имам следния код (изглед в django приложение): def home\_view(request): products\_g1 = models.Product.objects.filter(group=models.GROUP1) products\_g2 = models.Product.objects.filter(group=models.GROUP2) products\_g3 = models.Product.objects.filter(group=models.GROUP3) if 'product\_ids' in request.COOKIES: product\_ids = request.COOKIES['product\_ids'] counter = product\_ids.split('|') for i in counter: print(i) product\_count\_in\_cart = len(set(counter)) else: product\_count\_in\_cart = 0 context = {'products\_g1': products\_g1, 'products\_g2': products\_g2, 'products\_g3': products\_g3, 'product\_count\_in\_cart': product\_count\_in\_cart, } return render(request, 'main/index.html', context) # Изглед за добавяне на артикул към количката def add\_to\_cart\_view(request, pk): products\_g1 = models.Product.objects.filter(group=models.GROUP1) products\_g2 = models.Product.objects.filter(group=models.GROUP2) products\_g3 = models.Product.objects.filter(group=models.GROUP3) # за брояча на избраните продукти, изваждам всички id, добавени от клиента, от cookies if 'product\_ids' in request.COOKIES: product\_ids = request.COOKIES['product\_ids'] counter = product\_ids.split('|') product\_count\_in\_cart = len(set(counter)) else: product\_count\_in\_cart = 1 context = {'products\_g1': products\_g1, 'products\_g2': products\_g2, 'products\_g3': products\_g3, 'product\_count\_in\_cart': product\_count\_in\_cart, } response = render(request, 'main/index.html', context) # добавяне на артикул към cookies if 'product\_ids' in request.COOKIES: product\_ids = request.COOKIES['product\_ids'] if product\_ids == "": product\_ids = str(pk) else: product\_ids = product\_ids + "|" + str(pk) response.set\_cookie('product\_ids', product\_ids) else: response.set\_cookie('product\_ids', pk) product = models.Product.objects.get(id=pk) messages.info(request, product.name + ' added to cart successfully!') return response поясни ми го

**GPT-4o**

Кодът предоставя два изгледа (views) в Django приложение:

1. **home\_view**: Отговаря за рендването на началната страница (например списък с продукти).
2. **add\_to\_cart\_view**: Позволява добавяне на продукт към „количката за пазаруване“ и обновява броя на продуктите в нея.

Сега ще разгледам подробно как работи този код:

**1. home\_view**

Този изглед извлича продуктите от база данни, като ги групира по конкретни категории (group), дефинирани в модела. Освен това проверява „количката“, базирана на cookies, и изчислява броя продукти, които са добавени в нея.

**Разглеждане на кода:**

1. **Филтриране на продуктите по групи:**
2. products\_g1 = models.Product.objects.filter(group=models.GROUP1)
3. products\_g2 = models.Product.objects.filter(group=models.GROUP2)
4. products\_g3 = models.Product.objects.filter(group=models.GROUP3)
   * Използват се три QuerySet, които извличат продуктите съответно от три групи (GROUP1, GROUP2, GROUP3). Тези групи предполагам са дефинирани като стойности във вашата база (choices поле в модела Product).
5. **Проверка за съществуването на „количка“ в cookies:**
6. if 'product\_ids' in request.COOKIES:
7. product\_ids = request.COOKIES['product\_ids']
8. counter = product\_ids.split('|')
9. product\_count\_in\_cart = len(set(counter))
10. else:
11. product\_count\_in\_cart = 0
    * Ако в cookies е записан ключът product\_ids, се извличат стойностите на всички продукти в количката.
    * Стойностите се разделят, използвайки символа | като разделител (пример: 1|2|3|4).
    * Чрез set(counter) се премахват повторенията в списъка от ID-та.
    * product\_count\_in\_cart съдържа броя на различните продукти в количката.
    * Ако product\_ids не съществува (т.е. потребителят първо посещава сайта), броят продукти в количката е зададен на нула.
12. **Създаване на контекст и показване на данни в шаблон:**
13. context = {
14. 'products\_g1': products\_g1,
15. 'products\_g2': products\_g2,
16. 'products\_g3': products\_g3,
17. 'product\_count\_in\_cart': product\_count\_in\_cart,
18. }
19. return render(request, 'main/index.html', context)
    * Данните за продуктите и информацията за „количката“ се предават към шаблона main/index.html.

**2. add\_to\_cart\_view**

Този изглед добавя избрания продукт (идентифициран с pk) в „количката“ на потребителя и обновява cookies.

**Разглеждане на кода:**

1. **Извличане на продуктите от база по групи:**  
   Това е същият блок код като в home\_view, използван за извличане на продукти по групи (GROUP1, GROUP2, GROUP3).
2. **Проверка за броя на продуктите в „количката“:**
3. if 'product\_ids' in request.COOKIES:
4. product\_ids = request.COOKIES['product\_ids']
5. counter = product\_ids.split('|')
6. product\_count\_in\_cart = len(set(counter))
7. else:
8. product\_count\_in\_cart = 1
   * Ако потребителят вече има продукти в „количката“ (в cookies), те се извличат и броят се изчислява.
   * Ако няма, броят на продуктите в количката е зададен на 1, тъй като добавяме първия продукт.
9. **Обновяване на „количката“ в cookies:**
10. if 'product\_ids' in request.COOKIES:
11. product\_ids = request.COOKIES['product\_ids']
12. if product\_ids == "":
13. product\_ids = str(pk)
14. else:
15. product\_ids = product\_ids + "|" + str(pk)
16. response.set\_cookie('product\_ids', product\_ids)
17. else:
18. response.set\_cookie('product\_ids', pk)
    * Проверява се дали в cookies съществува product\_ids.
    * Ако съществува:
      + Записът се актуализира, като се добави ново ID към списъка с разделител |.
    * Ако не съществува:
      + За първи път създава нов запис и добавя текущото pk.
19. **Изпращане на съобщение:**
20. product = models.Product.objects.get(id=pk)
21. messages.info(request, product.name + ' added to cart successfully!')
    * Зарежда се продуктът, който е добавен в количката, и се изпраща потвърдително съобщение за потребителя чрез Django messages.
22. **Рендиране на изгледа:**
23. response = render(request, 'main/index.html', context)
24. return response
    * Изгледът отново показва обновената страница с продуктовите категории и броя продукти в количката.

**Как работи „количката“?**

Cookies се използва като временно хранилище за ID-тата на продуктите, които потребителят е „добавил“.

* product\_ids съхранява ID-тата на продуктите в текстов формат, разделени с | (напр. 1|2|3).
* ID-тата се използват за изчисляване на броя продукти в количката.
* „Количката“ не използва база данни, а се съхранява само в браузъра на потребителя.

**Мнения/Оптимизация:**

1. **Повтарящ се код:**  
   Филтрирането на продуктите (по групи) се прави няколко пъти и може да бъде обособено в отделна функция, която да се извиква, за да избегнете дублиране:
2. def get\_products\_by\_group():
3. products\_g1 = models.Product.objects.filter(group=models.GROUP1)
4. products\_g2 = models.Product.objects.filter(group=models.GROUP2)
5. products\_g3 = models.Product.objects.filter(group=models.GROUP3)
6. return products\_g1, products\_g2, products\_g3
7. **Сигурност:**
   * Cookies не са защитен метод за съхранение на информация, тъй като потребителят може лесно да манипулира стойностите. Вместо това е по-добре да използвате sessions или база данни за съхранение на данни.
8. **Премахване на дублиране в количката:**  
   Може да добавите проверка дали даден продукт вече е в количката, преди да го добавите.

Споделете, ако възникнат още въпроси! 😊



**Ти**

как да използвам session

**GPT-4o**

Използването на сесии (sessions) в Django е по-сигурен и по-добър начин за съхранение на данни като „количка за пазаруване“. Сесиите позволяват данните да се съхраняват на сървъра, докато в браузъра на потребителя се съхранява само идентификатор на сесията (cookie).

Ето как можете да използвате session, за да реализирате количката във вашия случай:

**1. Конфигуриране на сесиите в Django**

Django сесиите обикновено са активирани по подразбиране. Уверете се, че в настройките ви (settings.py) имате следните редове:

# Бекенд за сесии (по подразбиране използва база данни)

SESSION\_ENGINE = 'django.contrib.sessions.backends.db' # Съхранява сесиите в база данни

# Срок на годност на сесиите (по избор)

SESSION\_COOKIE\_AGE = 1209600 # 2 седмици, в секунди

SESSION\_SAVE\_EVERY\_REQUEST = True # Съхранява сесията при всяка заявка

**2. Използване на сесии в изгледите**

**Пример за home\_view:**

Тук ще съхранявате информацията за продуктите в session вместо cookies, като използвате списък от ID-та.

def home\_view(request):

# Извличане на продукти по групи от базата

products\_g1 = models.Product.objects.filter(group=models.GROUP1)

products\_g2 = models.Product.objects.filter(group=models.GROUP2)

products\_g3 = models.Product.objects.filter(group=models.GROUP3)

# Проверка за продукти в количката (съхранява се в сесия)

cart = request.session.get('cart', []) # Ако няма "cart", връща празен списък

product\_count\_in\_cart = len(set(cart)) # Изчисляване на уникалните продукти в количката

# Подготовка на контекст за рендиране

context = {

'products\_g1': products\_g1,

'products\_g2': products\_g2,

'products\_g3': products\_g3,

'product\_count\_in\_cart': product\_count\_in\_cart,

}

return render(request, 'main/index.html', context)

**Пример за add\_to\_cart\_view:**

Тук добавяме избрания продукт към „количката“, съхранявана в сесията.

def add\_to\_cart\_view(request, pk):

# Проверка дали съществува количка в сесията

cart = request.session.get('cart', []) # Ако няма "cart", връща празен списък

cart.append(pk) # Добавяме ID на продукта в количката

request.session['cart'] = cart # Обновяваме "cart" в сесията

# Извличане на продукти по групи (за показване)

products\_g1 = models.Product.objects.filter(group=models.GROUP1)

products\_g2 = models.Product.objects.filter(group=models.GROUP2)

products\_g3 = models.Product.objects.filter(group=models.GROUP3)

# Обновяване на броя на продуктите в количката

product\_count\_in\_cart = len(set(request.session['cart']))

# Подготовка на контекст за рендиране

context = {

'products\_g1': products\_g1,

'products\_g2': products\_g2,

'products\_g3': products\_g3,

'product\_count\_in\_cart': product\_count\_in\_cart,

}

# Съобщение за потвърждение

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

messages.info(request, f"{product.name} was added to the cart successfully!")

return render(request, 'main/index.html', context)

**Пример за изглед за изчистване на количката (по избор):**

Може би искате бутон или функционалност за изчистване на количката. Ето пример как да я изтриете от сесията:

def clear\_cart\_view(request):

if 'cart' in request.session:

del request.session['cart'] # Премахване на данните за количката от сесията

messages.info(request, "The cart has been cleared.")

return redirect('home') # Пренасочване към началната страница

**3. Обяснения**

* **request.session**:
  + Позволява съхраняването и достъпа до данни, специфични за сесията на даден потребител.
  + Данните в сесията се съхраняват като обикновен Python речник. Например:
  + request.session['ключ'] = стойност
  + Данните остават валидни, докато сесията е активна или не бъдат изтрити.
* **Количката (cart)**:
  + В примера количката е обикновен списък от ID-та на продукти (напр. [1, 2, 3]).
  + set(cart) премахва дублиращите се ID-та на продуктите.
* **Добавяне към сесията**:
  + Когато добавяте към количката, се извлича текущата стойност на cart и към нея се добавя нов продукт.
* **Изчистване или модифициране**:
  + Можете да премахвате продукти или изцяло да изчиствате количката със стандартни Python операции върху списъци/речници.

**4. Допълнителна оптимизация**

* Ако желаете да съхранявате повече информация (например количество продукти или детайли за продуктите), вместо списък от ID-та, може да използвате речник като:
* request.session['cart'] = {
* 1: {'quantity': 2, 'price': 19.99}, # Продукт с ID 1
* 2: {'quantity': 1, 'price': 15.50} # Продукт с ID 2
* }
* След това може да обработвате навсякъде данните по същия начин.

Това решение е по-сигурно, защото сесиите се съхраняват на сървъра и не могат да бъдат манипулирани директно от потребителя, както е при cookies. Ако имате нужда от допълнителна помощ или други примери, не се колебайте да питате! 😊



**Ти**

а ако използваме база данни как би изглеждал кода

**GPT-4o**

Ако използваме база данни за съхранение на информацията за количката, това е по-сигурен, устойчив и често срещан подход в приложението за електронна търговия. Ще трябва да създадем модел за „количка“ и да модифицираме изгледите, за да работят с този модел.

**1. Модели (Models)**

**Създаване на модел за количката:**

Ще създадем два модела:

* **Cart**: Представлява самата количка (свързана с потребител, ако е необходимо).
* **CartItem**: Представлява индивидуален продукт в количката.

Пример:

from django.db import models

from django.contrib.auth.models import User # Ако използвате потребители в приложението

class Cart(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE, null=True, blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return f"Cart ({self.user.username if self.user else 'Anonymous'})"

class CartItem(models.Model):

cart = models.ForeignKey(Cart, related\_name='items', on\_delete=models.CASCADE)

product = models.ForeignKey('Product', on\_delete=models.CASCADE) # Връзка с модела Product

quantity = models.PositiveIntegerField(default=1)

def \_\_str\_\_(self):

return f"{self.product.name} ({self.quantity})"

* **Cart**:
  + Връзка с потребител: *Anonymous* потребител може също да има количка (без потребител).
  + Количката съдържа много продукти чрез CartItem (много към едно).
* **CartItem**:
  + Връзка към конкретната количка чрез ForeignKey.
  + Връзка към Product модела.
  + Съхранява информация за броя (quantity) на дадения продукт.

**2. Изгледи (Views)**

След създаването на моделите, ще трябва да променим изгледите си, така че да използват тези модели.

**1. home\_view:**

Тук просто показваме продуктите и броя продукти в количката, като информацията за количката идва от базата данни.

def home\_view(request):

# Търсене или създаване на количка за текущия потребител

if request.user.is\_authenticated:

cart, created = Cart.objects.get\_or\_create(user=request.user)

else:

cart = None

products\_g1 = models.Product.objects.filter(group=models.GROUP1)

products\_g2 = models.Product.objects.filter(group=models.GROUP2)

products\_g3 = models.Product.objects.filter(group=models.GROUP3)

# Извличаме броя артикули в количката

if cart:

product\_count\_in\_cart = cart.items.count()

else:

product\_count\_in\_cart = 0

context = {

'products\_g1': products\_g1,

'products\_g2': products\_g2,

'products\_g3': products\_g3,

'product\_count\_in\_cart': product\_count\_in\_cart,

}

return render(request, 'main/index.html', context)

**2. add\_to\_cart\_view:**

Добавяне на продукт към „количката“.

def add\_to\_cart\_view(request, pk):

# Проверка дали потребителят има количка, ако не - създайте нова

if request.user.is\_authenticated:

cart, created = Cart.objects.get\_or\_create(user=request.user)

else:

return redirect('login') # Пренасочете анонимния потребител към страницата за вход

# Намиране на продукта

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

# Добавяне на продукта към количката

cart\_item, created = CartItem.objects.get\_or\_create(cart=cart, product=product)

if not created:

# Ако продуктът вече е в количката, увеличаваме неговото количество

cart\_item.quantity += 1

cart\_item.save()

messages.info(request, f"{product.name} was added to the cart successfully!")

return redirect('home') # Пренасочване към началната страница

**3. Изглед за показване на количката:**

Може би ще искате и специална страница, на която да покажете всички продукти в количката.

def cart\_view(request):

# Извличане на количката на текущия потребител

if request.user.is\_authenticated:

cart = Cart.objects.filter(user=request.user).first()

items = cart.items.all() if cart else []

else:

items = [] # Ако няма потребител, няма количка

context = {

'items': items,

}

return render(request, 'main/cart.html', context)

**4. Изглед за изчистване на количката:**

Изчистване на всички продукти в количката.

def clear\_cart\_view(request):

if request.user.is\_authenticated:

cart = Cart.objects.filter(user=request.user).first()

if cart:

cart.items.all().delete()

messages.info(request, "The cart has been cleared.")

return redirect('cart') # Пренасочване към страницата с количката

**3. Шаблони (Templates)**

**Шаблон за начална страница (main/index.html):**

<h1>Product Groups</h1>

<p>Products in cart: {{ product\_count\_in\_cart }}</p>

<h2>Group 1</h2>

<ul>

{% for product in products\_g1 %}

<li>

{{ product.name }} - {{ product.price }}

<a href="{% url 'add\_to\_cart' product.id %}">Add to Cart</a>

</li>

{% endfor %}

</ul>

**Шаблон за количка (main/cart.html):**

<h1>Your Cart</h1>

<ul>

{% for item in items %}

<li>

{{ item.product.name }} - Quantity: {{ item.quantity }} - Total: {{ item.quantity|multiply:item.product.price }}

</li>

{% endfor %}

</ul>

<a href="{% url 'clear\_cart' %}">Clear Cart</a>

**4. Администраторски панел (admin.py)**

Добавете новосъздадените модели в администрацията.

from django.contrib import admin

from .models import Cart, CartItem

@admin.register(Cart)

class CartAdmin(admin.ModelAdmin):

list\_display = ('user', 'created\_at', 'updated\_at')

@admin.register(CartItem)

class CartItemAdmin(admin.ModelAdmin):

list\_display = ('cart', 'product', 'quantity')

**5. Миграции**

Не забравяйте да създадете и приложите миграции за новите модели.

python manage.py makemigrations

python manage.py migrate

**6. Ползи от използването на база данни**

* **Сигурност**: Информацията за количката се съхранява на сървъра.
* **Устойчивост**: Данните за количката могат да се запазват дори след затваряне на браузъра.
* **Възможност за разширение**: Лесно можете да добавите нови функционалности като управление на количество, промоции и бърза поръчка.

Тази структура е подходяща за сериозно приложение, защото предоставя повече контрол и възможности. Кажете, ако трябва да добавим още нещо! 😊



**Ти**

това ми харесва - подходящо е за моя случай, но искам да може да се ползва и от анонимен потребител - т.е. да не се налага потребителят явно да се идентифицира. как би могло да стане това

**GPT-4o**

За да добавите функционалност, която позволява на анонимни потребители да използват количката, можете да добавите уникален идентификатор за сесията на всеки анонимен потребител. Това ще позволи създаването на отделна количка за всеки потребител, без да е необходимо той да се идентифицира чрез логин.

Подходът е следният:

**Промяна в моделите**

Ще добавим допълнително поле към модела Cart за съхраняване на уникален идентификатор за количката на анонимни потребители.

import uuid # Генерира уникални идентификатори

class Cart(models.Model):

user = models.OneToOneField(

User, on\_delete=models.CASCADE, null=True, blank=True

) # Връзка с потребител (по желание)

session\_id = models.CharField(

max\_length=40, null=True, blank=True

) # За анонимни потребители

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

if self.user:

return f"Cart for {self.user.username}"

else:

return f"Cart for session {self.session\_id}"

**Какво прави:**

1. Ако потребителят е идентифициран (user), количката се свързва с този потребител.
2. Ако потребителят е **анонимен**, количката се свързва със session\_id, което е уникално ID за текущата сесия.

**Изменения в изгледите**

Сега ще променим изгледите, за да поддържат анонимни потребители.

**1. home\_view: Добавяне на проверка за сесия**

Проверяваме дали потребителят е логнат или не, и съответно намираме/създаваме количката.

def home\_view(request):

# Търсене или създаване на количка за текущия потребител или сесия

if request.user.is\_authenticated:

cart, created = Cart.objects.get\_or\_create(user=request.user)

else:

# Уникален идентификатор на сесията за анонимния потребител

session\_id = request.session.session\_key

if not session\_id:

request.session.create() # Създаване на нова сесия, ако няма

session\_id = request.session.session\_key

cart, created = Cart.objects.get\_or\_create(session\_id=session\_id)

# Извличане на продуктите

products\_g1 = models.Product.objects.filter(group=models.GROUP1)

products\_g2 = models.Product.objects.filter(group=models.GROUP2)

products\_g3 = models.Product.objects.filter(group=models.GROUP3)

# Изчисляване на броя продукти в количката

product\_count\_in\_cart = cart.items.count()

# Подготовка на контекст за изгледа

context = {

'products\_g1': products\_g1,

'products\_g2': products\_g2,

'products\_g3': products\_g3,

'product\_count\_in\_cart': product\_count\_in\_cart,

}

return render(request, 'main/index.html', context)

**2. add\_to\_cart\_view: Връзка с количка за анонимен потребител**

Добавяме продукт към количката в базата данни, като вземаме предвид дали потребителят е логнат.

def add\_to\_cart\_view(request, pk):

# Проверка за потребител или сесия

if request.user.is\_authenticated:

cart, created = Cart.objects.get\_or\_create(user=request.user)

else:

session\_id = request.session.session\_key

if not session\_id:

request.session.create() # Създаване на нова сесия, ако няма

session\_id = request.session.session\_key

cart, created = Cart.objects.get\_or\_create(session\_id=session\_id)

# Намерете продукта, който ще се добави

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

# Добавяме продукта в количката

cart\_item, created = CartItem.objects.get\_or\_create(cart=cart, product=product)

if not created:

cart\_item.quantity += 1

cart\_item.save()

messages.info(request, f"{product.name} was added to the cart successfully!")

return redirect('home') # Пренасочване към началната страница

**3. cart\_view: Показване на количката**

Събират се продуктите в количката независимо дали потребителят е логнат.

def cart\_view(request):

# Намиране на количката за логнат или анонимен потребител

if request.user.is\_authenticated:

cart = Cart.objects.filter(user=request.user).first()

else:

session\_id = request.session.session\_key

if session\_id:

cart = Cart.objects.filter(session\_id=session\_id).first()

else:

cart = None

items = cart.items.all() if cart else []

context = {

'items': items,

}

return render(request, 'main/cart.html', context)

**4. clear\_cart\_view: Изчистване на количката**

Позволете да се изтрие количката както за логнат, така и за анонимен потребител.

def clear\_cart\_view(request):

if request.user.is\_authenticated:

cart = Cart.objects.filter(user=request.user).first()

else:

session\_id = request.session.session\_key

cart = Cart.objects.filter(session\_id=session\_id).first()

if cart:

cart.items.all().delete()

messages.info(request, "The cart has been cleared.")

return redirect('cart') # Пренасочване към страницата с количката

**Допълнения в settings.py**

Уверете се, че сесиите са правилно конфигурирани в settings.py. По подразбиране Django работи правилно със сесии, но ето ключовите настройки:

# Активиране на сесиите

SESSION\_ENGINE = 'django.contrib.sessions.backends.db' # Сесиите се съхраняват в базата данни

# Срок на валидност на сесията

SESSION\_COOKIE\_AGE = 1209600 # 2 седмици (в секунди)

# Сесията ще се пази в браузъра дори ако не е записана информация

SESSION\_SAVE\_EVERY\_REQUEST = True

**Как работи механизмът със сесии?**

* **Логнат потребител:**
  + Полето user в модела Cart свързва количката директно с потребителя.
  + Всичко, добавено в количката, се запазва за потребителя.
* **Анонимен потребител:**
  + Ако потребителят не е логнат, Django използва сесията за свързване. session\_id е уникален идентификатор за текущата сесия.
  + Сесията се идентифицира чрез cookie, запазено в браузъра на потребителя.
  + Ако потребителят затвори браузъра, количката ще се изчисти, освен ако SESSION\_COOKIE\_AGE не е променен.
* **Преминаване от анонимен към логнат потребител:**  
  При логване на анонимен потребител с пълноправен акаунт можете да преобразувате анонимната количка:

if request.user.is\_authenticated and request.session.session\_key:

session\_cart = Cart.objects.filter(session\_id=request.session.session\_key).first()

user\_cart, created = Cart.objects.get\_or\_create(user=request.user)

if session\_cart:

# Прехвърляне на продуктите от анонимната количка в потребителската количка

for item in session\_cart.items.all():

user\_cart\_item, created = CartItem.objects.get\_or\_create(

cart=user\_cart, product=item.product

)

if not created:

user\_cart\_item.quantity += item.quantity

user\_cart\_item.save()

session\_cart.delete() # Изтриване на анонимната количка

**Заключение**

С този подход анонимните потребители могат да добавят продукти към количката, без да се логват, и същевременно количката за логнати потребители остава напълно функционална. Това е добре интегрирано решение за гъвкаво управление на количката. Ако имате допълнителни въпроси или искате още оптимизации, пишете ми! 😊

Сега трябва да генерираме още поне две страници за завършване на поръчката. трябва да използваме форми. Едната форма трябва да бъде ModelForm, а другата просто Form. Трябва да покажем и как се използва валидиране на данни в django.

**GPT-4o**

Разбира се! Ще добавим две страници за завършване на поръчката, като използваме **ModelForm** за една от формите и **Form** за другата. Ще добавим също валидация на данни и ще покажем как работи тя в Django.

**Анализ и добавяне на нови функционалности**

**1. Актуализиране на models.py**

Първо, ще актуализираме моделите, за да добавим данни, свързани с поръчката. Ще добавим модел Order и OrderItem, за да съхраняваме информация за поръчката, включително асоциирани продукти.

**Актуализиран models.py:**

from django.db import models

from django.contrib.auth.models import User

from uuid import uuid4

class Product(models.Model):

name = models.CharField('Наименование', max\_length=40)

group = models.PositiveSmallIntegerField('Продуктова група', choices=PRODUCT\_TYPE, default=GROUP1)

product\_image = models.ImageField('Снимка', upload\_to='product\_image', null=True, blank=True)

price = models.DecimalField('Цена', max\_digits=10, decimal\_places=2, default=0.00)

description = models.TextField('Описание', default='', blank=True)

def \_\_str\_\_(self):

return self.name

class Order(models.Model):

user = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

# Полета за потребителски данни

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField()

phone = models.CharField(max\_length=20)

address = models.TextField()

notes = models.TextField(blank=True, default='')

def \_\_str\_\_(self):

return f"Order {self.id} by {self.user}" if self.user else f"Order {self.id} (Anonymous)"

class OrderItem(models.Model):

order = models.ForeignKey(Order, related\_name="items", on\_delete=models.CASCADE)

product = models.ForeignKey(Product, on\_delete=models.CASCADE)

quantity = models.PositiveIntegerField()

def \_\_str\_\_(self):

return f"{self.product.name} - {self.quantity}"

def get\_total(self):

return self.product.price \* self.quantity

**Обяснение:**

* **Order**: Съхранява данни за потребителя и поръчката като цяло.
* **OrderItem**: Съхранява продукти и количества, свързани с дадена поръчка.

**2. Създаване на форми**

В папката на приложението създайте файл forms.py, където ще дефинираме двете форми:

* **ModelForm** за данните за поръчката (модел Order),
* **Form** за допълнителна информация, например бележки към поръчката.

**forms.py:**

from django import forms

from .models import Order

class OrderForm(forms.ModelForm):

"""

ModelForm за събиране на информация за потребителя и адреса.

"""

class Meta:

model = Order

fields = ['first\_name', 'last\_name', 'email', 'phone', 'address']

widgets = {

'address': forms.Textarea(attrs={'rows': 3}),

}

# Добавяне на метод за валидация на имейл

def clean\_email(self):

email = self.cleaned\_data.get('email')

if not email.endswith('@example.com'): # Примерна валидация

raise forms.ValidationError("Имейлът трябва да бъде в домейна example.com")

return email

class NotesForm(forms.Form):

"""

Form за допълнителни данни (бележки).

"""

notes = forms.CharField(

label="Бележки към поръчката",

widget=forms.Textarea(attrs={'rows': 3}),

max\_length=500,

required=False

)

# Пример за потребителска валидация

def clean\_notes(self):

notes = self.cleaned\_data.get('notes')

if len(notes) > 500:

raise forms.ValidationError("Бележките не могат да съдържат повече от 500 символа")

return notes

**3. Актуализиране на views.py**

Ще добавим изгледи за:

* Попълване на основна информация за поръчката с OrderForm.
* Добавяне на бележки или потвърждение с NotesForm.
* Записване на поръчката в базата данни.

**Актуализиран views.py:**

from django.shortcuts import render, redirect

from .models import Cart, Order, OrderItem

from .forms import OrderForm, NotesForm

from django.contrib import messages

def checkout\_step\_one(request):

"""

Първа стъпка: Попълване на данни за поръчка

"""

cart = Cart.objects.filter(user=request.user).first() # Получаваме количката на потребителя

if not cart or not cart.items.exists():

messages.error(request, "Количката е празна!")

return redirect('cart')

if request.method == 'POST':

form = OrderForm(request.POST)

if form.is\_valid():

# Запазваме данните временно в сесията

request.session['order\_data'] = form.cleaned\_data # Съхраняваме полетата от формата

return redirect('checkout\_step\_two') # Към следващата стъпка

else:

form = OrderForm()

return render(request, 'main/checkout\_step\_one.html', {'form': form})

def checkout\_step\_two(request):

"""

Втора стъпка: Добавяне на бележки и финализиране

"""

order\_data = request.session.get('order\_data') # Извличаме данните от предходната стъпка

if not order\_data:

return redirect('checkout\_step\_one') # Ако няма данни, връщаме се към 1-ва стъпка

cart = Cart.objects.filter(user=request.user).first()

if request.method == 'POST':

form = NotesForm(request.POST)

if form.is\_valid():

# Създаваме поръчка в базата данни

order = Order.objects.create(

user=request.user if request.user.is\_authenticated else None,

\*\*order\_data,

notes=form.cleaned\_data['notes']

)

# Добавяме продуктите към поръчката

for item in cart.items.all():

OrderItem.objects.create(

order=order,

product=item.product,

quantity=item.quantity

)

cart.items.all().delete() # Изчистваме количката

messages.success(request, "Вашата поръчка беше завършена успешно!")

return redirect('home')

else:

form = NotesForm()

return render(request, 'main/checkout\_step\_two.html', {'form': form, 'cart': cart})

**4. Актуализиране на urls.py**

Добавете маршрути за двете стъпки в процеса на завършване на поръчката.

from django.urls import path

from .views import checkout\_step\_one, checkout\_step\_two

urlpatterns = [

path('checkout/step-one/', checkout\_step\_one, name='checkout\_step\_one'),

path('checkout/step-two/', checkout\_step\_two, name='checkout\_step\_two'),

]

**5. Шаблони**

**Шаблон за стъпка 1 (checkout\_step\_one.html):**

<h1>Завършване на поръчката - Стъпка 1</h1>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Продължи</button>

</form>

**Шаблон за стъпка 2 (checkout\_step\_two.html):**

<h1>Завършване на поръчката - Стъпка 2</h1>

<h3>Обща сума: {{ cart.items.all|get\_total }}</h3>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Финализирай поръчката</button>

</form>

**6. Тестване**

1. Уверете се, че всички миграции са приложени:
2. python manage.py makemigrations
3. python manage.py migrate
4. Тествайте процеса на попълване на данни за поръчката.