Uniwersytet Bielsko-Bialski

LABORATORIUM

Programowanie dla Internetu w technologii ASP.NET

Sprawozdanie nr 4

Warstwa serwisów

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

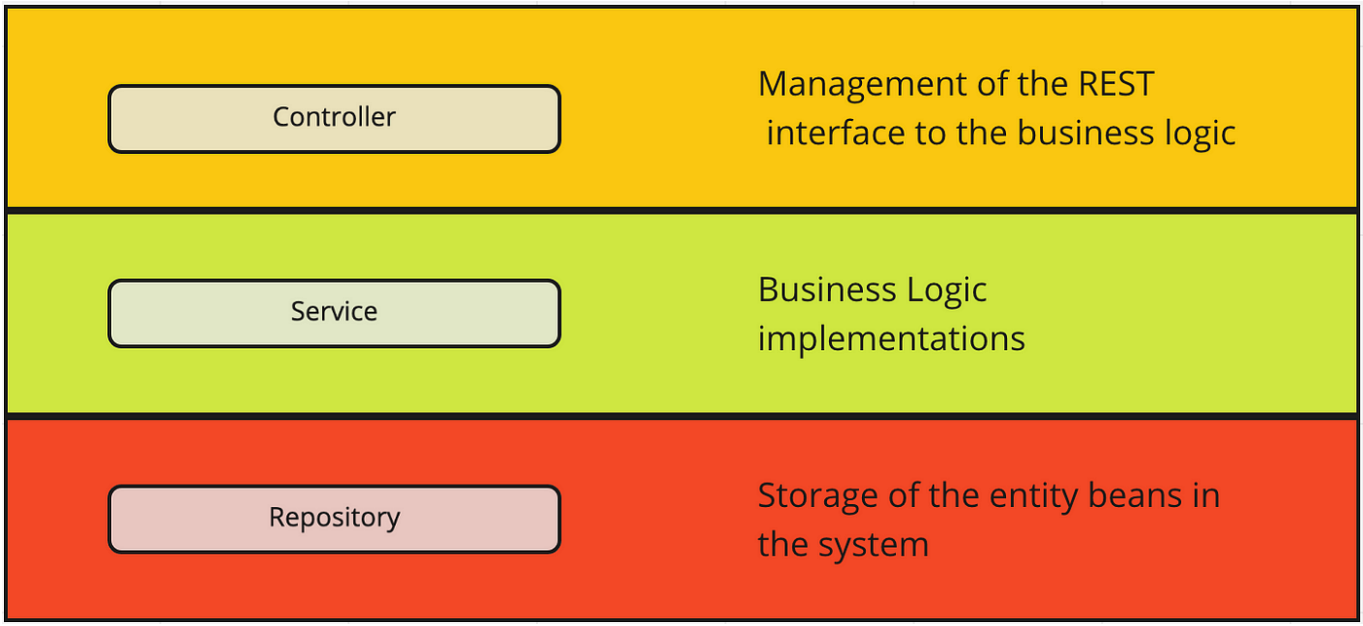
Kacper Lizak / 59443

Cel ćwieczenia

Celem ćwiczenia było dodanie warstwy serwisów do naszej aplikacji.

Wprowadzenie

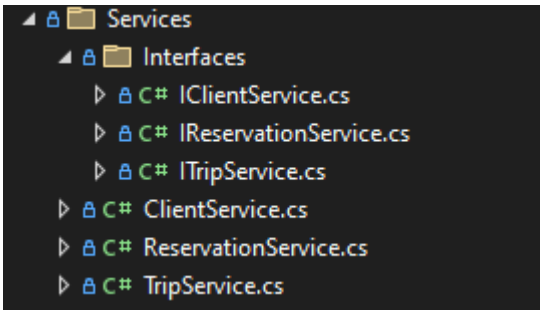
Serwisy w aplikacjach internetowych stosujemy, aby oddzielić logikę biznesową od warstwy prezentacji, co prowadzi do bardziej czytelnego i modułowego kodu. Serwisy umożliwiają łatwiejsze testowanie aplikacji poprzez izolację logiki biznesowej oraz zapewniają elastyczność poprzez wykorzystanie interfejsów i wstrzykiwanie zależności. Ich zastosowanie pozwala również na łatwe wprowadzanie zmian w implementacji, zachowując jednocześnie spójność z interfejsem



Realizacja

Podczas ćwiczenia utworzono folder dla serwisów oraz ich interfejsów, co pozwoliło na klarowne oddzielenie logiki biznesowej od warstwy kontrolerów. Przedstawiono przykładowy interfejs IClientService, definiujący podstawowe operacje na encji Client, takie jak pobieranie, dodawanie, aktualizowanie i usuwanie. Następnie zaimplementowano przykładowy serwis ClientService, który realizuje te operacje poprzez współpracę z repozytorium IClientRepository.

Utworzenie folderu dla Serwisów i ich interface'ów



Przykładowy interface ClientService

```
public interface IClientService
{
     Task<List<Client>> GetAllAsync();
     ValueTask<Client?> GetByIdAsync(int? id);
     Task InsertAsync(Client client);
     void Update(Client client);
     void Delete(Client client);
     Task SaveAsync();
     bool Exist(Client client);

}
```

Przykład ClientService

```
public class ClientService : IClientService
{
    private readonly IClientRepository _clientRepository;

    public ClientService(IClientRepository clientRepository)
    {
        this._clientRepository = clientRepository;
    }
    public void Delete(Client client)
    {
       _clientRepository.Delete(client);
    }

    public bool Exist(Client client)
    {
        return _clientRepository.Exist(client);
    }

    public async Task<List<Client>> GetAllAsync()
    {
        return await _clientRepository.GetAllAsync();
    }

    public async ValueTask<Client?> GetByIdAsync(int? id)
    {
```

```
        return await _clientRepository.GetById(id);
    }

    public async Task InsertAsync(Client client)
    {
        await _clientRepository.InsertAsync(client);
    }

    public async Task SaveAsync()
    {
        await _clientRepository.SaveAsync();
    }

    public void Update(Client client)
    {
         _clientRepository.Update(client);
    }
}
```

Modyfikacja controlera:

```
private readonly IClientService _clientServices;
public ClientsController(IClientService clientRepository)
{
    this._clientServices = clientRepository;
}
```

Wstrzyknięcie zależności w Program.cs

```
////////////// Services
builder.Services.AddScoped<IClientService, ClientService>();
builder.Services.AddScoped<ITripService, TripService>();
builder.Services.AddScoped<IReservationService, ReservationService>();
var app = builder.Build();
```

Wnioski

Warstwa serwisów umożliwia klarowne oddzielenie logiki biznesowej od warstwy prezentacji, co przekłada się na bardziej czytelny i łatwiejszy w zarządzaniu kod. Dodatkowo, stosowanie interfejsów i wstrzykiwanie zależności poprzez dependency injection sprawia, że aplikacja staje się bardziej elastyczna i łatwiejsza w testowaniu.