Bazy danych - Aplikacja hotelowa z użyciem PostgreSQL

Łukasz Wójcik & Daniel Troszczyński

25 czerwca 2024

Spis treści

1 API

1.1 Łączenie się z bazą danych

API nawiązuje połączenie z bazą danych poprzez użycie "connection string'a", który zawiera informacje inicjalizacyjne. Za jego pomocą baza danych przeprowadza walidację i nawiązuje połączenie z API. Funkcje odczytu korzystają z slaveConnectionstring a funkcje zapisu z masterConnectionString.

```
public class Db
{
    static public string masterConnectionString = "Server=localhost;Port=5431;Username=postgres;Password=password;Database=hotel;";
    static public string slaveConnectionString = "Server=localhost;Port=5433;Username=postgres;Password=password;Database=hotel;";
}
```

Rysunek 1: Łączenie się z bazą danych

1.2 Szyfrowanie SSL

Protokół SSL zapewnia uwierzytelnianie przy użyciu certyfikatów infrastruktury kluczy publicznych. Serwer musi podać certyfikat, który uwierzytelnia serwer na kliencie.

```
app.UseHttpsRedirection();
```

Rysunek 2: Protokół SSL

1.3 Zaimplementowane struktury danych

Następujące struktury danych zostały zaimplementowane, aby mogły być użyte jako zwracane wartości zapytań.

```
public enum RoomType : Int16 { SMALL, MEDIUM, BIG, VIP, DISABLED }

public class Room
{
    public Int16 RoomId { get; set; }

    public Int16 HotelId { get; set; }

    public RoomType RoomType { get; set; }

    public Int16 Floor { get; set; }

    public Int16 MaxCapacity { get; set; }

    public Room(Int16 room_id, Int16 hotel_id, RoomType room_type, Int16 floor, Int16 max_capacity)
    {
        RoomId = room_id;
        HotelId = hotel_id;
        RoomType = room_type;
        Floor = floor;
        MaxCapacity = max_capacity;
    }
}
```

Rysunek 3: Struktura 'Room'

```
public class Booking
{
    public Int16 BookingId { get; set; }

    public Int16 CustomerId { get; set; }

    public Int16 RoomId { get; set; }

    public DateTime RentDate { get; set; }

    public Int16 RentDuration { get; set; }

    public Int16 PersonCount { get; set; }

    public Booking(Int16 booking_id, Int16 customer_id, Int16 room_id, DateTime rent_date, Int16 rent_duration, Int16 person_count)
    {
        BookingId = booking_id;
        CustomerId = customer_id;
        RoomId = room_id;
        RentDate = rent_date;
        RentDuration = rent_duration;
        PersonCount = person_count;
    }
}
```

Rysunek 4: Struktura 'Booking'

1.4 Główne funkcjonalności

1.4.1 Rezerwacja pokoju

```
[HttpPost("AddBooking")]
Odwołania: 0
public async Task<int> AddBooking(short customerId, short roomId, DateTime rentDate, short rentDuration, short personCount)
{
    using var connection = new NpgsqlConnection(Db.masterConnectionString);
    await connection.OpenAsync();
    using var transaction = await connection.BeginTransactionAsync();

    try
    {
        using var cmd = new NpgsqlCommand
        {
            Connection = connection,
            Transaction = transaction,
            CommandText = "INSERT INTO booking (customer_id, room_id, rent_date, rent_duration, person_count) VALUES (@CustomerId, @RoomId, @RentDate, @RentDuration, @PersonCount);"
        };
        cmd.Parameters.AddWithValue("CustomerId", customerId);
        cmd.Parameters.AddWithValue("RoomId", roomId);
        cmd.Parameters.AddWithValue("RentDate", rentDate);
        cmd.Parameters.AddWithValue("RentDuration", rentDuration);
        cmd.Parameters.AddWithValue("PersonCount", personCount);

        var result = await cmd.ExecuteNonQueryAsync();
        await transaction.CommitAsync();
        return result;
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        throw;
    }
}
```

Rysunek 5: Rezerwacja pokoju

1.4.2 Usuwanie rezerwacji pokoju

```
[HttpDelete("RemoveBooking")]
Odwołania: 0
public async Task<int> RemoveBooking(short bookingId)
{
    using var connection = new NpgsqlConnection(Db.masterConnectionString);
    await connection.OpenAsync();
    using var transaction = await connection.BeginTransactionAsync();

    try
    {
        using var cmd = new NpgsqlCommand
        {
            Connection = connection,
            Transaction = transaction,
            CommandText = "DELETE FROM booking WHERE booking_id = @BookingId;"
        };
        cmd.Parameters.AddWithValue("BookingId", bookingId);

        var result = await cmd.ExecuteNonQueryAsync();
        await transaction.CommitAsync();
        return result;
    }
    catch (Exception)
    {
        await transaction.RollbackAsync();
        throw;
    }
}
```

Rysunek 6: Usuwanie rezerwacji pokoju

1.4.3 Przegladanie rezerwacji pokoi

```
[HttpGet("GetBookings")]
Odwołania: 0
public async Task<IEnumerable<Booking>> GetBookings()
{
    var result = new List<Booking>();
    using var connection = new NpgsqlConnection(Db.slaveConnectionString);
    await connection.OpenAsync();

    using var cmd = new NpgsqlCommand
    {
        Connection = connection,
        CommandText = "SELECT * FROM booking;"
    };

    using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        result.Add(new Booking(
            booking_id: reader.GetInt16(0),
            customer_id: reader.GetInt16(1),
            room_id: reader.GetInt16(2),
            rent_date: reader.GetDateTime(3),
            rent_duration: reader.GetInt16(4),
            person_count: reader.GetInt16(5)));
    }
    return result;
}
```

Rysunek 7: Przegladanie rezerwacji pokoi

1.4.4 Wyszukiwanie wolnych pokoi z filtrowaniem po ilości osób

Zaimplementowane zostały dwie wersje tej funkcji:

- Filtrowanie malejąco
- Filtrowanie rosnąco

```
[HttpGet("GetFreeRoomsByCapacityDesc")]
Odwołania: 0
public async Task<IEnumerable<Room>> GetFreeRoomsFilterRoomMaxCapacityDescending(DateTime rentStartDate, DateTime rentEndDate)
{
    var result = new List<Room>();
    using var connection = new NpgsqlConnection(Db.slaveConnectionString);
    await connection.OpenAsync();

    using var cmd = new NpgsqlCommand
    {
        Connection = connection,
        CommandText = @"
    SELECT * FROM room
    WHERE room_id NOT IN (
        SELECT room_id
        FROM booking
        WHERE rent_date <= @RentEndDate
        AND (rent_date + (rent_duration * INTERVAL '1 day')) >= @RentStartDate
    )
    ORDER BY max_capacity DESC;"
    };
    cmd.Parameters.AddWithValue("RentStartDate", rentStartDate);
    cmd.Parameters.AddWithValue("RentEndDate", rentEndDate);

    using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        result.Add(new Room(
            room_id: reader.GetInt16(0),
            hotel_id: reader.GetInt16(1),
            room_type: Enum.Parse<RoomType>(reader.GetString("room_type")),
            floor: reader.GetInt16(3),
            max_capacity: reader.GetInt16(4)));
    }
    return result;
}
```

Rysunek 8: Wyszukiwanie wolnych pokoi z filtrowaniem po ilości osób

1.4.5 Wyszukiwanie wolnych pokoi z filtrowaniem po standardzie

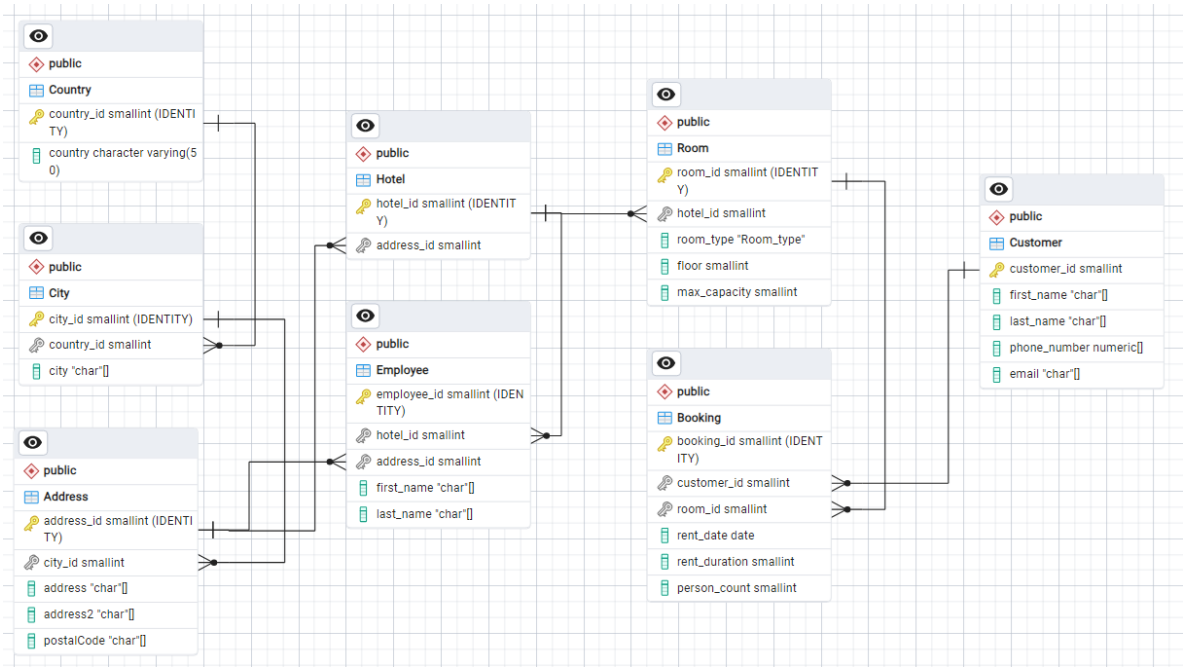
Zaimplementowane zostały dwie wersje tej funkcji:

- Filtrowanie malejąco
- Filtrowanie rosnąco

```
[HttpGet("GetFreeRoomsByTypeDesc")]
Odwołania: 0
public async Task<IEnumerable<Room>> GetFreeRoomsFilterRoomTypeDescending(DateTime rentStartDate, DateTime rentEndDate)
{
    var result = new List<Room>();
    using var connection = new NpgsqlConnection(Db.slaveConnectionString);
    await connection.OpenAsync();

    using var cmd = new NpgsqlCommand
    {
        Connection = connection,
        CommandText = @"
            SELECT * FROM room
    WHERE room_id NOT IN (
        SELECT room_id
        FROM booking
        WHERE rent_date <= @RentEndDate
        AND rent_date + (rent_duration * INTERVAL '1 day') >= @RentStartDate
    )
    ORDER BY room_type DESC;"
    };
    cmd.Parameters.AddWithValue("RentStartDate", rentStartDate);
    cmd.Parameters.AddWithValue("RentEndDate", rentEndDate);

    using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        result.Add(new Room(
            room_id: reader.GetInt16(0),
            hotel_id: reader.GetInt16(1),
            room_type: Enum.Parse<RoomType>(reader.GetString("room_type")),
            floor: reader.GetInt16(3),
            max_capacity: reader.GetInt16(4)));
    }
    return result;
}
```

Rysunek 9: Wyszukiwanie wolnych pokoi z filtrowaniem po standardzie

2 Schemat bazy danych



Rysunek 10: Schemat bazy danych

3 Zabezpieczenia SQL Injection

Dzieki wykorzystaniu funkcji cmd.Parameters.AddWithValue() zabezpieczono API przed niepożadanymi zapytaniami przez sql injection.

```
using var cmd = new NpgsqlCommand
{
    Connection = connection,
    Transaction = transaction,
    CommandText = "DELETE FROM booking WHERE booking_id = @BookingId;"
};
cmd.Parameters.AddWithValue("BookingId", bookingId);
```
[H]

Rysunek 11: Przykład zabezpieczenia przed SQL Injection

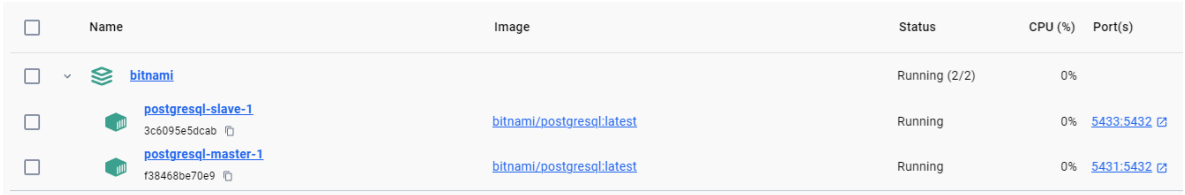
4 Zastosowanie transakcji

```
using var transaction = await connection.BeginTransactionAsync();

try
{
    using var cmd = new NpgsqlCommand
    {
        Connection = connection,
        Transaction = transaction,
        CommandText = "INSERT INTO customer (first_name, last_name, phone_number, email) VALUES
    };
    cmd.Parameters.AddWithValue("FirstName", firstName);
    cmd.Parameters.AddWithValue("LastName", lastName);
    cmd.Parameters.AddWithValue("PhoneNumber", phoneNumber);
    cmd.Parameters.AddWithValue("Email", email);

    var result = await cmd.ExecuteNonQueryAsync();
    await transaction.CommitAsync();
    return result;
}
catch (Exception)
{
    await transaction.RollbackAsync();
    throw;
}
```

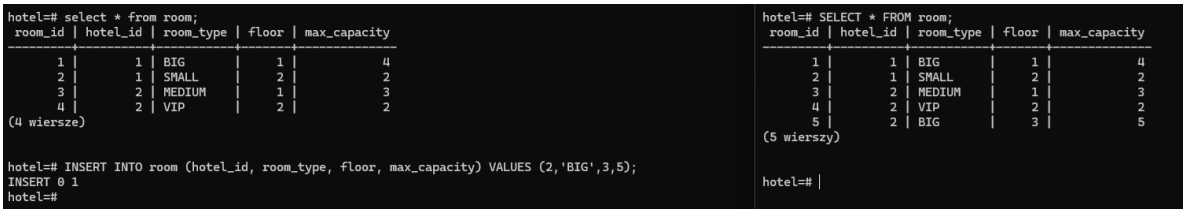
Rysunek 12: Zastosowanie transakcji

5 Replikacja Master-Slave

W Dockerze stworzono dwa kontenery postresql - jeden dla mastera i jeden dla slave. Skonfigurowano na nich replikacje master - slave oraz zmapowano ich porty tak aby można było dostać się do nich z zewnątrz.



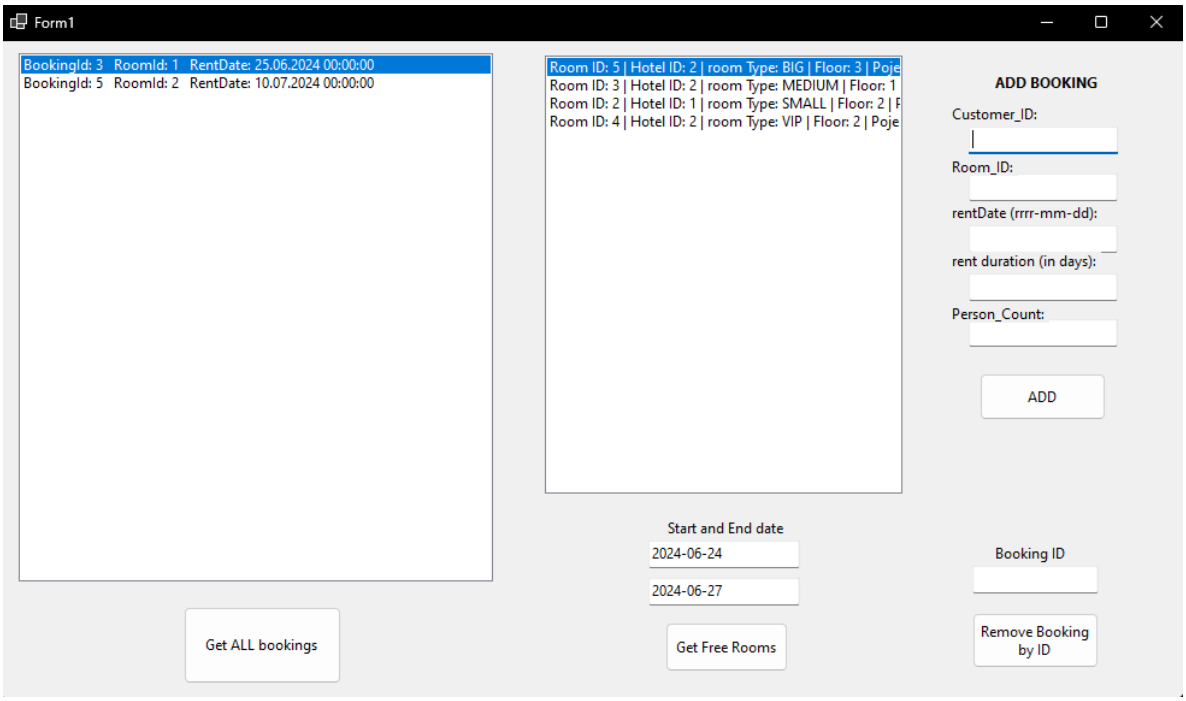
Rysunek 13: Stworzone kontenery w Dockerze



Rysunek 14: Działanie replikacji master slave

6 Aplikacja Desktopowa

6.1 Wygląd aplikacji



Rysunek 15: Wygląd aplikacji dostępowej

6.2 Przykład zapytania do API

Rysunek 16: Przykład działania API - Odczytanie wszystkich Bookingów

6.3 Uzycie API w aplikacji

Do pozyskiwania danych z API korzystamy z HttpClienta w C do wysyłania zapytań GET/POST itp. na adres naszego api.

```
private async void addbookingbutton_Click(object sender, EventArgs e)
{
    string call = "https://localhost:7269/HotelAssistant/AddBooking?" +
        "customerId=" + customerIDTEXT.Text + "&roomId=" + roomIdText.Text + "&rentDate=" + rentDateText.Text + "&rentDuration=" + rentDurationText.Text + "&personCoun

    var content = new StringContent("", Encoding.UTF8, "application/json");

    HttpResponseMessage response = await _httpClient.PostAsync(call, content);

    if (response.IsSuccessStatusCode)
    {
        string responseContent = await response.Content.ReadAsStringAsync();
        MessageBox.Show("Response content: " + responseContent, "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Error: " + response.StatusCode, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Rysunek 17: Przykład komunikacji z API