Anvendelse af "Port" mønsteret i Booking projektet

Vi skal nu i gang med at implementerer "Port" mønsteret fra "The Hexagonal Architecture".

"Port" mønsteret

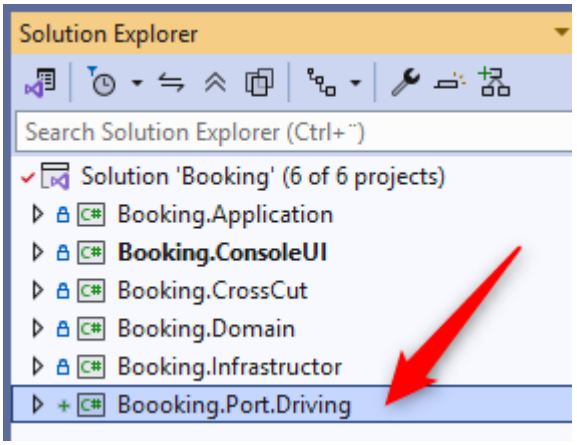
Den hexagonale arkitektur, også kendt som **Onion Architecture** og omtalt som **Ports and Adapters**, er et rent arkitektonisk mønster, der tillader input fra brugere eller eksterne systemer at komme ind i applikationen via en **Port** gennem en **Adapter**, og tillader output at blive sendt ud fra applikationen gennem en Port til en Adapter. Dette skaber et **abstraktionslag**, der beskytter kernen i en applikation og isolerer den fra ekstern interaktion.

Hvis man arbejder med softwaredesign, kan denne tilgang virkelig styrke modularitet og testbarhed.

Anvendelse af Port tankegangen støtter op omkring **The Entourage anti-pattern** og **The Stairway pattern**

I praksis indskydes et **Primære ports (driving ports)** lag. Disse definerer funktioner, som eksterne aktører (brugere, UI, CLI, tests) kan aktivere.

Konkret oprettes libary projektet `Boooking.Port.Driving` og samtidig skal der ændres i projekt referencerne således at `UI` og `Application` afhænger af `Boooking.Port.Driving`



`Booking.Application.csproj`

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\Booking.Domain\Booking.Domain.csproj" />
```

```
    <ProjectReference Include="..\Boooking.Port.Driving\Boooking.Port.Driving.csproj" /> <!-
-Ny reference til port-->
  </ItemGroup>

</Project>
```

`Booking.ConsoleUI.csproj`

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net9.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="9.0.8">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Booking.CrossCut\Booking.CrossCut.csproj" />
    <ProjectReference Include="..\Boooking.Port.Driving\Boooking.Port.Driving.csproj" /> <!-
-Ny reference til port-->
  </ItemGroup>

</Project>
```

Så flyttes de kontrakter der anvendes af UI til `Boooking.Port.Driving` og kontrakterne ændres således at kernen afkobles vha DTO objekter.

> **Note**
>
> En **DTO** – eller **Data Transfer Object** – er et simpelt objekt, der bruges til at **transportere data** mellem forskellige lag eller systemer i en applikation. Den indeholder kun data og **ingen forretningslogik**.
>
> DTO'er har flere fordele:
>
> - **Isolering af domænemodeller** DTO'er beskytter dine kerneobjekter (domænemodeller) mod at blive eksponeret direkte til eksterne systemer som UI eller API'er.
> - **Effektiv dataoverførsel** DTO'er kan sammensætte kun de nødvendige felter, hvilket reducerer mængden af data, der sendes over netværket – især vigtigt i distribuerede systemer.
> - **Tilpasning til klientens behov** Du kan lave forskellige DTO'er til forskellige formål, f.eks. . `UserSummaryDTO` og `UserDetailDTO`, afhængigt af hvad klienten har brug for.

- **Lettere vedligeholdelse og test** DTO'er gør det nemmere at teste og vedligeholde koden, fordi de er simple og uafhængige af forretningslogik.

Ved at inspicerer UI program.cs kan vi se hvilke kontrakter der anvendes udenfor kernen.

```
// See https://aka.ms/new-console-template for more information

using Booking.Application;
using Booking.CrossCut;
using Microsoft.Extensions.DependencyInjection;

Console.WriteLine("Hello, World!");

var serviceProvider = IocManager.RegisterService();
var bookingCommand = serviceProvider.GetService<IBookingCommand>(); // Reference til Core

bookingCommand.UpdateStartTid(1, DateTime.Now
                                    + new TimeSpan(0, 0, 30, 0)); // Kald til Core

Console.WriteLine("Done");
```

Det ses at `IBookingCommand` er det interface der anvendes af UI til at kalde ind i kernen (i dette tilfælde Application laget). Derfor skal `IBookingCommand` flyttes til `Boooking.Port.Driving` og der skal introduceres DTO'er til request og response data.

Boooking.Port.Driving

```
namespace Boooking.Port.Driving;

public interface IBookingCommand
{
    void UpdateStartTid(UpdateStartTidCommand command);
}

public record UpdateStartTidCommand(int Id, DateTime StartTid);
```

Application

`BookingCommandHandler` skal tilpasse introduktionen af `Boooking.Port.Driving`

```
using Boooking.Port.Driving;

namespace Booking.Application;
```

```
public class BookingCommandHandler : IBookingCommand
{
    private readonly IBookingRepository _repo;

    public BookingCommandHandler(IBookingRepository repo)
    {
        _repo = repo;
    }

    void IBookingCommand.UpdateStartTid(UpdateStartTidCommand command)
    {
        // Load
        var booking = _repo.GetBooking(command.Id);

        // Do
        booking.UpdateStartSlut(command.StartTid, booking.SlutTid);

        // Save
        _repo.SaveBooking(booking);
    }
}
```

UI

Tilpasset `program.cs`

```
// See https://aka.ms/new-console-template for more information

using Booking.Application;
using Booking.CrossCut;
using Boooking.Port.Driving;
using Microsoft.Extensions.DependencyInjection;

Console.WriteLine("Hello, World!");

var serviceProvider = IocManager.RegisterService();
var bookingCommand = serviceProvider.GetService<IBookingCommand>();

bookingCommand.UpdateStartTid(new UpdateStartTidCommand(1, DateTime.Now
                                                           + new TimeSpan(0, 0, 30, 0)));

Console.WriteLine("Done");
```

CrossCut

Tilpasning af `DependencyInjection.cs`

```
using Booking.Application;
using Booking.Domain.DomainService;
using Booking.Infrastructor;
using Booking.Infrastructor.Database;
using Boooking.Port.Driving;
using Microsoft.Extensions.DependencyInjection;

namespace Booking.CrossCut;

public static class DependencyInjection
{
    public static IServiceCollection AddBookingCore(this IServiceCollection services)
    {
        services.AddScoped<IBookingCommand, BookingCommandHandler>();
        services.AddScoped<IBookingRepository, BookingRepository>();
        services.AddScoped<IBookingOverlapCheck, BookingOverlapCheck>();

        // https://stackoverflow.com/questions/70273434/unable-to-resolve-service-for-type-%C2%A8microsoft-entityframeworkcore-dbcontextopti
        services.AddDbContext<BookingContext>();
        return services;
    }
}
```

Det store overblik