

Projekt - Sklep winylowy

Jan Ciężkowski, Dawid Kardacz, Jakub Pilch

Spis treści

Wstęp	3
Funkcje systemu	3
Schemat bazy	4
Konfiguracja Django i struktura projektu	7
Połączenie projektu z bazą danych	8
Routing i dostęp do danych	9
Widok strony głównej	11
Widok wyszukiwania	13
Widok panelu admina	15
Widok dodawania płyty	17
Widok rejestracji dostawy	19
Widok statystyk	21
Widok listy winyli	23
Widok kupowania	26
Widok zakupu	28

Wstęp

Projekt został przygotowany przy użyciu:

- SQLite
- Django
- Python 3

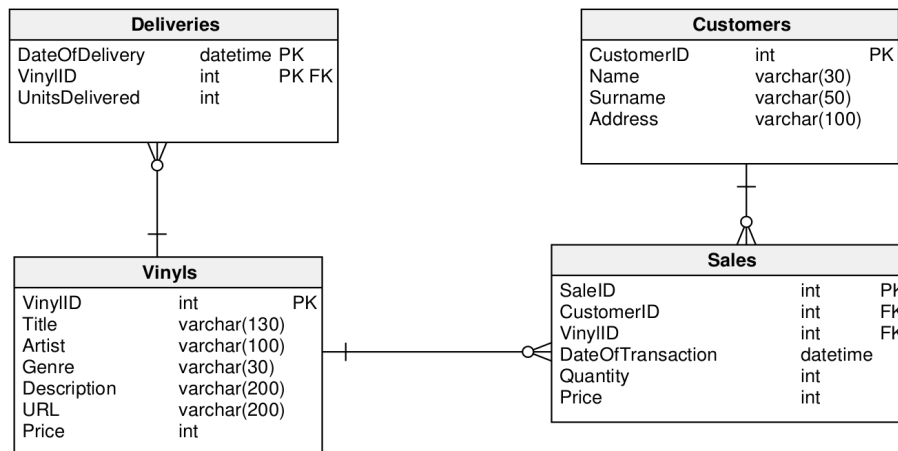
Tematem projektu jest sklep internetowy z płytami winylowymi - “*Vinylo*”.

Funkcje systemu

Funkcjonalności:

- Klient:
 - Zakup płyt
 - Przeglądanie wszystkich dostępnych płyt wraz z opcją wyszukiwania
- Admin:
 - Wyświetlanie statystyk dotyczących sprzedaży
 - Zarejestrowanie nowej płyty winylowej w bazie
 - Zarejestrowanie nowej dostawy w bazie

Schemat bazy danych



Schemat został przygotowany w Vertabelo. Następnie, również przy użyciu Vertabelo, wygenerowany został kod SQL (plik baza_create.sql). Zawartość pliku:

```
-- Table: Customers
CREATE TABLE Customers (
    CustomerID int NOT NULL CONSTRAINT Customers_pk PRIMARY KEY,
    Name varchar(30) NOT NULL,
    Surname varchar(50) NOT NULL,
    Address varchar(100) NOT NULL
);

-- Table: Deliveries
CREATE TABLE Deliveries (
    DateOfDelivery datetime NOT NULL,
    VinylID int NOT NULL,
    UnitsDelivered int NOT NULL,
    CONSTRAINT Deliveries_pk PRIMARY KEY (DateOfDelivery,VinylID),
    CONSTRAINT Deliveries_Vinyls FOREIGN KEY (VinylID)
    REFERENCES Vinyls (VinylID)
);

-- Table: Sales
CREATE TABLE Sales (
    SaleID int NOT NULL CONSTRAINT Sales_pk PRIMARY KEY,
    CustomerID int NOT NULL,
    VinylID int NOT NULL,
    DateOfTransaction datetime NOT NULL,
```

```

Quantity int NOT NULL,
Price int NOT NULL,
CONSTRAINT Sales_Vinyls FOREIGN KEY (VinylID)
REFERENCES Vinyls (VinylID),
CONSTRAINT Sales_Customers FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID)
);

-- Table: Vinyls
CREATE TABLE Vinyls (
    VinylID int NOT NULL CONSTRAINT Vinyls_pk PRIMARY KEY,
    Title varchar(130) NOT NULL,
    Artist varchar(100) NOT NULL,
    Genre varchar(30) NOT NULL,
    Description varchar(200) NOT NULL,
    URL varchar(200) NOT NULL,
    Price int NOT NULL
);

```

Kolejnym krokiem było utworzenie bazy danych. Do czego użyliśmy SQLite:

```
> sqlite3 VinylShop
```

Baza została utworzona w pliku VinylShop.db. Następnie po połączeniu się z bazą danych przy użyciu DataGripa oraz wcześniej wygenerowanego kodu utworzyliśmy potrzebne tabele.

Opis tabel

1. Tabela Vinyls

- VinylID - klucz główny, ID płyty
- Title - tytuł płyty
- Artist - nazwa artysty
- Genre - gatunek muzyczny
- Description - opis płyty
- URL - link do okładki (do pliku)
- Price - cena za sztukę

Uwaga: Ilość dostępnych sztuk jest obliczana na podstawie Sales oraz Deliveries

2. Tabela Sales

- SaleID - klucz główny, ID transakcji

- CustomerID - ID klienta dokonującego zakupu
- VinylID - ID kupowanej płyty
- DateOfTransaction - data transakcji
- Qunatity - ilość kupowanych płyt
- Price - cena za płytę w momencie dokonania transakcji

3. Tabela Deliveries

- DateOfDelivery - data dostawy
- VinylID - ID dostarczonej płyty
- UnitsDelieverd - ilość dostarczonych sztuk

4. Tabela Customers

- CustomerID - klucz główny, ID klienta
- Name - Imie
- Surname - Nazwisko
- Address - Adres

Konfiguracja Django i struktura projektu

Pierwszym krokiem była instalacja Django za pomocą:

```
python3 -m pip install Django
```

Utworzenie projektu:

```
django-admin startproject vinyl_shop
```

W ramach projektu utworzyliśmy naszą aplikację:

```
python3 manage.py startapp shop
```

W pliku settings.py w sekcji INSTALLED_APPS dodajemy naszą aplikację 'shop'. A w pliku vinyl_shop/urls.py dodajemy:

```
path('', include('shop.urls'))
```

Co odpowiada za przekierowywanie routingu do routingu aplikacji shop

W celu uruchomienia serwera wykonujemy polecenie:

```
python3 manage.py runserver
```

Struktura projektu:

vinyl_shop/	<- Główny folder projektu
manage.py	<- Plik zarządzający projektem i serwerem
VinylShop.db	<- Plik z bazą danych
vinyl_shop/	<- folder z plikami dotyczącymi samego projektu
__init__.py	
asgi.py	
settings.py	<- plik z ustawieniami projektu
urls.py	<- routing projektu
wsgi.py	
shop/	<- folder aplikacji
migrations/	<- migracje bazy danych
static/	<- folder z plikami statycznymi (img, css itp)
templates/	<- folder z plikami html
templatetags/	<- folder z tagami używanymi w templates
__init__.py	
admin.py	
apps.py	
models.py	<- plik zawierający definicje modelu bazy danych
tests.py	
views.py	<- plik z definicjami widoków i przetwarzaniem danych do nich
urls.py	<- routing aplikacji

Połączenie bazy z aplikacją

Żeby aplikacja wiedziała, żeby korzystać z naszej bazy danych musimy w pliku settings.py ustawić wskazanie na nasz plik. Wygląda to następująco:

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

Musimy też zadbać, aby model znajdujący się w shop/models.py odpowiadał strukturze naszej bazy. Do tego użyliśmy polecenia:

```
python manage.py inspectdb > models.py
```

Do pliku models.py został wygenerowany następujący kod:

```
from django.db import models

class Customers(models.Model):
    customerid = models.AutoField(db_column='CustomerID', primary_key=True)
    name = models.CharField(db_column='Name', max_length=30)
    surname = models.CharField(db_column='Surname', max_length=50)
    address = models.CharField(db_column='Address', max_length=100)

    class Meta:
        managed = False
        db_table = 'Customers'

class Deliveries(models.Model):
    dateofdelivery = models.DateTimeField(db_column='DateOfDelivery', primary_key=True)
    vinylid = models.ForeignKey('Vinyls', models.DO_NOTHING, db_column='VinylID')
    unitsdelivered = models.IntegerField(db_column='UnitsDelivered')

    class Meta:
        managed = False
        db_table = 'Deliveries'

class Sales(models.Model):
    saleid = models.AutoField(db_column='SaleID', primary_key=True)
    customerid = models.ForeignKey(Customers, models.DO_NOTHING, db_column='CustomerID')
    vinylid = models.ForeignKey('Vinyls', models.DO_NOTHING, db_column='VinylID')
    dateoftransaction = models.DateTimeField(db_column='DateOfTransaction')
    quantity = models.IntegerField(db_column='Quantity')
    price = models.IntegerField(db_column='Price')
```



```

class Meta:
    managed = False
    db_table = 'Sales'

class Vinyls(models.Model):
    vinylid = models.AutoField(db_column='VinylID', primary_key=True)
    title = models.CharField(db_column='Title', max_length=130)
    artist = models.CharField(db_column='Artist', max_length=100)
    genre = models.CharField(db_column='Genre', max_length=30)
    description = models.CharField(db_column='Description', max_length=200)
    url = models.CharField(db_column='URL', max_length=200)
    price = models.IntegerField(db_column='Price')

    class Meta:
        managed = False
        db_table = 'Vinyls'

```

Routing i dostęp do danych

Każda ze stron naszej aplikacji zdefiniowana jest w pliku shop/urls.py
Przykładowa struktura pliku:

```

urlpatterns = [
    path('vinyls/', views.allVinyls, name='allVinyls'),
    path('', views.mainPage, name='mainPage'),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

Każdy z routingów składa się z adresu, odnośnika do funkcji w pliku views.py oraz z nazwy.

Dodatkowo dołączony jest folder static, który możemy zdefiniować w pliku settings.py:

```

STATIC_ROOT = os.path.join(BASE_DIR, 'shop', 'static')
STATIC_URL = '/static/'

```

Przykładowa funkcja z pliku views - np mainPage:

```

def mainPage(request):
    template = loader.get_template("main.html")
    mainVinyls = Vinyls.objects.all().values()
    if len(mainVinyls) > 4:
        mainVinyls = random.sample(list(mainVinyls), 4)
    context = {
        'mainVinyls': mainVinyls,
    }
    return HttpResponse(template.render(context, request))

```

Najpierw z folderu templates jest wczytywany odpowiedni plik .html, który wygeneruje stronę internetową.

Jeśli strona korzysta z jakichś danych to przesyłamy je w zmiennej context, gdzie:

```
'nazwa_zmiennej_w_template' : nazwa_zmiennej_w_views
```

Same dane pobieramy z bazy danych, w tym przypadku pobieramy wartości wszystkich obiektów z kolekcji Vinyls, z których potem wybieramy losowo maksymalnie 4.

```
Vinyls.objects.all().values()
```

Jest równoznaczne z

```
SELECT * FROM Vinyls
```

Po przesłaniu danych do pliku .html możemy skorzystać z tagów udostępnionych przez Django

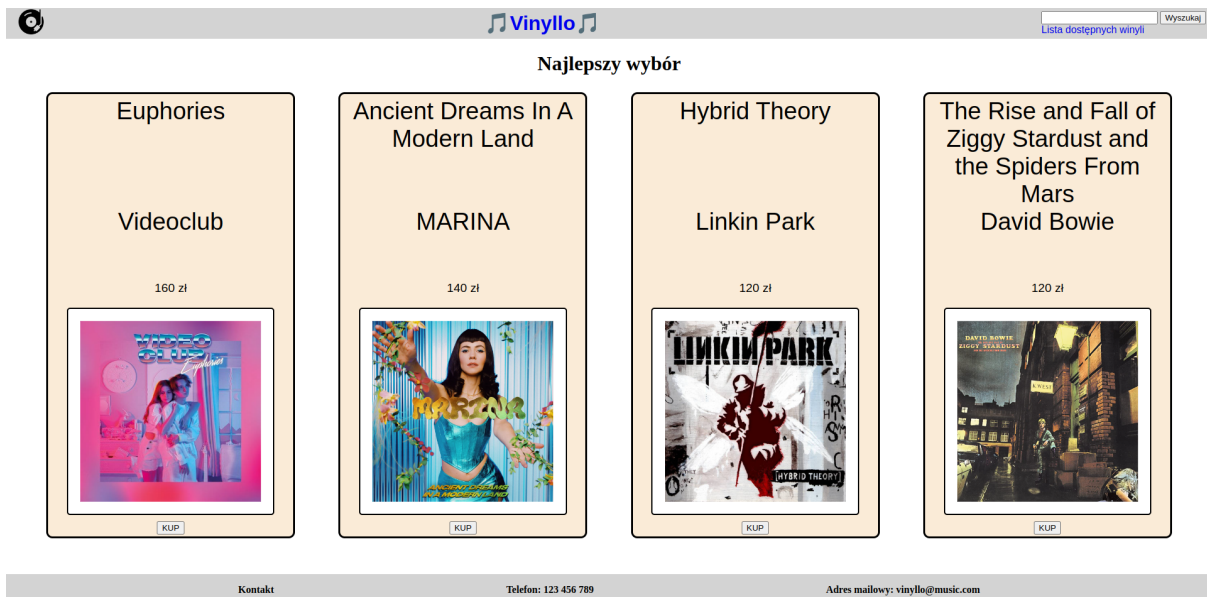
```
{% for vinyl in mainVinyls %}
    <div class="vinylContainer">
        <p>{{ vinyl.title }}</p>
        <p>{{ vinyl.artist }}</p>
        <p>{{ vinyl.price }} zł</p>
        
        <form method="GET" action="/basket">
            <input type="checkbox" name="buy" value="{{ vinyl.vinylid }}" style="display: none;" checked>
            <button type="submit">KUP</button>
        </form>
    </div>
{% endfor %}
```

Tutaj tagi for oraz endfor pozwalające iterować po obiektach, a za pomocą “.” możemy dostać się do odpowiedniego atrybutu w obiekcie. Dodatkowo używamy tagu static który pozwala na pobranie zdjęcia z odpowiedniego folderu/pliku z folderu static. “add” służy temu aby skontatenować ‘img’ oraz .url .Natomiast żeby z niego skorzystać musimy dodać

```
{% load static %}
```

w pliku html.

Widok strony głównej



W centrum znajdują się 4 losowe wybrane płyty wraz z podstawowymi informacjami oraz możliwością zakupu. Definicja funkcji w pliku views:

```
def mainPage(request):
    template = loader.get_template("main.html")
    mainVinyls = Vinyls.objects.all().values()
    if len(mainVinyls) > 4:
        mainVinyls = random.sample(list(mainVinyls), 4)
    context = {
        'mainVinyls': mainVinyls,
    }
    return HttpResponse(template.render(context, request))
```

(Opis działania przedstawiony na stronie 10)

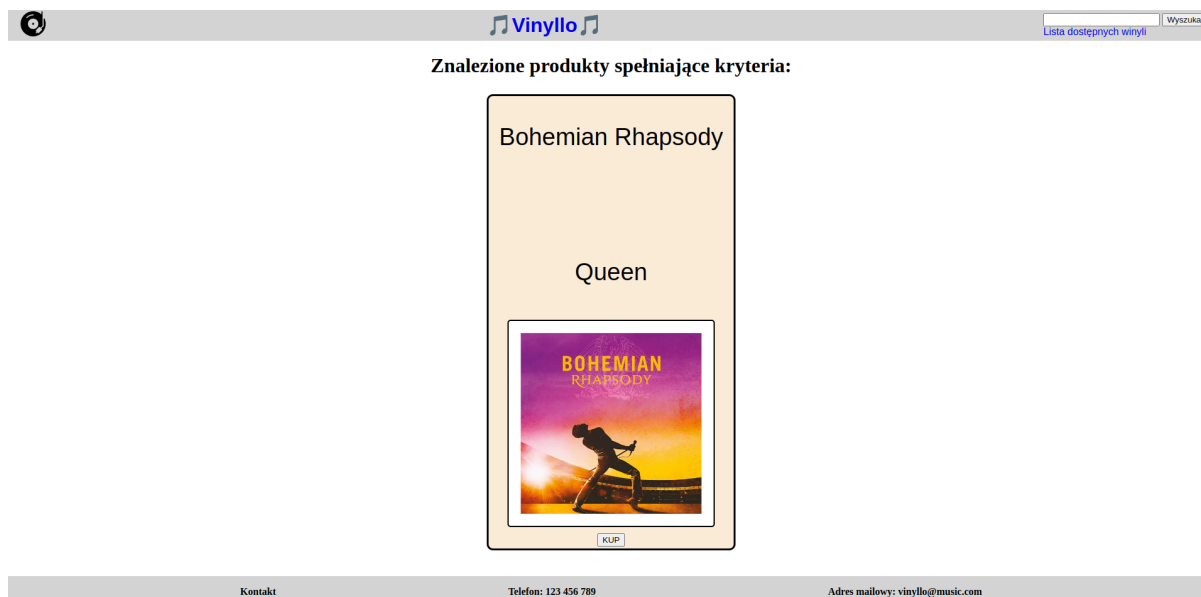
W lewym górnym rogu znajduje się pole wyszukiwania oraz dwa odnośniki do odpowiednio listy wszystkich dostępnych winyli oraz do panelu admina (widoczny tylko dla admina)

Przedstawienie w pliku html:

```
<div class="menus">
    <form method="POST" action="{% url 'search' %}" role="search">
        {% csrf_token %}
        <input type="text" name="searchbar"></input>
        <button type="submit">Wyszukaj</button>
    </form>
    {% if user.is_superuser %}
        <a href="/adminPanel">Panel Admina</a>
    {% endif %}
    <a href="/vinyls">Lista dostępnych winyli</a>
</div>
```

Jak widać odnośniki przekierowują nas na odpowiednie adresy, zdefiniowane w pliku `urls.py`. Natomiast w polu wyszukiwania wykorzystujemy metodę POST (do czego potrzebujemy `csrf_token`) do przesłania wpisanych danych do widoku `search`.

Widok wyszukiwania



Przykład wyglądu strony po wyszukaniu frazy “Queen”

Otrzymujemy wyniki wyszukiwania wraz z możliwością zakupu.

Definicja w pliku views.py:

```
def search(request):
    if request.method == 'POST':
        search_query = request.POST['searchbar']
        posts = Vinylns.objects.filter(Q(title__icontains=search_query) |
Q(artist__icontains=search_query) | Q(genre__icontains=search_query) |
Q(description__icontains=search_query))
        return render(request, 'search.html', {'query':search_query, 'posts': posts})
    else:
        return render(request, 'search.html', {})
```

Jak widać w zależności od tego czy użyta została metoda POST wysyłamy różne argumenty. Wyniki wyszukiwania są otrzymywane za pomocą metody filter która jako argumenty podaje warunki parameter__icontains, które zwracają obiekty które w parametrze zawierają nasze search_query. Wyszukiwanie występuje na przestrzeni tytułu, artysty, gatunku oraz opisu płyty.

Teraz w zależności od tego czy użyto metody POST i od tego czy znaleziono jakiegokolwiek wyniki otrzymujemy wyniki lub odpowiednią informację. Używamy do tego tagów if, else oraz for.

```
{% if query %}
<div>
  {% if posts %}
    <h1>Znalezione produkty spełniające kryteria:</h1>
    <div class="vinylPanel">
      {% for post in posts %}
        <div class="vinylContainer">
          <p>{{ post.title }}</p>
          <p>{{ post.artist }}</p>
          
          <form method="GET" action="/basket">
            <input type="checkbox" name="buy" style="display: none;" value="{{ post.vinylid }}" checked>
            <button type="submit">KUP</button>
          </form>
        </div>
      {% endfor %}
    </div>
  {% else %}
    <h1>Nie znaleziono poszukiwanej frazy</h1>
  {% endif %}
</div>
{% else %}
  <h1>Proszę wprowadzić poprawne wyszukiwanie</h1>
{% endif %}
```

Widok panelu admina

The screenshot shows the Vinylo admin panel. At the top, there's a navigation bar with a logo, a search bar, and links for 'Panel Admina' and 'Lista dostępnych winyli'. Below this is a link to 'Statystyki Sprzedaży'. The main content area contains two forms. The first form, titled 'Dodaj Płytę', has fields for 'Tytuł', 'Wykonawca', 'Gatunek', 'Opis', 'Link do okładki', and 'Cena' (with a currency icon). The second form, titled 'Zarejestruj dostawę', has fields for 'Tytuł', 'Wykonawca', and 'Liczba sztuk' (with a currency icon). Both forms have a 'Potwierdź' button. The footer contains links for 'Kontakt', 'Telefon: 123 456 789', and 'Adres mailowy: vinylo@music.com'.

W głównej części widoku mamy dostęp do statystyk sprzedaży płyt, formularz pozwalający dodać nową płytę do bazy oraz formularz pozwalający zarejestrować dostawę.

Definicja w pliku views.py

```
def adminPanel(request):  
    template = loader.get_template("adminPanel.html")  
    context = {}  
    return HttpResponse(template.render(context, request))
```


Plik html:



```
<h2>  
    <a href="/salesStats/">Statystyki Sprzedaży</a>  
</h2>  
<div class="formPanel">  
    <h2>Dodaj Płytę</h2>  
    <form action="/addVinyl/" method="GET">  
        <input type="text" name="title" placeholder="Tytuł">  
        <br>  
        <input type="text" name="artist" placeholder="Wykonawca">  
        <br>  
        <input type="text" name="genre" placeholder="Gatunek">  
        <br>  
        <input type="text" name="description" placeholder="Opis">  
        <br>  
        <input type="text" name="url" placeholder="Link do okładki">  
        <br>  
        <input type="number" name="price" placeholder="Cena">  
        <br>
```

```
        <button type="submit">Potwierdź</button>
    </form>
</div>
<div class="formPanel">
    <h2>Zarejestruj dostawę</h2>
    <form action="/addDelivery/" method="GET">
        <input type="text" name="title" placeholder="Tytuł">
        <br>
        <input type="text" name="artist" placeholder="Wykonawca">
        <br>
        <input type="number" name="units" placeholder="Liczba sztuk">
        <br>
        <button type="submit">Potwierdź</button>
    </form>
</div>
```

Odnosnik Statystyki Sprzedaży przenosi nas do odpowiedniego adresu. Formularze za pomocą metody GET przesyłają podane informacje do strony dodającej nowy winyl lub nową dostawę.

Widok dodawania płyty



Vinyllo

[Panel Admina](#) [Lista dostępnych winyli](#)

Pomyślnie dodano

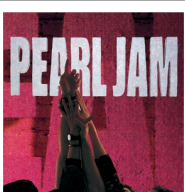
Tytuł: Ten

Wykonawca: Pearl Jam

Gatunek: Rock

Opis: Pierwszy studyjny album legendarnej grupy z Seattle

Cena: 160 zł



[Kontakt](#) [Telefon: 123 456 789](#) [Adres mailowy: vinyllo@music.com](#)

Otrzymujemy informację o dodanej płycie lub informację o nieudawanym dodawaniu. Definicja w pliku views.py:

```
def addVinyl(request):
    template = loader.get_template("addVinyl.html")
    vinyl = None
    if request.method == 'GET':
        title = request.GET.get('title')
        artist = request.GET.get('artist')
        genre = request.GET.get('genre')
        description = request.GET.get('description')
        url = request.GET.get('url')
        price = request.GET.get('price')
        if '' not in [title, artist, genre, description, url, price]:
            vinyl = Vinyls(title=title, artist=artist, genre=genre, description=description, url=url,
price=price)
            vinyl.save()
            context = {
                'vinyl': vinyl
            }
            return HttpResponse(template.render(context, request))
        else:
            return HttpResponse(template.render({}, request))
```

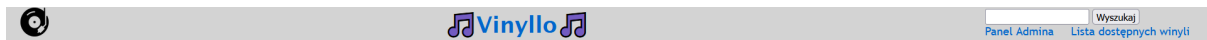
Tutaj również, w zależności od użytej metody oraz wczytanych danych, wysyłamy do strony vinyl utworzony z danych lub pusty obiekt.

Plik html:

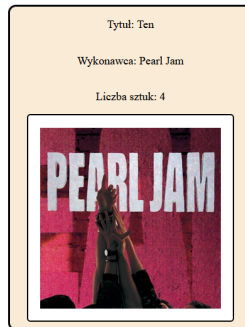
```
{% if vinyl %}
<h1>Pomyślnie dodano</h1>
<div class="vinylContainer">
  <p>Tytuł: {{vinyl.title}}</p>
  <p>Wykonawca: {{vinyl.artist}}</p>
  <p>Gatunek: {{vinyl.genre}}</p>
  <p>Opis: {{vinyl.description}}</p>
  <p>Cena: {{vinyl.price}} zł</p>
  
</div>
{%else%}
  <h2>Problem z dodawaniem, spróbuj ponownie</h2>
{%endif%}
```

Tutaj ponownie używamy tagów warunkowych oraz for.

Widok rejestracji dostawy

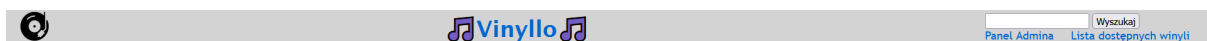


Pomyślnie zarejestrowano dostawę



Widok w przypadku udanej rejestracji dostawy

Aby zarejestrować dostawę, administrator wprowadza tytuł, wykonawcę oraz liczbę egzemplarzy. Ponieważ foreign key w tabeli Deliveries to VinylID, funkcja w pliku views.py wykrywa pożądaný produkt na podstawie tytułu i wykonawcy. W przypadku, gdy żaden winyl znajdujący się w bazie nie spełnia kryteriów, strona wyświetla następujący komunikat:



Nie udało się zarejestrować dostawy.

Czy dodałeś już album do bazy?

Czy wprowadziłeś poprawne dane?



Definicja w pliku views.py:

```
def addDelivery(request):  
    template = loader.get_template("addDelivery.html")  
    vinyl = None  
    if request.method == 'GET':
```

```

title = request.GET.get('title')
artist = request.GET.get('artist')
units = request.GET.get('units')
vinyl = Vinyls.objects.filter(Q(title=title) & Q(artist=artist)).first()
if vinyl:
    newdate = date.today()
    delivery = Deliveries(dateofdelivery=newdate, vinylid=vinyl, unitsdelivered=units)
    delivery.save()
    context = {
        'delivery': delivery,
        'vinyl': vinyl
    }
    return HttpResponse(template.render(context, request))
else:
    return HttpResponse(template.render({}, request))

```

Plik html:

```

{% if delivery %}
    <h1>Pomyślnie zarejestrowano dostawę</h1>
    <div class="vinylContainer">
        <p>Tytuł: {{vinyl.title}}</p>
        <p>Wykonawca: {{vinyl.artist}}</p>
        <p>Liczba sztuk: {{delivery.unitsdelivered}}</p>
        
    </div>
{% else %}
    <h1>Nie udało się zarejestrować dostawy.</h1>
    <p>Czy dodałeś już album do bazy?</p>
    <p>Czy wprowadziłeś poprawne dane?</p>
{% endif %}

```

Widok statystyk

Vinyllo					
			Wyszukaj		
Panel Admina			Lista dostępnych winyli		
Tytuł	Artysta	Gatunek	Cena za sztukę (zł)	Ilość sprzedanych sztuk	Przychód ze sprzedaży (zł)
Ancient Dreams In A Modern Land	MARINA	Pop	140	4	560
The Velvet Underground & Nico	The Velvet Underground & Nico	Rock	220	2	440
Enter the Wu-Tang (36 Chambers)	Wu-Tang Clan	Rap/Hip-Hop	120	3	360
Euphories	Videoclub	Pop	160	2	320
Bohemian Rhapsody	Queen	Rock	150	2	300
In the Court of the Crimson King	King Crimson	Rock	200	1	200
Abbey Road	The Beatles	Rock	150	1	150
Miłość w czasach popkultury	Myslovitz	Pop	100	1	100
Hybrid Theory	Linkin Park	Metal	120	0	0
ABBA Gold	ABBA	Pop	80	0	0
To Pimp A Butterfly	Kendrick Lamar	Rap/Hip-Hop	180	0	0
Ok Computer	Radiohead	Rock	150	0	0
Wish You Were Here	Pink Floyd	Rock	120	0	0
Madvillainy	Madvillain	Rap/Hip-Hop	160	0	0
Kid A	Radiohead	Rock	150	0	0
In Rainbows	Radiohead	Rock	140	0	0

Kontakt Telefon: 123 456 789 Adres mailowy: vinyllo@music.com

Widok pokazuje informacje na temat sprzedaży płyt posortowane po sumarycznych zyskach ze sprzedaży malejąco.

Definicja w pliku views.py:

```
def salesStats(request):
    template = loader.get_template("salesStats.html")
    queryset = Vinyls.objects.annotate(
        sprzedane=Coalesce(Sum('sales__quantity'), 0),
        total_price=ExpressionWrapper(
            Coalesce(Sum(F('sales__quantity') * F('price')), 0),
            output_field=IntegerField()
        )
    ).order_by('-total_price')
    context = {
        'stats': queryset,
    }
    return HttpResponse(template.render(context, request))
```

Przy pobieraniu danych z bazy korzystamy z `.annotate` które pozwala nam korzystać z funkcji agregujących i ich wyniki przypisać do nowych kolumn. `Coalesce` pozwala nam na ustawienie wartości 0 w przypadku gdy otrzymujemy `None`. Jak widać do zmiennej `sprzedane` przypisujemy sumę sztuk sprzedanych w tabeli `Sales`.

W przypadku `total_price` musimy dodatkowo użyć `ExpressionWrapper`, aby móc ustawić pole wyjściowe jako typ `IntegerField`. Znow sumujemy

sprzedane sztuki z tabeli Sales, sumę dodatkowo mnożymy przez cenę płyty żeby uzyskać zysk.

Sortowanie odbywa się za pomocą `order_by('-total_price')` gdzie "-" oznacza sortowanie malejąco.

Plik html:

```
<table class="statsTable">
  <tr>
    <th>Tytuł</th>
    <th>Artysta</th>
    <th>Gatunek</th>
    <th>Cena za sztukę (zł) </th>
    <th>Ilość sprzedanych sztuk</th>
    <th>Przychód ze sprzedaży (zł) </th>
  </tr>
  {% for vinyl in stats %}
  <tr>
    <td>{{vinyl.title}}</td>
    <td>{{vinyl.artist}}</td>
    <td>{{vinyl.genre}}</td>
    <td>{{vinyl.price}}</td>
    <td>{{vinyl.sprzedane}}</td>
    <td>{{vinyl.total_price}}</td>
  </tr>
  {% endfor %}
</table>
```

Utworzona zostaje tabela, gdzie dla każdego winyla tworzymy wiersz by wyświetlić wcześniej wygenerowane informacje.

Widok listy winyli



[Panel Admina](#) [Lista dostępnych winyli](#)

Sortuj według:

☐ Cena: malejąco
☐ Cena: rosnąco
☐ Tytuł: alfabetycznie
☐ Wykonawca: alfabetycznie

Filtruj według:

Gatunek:
☒ Rock
☒ Rap/Hip-Hop
☒ Jazz
☒ Pop
☒ Post-Punk
☒ Post-Rock
☒ Shoegaze
☒ Metal

Lista winyli



Tytuł: Miłość w czasach popkultury

Wykonawca: Myslovitz

Gatunek: Pop

Opis: Polski klasyk

Cena: 100 zł

Ilość dostępnych sztuk: 49



Tytuł: Bohemian Rhapsody

Wykonawca: Queen

Gatunek: Rock

Opis: Soundtrack z filmu

Cena: 150 zł

Ilość dostępnych sztuk: 18



Tytuł: Hybrid Theory

Wykonawca: Linkin Park

Gatunek: Metal

Z lewej strony mamy narzędzie filtrujące - sortujące, w którym możemy wybrać interesujące nas gatunki muzyczne, a także dokonać sortowania produktów według konkretnego kryterium.

Definicja w pliku views.py:

```
def allVinyls(request):
    vinyls = Vinyls.objects.none()
    if request.method == 'GET':
        genres = request.GET.getlist('genres')
        if genres:
            for mygenre in genres:
                vinyls = chain(vinyls, Vinyls.objects.filter(genre=mygenre))
        else:
            vinyls = chain(vinyls, Vinyls.objects.all())
        sort_method = request.GET.get('sort')
        if sort_method:
            if sort_method == "price_desc":
                vinyls = sorted(vinyls, key=operator.attrgetter('price'), reverse=True)
            elif sort_method == "price_asc":
                vinyls = sorted(vinyls, key=operator.attrgetter('price'))
            elif sort_method == "title":
                get_key = operator.attrgetter('title')
                vinyls = sorted(vinyls, key=lambda x: get_key(x).lower())
            elif sort_method == "artist":
                get_key = operator.attrgetter('artist')
                vinyls = sorted(vinyls, key=lambda x: get_key(x).lower())
        template = loader.get_template("allVinyls.html")
        context = {
            'vinyls': vinyls,
        }
        return HttpResponse(template.render(context, request))
```

23

Na początku generowany jest pusty zbiór winyli. Następnie, jeśli użyto metody GET to sprawdzamy, czy dokonano wyboru konkretnych gatunków. W przypadku, gdy nie wybrano żadnych gatunków generowana jest lista wszystkich produktów, w przeciwnym wypadku za pomocą metody chain po kolei do utworzonego wcześniej zbioru dodawane są produkty zgodne z poszczególnymi kryteriami. Następnie w zależności od wybranego sortowania używana jest metoda sorted z odpowiednim argumentem key.

Plik html:

```
<div class="sort_filter">
  <form method="GET" action="/vinyls">
    <div id="form_title">Sortuj według:</div>
    <input type="radio" name="sort" value="price_desc">Cena: malejąco<br>
    <input type="radio" name="sort" value="price_asc">Cena: rosnaco<br>
    <input type="radio" name="sort" value="title">Tytuł: alfabetycznie<br>
    <input type="radio" name="sort" value="artist">Wykonawca: alfabetycznie<br>
    <div id="form_title">Filtruj według:</div>
    Gatunek:<br>
    <input type="checkbox" name="genres" value="Rock" checked>Rock</input><br>
    <input type="checkbox" name="genres" value="Rap/Hip-Hop" checked>Rap/Hip-Hop</input><br>
    <input type="checkbox" name="genres" value="Jazz" checked>Jazz</input><br>
    <input type="checkbox" name="genres" value="Pop" checked>Pop</input><br>
    <input type="checkbox" name="genres" value="Post-Punk" checked>Post-Punk</input><br>
    <input type="checkbox" name="genres" value="Post-Rock" checked>Post-Rock</input><br>
    <input type="checkbox" name="genres" value="Shoegaze" checked>Shoegaze</input><br>
    <input type="checkbox" name="genres" value="Metal" checked>Metal</input><br>
    <button type="submit">Zatwierdź</button>
  </form>
</div>
<div class="mainPanel">
  <h1>Lista winyli</h1>
  <div class="vinylPanel">
    {% for vinyl in vinyls %}
    <div class="vinylContainer">
      
      <div class="infoPanel">
        <p>Tytuł: {{vinyl.title}}</p>
        <p>Wykonawca: {{vinyl.artist}}</p>
        <p>Gatunek: {{vinyl.genre}}</p>
        <p>Opis: {{vinyl.description}}</p>
        <p>Cena: {{vinyl.price}} zł</p>
        <p>Ilość dostępnych sztuk: {{vinyl | get_units}}</p>
        <form method="GET" action="/basket">
          <input type="checkbox" name="buy" value="{{ vinyl.vinylid }}" style="display:
none;" checked>
          <button type="submit">KUP</button>
        </form>
      </div>
    </div>
    {% endfor %}
  </div>
</div>
```


Formularz GET posiada przyciski typu radio dla opcji sortowania, przyciski typu checkbox dla opcji filtrowania oraz przycisk typu submit. W panelu poniżej następuje użycie for w celu wygenerowania informacji dla każdej z płyt. Dodatkowo korzystamy z funkcji `get_units`, która zdefiniowana została w pliku `templatetags/customtags.py`:

```
@register.filter
def get_units(vinyl):
    vid = vinyl.vinylid
    unitsDelivered = Deliveries.objects.filter(vinylid=vid).aggregate(total=Sum('unitsdelivered'))['total']
    if unitsDelivered is None:
        unitsDelivered = 0
    unitsSold = Sales.objects.filter(vinylid=vid).aggregate(total=Sum('quantity'))['total']
    if unitsSold is None:
        unitsSold = 0
    return unitsDelivered - unitsSold
```

Wyszukuje ona ilość dostarczonych oraz sprzedanych płyt i na ich podstawie oblicza dostępną ilość płyt. Aby korzystać z funkcji musimy dodać dekorator: `@register.filer`, a w pliku html dodajemy:

```
{% load customtags %}
```

Widok kupowania

The screenshot shows a web browser window with the Vinyllo application. The header includes the Vinyllo logo and a search bar. The main content area displays a vinyl record cover for 'Andy Warhol' (a banana) and a form titled 'Szczegóły transakcji' (Transaction Details). The form contains four input fields: 'Imię' (Name), 'Nazwisko' (Surname), 'mail' (Email), and 'Liczba egzemplarzy' (Number of copies), followed by a 'Potwierdź' (Confirm) button. The footer contains contact information: 'Kontakt', 'Telefon: 123 456 789', and 'Adres mailowy: vinyllo@music.com'.

Po wciśnięciu któregośkolwiek z przycisków “KUP” przechodzimy do widoku kupowania, gdzie widzimy podstawowe informacje o płycie oraz formularz z danymi.

Widok w pliku views.py:

```
def basket(request):
    template = loader.get_template("basket.html")
    myvinylid = request.GET.get("buy")
    if request.method == 'GET':
        vinyl = Vinyls.objects.filter(vinylid=myvinylid).first()
        context = {
            'vinyl': vinyl
        }
        return HttpResponse(template.render(context, request))
    return HttpResponse(template.render({}, request))
```

Plik html:

```
{% if vinyl %}
{% if vinyl.is_available %}
<h1>Kup winyl</h1>
<div class="vinylContainer">
    <div class="vinyl_title">{{ vinyl.title }}</div>
    <p class="vinyl_artist">{{ vinyl.artist }}</p>
    <p class="vinyl_price">{{ vinyl.price }} zł</p>
    <p class="vinyl_price">Ilość dostępnych sztuk: {{vinyl | get_units}}</p>
    
</div>
<div class="formPanel">
```

```

<h2>Szczegóły transakcji</h2>
<form action="/purchased/" method="GET">
  <input type="checkbox" name="buy" value={{ vinyl.vinylid }} style="display: none;" checked>
  <input type="text" name="firstname" placeholder="Imię">
  <br>
  <input type="text" name="lastname" placeholder="Nazwisko">
  <br>
  <input type="text" name="mail" placeholder="mail">
  <br>
  <input type="number" name="quantity" placeholder="Liczba egzemplarzy" min="1" max="{{ vinyl
| get_units }}">
  <br>
  <button type="submit">Potwierdź</button>
</form>
</div>
{% else %}
  <h1>Przepraszamy, ten produkt jest już niedostępny</h1>
  <a href="/">Powrót na stronę główną</a>
{% endif %}
{% else %}
  <h1>Wystąpił błąd</h1>
{% endif %}

```

Widok zakupu:



Otrzymujemy informacje zwrotną dotyczącą zakupu.

Definicja w pliku views.py:

```
def purchased(request):
    template = loader.get_template("purchased.html")
    vinyl = Vinyls.objects.filter(vinylid=request.GET.get("buy")).first()
    if request.method == 'GET':
        newdate = date.today()
        name = request.GET.get('firstname')
        surname = request.GET.get('lastname')
        mail = request.GET.get('mail')
        customer = Customers.objects.filter(Q(name=name) & Q(surname=surname) &
Q(address=mail)).first()
        if not customer:
            customer = Customers(name=name, surname=surname, address=mail)
            customer.save()
        quantity = request.GET.get('quantity')
        if int(quantity) <= get_units(vinyl):
            sale = Sales(customerid=customer, vinylid=vinyl, dateoftransaction=newdate,
quantity=int(quantity),price=vinyl.price)
            sale.save()
            return HttpResponse(template.render({'vinyl': vinyl}, request))
        else:
            return HttpResponse(template.render({}, request))
```

Dane klienta wyszukiwane są w bazie, jeśli dany klient istnieje jego ID będzie przypisane do transakcji, jeśli użytkownika nie ma w bazie to zostaje on utworzony.

Plik html:

```
{% if vinyl %}
    <h1>Gratulujemy zakupu!</h1>
{% else %}
    <h1>Coś poszło nie tak</h1>
```

```
{% endif %}  
  <a href="/">powrót na stronę główną</a>
```