Servicelager för Tournaments APIet

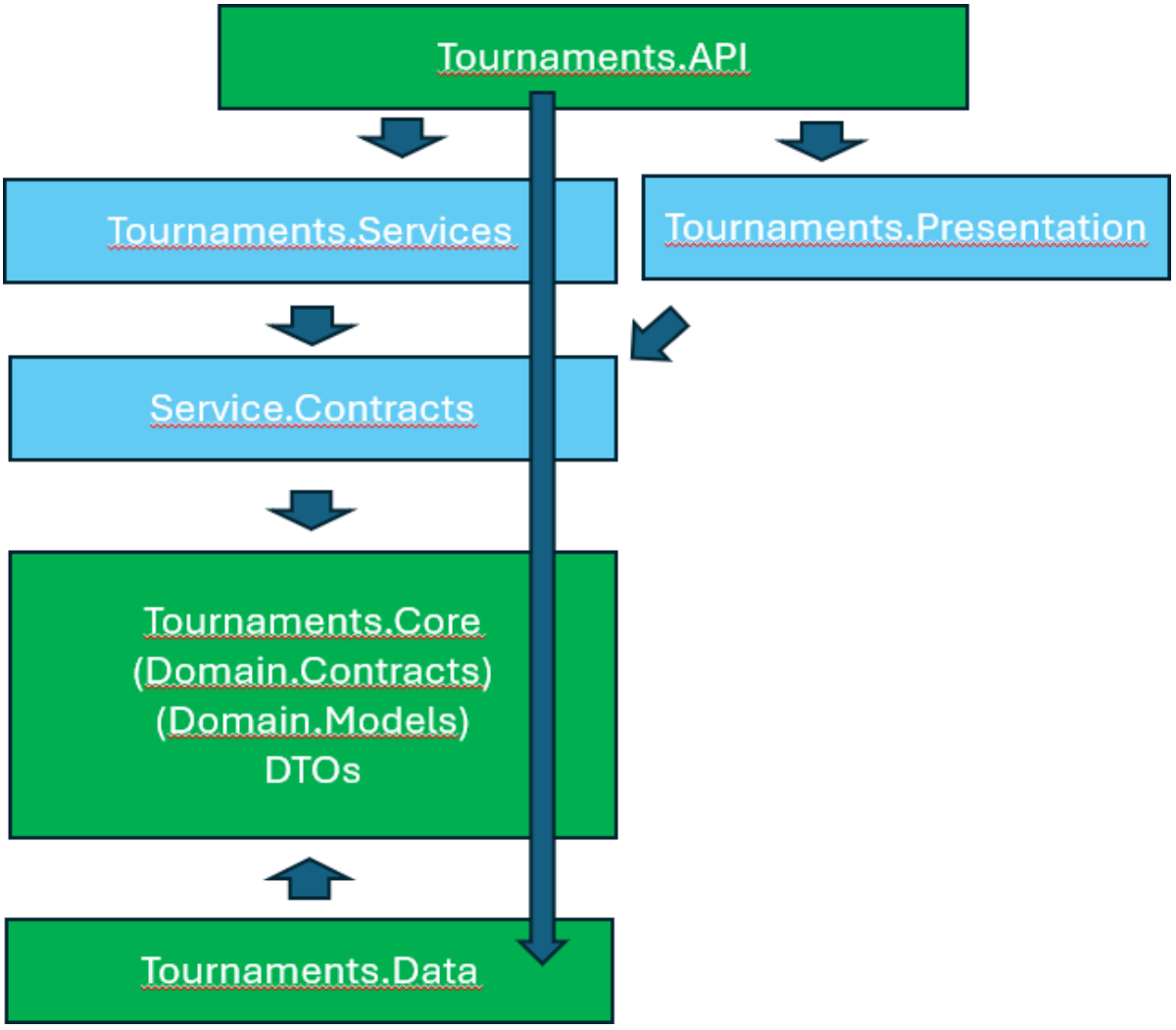
Ni har redan implementerat en grundläggande arkitektur Nu ska ni utöka funktionaliteten i ert turneringssystem och följa den strukturerade arkitekturen enligt bilden nedan.

De gröna klasserna är de som ni redan har på plats. Jag har förenklat arkitekturen för er då jag slagit ihop Domain.Models samt Domain.Contracts till ert nuvarande Tournaments.Core projekt

Ni behåller era DTOs i Core projektet. Så Shared projektet finns inte med i den här uppgiften.

(Ni får givetvis separera Core projektet till Domin.Models och Domain.Contracts precis som vi gjort under genomgångarna men det är inget krav)

Här kommer fokus vara på att implementera ett service lager. Samt börja arbeta med ett Presentations lager! Samt lite annat så klart.



Projektstruktur och Ansvarsområden

Tournament.API

Ansvar: Uppstart och konfiguration.

- Registrera services till Dependency Injection, AutoMapper, IServiceManager och andra services.
- Konfigurera och starta applikationen.
- Hantera request pipeline (Middleware)

Tournament.Presentation

Ansvar: Hantera controllers och exponera API:t endpoints

- Flytta TournamentsController och GamesController hit.
- Hantera HTTP-request och response

Tournament.Services

Ansvar: Implementera affärslogik.

- Hantera regler för turneringar och matcher.
- Interagera med Unit of Work och repositories via **IUnitOfWork**

Service.Contracts

Ansvar: Definiera interface för servicelagret.

- Interfaces för services (t.ex., **ITournamentService**, **IGameService**).

Tournaments.Core (Sammanslagning av Domain.Model och Domain.Contracts + DTOs)

Entities och DTOs folder (Domain.Models i vårt gemensamma projekt)

Innehåller modell klasser

- Entiteter: **Tournament** och **Game**

- DTOs för request och response

Contracts folder (Domain.Contracts i vårt gemensamma projekt)

- Interfaces för repositories och Unit of Work.

Tournament.Data

Ansvar: Implementera repositories, Unit of Work och AutoMapper-konfiguration.

- Hantera dataåtkomst.
- Konfigurera och tillhandahålla AutoMapper-profile

Krav

- Ni ska ha en **Servicemanager** som exponerar två interface **ITournamentService** och **IGameService**
- Ni ska implementera dessa två Services med passande metoder för de behov ni har. Tex hämta alla Tournaments

- I Servicelagret ska mappningen ske från Entiteter till DTOs (Ni väljer själva hur)
- **ServicManager** ska implementera interfacet **IServiceManager**
- Ni ska i både **TournamentsController** och **GameController** injecta **IServicemanager** med dependency injection

- Det får inte finnas några referenser till **Context**, **UnitOfWork** eller **Automapper** i någon av kontrollerna. Ni ska inte heller injecta någon av dessa services.
- Alla endpoints som returnerar kollektioner ska stödja möjligheten att ange hur många objekt som ska ingå i en respons (till exempel antal turneringar).
- Klienten kan specificera detta genom en queryparameter, exempelvis ?pageSize=20.
- API:et ska maximalt returnera **100 objekt per förfrågan**. Om klienten begär fler än 100 objekt (t.ex. pageSize=110), ska API:et istället returnera **100 objekt**, eftersom detta är det maximala stödet.
- Om klienten inte anger någon specifik pageSize, ska API:et använda en **standardstorlek** för antalet objekt som returneras (t.ex. 20 objekt per sida).
- Utöver själva objekten (DTO:er) ska API:et inkludera följande metadata:

  Totalt antal sidor (`totalPages`).

Nuvarande sidstorlek (`pageSize`).

Nuvarande sida (`currentPage`).

Totalt antal objekt (`totalItems`).

Denna information kan antingen läggas i **responsens body** eller i **headers**.

- Klienten ska kunna skicka information om sidstorlek (`pageSize`) och nuvarande sida (`page`) som **queryparametrar** i sitt request, exempelvis:

  GET /api/tournaments?page=2&pageSize=50

- Felhantering i Servicelager via exception alternativt custom resultobjekt.
- Implementera en affärsregel som begränsar antalet **Games** till max 10 per turnering. Valideringen ska göras när en ny match skapas eller läggs till.
- Vid olika typer av fel ska responet fortfarande vara av typen **ProblemDetails** med korrekt statuskod.
- Vid fel ska även **content-type** sättas till **Application/Json** i responsets header(För problemdetails)

Vissa saker har vi ännu inte gått igenom men så är det oftast att man ställs inför problem/uppgifter man inte gjort innan. En stor del av arbetet som utvecklare består av att söka information. Givetvis ska vi gå igenom dessa saker med men vi vill att ni försöker själva först!

Lycka till