

Uniwersytet Bielsko-Bialski

LABORATORIUM

Programowanie dla Internetu w technologii ASP.NET

Sprawozdanie nr 6

Validator i AutoMapper

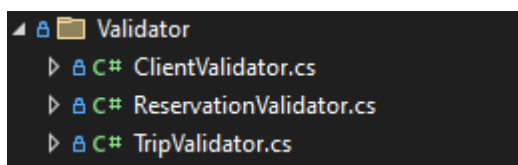
Cel ćwiczenia

Celem ćwiczenia było dodanie validatora i automappera do naszego projektu w ASP.NET

Wprowadzenie

Validator w ASP.NET jest narzędziem służącym do sprawdzania poprawności danych wejściowych w formularzach i żądaniach HTTP. Zapewnia to integralność danych i zabezpiecza przed nieprawidłowymi danymi wprowadzanymi przez użytkowników. Automapper natomiast to biblioteka ułatwiająca mapowanie danych między różnymi modelami w aplikacji, redukując ilość powtarzalnego kodu i zwiększając czytelność oraz elastyczność implementacji. Ich wspólne zastosowanie w projekcie ASP.NET przyczynia się do zwiększenia niezawodności, łatwości utrzymania i szybkości rozwoju aplikacji poprzez automatyzację procesów walidacji i mapowania danych.

Stworzenie folderu dla Validator'ów



Przykładowy validator dla Client

```
public class ClientValidator : AbstractValidator<ClientViewModel>
{
    public ClientValidator()
    {
        RuleFor(x => x.FirstName).NotEmpty().WithMessage("First name is required");
        RuleFor(x => x.LastName).NotEmpty().WithMessage("Last name is required");
        RuleFor(x => x.Email).NotEmpty().WithMessage("Email is required");
        RuleFor(x => x.Email).EmailAddress().WithMessage("Email is not valid");
        RuleFor(x => x.Phone).NotEmpty().WithMessage("Phone is required");
        RuleFor(x => x.Phone).Length(9).WithMessage("Phone must have 9 digits");
    }
}
```

Użycie validatora w kontrolerze

```
public async Task <IActionResult>
Create([Bind("IdClient,FirstName,LastName,Email,Phone")] ClientViewModel
clientViewModel)
{
    var _clientValidatorR = _clientValidator.Validate(clientViewModel);
    if (_clientValidatorR.IsValid)
    {
        var client = new Client
        {
            FirstName = clientViewModel.FirstName,
            LastName = clientViewModel.LastName,
            Email = clientViewModel.Email,
            Phone = clientViewModel.Phone,
            IdClient = clientViewModel.IdClient

        };
        await _clientServices.InsertAsync(client);
        await _clientServices.SaveAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(clientViewModel);
}
```

```
public async Task <IActionResult> Edit(int id,
[Bind("IdClient,FirstName,LastName,Email,PhoneNumber")] ClientViewModel
clientViewModel)
{
    if (id != clientViewModel.IdClient)
    {
        return NotFound();
    }
    var result= _clientValidator.Validate(clientViewModel);
    if (!result.IsValid)
    {
        foreach (var failure in result.Errors)
        {
            ModelState.AddModelError(failure.PropertyName, failure.ErrorMessage);
        }
    }
    if (result.IsValid)
    {
        var client = new Client
        {
            IdClient = clientViewModel.IdClient,
            FirstName = clientViewModel.FirstName,
            LastName = clientViewModel.LastName,
            Email = clientViewModel.Email,
            Phone = clientViewModel.Phone

        };
        try
```

```

    {
        _clientServices.Update(client);
        await _clientServices.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ClientExists(client.IdClient))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View(clientViewModel);
}

```

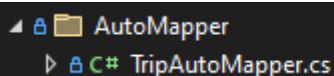
Dodanie validator'ów do program.cs

```

// Validators
builder.Services.AddScoped<IValidator<ClientViewModel>, ClientValidator>();
builder.Services.AddScoped<IValidator<TripViewModel>, TripValidator>();
builder.Services.AddScoped<IValidator<ReservationViewModel>, ReservationValidator>
();

```

Utworzenie folderu dla AutoMapper



```

└─ AutoMapper
   └─ TripAutoMapper.cs

```

Kod AutoMapper

```

public class TripAutoMapper : Profile
{
    public TripAutoMapper()
    {
        CreateMap<TripViewModel, Trip>()
            .ForMember(x => x.From, opt => opt.MapFrom(src =>
src.From.ToUpperInvariant()))
            .ForMember(x => x.To, opt => opt.MapFrom(src =>
src.To.ToUpperInvariant()))
            .ForMember(x => x.StartTrip, opt => opt.MapFrom(src =>
src.StartTrip.ToUniversalTime()))
            .ForMember(x => x.EndTrip, opt => opt.MapFrom(src =>
src.EndTrip.ToUniversalTime()))

```

```
        .ReverseMap();
        CreateMap<ClientViewModel, Client>()
            .ForMember(ClientViewModel => ClientViewModel.FirstName, opt =>
opt.MapFrom(src => src.FirstName.ToUpperInvariant()))
            .ForMember(ClientViewModel => ClientViewModel.LastName, opt =>
opt.MapFrom(src => src.LastName.ToUpperInvariant()))
            .ReverseMap();
        CreateMap<ReservationViewModel, Reservation>()
            .ForMember(ReservationViewModel =>
ReservationViewModel.ReservationDate, opt => opt.MapFrom(src =>
src.ReservationDate.ToUniversalTime()))
            .ReverseMap();
    }
}
```

Dodanie AutoMappera do buildera

```
builder.Services.AddAutoMapper(options =>
{
    options.AddProfile<TripAutoMapper>();
});
```

Przykład AutoMappera w ClientController

```
public async Task<IActionResult> Index()
{
    var clients = await _clientServices.GetAllAsync();

    var clientsList = _mapper.Map<List<Client>, List<ClientViewModel>>(clients);

    return View(clientsList);
}
```

Wnioski:

Użycie walidatora pozwala na skuteczną weryfikację danych wejściowych, zapewniając poprawność i kompletność informacji przekazywanych do aplikacji. Z kolei wykorzystanie AutoMappera usprawnia proces mapowania danych między obiektami, co redukuje ilość powtarzalnego kodu i ułatwia zarządzanie aplikacją poprzez automatyzację tego procesu. Kombinacja tych narzędzi znacząco zwiększa niezawodność i czytelność kodu oraz przyspiesza rozwój oprogramowania.