Uniwersytet Bielsko-Bialski

# LABORATORIUM

# Programowanie dla Internetu w technologii ASP.NET

**Sprawozdanie nr 4**

Serwisy

GRUPA: 2B / SEMESTR: 6 / ROK: 3

Igor Gawłowicz / 59096

## Warstwa serwisów

Na ostatnich laboratoriach naszym zadaniem było rozdzielenie pracy aplikacji na warstwę serwisów.

Wprowadzenie warstwy serwisów do aplikacji ASP.NET pozwala na lepszą organizację kodu, zwiększa jego czytelność i ułatwia utrzymanie aplikacji w przyszłości. Dodatkowo, separacja logiki biznesowej od logiki warstwy prezentacji pozwala na bardziej elastyczną rozbudowę i dostosowanie aplikacji do zmieniających się wymagań biznesowych.

Aby zaimplementować je w naszym programie utworzyłem nową ścieżkę w projekcie która nazwałem `Services` po czym dla każdego z koniecznych serwisów utworzyłem interfejs a także serwis sam w sobie

każdy z tych serwisów korzysta z wcześniej utworzonych repozytoriów, przez co kod w środku kontrolera uległ znacznemu zmniejszeniu poprzez rozdzielenie pracy programu na kolejną warstwę

Serwis odpowiedzialny za klientów oraz jego interfejs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using TripApp.Models;

namespace TripApp.Services
{
    public interface IClientService
    {
        Task<Client> GetOrCreateAsync(string name, string email, string phone);
    }

}
```

```
using System;
using System.Threading.Tasks;
using TripApp.Models;
using TripApp.Repositories;

namespace TripApp.Services
{
    public class ClientService : IClientService
    {
        private readonly IClientRepository _clientRepository;

        public ClientService(IClientRepository clientRepository)
        {
            _clientRepository = clientRepository;
        }

        public async Task<Client> GetOrCreateAsync(string name, string email,
string phone)
        {
```

```
            var existingClient = await _clientRepository.GetByEmailAsync(email);

            if (existingClient == null)
            {
                var newClient = new Client
                {
                    Name = name,
                    Email = email,
                    Phone = phone
                };
                await _clientRepository.AddAsync(newClient);
                return newClient;
            }

            existingClient.Name = name;
            existingClient.Phone = phone;
            await _clientRepository.UpdateAsync(existingClient);

            return existingClient;
        }
    }

}
```

Serwis odpowiedzialny za rezerwacje oraz jego interfejs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using TripApp.Models;

namespace TripApp.Services
{
    public interface IReservationService
    {
        Task CreateReservationAsync(Client client, int tripId);
    }
}
```

```
using System;
using System.Threading.Tasks;
using TripApp.Models;
using TripApp.Repositories;

namespace TripApp.Services
{

    public class ReservationService : IReservationService
    {
        private readonly IReservationRepository _reservationRepository;
```

```
        public ReservationService(IReservationRepository reservationRepository)
        {
            _reservationRepository = reservationRepository;
        }

        public async Task CreateReservationAsync(Client client, int tripId)
        {
            var reservation = new Reservation
            {
                ClientId = client.ClientId,
                ReservationDate = DateTime.Now,
                TripId = tripId
            };
            await _reservationRepository.AddAsync(reservation);
        }
    }
}
```

Serwis odpowiedzialny za wycieczki i jego interfejs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using TripApp.Models;

namespace TripApp.Services
{
    public interface ITripService
    {
        Task<IEnumerable<Trip>> GetAllTripsAsync();
        Task<Trip> GetTripByIdAsync(int id);
        Task AddTripAsync(Trip trip);
        Task UpdateTripAsync(Trip trip);
        Task DeleteTripAsync(int id);
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;
using TripApp.Models;
using TripApp.Repositories;

namespace TripApp.Services
{
    public class TripService : ITripService
    {
        private readonly ITripRepository _tripRepository;

        public TripService(ITripRepository tripRepository)
```

```
        {
            _tripRepository = tripRepository;
        }

        public async Task<IEnumerable<Trip>> GetAllTripsAsync()
        {
            return await _tripRepository.GetAllAsync();
        }

        public async Task<Trip> GetTripByIdAsync(int id)
        {
            return await _tripRepository.GetByIdAsync(id);
        }

        public async Task AddTripAsync(Trip trip)
        {
            await _tripRepository.AddAsync(trip);
        }

        public async Task UpdateTripAsync(Trip trip)
        {
            await _tripRepository.UpdateAsync(trip);
        }

        public async Task DeleteTripAsync(int id)
        {
            await _tripRepository.DeleteAsync(id);
        }
    }
}
```

Następnie musiałem zmodyfikować wszystkie kontrolery ponieważ jak wspomniałem wcześniej spora część kodu która była wcześniej w kontrolerze znalazła się teraz w serwisach.

Nie możemy zapomnieć też o zarejestrowaniu tych serwisów w aplikacji

```
builder.Services.AddScoped<IReservationService, ReservationService>();
builder.Services.AddScoped<IClientService, ClientService>();
builder.Services.AddScoped<ITripService, TripService>();
```

Po tym wszystkim aplikacja działa jak wcześniej jednak w tle jest bardziej rozbita na warstwy, dzięki czemu każda sfera programu jest zamknięta i hermetyczna.

## Wnioski

1. **Separacja odpowiedzialności (Separation of Concerns)**: Warstwa serwisów pomaga oddzielić logikę biznesową od logiki warstwy kontrolerów. Dzięki temu kontrolery stają się bardziej szczupłe i odpowiedzialne tylko za przetwarzanie żądań HTTP oraz przekazywanie danych do i z warstwy serwisów.

2. **Łatwiejsze testowanie (Testability)**: Warstwa serwisów umożliwia łatwiejsze testowanie aplikacji. Logika biznesowa w serwisach może być testowana bez konieczności symulowania żądań HTTP i kontekstu ASP.NET. Możemy testować logikę biznesową w warstwie serwisów za pomocą jednostkowych testów.

3. **Ponowne użycie kodu (Code Reusability)**: Przeniesienie logiki biznesowej do warstwy serwisów umożliwia jej ponowne użycie w różnych miejscach w aplikacji. Na przykład jeśli mamy dwie różne akcje kontrolera, które wymagają tej samej logiki biznesowej, możemy ją zaimplementować w jednym serwisie i wykorzystać w obu akcjach.

4. **Łatwiejsze zarządzanie zależnościami (Dependency Management)**: Korzystanie z wstrzykiwania zależności (Dependency Injection) w warstwie serwisów ułatwia zarządzanie zależnościami. Kontrolery nie muszą bezpośrednio tworzyć instancji serwisów, ale mogą je otrzymać wstrzyknięte przez kontener DI.

5. **Skalowalność i modularność (Scalability and Modularity)**: Wprowadzenie warstwy serwisów ułatwia skalowanie aplikacji poprzez podział jej na moduły. Każdy moduł może mieć swoją własną warstwę serwisów, co ułatwia zarządzanie kodem i jego rozbudowę w przyszłości.