**Реализация программного продукта**

Процесс создания включает в себя несколько стадий:

– создание функций приложения,

– создание функций интеграции.

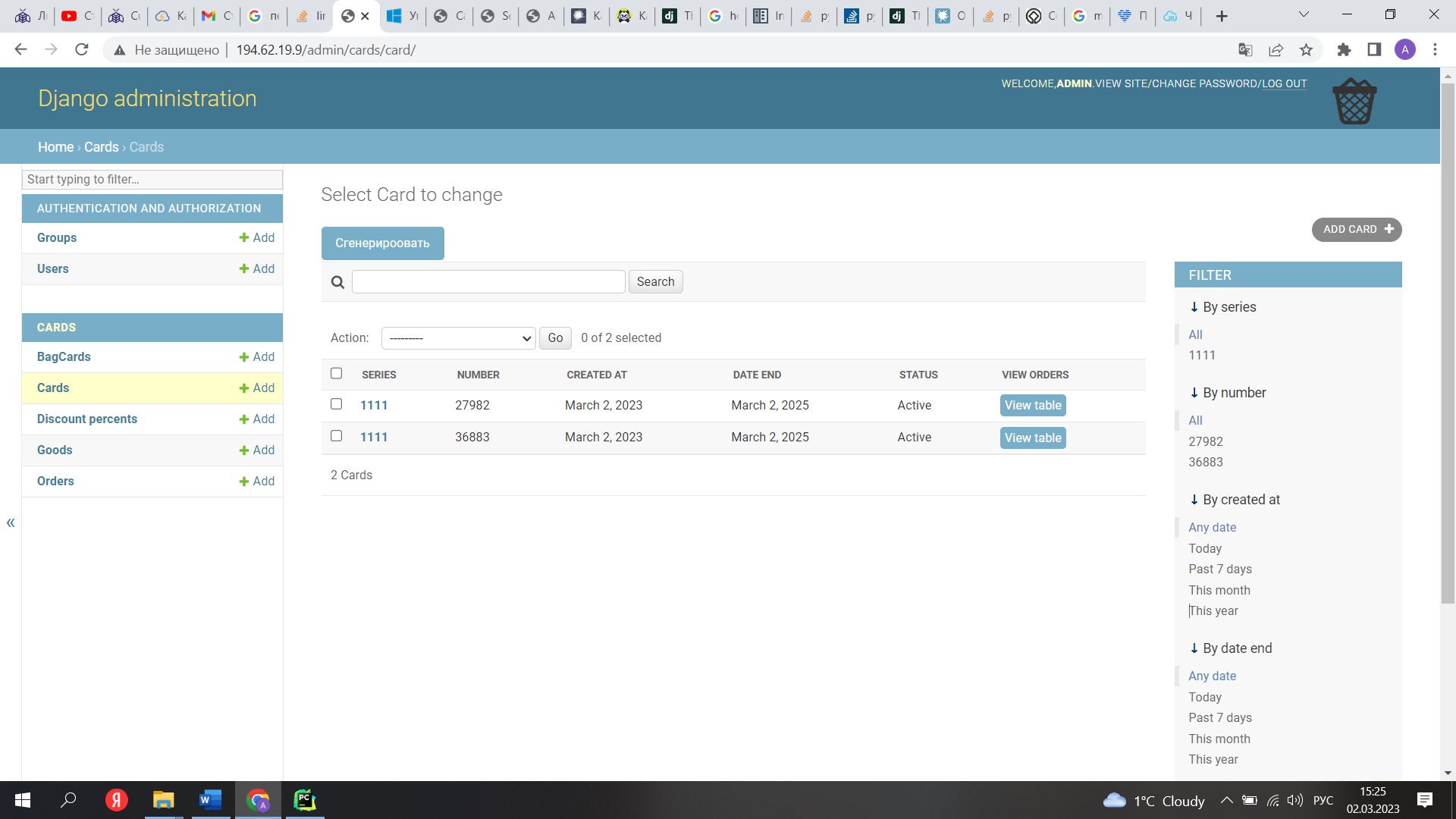
**1.Функции приложения**

* Список карт с полями: серия, номер, дата выпуска, дата окончания активности, статус(рисунок 1.1,листинги 1.1-1.2),
  + поиск по этим же полям(рисунок 1.1,листинги 1.1-1.2),
* генератор карт, с указанием серии и количества генерируемых карт, а срока активности «с-по»(рисунки 1.2,листинги 1.3-1.7),
* удаление карты -сперва в корзину с возможностью восстановления(рисунки 1.3 – 1.7,листинги 1.8-1.12),
* просмотр профиля карты с историей покупок по ней (рисунки 1.8-1.9,листинги 1.13),
* активация/деактивация карты(рисунки 1.10,листинги 1.14),
* после истечения срока активности карты, у карты проставляется статус "просрочена"(листинги 1.15).

**Дополнительный функционал:**

* добавлена кастомная кнопка для открытия ордеров записи на странице списка карт(рисунки 1.11,листинги 1.16),
* добавлена история изменений карты, для просмотра изменений даты действия карты и изменений статусов в течении времени использования(рисунки 1.12,листинги 1.17).

Так выглядит первая функция «Список карт» (рисунок 3.1).



**Рисунок 1.1 – Список карт**

На форме списка карт пользователь может выбрать дальнейшее действие:

– перейти на страницу «Профиль карты»,

– отсортировать карты по всем отображаемым полям,

– перейти на страницу «Генератор карт»,

– перейти на страницу «Корзина»,

– активировать и деактивировать список карт.

– активировать и деактивировать карту на страницу профиля.

– перейти на страницу «Корзина».

– перейти на страницу «Корзина».

Текст программного кода, отвечающего за работу некоторых кнопок, расположенных на главной форме, представлен в листингах 3.1. – 3.4.

**Листинг 1.1 – Список карт с полями, сортировкой и историей изменений(файл admins.py)**

from django.contrib import admin

from simple\_history.admin import SimpleHistoryAdmin

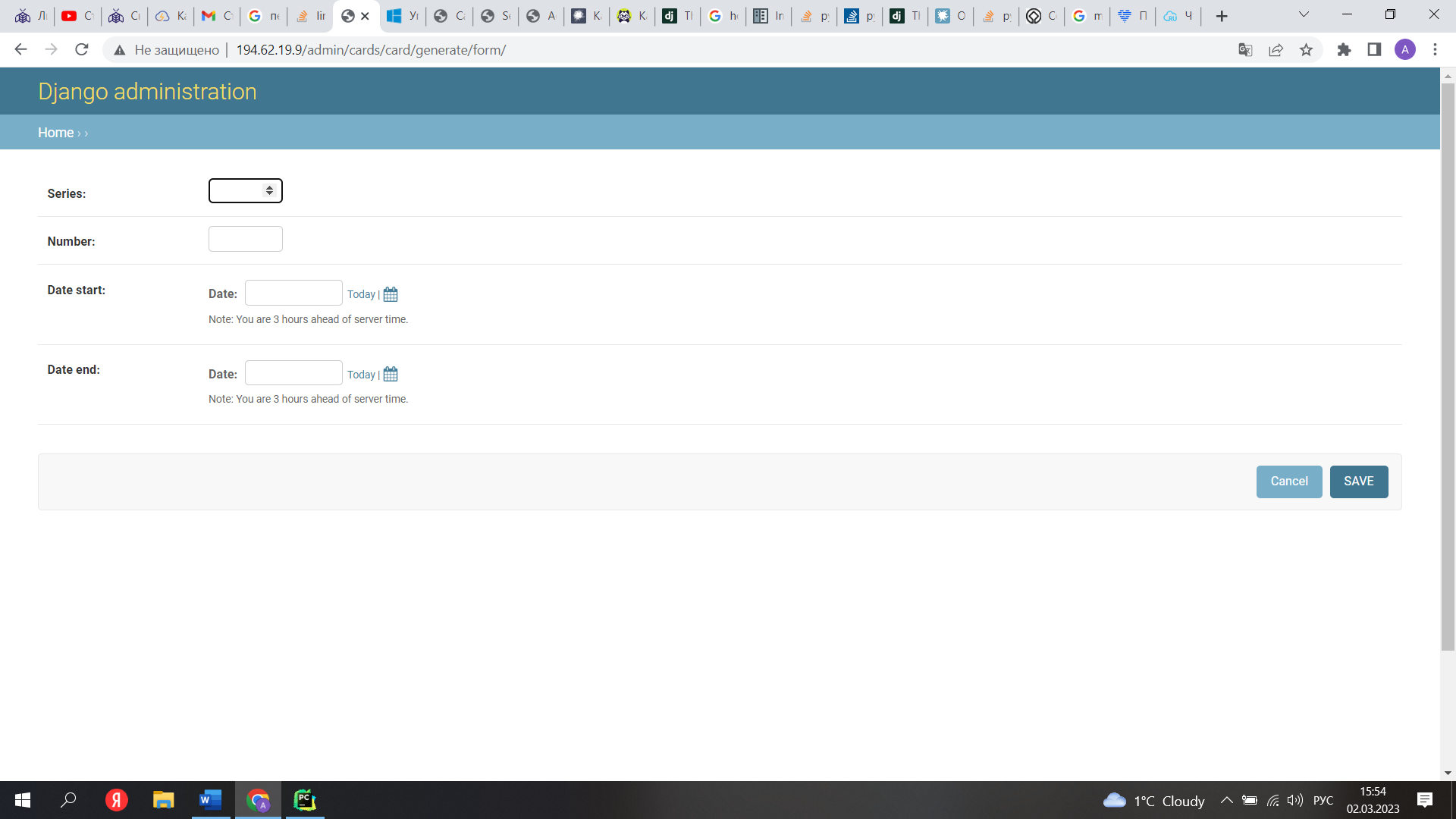
class WebsiteHistoryAdmin(SimpleHistoryAdmin):  
 history\_list\_display = ('series', 'number', 'created\_at', 'date\_end', 'status', 'view\_orders',)  
 list\_display = ('series', 'number','created\_at', 'date\_end', 'status', 'view\_orders',)  
 list\_filter = ('series', 'number', 'created\_at', 'date\_end', 'status',)  
 search\_fields = ('series', 'number')  
 actions = ['activate\_status\_actions', 'deactivate\_status\_actions']  
 exclude = ['date\_end']  
 list\_per\_page = 200  
 change\_list\_template = "admin/cards/cards\_change\_list.html"

**Листинг 1.2 – маршрутизация админки (файл urls.py)**

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

По нажатию на кнопку «Сгенерировать» открывается форма для ввода данных генерации карт.

Форма «Генератор карт» (рисунок 1.2).



**Рисунок 1.2 – форма «Генератор карт»**

На форме «Генератор карт» пользователь может выполнить следующие действия:

– перейти на страницу «Список карт» с выполнением генерации списка карт,

– перейти на страницу «Список карт» и закрыть форму генерации.

**Листинг 1.3 – Добавление кастомной маршрутизации и функция редиректа на страницу формы генератора карт(файл admin.py)**

def generate\_cards\_form(self, request):  
 return render(request, 'admin/cards/generate.html')  
  
def get\_urls(self):  
 urls = super(WebsiteHistoryAdmin, self).get\_urls()  
 custom\_urls = [  
 path('generate/form/', self.generate\_cards\_form, name='generate\_cards\_form'),  
 ]  
 return custom\_urls + urls

**Листинг 1.4 – добавление шаблона для генератора карт(файл generate.html)**

{% extends "admin/base\_site.html" %}  
{% load i18n admin\_urls static admin\_modify %}  
{% block extrahead %}{{ block.super }}  
 <script src="{% url 'admin:jsi18n' %}"></script>  
 <script src="/admin/jsi18n/"></script>  
 <script src="/static/admin/js/vendor/jquery/jquery.js"></script>  
 <script src="/static/admin/js/calendar.js"></script>  
 <script src="/static/admin/js/jquery.init.js"></script>  
 <script src="/static/admin/js/admin/DateTimeShortcuts.js"></script>  
 <script src="/static/admin/js/core.js"></script>  
 <script src="/static/admin/js/admin/RelatedObjectLookups.js"></script>  
 <script src="/static/admin/js/actions.js"></script>  
 <script src="/static/admin/js/urlify.js"></script>  
 <script src="/static/admin/js/prepopulate.js"></script>  
 <script src="/static/admin/js/vendor/xregexp/xregexp.js"></script>  
 {{ media }}  
{% endblock %}  
{% block extrastyle %}{{ block.super }}  
 <link rel="stylesheet" href="{% static "admin/css/forms.css" %}">{% endblock %}  
{% block coltype %}colM{% endblock %}  
{% block bodyclass %}{{ block.super }} app-{{ opts.app\_label }} model-{{ opts.model\_name }} change-form{% endblock %}  
{% if not is\_popup %}  
 {% block breadcrumbs %}  
 <div class="breadcrumbs">  
 <a href="{% url 'admin:index' %}">{% translate 'Home' %}</a>  
 &rsaquo; {% if has\_view\_permission %}  
 <a href="{% url opts|admin\_urlname:'changelist' %}">{{ opts.verbose\_name\_plural|capfirst }}</a>{% else %}  
 {{ opts.verbose\_name\_plural|capfirst }}{% endif %}  
 &rsaquo;  
 {% if add %}{% blocktranslate with name=opts.verbose\_name %}Add {{ name }}{% endblocktranslate %}{% else %}  
 {{ original|truncatewords:"18" }}{% endif %}  
 </div>  
 {% endblock %}  
{% endif %}  
{% block content %}  
 <div id="content-main">  
 <form action="{% url 'generate\_cards' %}" method="post" id="card\_form" novalidate="">  
 {% csrf\_token %}  
 <div>  
 <fieldset class="module aligned ">  
 <div class="form-row field-series">  
 <div>  
 <label class="required" for="id\_series">Series:</label>  
 <input type="number" name="series" class="vIntegerField" required="" id="id\_series">  
 </div>  
 </div>  
 <div class="form-row field-number">  
 <div>  
 <label class="required" for="id\_number">Number:</label>  
 <input type="number" name="number" class="vIntegerField" required="" id="id\_number">  
 </div>  
 </div>  
 <div class="form-row field-date\_start">  
 <div>  
 <label class="required" for="id\_date\_start\_0">Date start:</label>  
 <p class="datetime">  
 Date: <input type="text" name="date\_start\_0" class="vDateField" size="10" required=""  
 id="id\_date\_start">  
 </div>  
 </div>  
 <div class="form-row field-date\_end">  
 <div>  
 <label class="required" for="id\_date\_end\_0">Date end:</label>  
 <p class="datetime">  
 Date: <input type="text" name="date\_end\_0" class="vDateField" size="10" required=""  
 id="id\_date\_end">  
 </div>  
 </div>  
 </fieldset>  
  
 <div class="submit-row">  
 <input type="submit" value="Save" class="default" name="\_save">  
 <input type="submit" value="Cancel" name="\_cancel">  
 </div>  
  
 <script id="django-admin-form-add-constants" src="/static/admin/js/change\_form.js"  
 data-model-name="card" async="">  
 </script>  
  
 <script id="django-admin-prepopulated-fields-constants" src="/static/admin/js/prepopulate\_init.js"  
 data-prepopulated-fields="[]">  
 </script>  
 </div>  
 </form>  
 {% block object-tools %}  
 {% endblock %}  
 <form {% if has\_file\_field %}enctype="multipart/form-data" {% endif %}{% if form\_url %}action="{{ form\_url }}" {% endif %}method="post"  
 id="{{ opts.model\_name }}\_form" novalidate>{% csrf\_token %}{% block form\_top %}{% endblock %}  
 <div>  
 {% if is\_popup %}<input type="hidden" name="{{ is\_popup\_var }}" value="1">{% endif %}  
 {% if to\_field %}<input type="hidden" name="{{ to\_field\_var }}" value="{{ to\_field }}">{% endif %}  
 {% if save\_on\_top %}{% block submit\_buttons\_top %}{% submit\_row %}{% endblock %}{% endif %}  
 {% if errors %}  
 <p class="errornote">  
 {% blocktranslate count counter=errors|length %}Please correct the error below.{% plural %}  
 Please correct the errors below.{% endblocktranslate %}  
 </p>  
 {{ adminform.form.non\_field\_errors }}  
 {% endif %}  
 {% block field\_sets %}  
 {% for fieldset in adminform %}  
 {% include "admin/includes/fieldset.html" %}  
 {% endfor %}  
 {% endblock %}  
 {% block after\_field\_sets %}{% endblock %}  
 {% block inline\_field\_sets %}  
 {% for inline\_admin\_formset in inline\_admin\_formsets %}  
 {% include inline\_admin\_formset.opts.template %}  
 {% endfor %}  
 {% endblock %}  
 {% block after\_related\_objects %}{% endblock %}  
 {% block submit\_buttons\_bottom %}{% endblock %}  
 {% block admin\_change\_form\_document\_ready %}  
 <script id="django-admin-form-add-constants"  
 src="{% static 'admin/js/change\_form.js' %}"  
 {% if adminform and add %}  
 data-model-name="{{ opts.model\_name }}"  
 {% endif %}  
 async>  
 </script>  
 {% endblock %}  
 {# JavaScript for prepopulated fields #}  
 <script id="django-admin-form-add-constants" src="/static/admin/js/change\_form.js"  
 data-model-name="card" async="">  
 </script>  
 <script id="django-admin-prepopulated-fields-constants" src="/static/admin/js/prepopulate\_init.js"  
 data-prepopulated-fields="[]">  
 </script>  
 </div>  
 </form>  
 </div>  
{% endblock %}

**Листинг 1.5 – добавление маршрутизации для генератора карт(файл urls.py)**

path('generate/', views.generate\_cards, name='generate\_cards'),

**Листинг 1.6 – добавление обработчика формы генерации карт (файл views.py)**

from django.http import HttpResponseRedirect  
from .models import gen\_smth

def generate\_cards(request):  
 print(request.POST)  
 if request.POST['number'] == '':  
 return HttpResponseRedirect("/cards/card/")  
 for i in range(int(request.POST['number'])):  
 percent = DiscountPercent.objects.get(name='default')  
 card = Card(series=request.POST['series'],number=gen\_smth(3),created\_at=request.POST['date\_start\_0'],date\_end=request.POST['date\_end\_0'],summa\_purchases=0,status='Active',percent=percent)  
 card.save()  
 return HttpResponseRedirect("/admin/cards/card/")

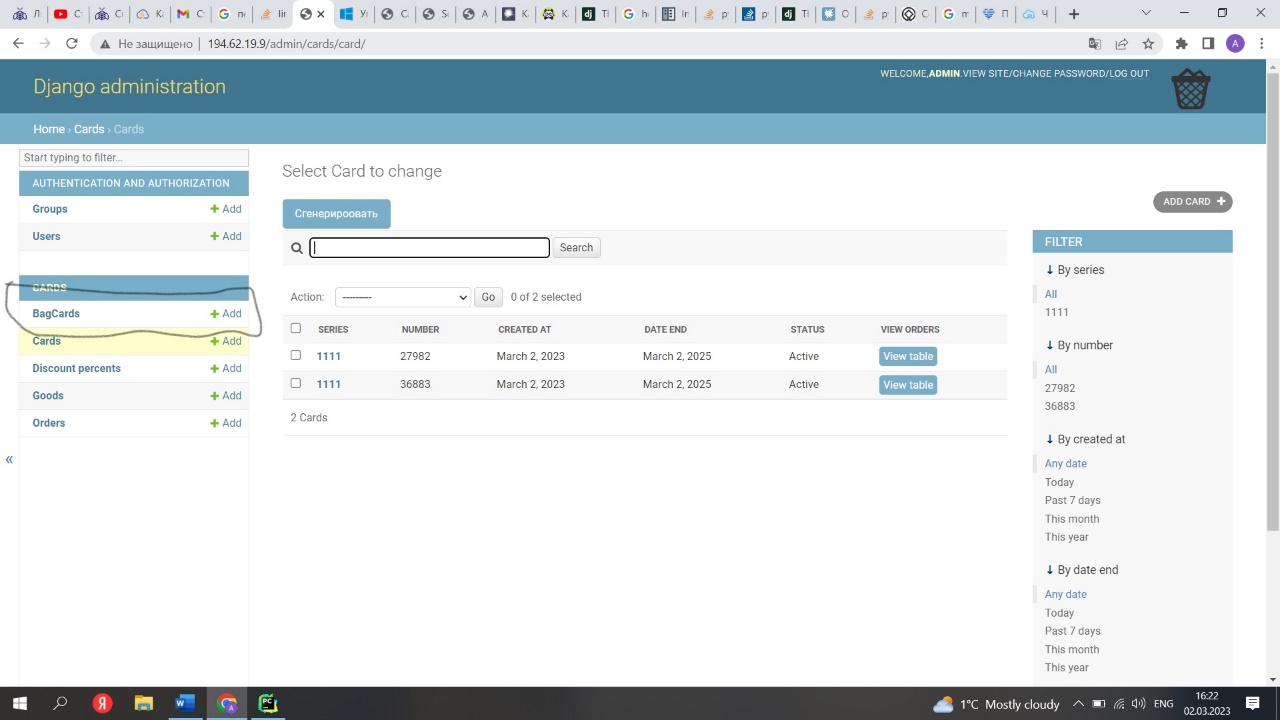
**Листинг 1.7 – добавление функции для генерации уникальных 4-х значных номеров карты (файл models.py)**

import random

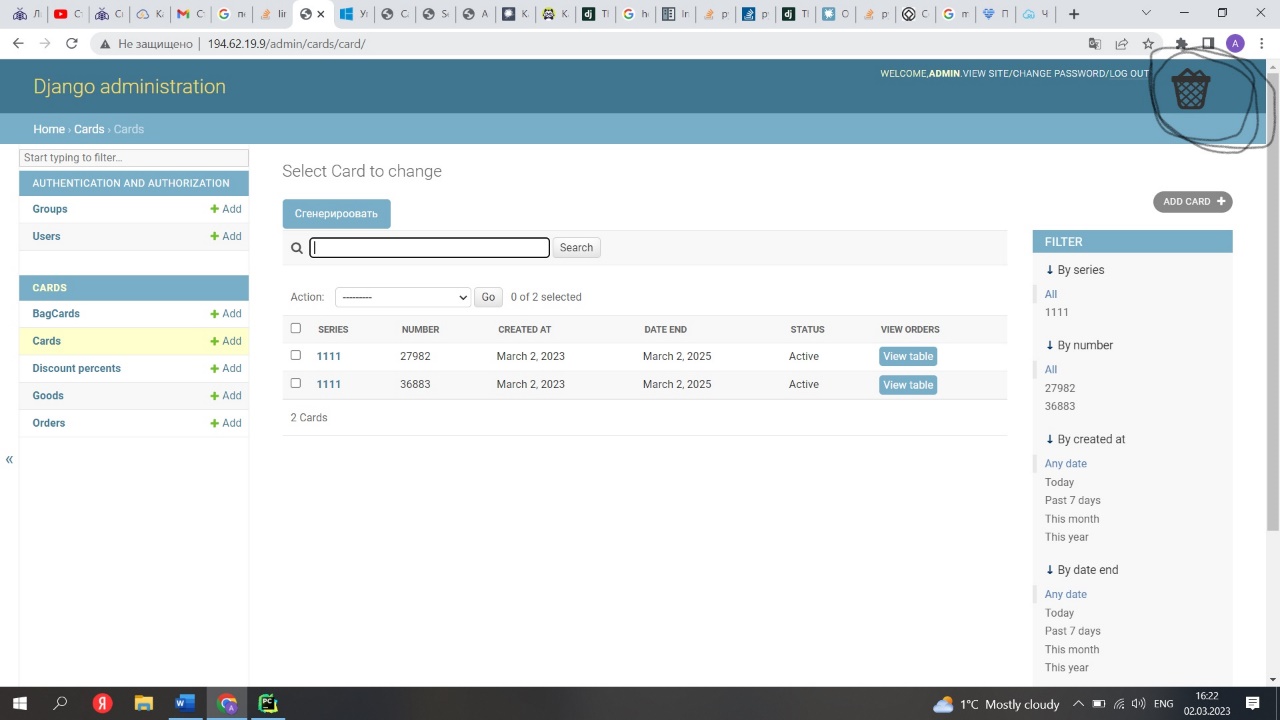
def sum\_digits(x):  
 res = 0  
  
 while x:  
 res += x % 10  
 x //= 10  
  
 return res  
  
  
def gen\_smth(N):  
 digit = random.randint(1, 9)  
 mid = random.randint(0, 10\*\*N - 1)  
 if sum\_digits(mid) % 2:  
 mid -= 1  
 rez = (digit \* 10\*\*N + mid) \* 10 + digit  
 if Card.objects.filter(number=rez).exists() and BagCards.objects.filter(number=rez).exists():  
 gen\_smth(3)  
 return rez

По нажатию на кнопку «Save» открывается страница списка карт

На странице списка карт есть 2 способа попасть на страницу со списком карт находящихся в корзине (рисунок 3.3, 3.4).



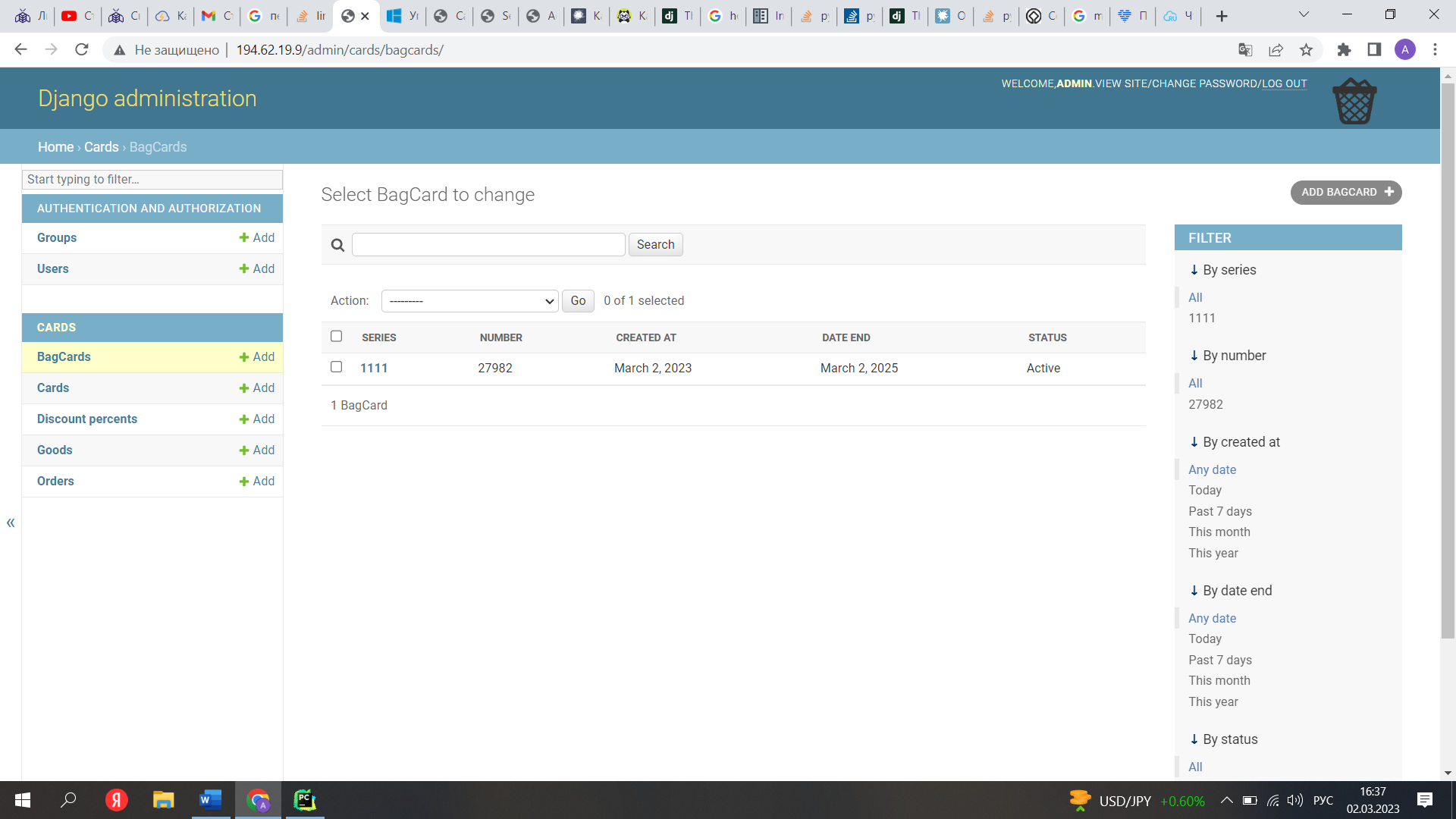
**Рисунок 1.3 – Кнопка перехода в корзину**



**Рисунок 1.4 – Иконка перехода в корзину**

По нажатию на кнопку или иконку для перехода в корзину открывается страница со списком карт находящихся в корзине.

Списком карт находящихся в корзине (рисунок 3.5).



**Рисунок 1.5 – Списком карт в корзине**

Текст программного кода, отвечающего за работу страниц, встроенных в кнопку и иконку редиректа и страницу списка карт в корзине представлен в листингах 1.8. – 1. 9 .

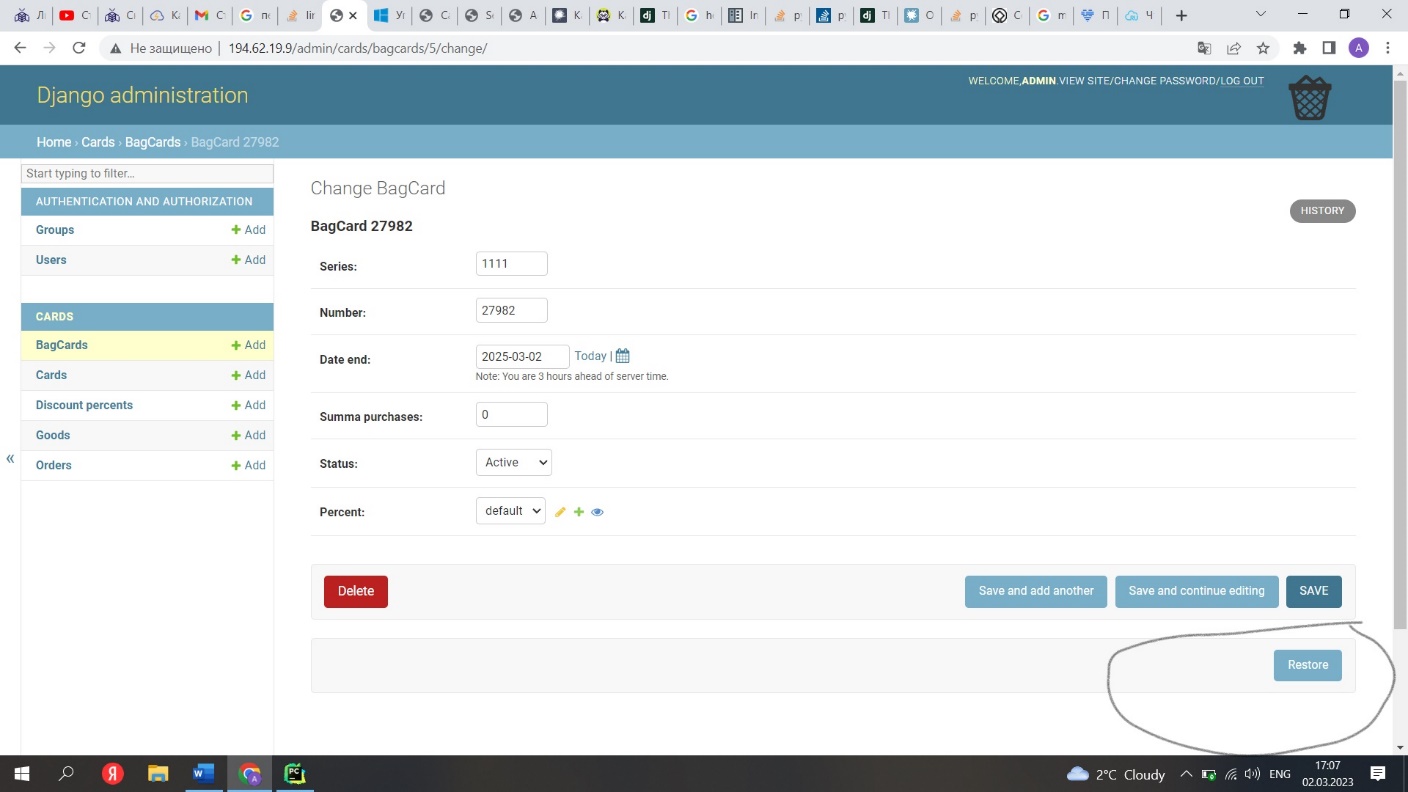
**Листинг 1.8 – переопределение шаблона base.html для добавления иконки корзины (файл base.html)**

<a href="{% url 'admin:cards\_bagcards\_changelist' %}" style="border-bottom: none;"><img src="{% static 'img/bag.png' %}" style="width: 100px; height: 50px"></a>

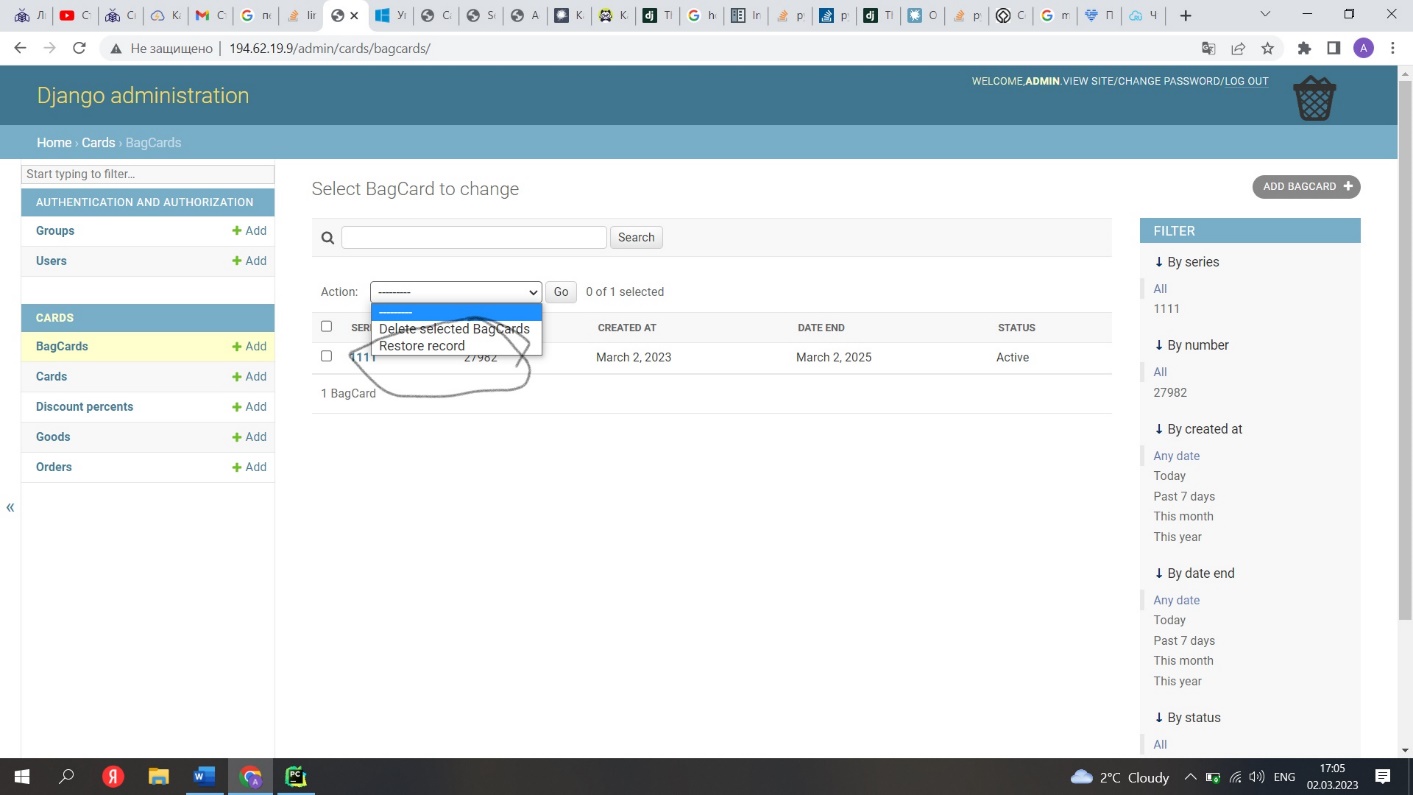
**Листинг 1.9 – Переопределение методов ModelAdmin(delete model(), delete\_queryset()) для мягкого удаления (файл admin.py)**

def delete\_model(self, request, change):  
 add\_in\_bag(change)  
 print(BagCards.objects.all())  
 super().delete\_model(request, change)  
  
def delete\_queryset(self, request, queryset):  
 print(queryset)  
 for obj in queryset:  
 add\_in\_bag(obj)  
 super().delete\_queryset(request, queryset)

Также есть функции для восстановления карты из корзины на странице профиля карты и восстановления списка карт на странице списка карт в корзине

Страница «Профиль карты» кнопка «Restore» (рисунок 1.6) и страница «Список карт в корзине» кнопка «Restore record» (рисунок 1.7)

**Рисунок 1.6 – Страница «Профиль карты» кнопка «Restore»**



**Рисунок 1.7 – страница «Список карт в корзине» кнопка «Restore record»**

**Листинг 1.10 – добавление логики отображения на страницу списка карт в корзине(файл admin.py)**

@admin.register(BagCards)  
class BagCardsAdmin(SimpleHistoryAdmin):  
 list\_display = ('series', 'number', 'created\_at', 'date\_end', 'status',)  
 list\_filter = ('series', 'number', 'created\_at', 'date\_end', 'status',)  
 search\_fields = ('series', 'number')  
 actions = ['restore\_actions']  
 change\_form\_template = "admin/cards/my\_change\_form.html"

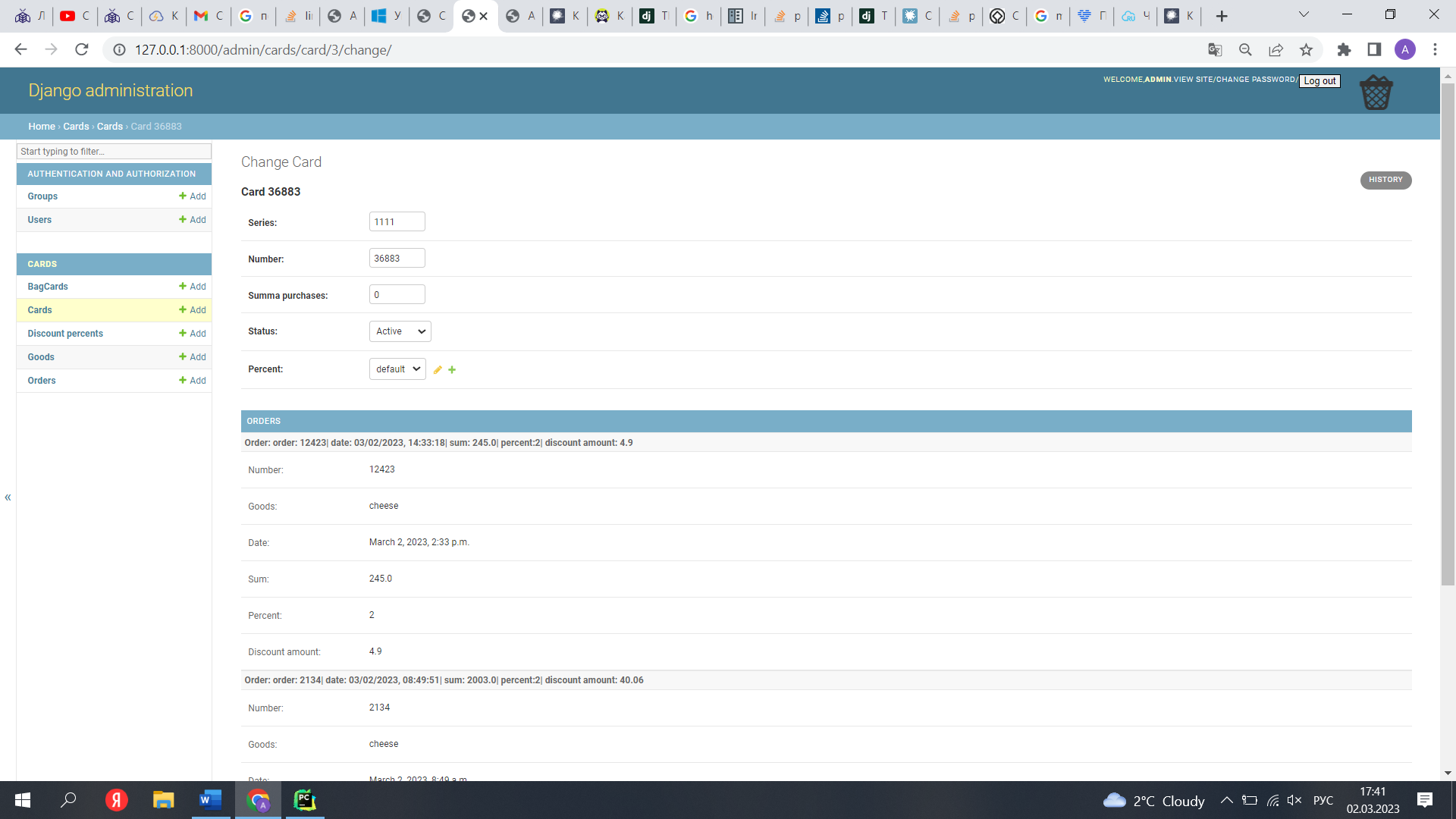
**Листинг 1.11 – Переопределение страницы change\_form.html для корзины(файл my\_change\_form.html)**

{% extends 'admin/change\_form.html' %}  
  
{% block submit\_buttons\_bottom %}  
 {{ block.super }}  
 <div class="submit-row">  
 <input type="submit" value="Restore" name="restore">  
 </div>  
{% endblock %}

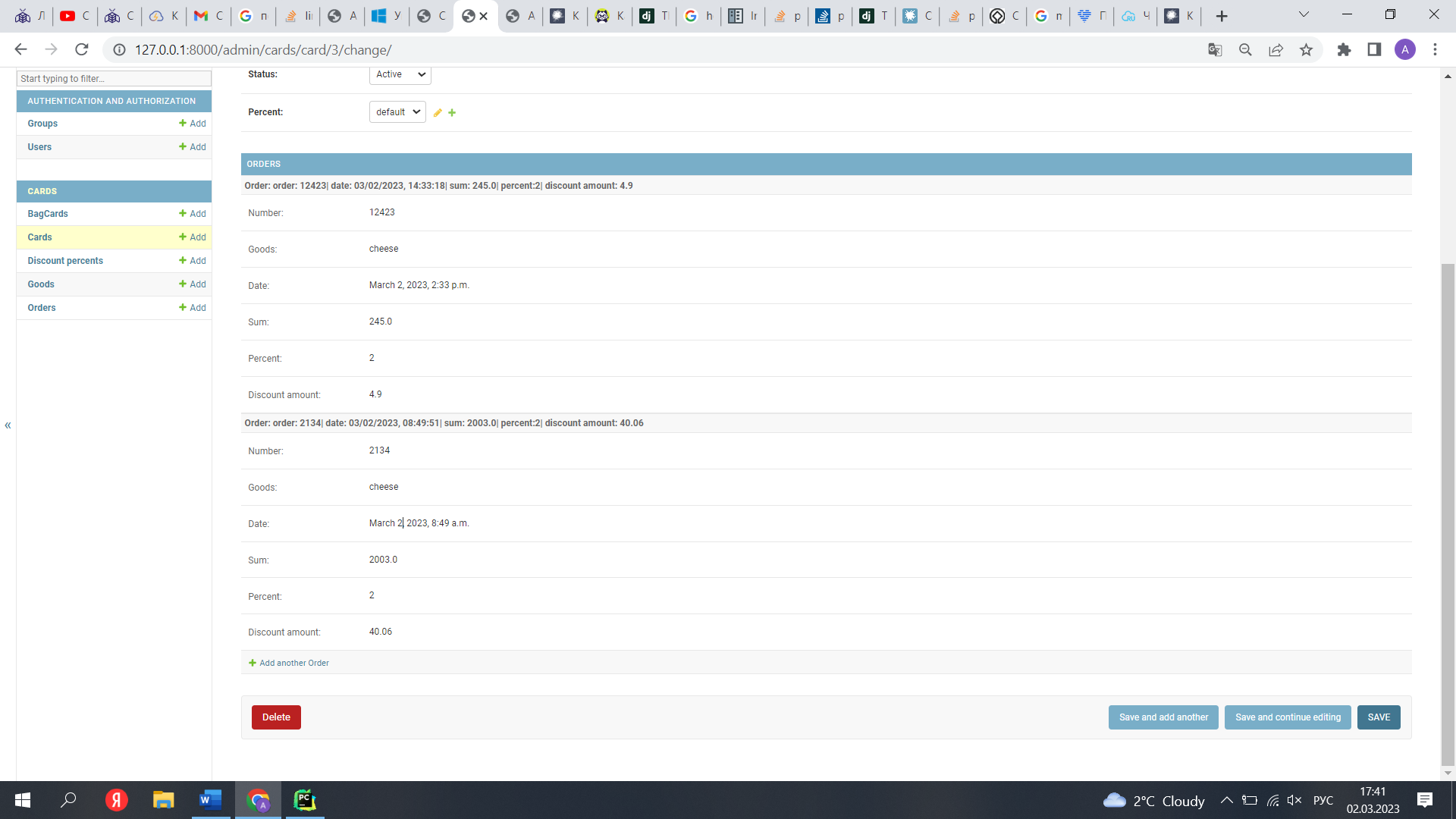
**Листинг 1.12 – добавление кастомной кнопки в страницу профиля корзины карт и action на страницу списка карт в корзине(файл admin.py)**

def restore\_actions(self, form, queryset):  
 for obj in queryset:  
 restore\_in\_cards(obj)  
 return format\_html(  
 '<a class="button" href="{}">Restore</a>',  
 reverse('admin:cards\_card\_changelist'),  
 )  
  
restore\_actions.short\_description = 'Restore record'  
restore\_actions.allow\_tags = True  
  
def response\_change(self, request, obj):  
 if "restore" in request.POST:  
 restore\_in\_cards(obj)  
 self.message\_user(request, "Restoring complete")  
 return HttpResponseRedirect("/admin/cards/card/")  
 return super().response\_change(request, obj)

Перейдем к странице профиля карты с историей ордеров по ней (рисунок 1.8-1.9)



**Рисунок 1.8 – Профиль карты часть 1**



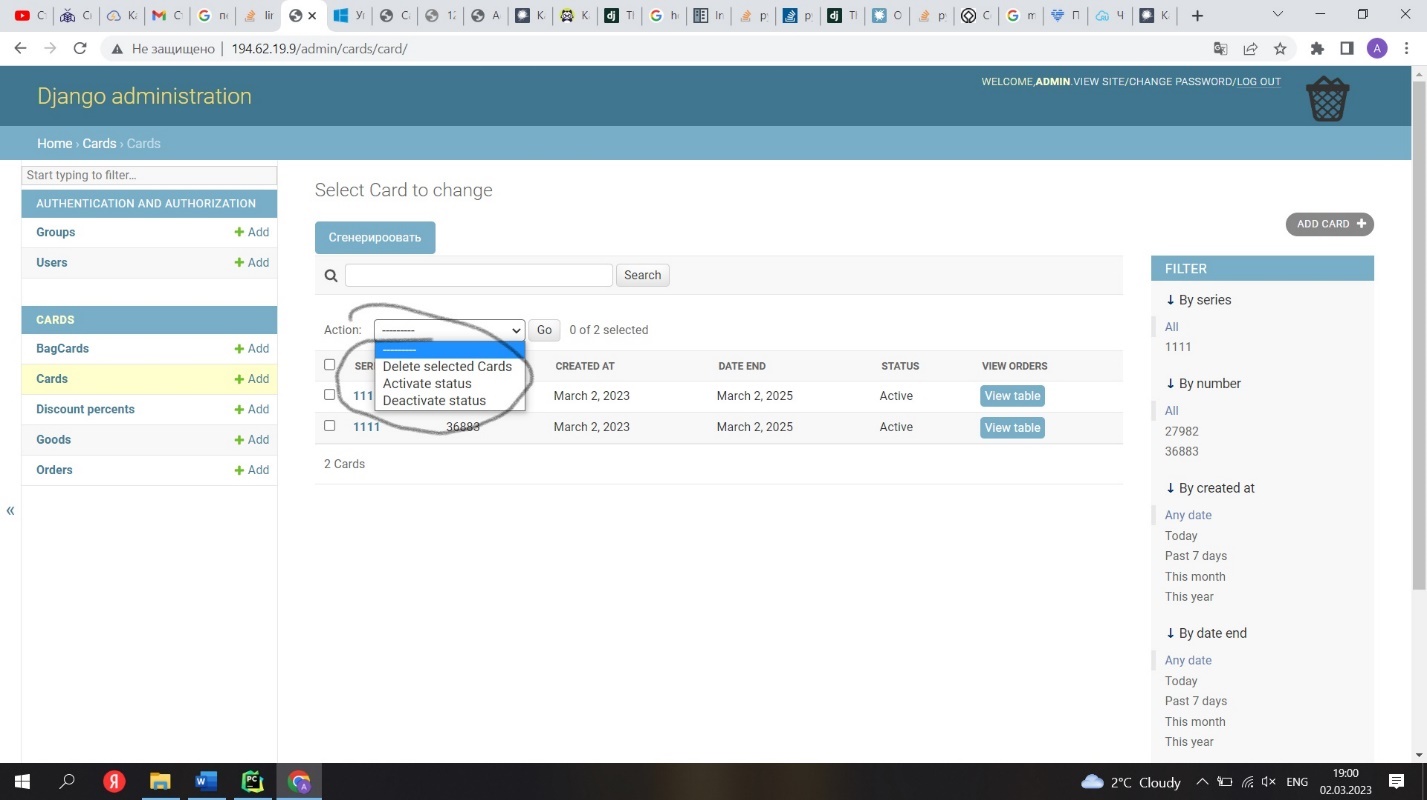
**Рисунок 1.9 – Профиль карты часть 2**

**Листинг 1.13 – добавление добавления списка ордеров без возможности редактирования на страницу профиля(файл admin.py)**

class OrdersInline(admin.StackedInline):  
 extra = 0  
 model = Orders  
 readonly\_fields = ('number','goods','date','sum','percent','discount\_amount')  
 can\_delete = False  
 exclude = ['bag\_id']

class WebsiteHistoryAdmin(SimpleHistoryAdmin):  
  
 change\_list\_template = "admin/cards/cards\_change\_list.html"  
 inlines = [  
 OrdersInline,  
 ]

Также добавлены actions на странице списка карт пользователей (рисунок 1.10)



**Рисунок 1.10 – Actions для активации и деактивации карты**

**Листинг 1.14 – добавление actions для активации и деактивации карты (файл admin.py)**

def activate\_status\_actions(self, request, queryset):  
 count = 0  
 for obj in queryset:  
 count +=1  
 obj.status = 'Active'  
 obj.save()  
 self.message\_user(request, f"Установлен статус Active для {count} записей")  
 return reverse('admin:cards\_card\_changelist'),  
  
  
activate\_status\_actions.short\_description = 'Activate status'  
activate\_status\_actions.allow\_tags = True  
  
def deactivate\_status\_actions(self, request, queryset):  
 count = 0  
 for obj in queryset:  
 count +=1  
 obj.status = 'Inactive'  
 obj.save()  
 self.message\_user(request, f"Установлен статус Inactive для {count} записей")  
 return reverse('admin:cards\_card\_changelist'),  
  
  
deactivate\_status\_actions.short\_description = 'Deactivate status'  
deactivate\_status\_actions.allow\_tags = True

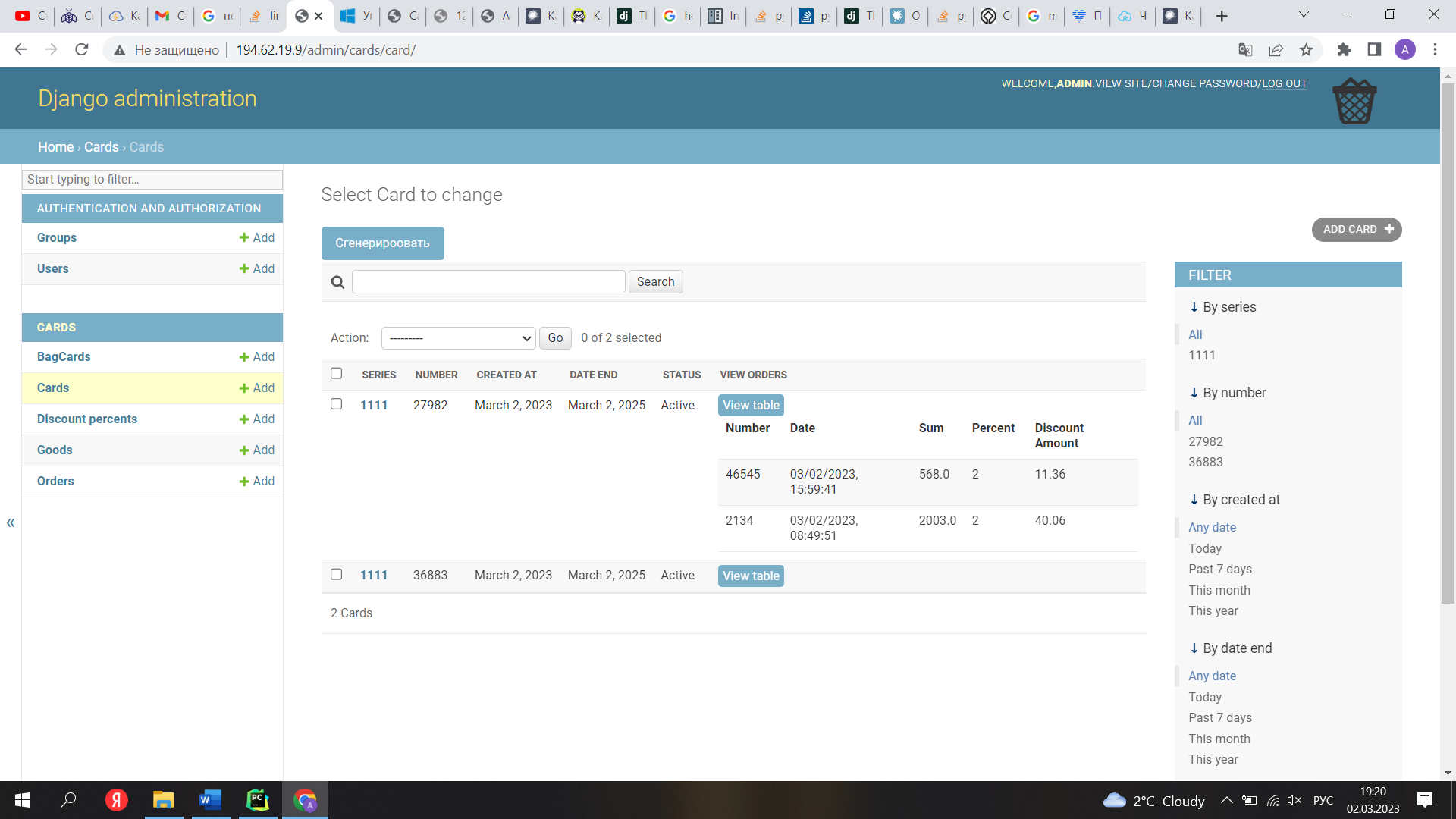
Функция проставления статуса просрочена при истечении времени использования карты представлена на листинге 1.15

**Листинг 1.15 – Функция проверки статуса(файл views.py)**

import schedule  
import datetime

def status\_check():  
 cards = Card.objects.all()  
 for i in cards:  
 if i.status == 'Active' or i.status == 'Inactive' or i.status == 'Frozen':  
 if i.date\_end >= datetime.date.today():  
 i.status = 'Overdue'  
 i.save()  
  
  
schedule.every().day.at("08:00").do(status\_check)

Добавлена кастомная кнопка для просмотра ордеров записи на странице списка товаров (рисунок 1.11)



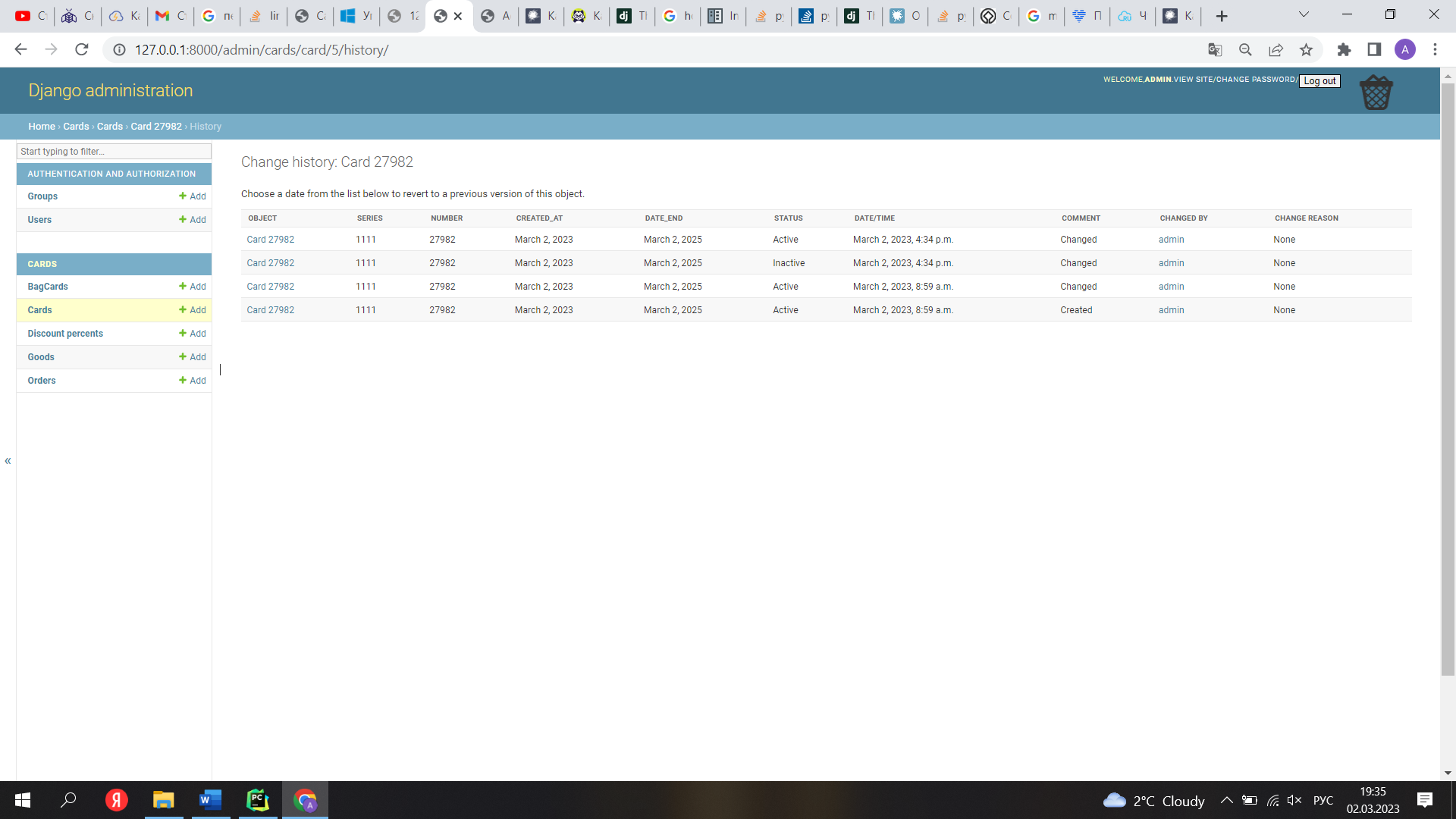
**Рисунок 1.11 – Кастомная кнопка показа ордеров**

**Листинг 1.16 – Кастомная кнопка показа ордеров(файл admin.py)**

class WebsiteHistoryAdmin(SimpleHistoryAdmin):  
 history\_list\_display = (''view\_orders')  
 list\_display = ('view\_orders')

def view\_orders(self, obj):  
 html = '"<table><tr><th>Number</th><th>Date</th>' \  
 '<th>Sum</th><th>Percent</th><th>Discount Amount</th></tr>'  
 if not obj.order.all():  
 html += '</table>"'  
 html\_end = f'<a class="button" id="button\_table" onclick="func\_{obj.id}({obj.id})">View table</a>' \  
 f'<div id="table\_{obj.id}"></div>' \  
 '<script>' \  
 'var count = 0;' \  
 f'function func\_{obj.id}(id) ' \  
 '{{{{' \  
 f'var elem = document.getElementById("table\_"+String(id));' \  
 'if(count === 0) {{{{elem.innerHTML = {0};count++;}}}}' \  
 'else {{{{elem.innerHTML = "";count--}}}}' \  
 '}}}}' \  
 '</script>'.format(html)  
 return format\_html(html\_end)  
 else:  
 for i in obj.order.all():  
 html\_columns = '<tr><td>{0}</td><td>{1}</td><td>{2}</td><td>{3}</td><td>{4}</td></tr>'.format(  
 i.number,  
 i.date.strftime("%m/%d/%Y, %H:%M:%S"),  
 i.sum,  
 i.percent,  
 i.discount\_amount  
 )  
 html += html\_columns  
 html += '</table>"'  
  
 html\_end = f'<a class="button" id="button\_table" onclick="func\_{obj.id}({obj.id})">View table</a>' \  
 f'<div id="table\_{obj.id}"></div>' \  
 '<script>' \  
 'var count = 0;' \  
 f'function func\_{obj.id}(id) ' \  
 '{{{{' \  
 f'var elem = document.getElementById("table\_"+String(id));' \  
 'if(count === 0) {{{{elem.innerHTML = {0};count++;}}}}' \  
 'else {{{{elem.innerHTML = "";count--}}}}' \  
 '}}}}' \  
 '</script>'.format(html)  
 return format\_html(html\_end)

Дополнительная функция истории изменений карты(рисунок 1.12)



**Рисунок 1.12 – История изменений карты**

**Листинг 1.16 –Добавления модуля истории(файл settings.py)**

INSTALLED\_APPS = [  
 'simple\_history',  
]

**Листинг 1.17 – История изменений карты(файл admin.py)**

from simple\_history.admin import SimpleHistoryAdmin

class WebsiteHistoryAdmin(SimpleHistoryAdmin):  
 history\_list\_display = ('series', 'number', 'created\_at', 'date\_end', 'status')

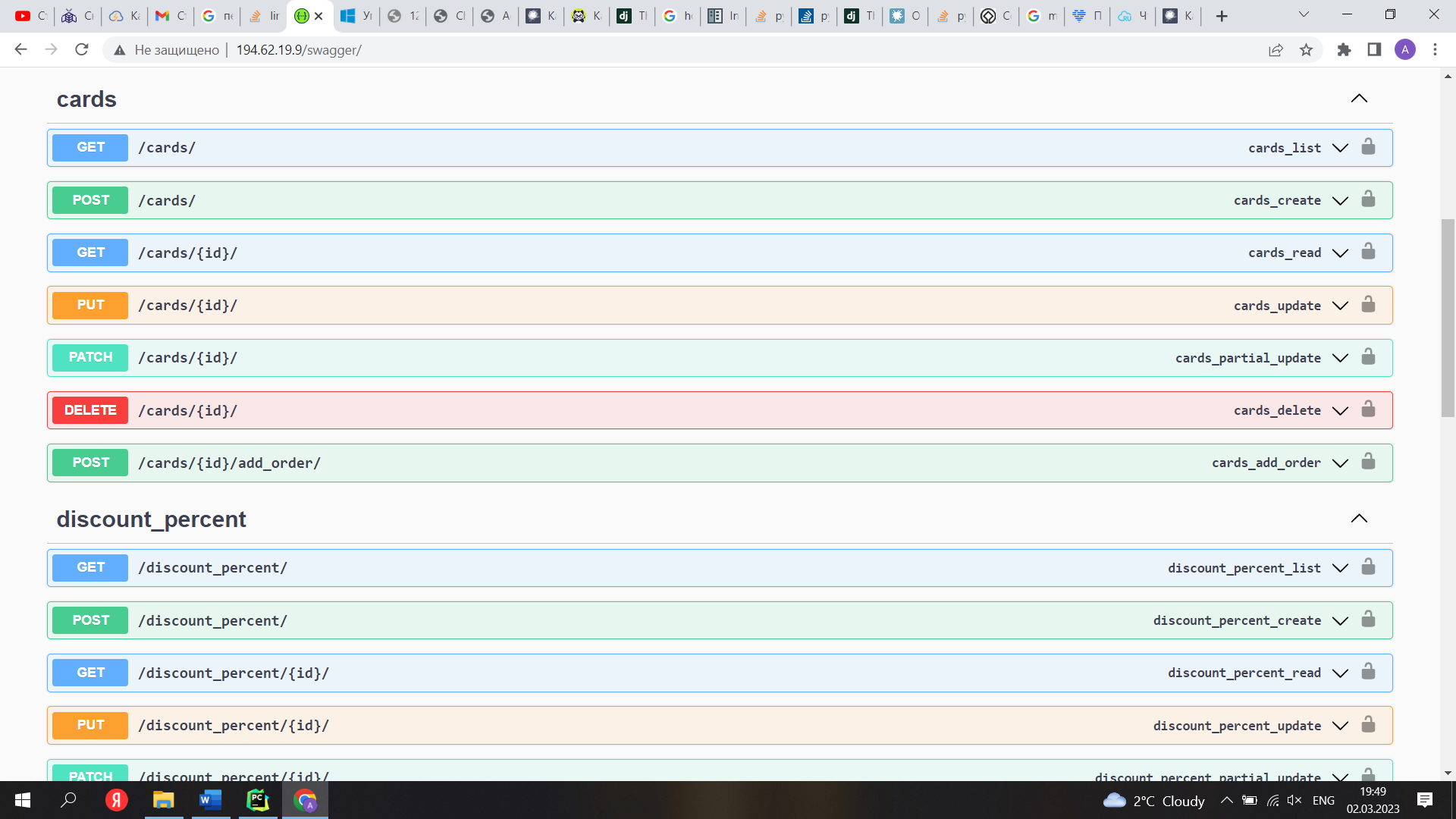
**2.Функции интеграции**

* необходимо реализовать REST API для интеграции с сервисом(рисунок 2.1-2.2 ,листинги 2.1-2.2),
* получение информации по номеру карты (информация по карте, информация по заказам и товарам карты с возможностью фильтрации по датам) (рисунок 2.3-2.4 ,листинги 2.3-2.5),
* записи информации по заказам и товарам для определенной карты(рисунок 2.5 ,листинги 2.6).

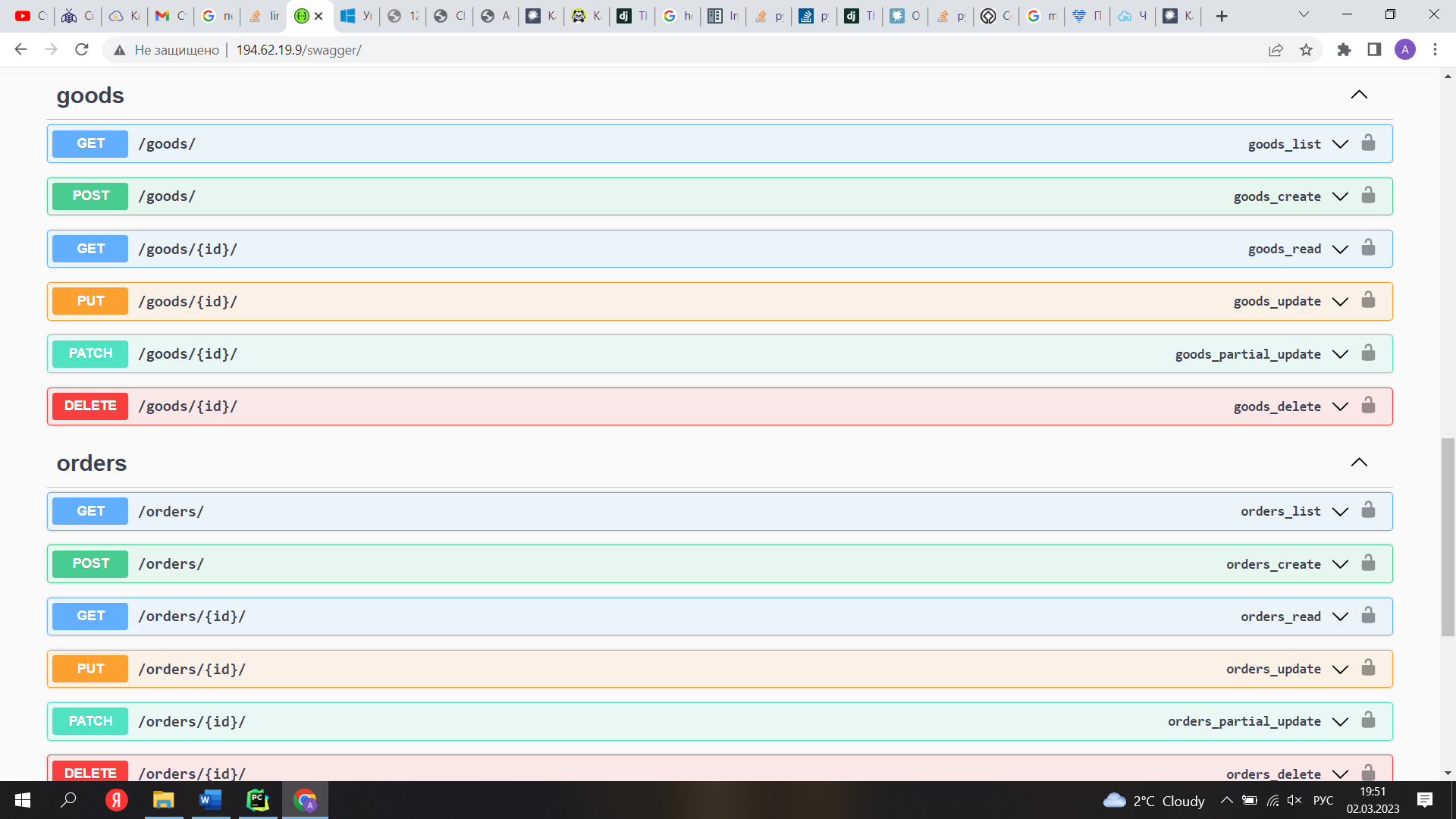
Дополнительные Функции интеграции

* добавлен swagger для наглядности отображения запросов и автодокументирования их (листинги 2.7),

REST API отражен в следующих запросах «Список запросов» (рисунок 2.1-2.2).



**Рисунок 2.1 – Список запросов часть 1**



**Рисунок 2.2 – Список запросов часть 2**

**Листинг 2.1 – Реализация запросов(файл views.py)**

from rest\_framework.viewsets import ModelViewSet

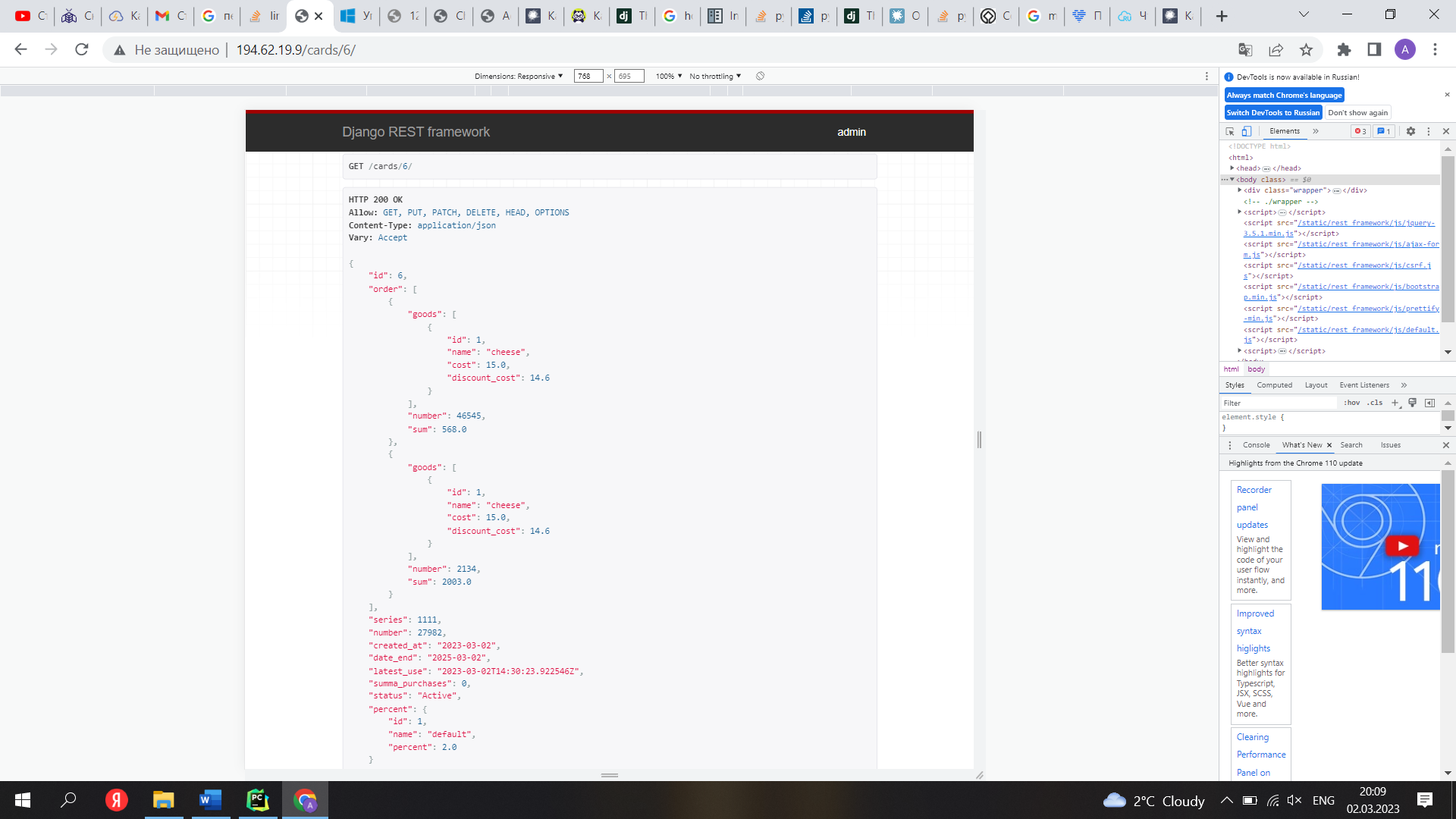
class DiscountPercentViewSet(ModelViewSet):  
 queryset = DiscountPercent.objects.all()  
 serializer\_class = DiscountPercentSerializer  
  
  
class OrdersViewSet(ModelViewSet):  
 serializer\_class = OrdersSerializer  
 filter\_backends = (DjangoFilterBackend,)  
 filterset\_class = OrdersFilter  
  
  
class GoodsViewSet(ModelViewSet):  
 queryset = Goods.objects.all()  
 serializer\_class = GoodsSerializer  
  
  
class CardViewSet(ModelViewSet):  
 queryset = Card.objects.all()  
 serializer\_class = CardSerializer  
 filter\_backends = (DjangoFilterBackend,)  
 filterset\_class = CardsFilter

**Листинг 2.2 –Маршрутизация запросов(файл urls.py)**

from django.contrib import admin  
from django.urls import path, include, re\_path  
from . import views  
from rest\_framework import routers

router = routers.SimpleRouter()  
router.register('cards', CardViewSet, basename='cards')  
router.register('orders', OrdersViewSet, basename='orders')  
router.register('goods', GoodsViewSet, basename='goods')  
router.register('discount\_percent', DiscountPercentViewSet, basename='discount\_percent')

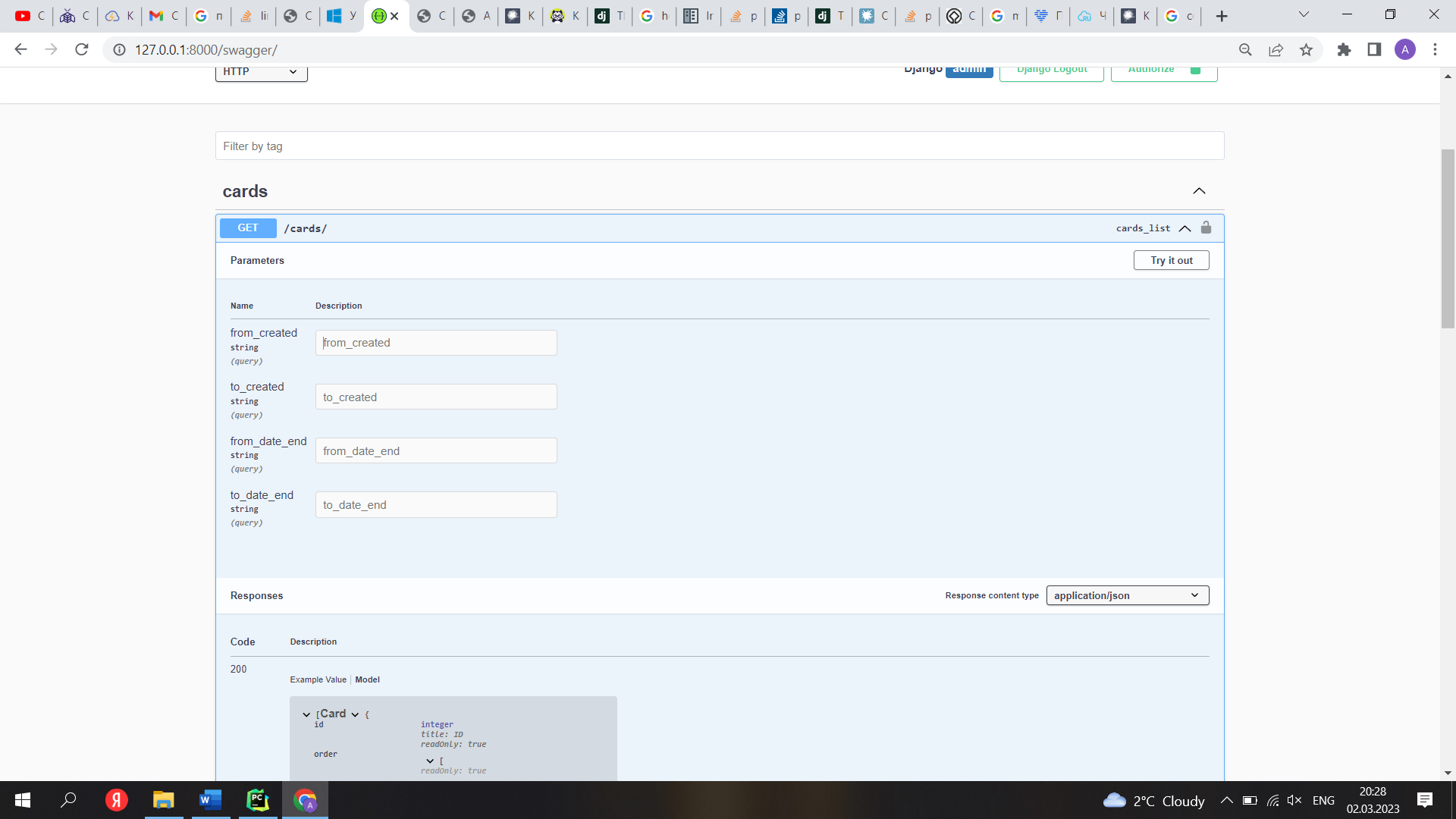
Далее реализован запрос на получение информации по номеру карты (информация по карте, информация по заказам и товарам карты с возможностью фильтрации по датам). Получение всей информации по карте реализовано с помощью вложенных сериализаторов ModelAdmin.(рисунок 2.3)



**Рисунок 2.3 – Информация по запросу для конкретной карты**

**Листинг 2.3 –Реализация сериализаторов(файл serializers.py)**

from rest\_framework import serializers  
from .models import Card, Orders, DiscountPercent, Goods  
  
  
class GoodsSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Goods  
 fields = '\_\_all\_\_'  
  
  
class OrdersSerializer(serializers.ModelSerializer):  
 goods = GoodsSerializer(many=True)  
  
 class Meta:  
 model = Orders  
 fields = ('goods','number', 'sum',)  
  
  
class DiscountPercentSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = DiscountPercent  
 fields = '\_\_all\_\_'  
  
  
class CardSerializer(serializers.ModelSerializer):  
 order = OrdersSerializer(read\_only=True,many=True)  
 class Meta:  
 model = Card  
 fields = '\_\_all\_\_'  
 depth = 2



**Рисунок 2.4 – сортировка карт по дате**

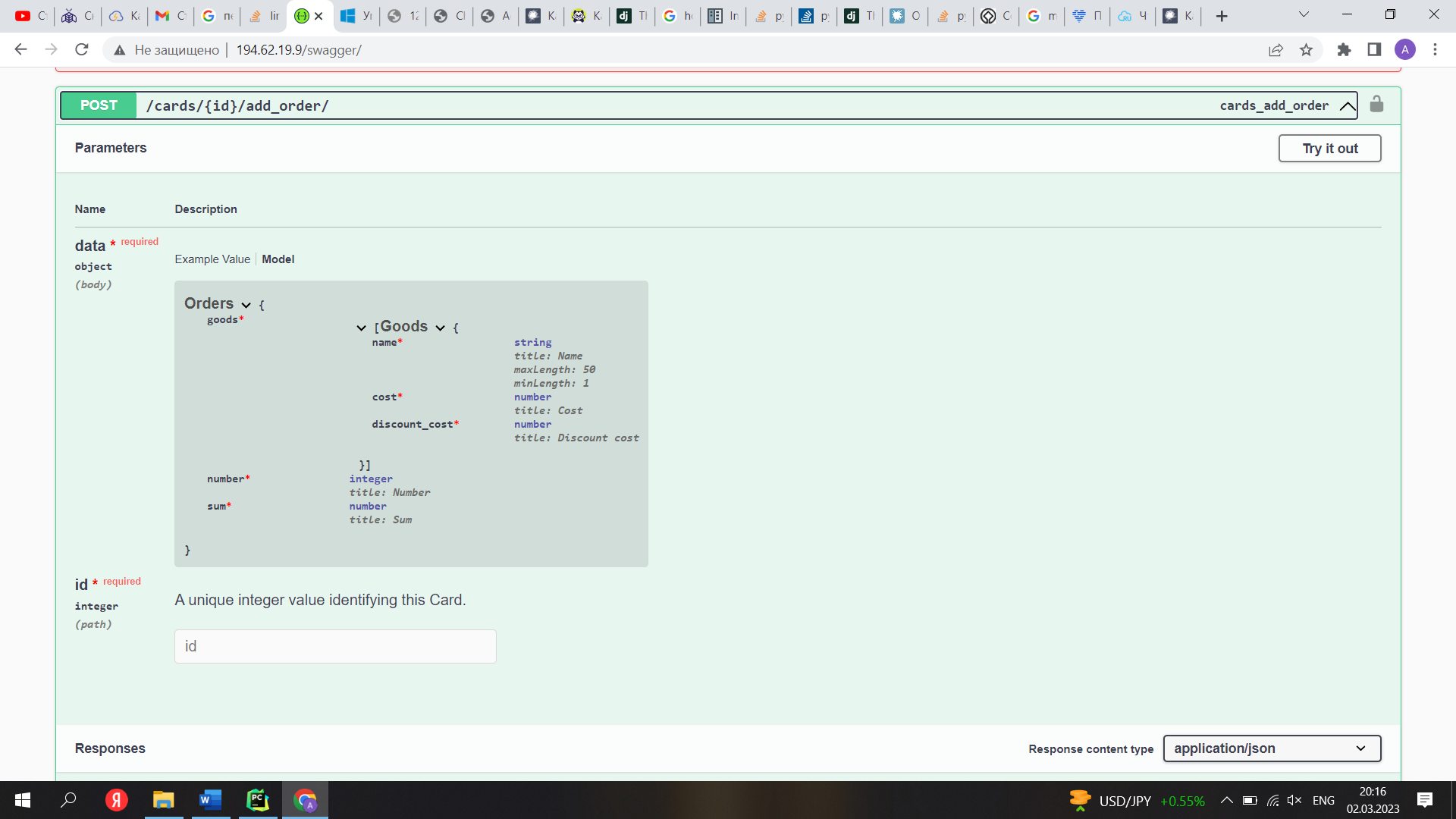
**Листинг 2.4 –Реализация классов сортировки в отдельном сервисе(файл filter\_class.py)**

from django\_filters import DateTimeFilter, FilterSet  
from ..models import Orders, Card  
  
  
class OrdersFilter(FilterSet):  
 from\_date = DateTimeFilter(field\_name='date',  
 lookup\_expr='gte')  
 to\_date = DateTimeFilter(field\_name='date',  
 lookup\_expr='lte')  
 class Meta:  
 model = Orders  
 fields = ['from\_date','to\_date',]  
  
  
class CardsFilter(FilterSet):  
 from\_created= DateTimeFilter(field\_name='created\_at',  
 lookup\_expr='gte')  
 to\_created = DateTimeFilter(field\_name='created\_at',  
 lookup\_expr='lte')  
 from\_date\_end = DateTimeFilter(field\_name='date\_end',  
 lookup\_expr='gte')  
 to\_date\_end = DateTimeFilter(field\_name='date\_end',  
 lookup\_expr='lte')  
 class Meta:  
 model = Card  
 fields = ['from\_created','to\_created','from\_date\_end','to\_date\_end']

**Листинг 2.5 –Подключение классов сортировки (файл views.py)**

class CardViewSet(ModelViewSet):  
 filter\_backends = (DjangoFilterBackend,)  
 filterset\_class = CardsFilter

Записи информации по заказам и товарам для определенной карты(рисунок 2.4)



**Рисунок 2.5 – Дополнительный запрос по карте с созданием ордера**

**Листинг 2.6 – Функция реализации дополнительного запроса по карте с созданием ордера(файл views.py)**

class CardViewSet(ModelViewSet):

@action(detail=True, methods=['post'], serializer\_class=OrdersSerializer)  
def add\_order(self, request, pk):  
 card = self.get\_object()  
 if card.status != 'Overdue':  
 serializer = OrdersSerializer(data=request.data)  
 try:  
 percent = default\_percent()  
 order = Orders.objects.create(  
 number=int(serializer.initial\_data['number']),  
 percent=percent.percent,  
 discount\_amount=(serializer.initial\_data['sum'] \* percent.percent) / 100,  
 sum=serializer.initial\_data['sum'],  
 card=card  
 )  
 for obj in serializer.initial\_data['goods']:  
 try:  
 goods = Goods.objects.get(name=obj['name'])  
 except:  
 goods = False  
 if goods:  
 order.goods.add(goods)  
 goods.orders\_set.add(order)  
 order.save()  
 pass  
 else:  
 goods = Goods(  
 name=obj['name'],  
 cost=obj['cost'],  
 discount\_cost=obj['discount\_cost']  
 )  
 goods.save()  
 goods.orders\_set.add(order)  
 order.save()  
 return Response({'status': 'adding order complete'})  
 except:  
 if serializer.is\_valid():  
 order = Orders.objects.create(  
 number=serializer.validated\_data['number'],  
 sum=serializer.validated\_data['sum'],  
 )  
 else:  
 return Response(serializer.errors,  
 status=status.HTTP\_400\_BAD\_REQUEST)

Дополнительная функция подключение swagger.

**Листинг 2.7 –Подключение swagger файл views.py)**

from drf\_yasg.views import get\_schema\_view  
from drf\_yasg import openapi  
from rest\_framework import permissions  
  
  
schema\_view = get\_schema\_view(  
 openapi.Info(  
 title="Snippets API",  
 default\_version='v1',  
 description="CarSharing API description",  
 contact=openapi.Contact(email="contact@snippets.local"),  
 license=openapi.License(name="BSD License"),  
 ),  
 public=True,  
 permission\_classes=[permissions.AllowAny],  
)

**3. Процесс разворачивания сайта на сервере**

**Оформление хостинга**

Заходим на сайт hostfly.by, авторизируемся и начинаем оформлять VPS хостинг. При оформлении можно взять пробный период на 10 дней. Выбираем тариф который нам нужен и заканчиваем оформление после чего нам приходит письмо на электронную почту со всеми данными.

**Хостинг оформлен**

Теперь переходим в putty или другой сервис для подключения к серверам по ssh и подключаемся к нашему серверу при помощи данных для входа которые нам пришли на почту при оформлении.

Для начала обновляем локальную базу пакетов при помощи команд:

sudo apt update  
sudo apt upgrade

Для установки необходимого набора пакетов и зависимостей выполним следующую команду:

sudo apt install python3-pip python3-dev libpq-dev nginx curl

В результате у вас будут установлены средства разработки Python, необходимые для сборки Gunicorn, pip, а также веб-сервер Nginx.

**Создание виртуальной среды Python для проекта**

Сначала необходимо настроить доступ к команде virtualenv, которую можно установить с помощью pip:

sudo -H pip3 install --upgrade pip  
sudo -H pip3 install virtualenv

Создайте и перейдите в каталог, где вы собираетесь хранить файлы вашего проекта. В нашем примере используется название каталога testproj:

mkdir ~/testproj   
~/ testproj

В каталоге проекта создайте виртуальную среду Python. Выберете название среды, которое имеет отношение к вашему проекту, в нашем примере мы используем название testprojenv:

virtualenv testprojenv

Нам нужно активировать виртуальную среду, указав название нашей среды:

s testprojenv/bin/activate

### Создание и настройка проекта Django

Через Filezilla подключаемся к серверу и в нашу папку testproj закидываем наш проект созданный локально

Для запуска созданного проекта необходимо настроить его параметры. Откройте файл настроек в текстовом редакторе, например nano:

nano ~/cards/cards/settings.py

Найдите директиву ALLOWED\_HOSTS и в качестве ее значения укажите IP-адрес или доменное имя сервера:

ALLOWED\_HOSTS = [‘194.62.19.9’]

Затем переместитесь в конец файла и добавьте параметр, указывающий, где должны быть размещены статические файлы. Это необходимо для того, чтобы Nginx мог правильно обрабатывать запросы к этим элементам:

STATIC\_URL =‘/static/’    
STATIC\_ROOT = os.path.join(BASE\_DIR, ‘static/’)

Перенесите исходную схему базы данных в базу данных используя скрипт управления:

~/cards/manage.py makemigrations

Python manage.py migrate –run-syncdb

Создайте административного пользователя для проекта, набрав:

~/cards/manage.py createsuperuser

Соберите весь статический контент в каталог, который был настроен:

~/cards/manage.py collectstatic

С помощью ufw откройте порт 8000 на котором по умолчанию будет запущен Django:

sudo ufw allow 8000

Наконец, вы можете протестировать проект, запустив сервер Django с помощью этой команды:

~/cards/manage.py runserver 0.0.0.0:8000

В любом удобном веб-браузере перейдите на ваше доменное имя или IP-адрес с указанием порта 8000:  
*http://194.62.19.9:8000*

### Тестирование работы Gunicorn

Перед тем, как покинуть виртуальную среду, необходимо протестировать Gunicorn, чтобы убедиться, что программа может обслуживать приложение. Это можно сделать, перейдя в каталог проекта и используя gunicorn для загрузки модуля WSGI:

 ~/cards

Gunicorn –bind 0.0.0.0:8000 cards.wsgi

В результате Gunicorn будет запущен на том же интерфейсе, на котором работал сервер разработки Django. Вы можете вернуться в браузер и снова протестировать приложение.

Теперь настройка приложения Django завершена. Необходимо выйти из виртуальной среды, набрав:

Deactivate

### Создание systemd Socket и Service Files для Gunicorn

Проверка показала, что Gunicorn может взаимодействовать с приложением Django, но необходимо реализовать более надежный способ запуска и остановки сервера приложений. Для этого нужно использовать файл службы systemd.

Создайте и откройте служебный файл systemd для Gunicorn в текстовом редакторе, например nano:

sudo nano /etc/systemd/system/gunicorn.service

Вставьте следующие строки:

[Unit]  
Description=gunicorn daemon  
After=network.target  
  
[Service]  
User=root  
Group=www-data  
WorkingDirectory=/var/www/testproj/cards   
ExecStart=/var/www/testproj/testprojenv/bin/gunicorn --access-logfile - --workers 3 --bind unix:/var/www/testproj cards/cards.sock cards.wsgi:application  
  
[Install]  
WantedBy=multi-user.target

Раздел [Unit] используется для указания метаданных и зависимостей. Здесь находится описание сервиса и информация для системы инициализации. В разделе [Service] определяется пользователь и группа, от имени которых необходимо выполнять работу. Группа www-data используется не случайно, чтобы Nginx мог легко общаться с Gunicorn. Затем в WorkingDirectory указывается рабочий каталог и в ExecStart команда для запуска сервиса. Наконец, раздел [Install] указывает, чтобы служба стартовала, когда обычная многопользовательская система запущена:

Теперь можно запустить созданную службу Gunicorn и включить ее так, чтобы она запускалась при загрузке:

sudo systemctl start gunicorn  
sudo systemctl enable gunicorn

**Проверка Gunicorn Socket file**

Для начала необходимо проверить состояние процесса, чтобы узнать, удалось ли его запустить:

sudo systemctl status gunicorn

Затем проверьте наличие файла cloudproject.sock в каталоге вашего проекта:

ls /var/www/testproj/cards

Output: manage.py cloudproject cloudenv cloudproject.sock static

Если вы не нашли файл cloudproject.sock в каталоге, это означает, что Gunicorn не смог правильно запуститься. Проверьте журналы процесса Gunicorn, набрав:

sudo journalctl -u gunicorn

### Настройка Nginx

Теперь, когда Gunicorn настроен, нужно настроить Nginx для передачи трафика процессу. Создайте и откройте новый файл в каталоге сайтов Nginx:

sudo nano /etc/nginx/sites-available/cards

Внутри создайте блок сервера, вставив следующие строки:

server {  
listen 80;  
server\_name ;  
  
location = /favicon.ico { access\_log off; log\_not\_found off; }  
location /static/ {  
root /var/www/testproj/cards;  
}  
  
location / {  
include proxy\_params;  
proxy\_pass http://unix:/var/www/testproj/cards/cards.sock;  
}  
}

В блоке указано, что nginx должен прослушивать порт 80 и отвечать на доменное имя или IP-адрес сервера. Далее указано игнорировать любые проблемы с поиском значка и путь до статических ресурсов. Последний блок location / {} соответствует всем другим запросам.

Сохраните и закройте файл после внесения изменений. Теперь мы можем включить файл, связав его с каталогом сайтов:

sudo ln -s /etc/nginx/sites-available/cards /etc/nginx/sites-enabled

Далее необходимо проверить конфигурацию Nginx на наличие синтаксических ошибок, набрав:

sudo nginx -t

Если ошибки не найдены, перезапустите Nginx, набрав:

sudo systemctl restart nginx

Наконец, нужно открыть брандмауэр для нормального трафика через порт 80. Поскольку доступ к серверу разработки больше не нужен, нужно удалить правило, которое открывает порт 8000:

sudo ufw delete allow 8000  
sudo ufw allow 'Nginx Full'

В этом руководстве создается проект Django в собственной виртуальной среде. Мы настроили Gunicorn для перевода клиентских запросов, чтобы Django мог их обработать. После этого мы настроили Nginx в качестве обратного прокси-сервера для обработки клиентских подключений и предоставления правильного проекта в зависимости от запроса клиента.

**4. Дополнительные данные**

**Ссылка на гитхаб: https://github.com/Alexsandr007/test\_project\_cards**

**Ссылка на сайт на хостинге: http://194.62.19.9/admin/**

**Логин от админки: admin**

**Пароль от админки: admin**