div class="form-group">

<input type="number" class="form-control" name="number" id="number">

</div>

<div class="form-group">

<button type="submit" id="submit\_btn" class="btn btn-success btn-buy"

data-product\_id = "{{ product.id}}"

data-name = "{{ product.name }}"

data-price = "{{ product.price }}"

>

Купить

</button>

</div>

</form>

</div>

</div>

{% endblock content %}

На этой странице сделана обычная верстка и вывод соответствующих полей из нашей БД. Кроме того, была создана кнопка "купить" и поле, для ввода количества товара, сколько мы хотим купить.

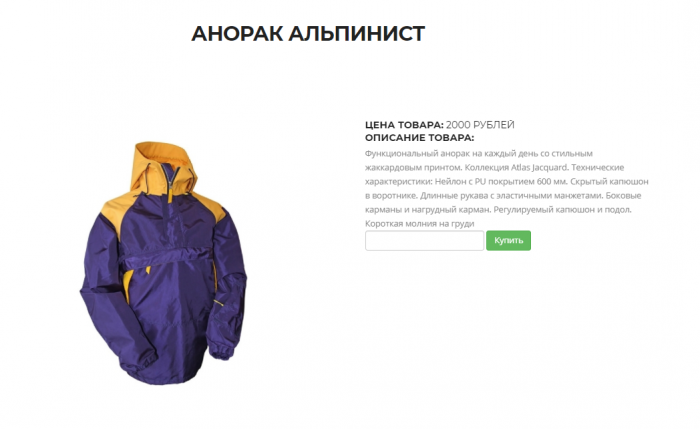


Рисунок 9 – Результат создания страницы товара

Это пригодиться нам для реализации корзины и добавления товара в нее. Сам итоговый результат представлены на рисунке 9.

# **ГЛАВА 4 СОЗДАНИЕ КОРЗИНЫ**

Я уже говорил в третей главе, что на странице товаров я создал кнопку покупки и поле для добавления количество товаров, поэтому этот момент в этом разделе опускаем. Сама корзина реализована с помощью jQuery. Что такое вообще jQuery? **jQuery** — набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Первым делом, мы подключаем jQuery к нашему проекту путем написания команды в **static** (в моем случае это файл main.html, в который мы подключали все файлы в первой части, а именно: Js,css и так далее). После этого, в папке js мы создаем файл scripts.js. В данном файле был прописан код, который позволяет добавлять и удалять товар из корзины. Помимо этого, он еще позволял показывать сколько было взято товаров, его название, цену. Его код представлен ниже:

$(document).ready(function(){

var form = $('#form\_buying\_product');

console.log(form);

function basketUpdating(product\_id, nmb, is\_delete){

var data = {};

data.product\_id = product\_id;

data.nmb = nmb;

var csrf\_token = $('#form\_buying\_product [name="csrfmiddlewaretoken"]').val();

data["csrfmiddlewaretoken"] = csrf\_token;

if (is\_delete){

data["is\_delete"] = true;

}

var url = form.attr("action");

console.log(data)

$.ajax({

url: url,

type: 'POST',

data: data,

cache: true,

success: function (data) {

console.log("OK");

console.log(data.products\_total\_nmb);

if (data.products\_total\_nmb || data.products\_total\_nmb == 0){

$('#basket\_total\_nmb').text("("+data.products\_total\_nmb+")");

console.log(data.products);

$('.basket-items ul').html("");

$.each(data.products, function(k, v){

$('.basket-items ul').append('<li>'+ v.name+', ' + v.nmb + 'шт. ' + 'по ' + v.price\_per\_item + 'руб ' +

'<a class="delete-item" href="" data-product\_id="'+v.id+'">x</a>'+

'</li>');

});

}

},

error: function(){

console.log("error")

}

})

}

form.on('submit', function(e){

e.preventDefault();

console.log('123');

var nmb = $('#number').val();

console.log(nmb);

var submit\_btn = $('#submit\_btn');

var product\_id = submit\_btn.data("product\_id");

var name = submit\_btn.data("name");

var price = submit\_btn.data("price");

console.log(product\_id );

console.log(name);

basketUpdating(product\_id, nmb, is\_delete=false)

});

function showingBasket(){

$('.basket-items').toggleClass('hidden');

};

$('.basket-container').on('click, hover',function(e){

e.preventDefault();

showingBasket();

});

$('.basket-container').mouseover(function(){

showingBasket();

});

$('.basket-container').mouseout(function(){

showingBasket();

});

$(document).on('click', '.delete-item', function(e){

e.preventDefault();

product\_id = $(this).data("product\_id")

nmb = 0;

basketUpdating(product\_id, nmb, is\_delete=true)

})

function calculatingBasketAmount(){

var total\_order\_amount = 0;

$('.total-product-in-basket-amount').each(function() {

total\_order\_amount = total\_order\_amount + parseFloat($(this).text());

});

console.log(total\_order\_amount);

$('#total\_order\_amount').text(total\_order\_amount.toFixed(2));

};

$(document).on('change', ".product-in-basket-nmb", function(){

var current\_nmb = $(this).val();

console.log(current\_nmb);

var current\_tr = $(this).closest('tr');

var current\_price = parseFloat(current\_tr.find('.product-price').text()).toFixed(2);

console.log(current\_price);

var total\_amount = parseFloat(current\_nmb\*current\_price).toFixed(2);

console.log(total\_amount);

current\_tr.find('.total-product-in-basket-amount').text(total\_amount);

calculatingBasketAmount();

});

calculatingBasketAmount();

});

А также в файле **views.py** тоже был написан код, который позволял сохранять данные на стороне сервера:

def basket\_adding(request):

return\_dict = dict()

session\_key = request.session.session\_key

data = request.POST

product\_id = data.get("product\_id")

nmb = data.get("nmb")

is\_delete = data.get("is\_delete")

if is\_delete == 'true':

ProductInBasket.objects.filter(id=product\_id).update(is\_active=False)

else:

new\_product, created = ProductInBasket.objects.get\_or\_create(session\_key=session\_key, product\_id=product\_id,

is\_active=True, defaults={"nmb": nmb})

if not created:

new\_product.nmb+= int(nmb)

new\_product.save(force\_update=True)

products\_in\_basket = ProductInBasket.objects.filter(session\_key=session\_key, is\_active=True)

products\_total\_nmb = products\_in\_basket.count()

return\_dict["products\_total\_nmb"] = products\_total\_nmb

return\_dict["products"] = list()

for item in products\_in\_basket:

product\_dict = dict()

product\_dict["id"] = item.id

product\_dict["name"] = item.product.name

product\_dict["price\_per\_item"] = item.price\_per\_item

product\_dict["nmb"] = item.nmb

return\_dict["products"].append(product\_dict)

return JsonResponse(return\_dict)

Помимо этого, была создана корзина в шапке сайта, при наведении на которую, она разворачивалась и там были показаны наши товары, а именно: название, цена за штуку, количество, а также был крестик, позволяющий удалить товар из корзины. Также, я поработал с библиотекой Ajax. **Ajax** — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. То есть, я создал в Django-Admin Panel еще одну БД (как создавать БД я рассказывал выше), в которой отображалась добавленные товары в корзине. И это как раз и было реализовано через Ajax. А для чего это нужно? Это нужно по большей части для анализа данных, какие товары добавляют больше всего, какие удаляют и тд. Итог выполнения представлен на рисунке 10.

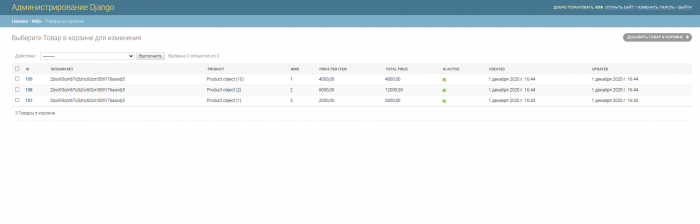


Рисунок 10 – Отображение добавленных товаров в корзину в Django-Admin Panel

Как мы видим, в БД показывается ID товара, кол-во, цену за шт, итоговую цену, дату добавления и т.д. Помимо этого, мы видим, что в поле "Is active" стоит зеленая галочка. Это значит, что товар у человека до сих пор находится в корзине, а если товар удалить, то будет красный крестик. Но в ходе создания корзины, я столкнулся с очень серьезной проблемой, а именно: при обновлении страницы или переходе на другую товар пропадал из корзины.

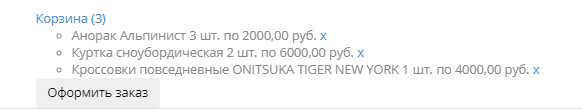


Рисунок 11 – Итог реализации корзины

Эту проблему я решил с помощью context processors. **Context processors** - это файл, который есть на всех страницах сайта. То есть, при перемещении по сайту, обновлении страницы, товары оставались в корзине и показывалось, сколько их там. Код контекстного процессора представлен ниже, а итоговый результат реализации корзины на рисунке 11.

from .models import ProductInBasket

def getting\_basket\_info(request):

session\_key = request.session.session\_key

if not session\_key:

#workaround for newer Django versions

request.session["session\_key"] = 123

#re-apply value

request.session.cycle\_key()

products\_in\_basket = ProductInBasket.objects.filter(session\_key=session\_key, is\_active=True)

products\_total\_nmb = products\_in\_basket.count()

return locals()

# **ГЛАВА 5 СОЗДАНИЕ ОФОРМЛЕНИЯ ЗАКАЗА**

После создания корзины, которая показывала, какие товары мы добавили, их цену и их количество, я приступил к реализации страницы оформления заказа, на которую мы уже переходили при нажатии на кнопку "Оформить заказ", расположенную в самой корзине. Как создавать страницу, я уже рассказывал в одной из предыдущих частей, на примере создания страницы товара и витрины. После того, как я создал страницу, там в ход пошла уже обычная верстка, а далее, я уже начал делать вывод товаров из корзины на нашу созданную страницу в виде таблицы. Для этого, в файле **views.py** я написал код, который у меня доставал все записи из модели **"ProductInBasket"**, которая хранит все данные о товарах в корзине. Сама модель расположена в **models.py**. Также в файле checkout.html я сделал отрисовку полученных данных на самой странице. Но кроме вывода товаров из корзины, в данном файле была прописана также логика работы страницы, которая позволяла изменять количество товаров, считала стоимость по каждому товару и считала уже итоговую стоимость всего заказа в целом. Помимо этого, если у покупателя нету товаров в корзине, то при переходе на страницу, ему пишет "в корзине нет товаров".

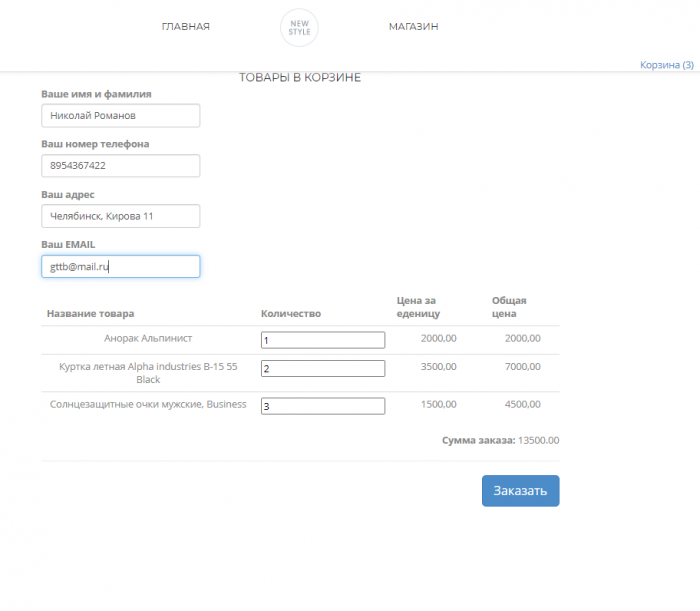


Рисунок 12 – Оформление заказа

Но это была проделана только половина работы. Основным этапом в данной части является написание back-end части, а именно, необходимо было, чтобы в Django-Admin Panel передавалась вся информация о заказчике, а также информация о самом заказе.



Рисунок 13 – Передача заказа в Django-Admin Panel

И в ходе этой работы, была создана и мигрирована таблица в базу данных (как это делается я уже рассказывал в одной из глав), куда, при нажатии на кнопку "Заказать", отправлялась вся информация о заказе и покупателе, а также дата его создания. Результат и сами коды, представлены ниже.

**Views.py**

def checkout(request):

session\_key = request.session.session\_key

products\_in\_basket = ProductInBasket.objects.filter(session\_key=session\_key, is\_active=True)

print (products\_in\_basket)

for item in products\_in\_basket:

print(item)

form = CheckoutContactForm(request.POST or None)

if request.POST:

print(request.POST)

if form.is\_valid():

print("yes")

data = request.POST

name = data.get("name", "3423453")

phone = data["phone"]

address = data["address"]

email = data["email"]

user, created = User.objects.get\_or\_create(username=phone, defaults={"first\_name": name})

order = Order.objects.create(user=user, customer\_name=name, customer\_phone=phone,customer\_address=address,customer\_email=email)

for name, value in data.items():

if name.startswith("product\_in\_basket\_"):

product\_in\_basket\_id = name.split("product\_in\_basket\_")[1]

product\_in\_basket = ProductInBasket.objects.get(id=product\_in\_basket\_id)

print(type(value))

product\_in\_basket.nmb = value

product\_in\_basket.order = order

product\_in\_basket.save(force\_update=True)

ProductInOrder.objects.create(product=product\_in\_basket.product, nmb = product\_in\_basket.nmb,

price\_per\_item=product\_in\_basket.price\_per\_item,

total\_price = product\_in\_basket.total\_price,

order=order)

return HttpResponseRedirect(request.META['HTTP\_REFERER'])

else:

print("no")

return render(request, 'web/checkout.html', locals())

**Forms.py**

class CheckoutContactForm(forms.Form):

name = forms.CharField(required=True)

phone = forms.CharField(required=True)

address = forms.CharField(required=True)

email = forms.CharField(required=True)

**Checkout.html**

{% extends 'web/main.html' %}

{% load static %}

{% block title\_page %}

<title>NewStyle</title>

{% endblock title\_page %}

{% block content %}

<div class="container">

<div class="row">

{% if products\_in\_basket %}

<div class="col-lg-8 col-lg-offset-2">

<h3 class="text-center">Товары в корзине</h3>

{{ request.session.basket.products }}

<form class="form-horizontal" method="post">{% csrf\_token %}

<div class="form-group">

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

<label>Ваше имя и фамилия</label>

<span class="form-error">{{ form.name.errors }}</span>

<input type="text" name="{{ form.name.html\_name }}" class="form-control"

value="{{ request.POST.name }}"

>

</div>

</div>

<div class="form-group">

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

<label>Ваш номер телефона</label>

<span class="form-error">{{ form.phone.errors }}</span>

<input type="text" name="{{ form.phone.html\_name }}" class="form-control"

value="{{ request.POST.phone }}"

>

</div>

</div>

<div class="form-group">

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

<label>Ваш адрес</label>

<span class="form-error">{{ form.address.errors }}</span>

<input type="text" name="{{ form.address.html\_name }}" class="form-control"

value="{{ request.POST.address }}"

>

</div>

</div>

<div class="form-group">

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

<label>Ваш EMAIL</label>

<span class="form-error">{{ form.email.errors }}</span>

<input type="text" name="{{ form.email.html\_name }}" class="form-control"

value="{{ request.POST.email }}"

>

</div>

</div>

<table class="table text-center">

<thead>

<tr>

<th>Название товара</th>

<th>Количество</th>

<th>Цена за еденицу</th>

<th>Общая цена</th>

</tr>

</thead>

<tbody>

{% for product\_in\_basket in products\_in\_basket %}

<tr>

<td>

{{ product\_in\_basket.product.name }}

</td>

<td>

<input value="{{ product\_in\_basket.nmb }}" type="number"

class="product-in-basket-nmb" name="product\_in\_basket\_{{ product\_in\_basket.id }}">

</td>

<td>

<span class="product-price">

{{ product\_in\_basket.price\_per\_item}}

</span>

</td>

<td>

<span class="total-product-in-basket-amount">

{{ product\_in\_basket.total\_price }}

</span>

</td>

</tr>

{% endfor %}

</tbody>

</table>

<div class="text-right">

<b>Сумма заказа: </b> <span id="total\_order\_amount">54</span>

</div>

<hr>

<div class="text-right">

<button type="submit" class="btn btn-primary btn-lg">Заказать</button>

</div>

</form>

</div>

{% else %}

<h3 class="text-center">В Вашей корзине нет товаров</h3>

{% endif %}

</div>

</div>

{% endblock %}

# **ГЛАВА 6 РЕГИСТРАЦИЯ И АВТОРИЗАЦИЯ ПОЛЬЗОВАТЕЛЕЙ**

Следующим делом, я приступил к созданию процесса регистрации и авторизации пользователей. Для этого, я создал файл registration.html, который отвечал у меня за саму регистрацию. Кроме этого, через терминал было установлено django-crispy-forms. Что это такое? Django-crispy-forms - это приложение, позволяющее работать с формами Django, а также отображать их без написания лишнего кода и с минимальными установками, что очень упрощает работу. После того, как данное приложение было установлено, его необходимо подключить к форме и в установленные приложения. Подключение к странице я реализовал с помощью команды **{% load crispy\_forms\_tags %}** в файле registration.html. Ну а далее, я создал саму форму с методом POST. Сам код представлен ниже:

{% extends 'web/main.html' %}

{% load crispy\_forms\_tags %}

{% block title\_page %}

<title> Регистрация на сайте </title>

{% endblock title\_page %}

{% block content %}

<div class="container" style="margin-top: 5%;">

<div class="form-section">

<h2>Регистрация</h2>

<form method="POST">

{% csrf\_token %}

{{ form|crispy }}

<button type="submit" class="btn btn-online-warning">Зарегестрироваться</button>

</form>

<hr>

Уже зарегестрированы? <a href="{% url 'login' %}">Войти</a>

</div>

</div>

{% endblock content %}

Далее, нам необходимо добавить метод views и urls ссылку на страницу. Как это делается я показывал в предыдущих частях. Единственным моментом является то, что вначале нам необходимо подключить определенные библиотеки в urls.py для взаимодействия с методами регистрации, которые уже встроены в Django: **from web import views as userViews и from django.contrib.auth import views as authViews**, после чего добавляем сам метод регистрации во views.py. Но вначале, нам также необходимо подключить определенные библиотеки: **messages (для отправки сообщений, уведомлений) и login\_required (позволяет проверять находится ли пользователь в профиле или нет, чтобы не предлагать регистрацию или вход). Код представлен ниже:**

from django.contrib import messages

from django.contrib.auth.decorators import login\_required

from .forms import UserOurRegistration

def register(request):

if request.method == "POST":

form = UserOurRegistration(request.POST);

if form.is\_valid():

form.save()

username = form.cleaned\_data.get('username');

return redirect('home')

else:

form = UserOurRegistration()

return render(request, 'web/registration.html', {'form': form})

Помимо этого, в файле forms.py была написана сама форма регистрации и авторизации. Для этого мы импортируем библиотеку forms, таблицу зарегистрированных пользователей (User) и библиотеку для создания собственной формы UserCreationForm. После чего создаем форму.

from django import forms

from .models import \*

from django import forms

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

class UserOurRegistration(UserCreationForm):

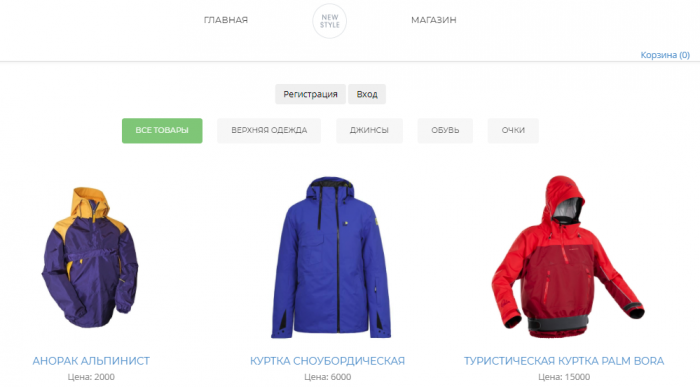
email = forms.EmailField(required=True)

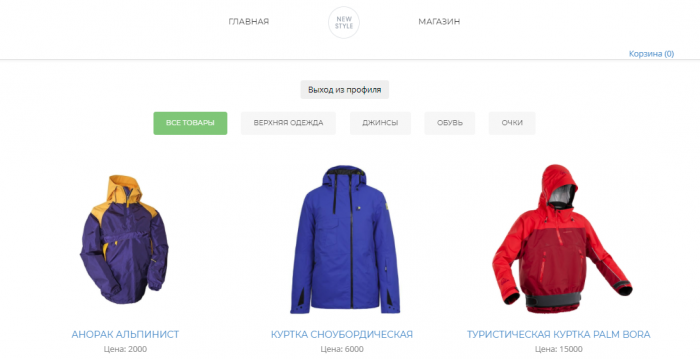
class Meta:

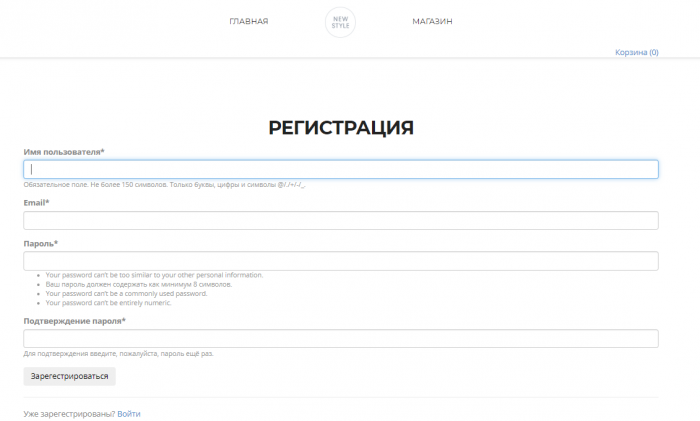
model = User

fields = ['username','email', 'password1', 'password2']

Также была добавлена html страница и urls на страницу авторизации, если человек уже зарегистрирован, ну и кнопка выхода из профиля, которая перекидывала нас на начальную страницу.







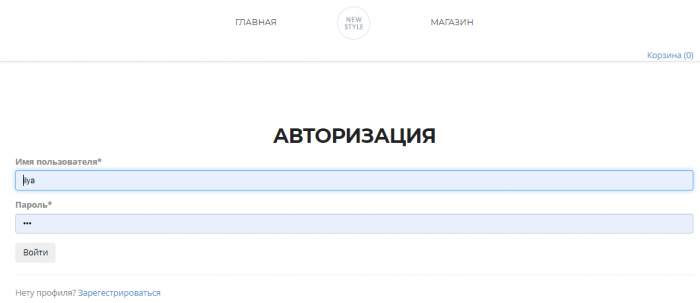


Рисунок 14 – Результат реализации регистрации и авторизации

**Urls.py**

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

from web import views as userViews

from django.contrib.auth import views as authViews

urlpatterns = [

path('admin/', admin.site.urls),

path('',include('web.urls') ),

path('registration/', userViews.register,name="reg"),

path('login/', authViews.LoginView.as\_view(template\_name='web/login.html'), name="login"),

path('exit/', authViews.LogoutView.as\_view(template\_name='web/index.html'), name="exit"),

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Login.html**

{% extends 'web/main.html' %}

{% load crispy\_forms\_tags %}

{% block title\_page %}

<title> Авторизация на сайте </title>

{% endblock title\_page %}

{% block content %}

<div class="container" style="margin-top: 5%;">

<div class="form-section">

<h2>Авторизация</h2>

<form method="POST">

{% csrf\_token %}

{{ form|crispy }}

<button type="submit" class="btn btn-online-warning">Войти</button>

</form>

<hr>

Нету профиля? <a href="{% url 'reg' %}">Зарегестрироваться</a>

</div>

</div>

{% endblock %}

# **ВЫВОД**

Исходя из проделанной мною работой, я могу смело сказать, что фреймворкDjango не зря является одним из самых популярных в мире. Потому что у него очень развитая экосистема, он постоянно изменяется и усовершенствуется, в случае возникновения трудностей найти ответ на конкретный вопрос не составляет труда, ведь тысячи специалистов уже решали такие же проблемы до нас и делились своим опытом в интернете. Помимо этого, Django поддерживает использование различных библиотек при разработке веб-приложений.

# **БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Документация Django