Sprawozdanie z miniprojektu

System rezerwacji biletów

Maciej Trznadel, Patryk Blacha

9 czerwca 2025

Spis treści

1 Wstęp

Link do repozytorium: https://github.com/PatrykBlacha/System-do-rezerwacji-bilet-w.git

Celem projektu było stworzenie prostego systemu do rezerwacji biletów na wydarzenia. Aplikacja umożliwia użytkownikom przeglądanie dostępnych wydarzeń, zakup biletów, przypisanie ich do uczestników oraz przegląd i anulowanie zamówień.

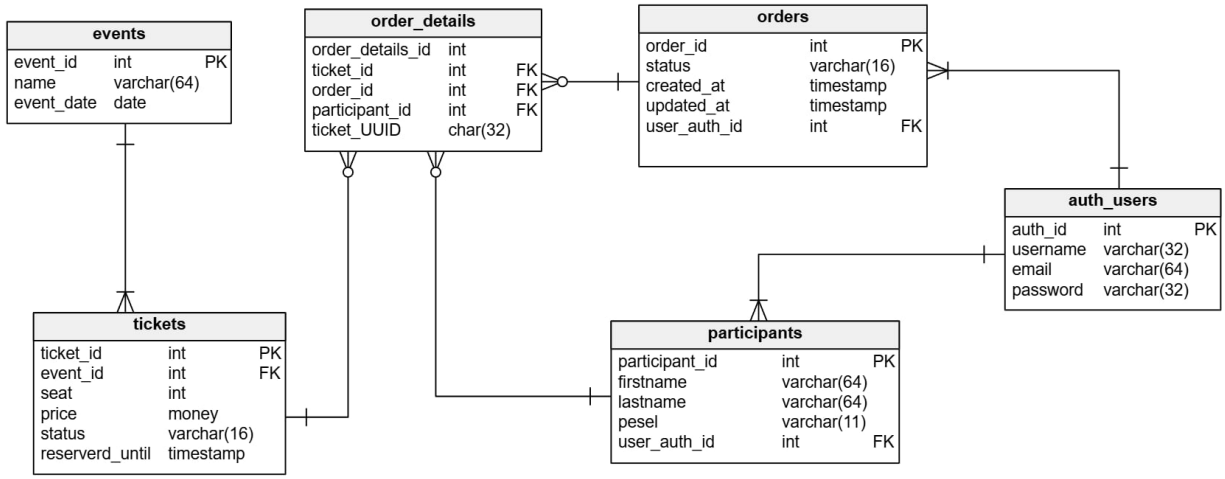
2 Opis systemu

System został zaimplementowany z wykorzystaniem frameworka **Django (Python)** oraz relacyjnej bazy danych **PostgreSQL**. Główne komponenty systemu to:

- **Modele bazodanowe (Django ORM)** – definiujące strukturę danych oraz relacje między tabelami (wydarzenia, bilety, uczestnicy, zamówienia).
- **Widoki i logika biznesowa (views.py)** – obsługujące operacje CRUD, finalizację zamówienia oraz przypisanie danych uczestników.
- **Transakcyjność i współbieżność** – zaimplementowana przy pomocy `@transaction.atomic` i `select_for_update`, co pozwala uniknąć konfliktów przy jednoczesnych operacjach na biletach.
- **Szablony HTML (Django Templates)** – odpowiedzialne za prezentacje i interakcję z użytkownikiem.
- **System logowania i kont użytkowników** – wykorzystuje wbudowany model `auth_user` z Django.

3 Baza danych

Poniżej przedstawiamy schemat naszej bazy danych:



Rysunek 1: Schemat bazy danych

Opis tabel

Tabela: `events`
Tabela przechowuje informacje o wydarzeniach, na które można zakupić bilety. Zawiera nazwę wydarzenia oraz datę, kiedy się ono odbywa.

- `name` – nazwa wydarzenia
- `event_date` – data wydarzenia

Tabela: `tickets`
Tabela reprezentuje bilety przypisane do konkretnego wydarzenia. Każdy bilet ma przypisane miejsce, cenę oraz status określający jego dostępność.

- `event` – klucz obcy do tabeli `events`
- `seat` – numer miejsca
- `price` – cena biletu
- `status` – status biletu (dostępny, zarezerwowany, sprzedany)
- `reserved_until` – opcjonalne pole określające czas rezerwacji

Tabela: `participants`
Tabela przechowuje dane uczestników wydarzenia. Każdy uczestnik jest powiązany z kontem użytkownika w systemie (Jedno konto może być powiazane z wieloma uczestnikami) i może mieć przypisane dane osobowe potrzebne do identyfikacji osoby podczas wejścia na wydarzenie.

- `first_name`, `last_name` – imię i nazwisko uczestnika
- `pesel` – numer PESEL
- `user` – klucz obcy do konta użytkownika w systemie `auth_user`

Tabela: `orders`
Reprezentuje zamówienie złożone przez użytkownika. Zawiera informacje o statusie zamówienia oraz znacznik czasu jego utworzenia i modyfikacji.

- `user` – klucz obcy do zalogowanego użytkownika systemu auth$_u ser skadajcego zamwienie$`status`$\breve{}$ statu
- `created_at`, `updated_at` – daty utworzenia i ostatniej modyfikacji

Tabela: `orders_details`
Tabela szczegółów zamówienia wiążaca bilet z uczestnikiem. Zawiera również unikalny identyfikator biletu (UUID) przypisanego do danego uczestnika. Taki indentyfikator mógłby służyć do sprawdzenia czy bilet jest prawdziwy.

- `order` – klucz obcy do zamówienia
- `participant` – klucz obcy do uczestnika przypisanego do biletu

- **ticket** – przypisany bilet
- **ticket_UUID** – unikalny identyfikator biletu

Tabela: auth_user
Tabela służąca do logowania użytkowników, dostarczana przez Django w ramach wbudowanego systemu uwierzytelniania. Dla uproszczenia przedstawiono tylko istotne kolumny wykorzystywane w projekcie.

- **username** – nazwa użytkownika systemu
- **email** – email użytkownika systemu
- **password** – hasło użytkownika systemu

Schemat bazy danych został zaimplementowany w Django z wykorzystaniem wbudowanego ORM. Poniżej znajdują się klasy modeli odpowiadające poszczególnym tabelom bazy danych.

```
1
2  import uuid
3
4  from django.contrib.auth.models import User
5  from django.db import models
6
7  class Event(models.Model):
8      name = models.CharField(max_length=64)
9      event_date = models.DateField()
10     def __str__(self):
11         return self.name
12     class Meta:
13         db_table = 'events'
14
15 class Ticket(models.Model):
16     STATUS_CHOICES = [
17         ('available', 'Available'),
18         ('reserved', 'Reserved'),
19         ('sold', 'Sold')
20     ]
21     event = models.ForeignKey(Event, on_delete=models.CASCADE)
22     seat = models.IntegerField()
23     price = models.DecimalField(max_digits=8, decimal_places=2)
24     status = models.CharField(max_length=16, choices=STATUS_CHOICES)
25     reserved_until = models.DateTimeField(null=True, blank=True)
26     def __str__(self):
27         return f"Ticket: {self.id} for {self.event.name}, seat: {self.seat},
           price: {self.price}"
28     class Meta:
29         db_table = 'tickets'
30
31 class Participant(models.Model):
32     first_name = models.CharField(max_length=64, null=True, blank=True)
33     last_name = models.CharField(max_length=64, null=True, blank=True)
34     pesel = models.CharField(max_length=11, null=True, blank=True)
35     user = models.ForeignKey(User, on_delete=models.CASCADE)
36     def __str__(self):
37         return f"{self.first_name} {self.last_name}"
```

```
38      class Meta:
39          db_table = 'participants'
40
41  class Order(models.Model):
42      STATUS_CHOICES = [
43          ('pending', 'Pending'),
44          ('completed', 'Completed'),
45          ('canceled', 'Canceled'),
46      ]
47      status = models.CharField(max_length=16, choices=STATUS_CHOICES)
48      created_at = models.DateTimeField(auto_now_add=True)
49      updated_at = models.DateTimeField(auto_now=True)
50      user = models.ForeignKey(User, on_delete=models.CASCADE)
51      def __str__(self):
52          return f"Order {self.id}"
53      class Meta:
54          db_table = 'orders'
55
56  class OrderDetails(models.Model):
57      order = models.ForeignKey(Order, on_delete=models.CASCADE)
58      participant = models.ForeignKey(Participant, on_delete=models.CASCADE)
59      ticket = models.ForeignKey(Ticket, on_delete=models.CASCADE)
60      ticket_UUID = models.UUIDField(default=uuid.uuid4, editable=False)
61      def __str__(self):
62          return f"Order by {self.participant} for ticket {self.ticket}"
63      class Meta:
64          db_table = 'orders_details'
```

Listing 1: models.py

4 Implementacja widoków

Wszystkie pliki html rozszerzają nasz plik bazowy:

```
1       {% load static %}
2   <!DOCTYPE html>
3   <html>
4   <head>
5       <title>{% block title %}My Tickets App{% endblock %}</title>
6       <link rel="stylesheet" href="{% static 'tickets_app/style.css' %}">
7   </head>
8   <body>
9       <header>
10      {% if user.is_authenticated %}
11          <p>Zalogowany jako: {{ user.username }} ({{ user.email }})</p>
12          <nav>
13              <a href="{% url 'home' %}">Home</a> |
14              <a href="{% url 'cart' %}">        Koszyk</a> |
15              <a href="{% url 'my_orders' %}">        Moje zamówienia</a>
16          </nav>
17          <form method="post" action="{% url 'logout' %}" style="display:inline;">
18              {% csrf_token %}
19              <button type="submit">Wyloguj się</button>
20          </form>
21      {% else %}
22          <p>Nie jesteś zalogowany. <a href="{% url 'login' %}">Zaloguj się </a> (<
                a href="{% url 'register' %}">Zarejestruj się</a>)</p>
23      {% endif %}
24  </header>
```

```
25
26      <main>
27          {% block content %}
28          {% endblock %}
29      </main>
30  </body>
31  </html>
```

Listing 2: base.html

Wykorzystaliśmy również style:

```
1  body {
2      font-family: Arial, sans-serif;
3      background: #f0f2f5;
4      margin: 40px;
5      color: #333;
6  }
7  h1 {
8      color: #2c3e50;
9      margin-bottom: 20px;
10     border-bottom: 2px solid #2980b9;
11     padding-bottom: 10px;
12 }
13 ul {
14     list-style: none;
15     padding-left: 0;
16 }
17 li {
18     background: white;
19     margin-bottom: 12px;
20     padding: 12px 20px;
21     border-radius: 6px;
22     box-shadow: 0 1px 3px rgba(0,0,0,0.1);
23     transition: background-color 0.3s;
24 }
25 li:hover {
26     background-color: #d6e9f8;
27 }
28 a {
29     text-decoration: none;
30     color: #2980b9;
31     font-weight: 600;
32 }
33 a:hover {
34     text-decoration: underline;
35 }
36 p {
37     font-style: italic;
38     color: #777;
39 }
```

Listing 3: style.css

4.1 Widok strony głównej

Pokazują nam się tutaj dostępne wydarzenia:

```
1      {% extends "base.html" %}
2  {% load static %}
3
```

```
{% block title %}Tickets_app{% endblock %}

{% block content %}
    <h1>Upcoming Events</h1>

    {% if latest_events %}
        <ul>
        {% for event in latest_events %}
            <li>
                <a href="{% url 'tickets' event.id %}">{{ event.name }}</a>
            </li>
        {% endfor %}
        </ul>
    {% else %}
        <p>No polls are available.</p>
    {% endif %}
{% endblock %}
```

Listing 4: index.html

Obsługa w DjJango:

```
class IndexView(generic.ListView):
    template_name = 'tickets_app/index.html'
    context_object_name = 'latest_events'

    def get_queryset(self):
        now = timezone.now()
        unlock_reserved_tickets()
        return Event.objects.all().filter(event_date__gt=now).order_by('
            event_date')
```

Listing 5: Striba główna

4.2 Widok rejestracji oraz logowania

```
{% extends "base.html" %}

{% block title %}Rejestracja{% endblock %}

{% block content %}
  <h2>Rejestracja</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Zarejestruj się</button>
  </form>
{% endblock %}
```

Listing 6: register.html

Logika rejestracji:
Widok formularza zdefiniowany został w ten sposób:

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class CustomUserCreationForm(UserCreationForm):
    first_name = forms.CharField(max_length=64, required=True)
```

```
    last_name = forms.CharField(max_length=64, required=True)
    email = forms.EmailField(required=True)
    address = forms.CharField(max_length=128, required=False)  # dodaj adres (opcjonalnie)

    class Meta:
        model = User
        fields = ("username", "first_name", "last_name", "email", "address", "password1", "password2")
```

Listing 7: forms.py

```
    def register(request):
    if request.method == "POST":
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save(commit=False)
            user.first_name = form.cleaned_data['first_name']
            user.last_name = form.cleaned_data['last_name']
            user.email = form.cleaned_data['email']
            user.save()

            login(request, user)
            return redirect('home')
    else:
        form = CustomUserCreationForm()
    return render(request, 'registration/register.html', {'form': form})
```

Listing 8: widok do rejestracji

Logika logowania jest zapewniona automatycznie przez DJango.

Widok do logowania w html:

```
    {% extends "base.html" %}

{% block title %}Logowanie{% endblock %}

{% block content %}
<h2>Zaloguj się</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Zaloguj</button>
</form>
{% endblock %}
```

Listing 9: login.html

4.3 Widok do rezerwowania biletów

Zaznaczamy które bilety chcemy kupić a następnie są one rezerwowane, dla naszego użtkownika.

```

{% extends "base.html" %}
{% load static %}

{% block title %}{{ event.name }} - Wybierz bilety{% endblock %}

{% block content %}
  <h1>{{ event.name }}</h1>
```

```
  <h2>{{ event.event_date }}</h2>

  <h3>Dostępne bilety:</h3>

  {% if tickets %}
    <form method="post" action="">
      {% csrf_token %}
      <table>
        <thead>
          <tr>
            <th>Wybierz</th>
            <th>Miejsce</th>
            <th>Cena</th>
          </tr>
        </thead>
        <tbody>
          {% for ticket in tickets %}
            <tr>
              <td>
                <input type="checkbox" name="ticket_ids" value="{{ ticket.id }}"
                    id="ticket_{{ ticket.id }}">
              </td>
              <td><label for="ticket_{{ ticket.id }}">{{ ticket.seat }}</label></
                  td>
              <td>${{ ticket.price }}</td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
      <button type="submit">Dodaj do koszyka</button>
    </form>
  {% else %}
    <p>Brak dostępnych biletów.</p>
  {% endif %}

  <p>
    <a href="{% url 'home' %}">    Powrót do wydarzeń</a>
  </p>
{% endblock %}
```

Listing 10: bilety.html

Widok w DJango:

```

@login_required
@transaction.atomic
def tickets_view(request, event_id):
    event = get_object_or_404(Event, pk=event_id)
    tickets = Ticket.objects.select_for_update().filter(event=event, status='
        available').order_by('seat')

    if request.method == "POST":
        selected_ticket_ids = request.POST.getlist('ticket_ids')

        if not selected_ticket_ids:
            messages.error(request, "Nie wybrano żadnych biletów.")
            return redirect('tickets', event_id=event_id)

        try:
            with transaction.atomic():
                order = Order.objects.create(user=request.user, status='pending')
```

```
18
19                  for ticket_id in selected_ticket_ids:
20                      ticket = Ticket.objects.select_for_update().get(id=ticket_id)
21                      if ticket.status != 'available':
22                          raise Exception(f"Bilet {ticket.seat} jest już niedostę
                               pny.")
23
24                      participant = Participant.objects.create(user=request.user)
25                      ticket.status = 'reserved'
26                      ticket.reserved_until = timezone.now() + timedelta(minutes
                           =10)
27                      ticket.save()
28
29                      OrderDetails.objects.create(
30                          order=order,
31                          participant=participant,
32                          ticket=ticket
33                      )
34
35              messages.success(request, "Bilety zostały dodane do koszyka.")
36              return redirect('cart')
37
38          except Exception as e:
39              messages.error(request, str(e))
40              return redirect('tickets', event_id=event_id)
41
42      return render(request, 'tickets_app/tickets.html', {'event': event, "tickets"
            : tickets})
```

Listing 11: widok do rezerwowania biletów

Jeżeli klient nie zdecyduje się zrealizować rezerwacji wówczas bilet po 10min wróci ze statusem available, aby ta funkcja się wywołała gdy jakikolwiek użytkownik otworzy główną stronę z wydarzeniami.

```
1   def unlock_reserved_tickets():
2   now = timezone.now()
3
4   expired_tickets = Ticket.objects.filter(
5       status='reserved',
6       reserved_until__lt=now
7   )
8
9   for ticket in expired_tickets:
10      ticket.status = 'available'
11      ticket.reserved_until = None
12      ticket.save()
13
14      details = OrderDetails.objects.filter(ticket=ticket)
15      for detail in details:
16          order = detail.order
17          order_tickets = OrderDetails.objects.filter(order=order).
                select_related('ticket')
18
19          if all(d.ticket.status == 'available' for d in order_tickets):
20              order.status = 'canceled'
21              order.save()
```

Listing 12: Odblokowywanie rezerwacji

4.4 Kupowanie biletu

Rezerwacja biletu automatycznie przekierowuje nas do możliwości jego kupienia w koszyku. Tutaj musimy podać imię, nazwisko, pesel osób które będą wchodzić na dany bilet.

```
    {% extends "base.html" %}
{% block title %}Twój koszyk{% endblock %}

{% block content %}
<h2>Twój koszyk</h2>

{% if order_details|length > 0 %}
<form method="post">
    {% csrf_token %}
    {% for detail in order_details %}
        <div style="border:1px solid #ccc; padding:10px; margin-bottom:10px;">
            <p><strong>Miejsce:</strong> {{ detail.ticket.seat }} | <strong>Cena:
                </strong> {{ detail.ticket.price }}</p>
            <input type="text" name="first_name_{{ detail.participant.id }}"
                value="{{ detail.participant.first_name }}" placeholder="Imię"
                required>
            <input type="text" name="last_name_{{ detail.participant.id }}" value
               ="{{ detail.participant.last_name }}" placeholder="Nazwisko"
                required>
            <input type="text" name="pesel_{{ detail.participant.id }}" value="{{
                detail.participant.pesel }}" placeholder="PESEL" required>
        </div>
    {% endfor %}
    <button type="submit">Finalizuj zakup</button>
</form>
{% else %}
    <p>Koszyk jest pusty.</p>
{% endif %}
{% endblock %}
```

Listing 13: cart.html

```
    @login_required
def cart_view(request):
    try:
        order = Order.objects.get(user=request.user, status='pending')
    except Order.DoesNotExist:
        order = None
        order_details = []
    else:
        order_details = OrderDetails.objects.filter(order=order).select_related('
            ticket', 'participant')

        if request.method == "POST":
            for detail in order_details:
                participant = detail.participant
                participant.first_name = request.POST.get(f'first_name_{
                    participant.id}', '')
                participant.last_name = request.POST.get(f'last_name_{participant
                    .id}', '')
                participant.pesel = request.POST.get(f'pesel_{participant.id}', '
                    ')
                participant.save()
            return redirect('finalize_cart')

    return render(request, 'tickets_app/cart.html', {
```

```
        'order': order,
        'order_details': order_details
    })
```

Listing 14: Widok w Django

Funkcja finalizecart która finalizuje zakup i zamienia rezerwację w kupno:

```
@login_required
@transaction.atomic
def finalize_cart(request):
    try:
        order = Order.objects.select_for_update().get(user=request.user, status='pending')
    except Order.DoesNotExist:
        messages.error(request, "Brak zamówienia do finalizacji.")
        return redirect('home')

    order_details = OrderDetails.objects.filter(order=order).select_related('ticket', 'participant')

    for detail in order_details:
        ticket = detail.ticket
        ticket.status = 'sold'
        ticket.save()

    order.status = 'completed'
    order.updated_at = timezone.now()
    order.save()

    messages.success(request, "Zakup zakończony sukcesem!")
    return redirect('my_orders')
```

Listing 15: funkcja do finalizacji

4.5 Widok zamówień użytkownika

Pokazuje jakie zamówienia ma użytkownik, może on je anulować tutaj.

```
{% extends "base.html" %}

{% block title %}Moje Zamówienia{% endblock %}

{% block content %}
  <h2>Moje Zamówienia</h2>
  {% if orders %}
    <ul>
      {% for order in orders %}
        <li>
          Zamówienie #{{ order.id }} |
          Status: {{ order.status }} |
          Data: {{ order.updated_at|date:"Y-m-d H:i" }} |
          <a href="{% url 'order_details' order.id %}">Szczegóły</a>

          {% if order.status != "canceled" %}
            <form action="{% url 'cancel_order' order.id %}" method="post" style="display:inline;">
              {% csrf_token %}
              <button type="submit" onclick="return confirm('Na pewno chcesz anulować to zamówienie?');">
                Anuluj zamówienie
```

```
                </button>
              </form>
            {% endif %}
          </li>
        {% endfor %}
      </ul>
    {% else %}
      <p>Brak zamówień.</p>
    {% endif %}
{% endblock %}
```

Listing 16: my_orders.html

W DJango:

```
    @login_required
def my_orders(request):
    orders = Order.objects.filter(user=request.user).order_by('-updated_at')

    return render(request, 'tickets_app/my_orders.html', {'orders': orders})
```

Listing 17: Obsługa wyświetlania zamówień

```
   @require_POST
@login_required
def cancel_order(request, order_id):
    user = request.user

    try:
        order = Order.objects.get(id=order_id, user=user)

        if order.status == 'canceled':
            messages.warning(request, "Zamówienie już zostało anulowane.")
            return redirect('my_orders')

        order_details = OrderDetails.objects.filter(order=order)

        for detail in order_details:
            ticket = detail.ticket
            ticket.status = 'available'
            ticket.reserved_until = None
            ticket.save()

        order.status = 'canceled'
        order.save()

        messages.success(request, "Zamówienie zostało anulowane.")
    except Order.DoesNotExist:
        messages.error(request, "Nie znaleziono zamówienia.")
    except Exception as e:
        messages.error(request, f"Wystąpił błąd: {e}")

    return redirect('my_orders')
```

Listing 18: Obsługa zwrotu zamówienia

Można również wyświtlić szczegóły zamówienia:

```
{% extends "base.html" %}

{% block title %}Szczegóły Zamówienia{% endblock %}

```

```
{% block content %}
  <h2>Zamówienie #{{ order.id }}</h2>
  <p>Status: {{ order.status }}</p>
  <p>Data: {{ order.updated_at|date:"Y-m-d H:i" }}</p>

  <h3>Bilety:</h3>
  <ul>
    {% for detail in details %}
      <li>
        Wydarzenie: {{ detail.ticket.event.name }} |
          Data: {{ detail.ticket.event.event_date }} |
        Miejsce: {{ detail.ticket.seat }} |
        Uczestnik: {{ detail.participant.first_name }} {{ detail.participant.
           last_name }} |
          UUID: {{ detail.ticket_UUID }}
      </li>
    {% endfor %}
  </ul>

  <a href="{% url 'my_orders' %}">    Powrót do zamówień</a>
{% endblock %}
```

Listing 19: order_details.html

Obsługa w DJango:

```
    @login_required
def order_details(request, order_id):
    order = get_object_or_404(Order, id=order_id, user=request.user)
    details = OrderDetails.objects.filter(order=order).select_related('ticket', '
       participant')

    return render(request, 'tickets_app/order_details.html', {
        'order': order,
        'details': details
    })
```

Listing 20: Obsługa zwrotu zamówienia

5 Podsumowanie

Korzystanie z DJango bardzo pomogło nam w utrzymaniu czytelności projektu oraz przejrzystej logiki. Zapisaliśmy również procedury, widoki, triggery i funkcje w czystym PostreSQL podczas robienia projektu. Korzystanie z nich jednak okazało się dużo mniej wygodnie niż zapewnienie tych samych funkcjonalności w DJango.