Poject TrainingManager

Introductie en opgave

De bedoeling van dit project is om een applicatie te maken waarin we trainingsgegevens kunnen beheren. We starten met het beschrijven van de context en de business regels die daar bij horen. Verder volgt een woordje uitleg over de gekozen architectuur waarbij we gebruik maken van een gelaagde aanpak en ons gedeeltelijk baseren op de 'clean architecture'.

Het project wordt in de volgende paragrafen meer in detail uitgelegd en aspecten die in vorige leerpaden niet aan bod zijn gekomen worden verder uitgelegd.

De bedoeling is om dit project af te werken en de taken die nog open staan zijn :

- Het testen van de business laag (unit test / integration test)
- Het ontwerpen en maken van een User Interface via WPF.

Context

Voor dit project ontwikkelen we een tool die ons in staat stelt onze trainingen bij te houden. We onderscheiden daarbij looptrainingen en fietstrainingen. Voor een looptraining zijn mogelijk de volgende gegevens beschikbaar :

- het tijdstip (datum + tijdstip) wanneer de training is gestart (verplicht)
- De afstand (in m) (verplicht)
- De tijdsduur van de training (verplicht)
- Gemiddelde snelheid (niet verplicht), indien niet meegegeven wordt de gemiddelde snelheid afgeleid op basis van de vorige twee parameters
- De type training (verplicht), we onderscheiden hierbij 'uithouding', 'interval' en 'recuperatietraining'
- Commentaar, een vrij veld waarin info over de training kan meegegeven worden (niet verplicht)
- De looptraining krijgt ook een identificator (verplicht)

Voor een fietstraining worden de volgende gegevens voorzien :

- het tijdstip (datum + tijdstip) wanneer de training is gestart (verplicht)
- De afstand (in km) (niet verplicht)
- De tijdsduur van de training (verplicht)
- Gemiddelde snelheid (niet verplicht), indien niet meegegeven wordt de gemiddelde snelheid afgeleid op basis van de vorige twee parameters als de afstand is meegegeven
- Gemiddelde wattage (niet verplicht)
- De type training (verplicht), we onderscheiden hierbij 'uithouding', 'interval' en 'recuperatietraining'

- Commentaar, een vrij veld waarin info over de training kan meegegeven worden (niet verplicht)
- Fietstype (verplicht), we onderscheiden hier IndoorBike, RacingBike, CityBike en MountainBike
- De fietstraining krijgt een identificator (verplicht)

Opmerking : voor fietstrainingen is de afstand niet verplicht, omdat dat voor indoor trainingen minder relevant is, daar is vooral het vermogen (Watt) en tijdsduur belangrijk.

De tool die we wensen te ontwikkelen moet over een user interface beschikken die nieuwe trainingen kan toevoegen, maar die ook een overzicht kan geven van de verschillende trainingen op maandbasis. Dit overzicht kan zowel enkel fiets- of looptrainingen bevatten als een combinatie van beiden. Wanneer we dit overzicht tonen is het belangrijk dat de trainingen gesorteerd zijn op datum. Bij elke overzicht wensen we ook steeds de 'beste' trainingen te tonen (niet specifiek voor de geselecteerde maand). Voor de looptrainingen zijn dat dan de sessie met de langste afstand (indien er meerdere zijn, sorteren we op gemiddelde snelheid) en de sessie met de hoogste gemiddelde snelheid. Wat betreft de fiets sessies wensen we telkens de sessie te zien met de langste afstand (ook hier geldt dat als er meerdere zijn we diegene kiezen met de hoogste gemiddelde snelheid), de sessie met de hoogste gemiddelde snelheid (indien meerdere sorteren op afstand) en de sessie met de hoogste wattage (indien meerdere sorteren op tijdsduur).

Daarnaast moet het ook mogelijk zijn om de laatste sessies te tonen (instelbaar hoeveel).

Als laatste requirement moet het ook mogelijk zijn om zowel fiets- als looptrainingen te verwijderen.

Technologiekeuze

De tool moet ontwikkeld worden in C# met .NET Core. Voor de user interface wordt er gebruik gemaakt van WPF en voor de opslag van de gegevens maken we gebruik van SQL server en Entity Framework Core.

Architectuur

Het hoofddoel van architectuur is om de levenscyclus van de software te ondersteunen. Een goede architectuur zorgt ervoor dat het systeem eenvoudig te begrijpen is, eenvoudig te ontwikkelen, eenvoudig is in onderhoud en eenvoudig inzetbaar (deployable) is. Het minimaliseren van de lifetime cost en het verhogen van de productiviteit van de programmeur staan dus centraal.

Er zijn verschillende architecturen beschreven elk met hun voor- en nadelen en afhankelijk van de omstandigheden worden er keuzes gemaakt. In dit project baseren we ons op een aantal aspecten van 'clean architecture' en splitsen we op in verschillende lagen. We gaan daarbij een presentatielaag onderscheiden, een business laag – die we centraal zetten – en een data laag. Voor de datalaag gaan we gebruik maken van het 'repository pattern' en het 'unit of work' pattern.

Schematisch ziet onze oplossing er als volgt uit : zowel de databank als de user interface zijn afhankelijk van de business rules

(Clean Architecture – A craftsman's guide to software structure and design. Robert C. Martin. Prentice Hall.)

Ontwerp

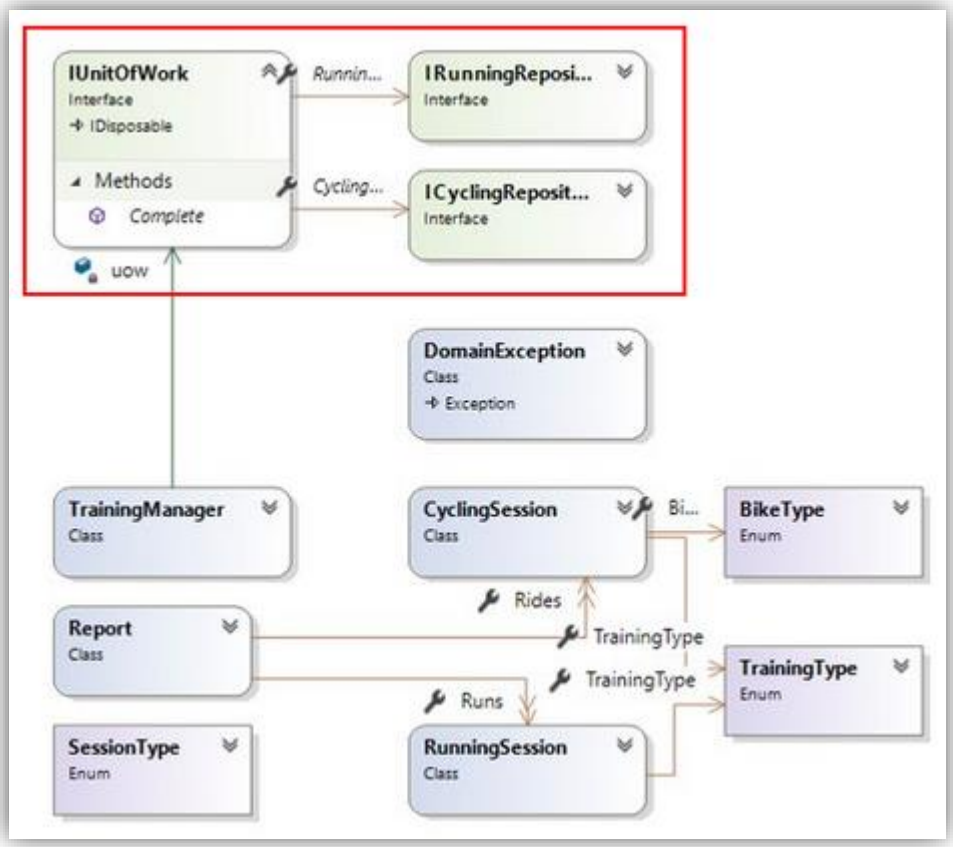
We starten met de aanmaak van een 'blank solution' in Visual Studio waarin we voor elk van de lagen een apart project aanmaken. Zowel voor de datalaag als de domeinlaag kiezen we voor een library project.

Business layer / domain layer

In onze domeinlaag vinden we de volgende elementen :

- CyclingSession en RunningSession die de informatie over de loop- en fietstrainingen bevat
- Enums voor het type fiets, het type training en het type sessie
- Een klasse Report die voor een bepaalde periode niet alleen alle trainingsessies bevat, maar ook de 'beste' trainingen, een aantal totalen en een tijdslijn van trainingen
- Er is een specifieke exceptions klasse voorzien
- De TrainingManager klasse die er voor zorgt dat de user stories kunnen worden uitgevoerd

Daarnaast definiëren we in onze domeinlaag ook de link naar de datalaag (repository/unit of work pattern) door middel van het definiëren van de nodige interfaces. Door gebruik te maken van interfaces maken we onze domeinlaag onafhankelijk van de datalaag. In onze IUnitOfWork interface voorzien we enkel een methode Complete en twee properties die de IRunningRepository en ICyclingRepository implementeren.



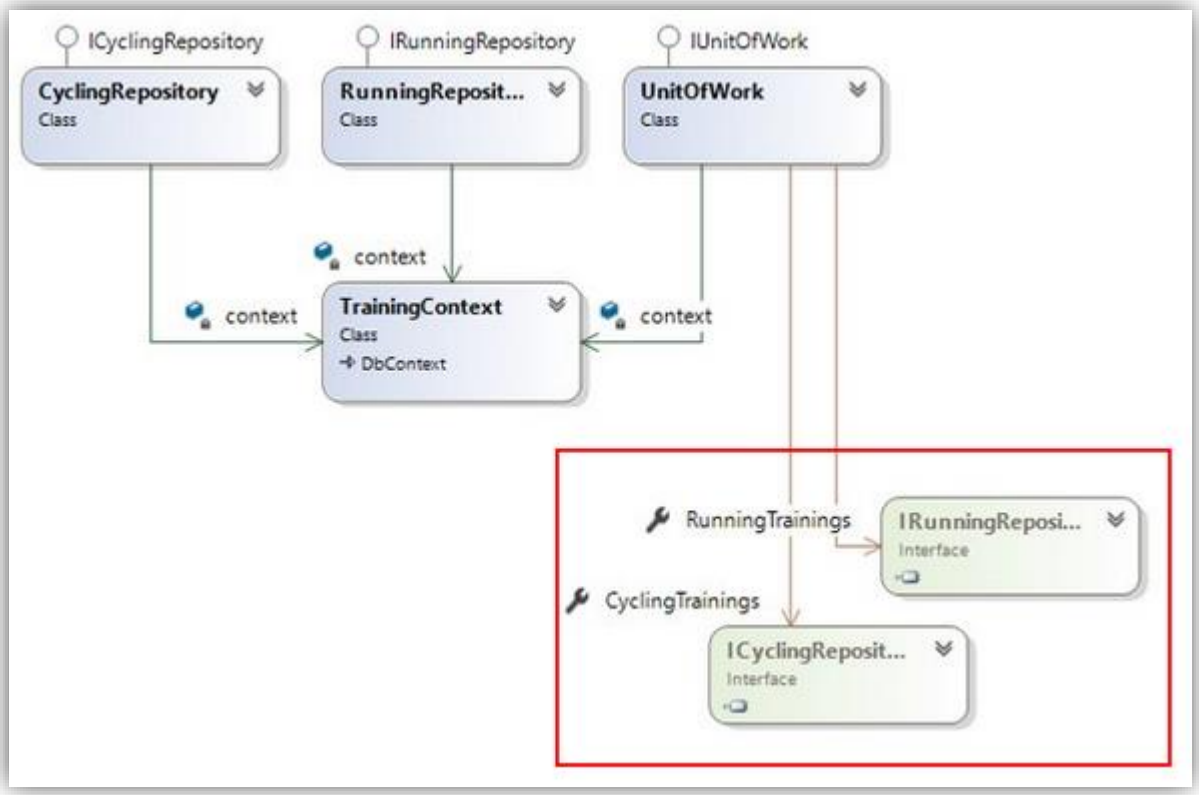
In de klasse Report is er een variabele TimeLine die alle trainingen (zowel fietsen als lopen) bevat en gesorteerd is op basis van de starttijd van de training. De list bestaat uit Tuples met als eerste item het type sessie (fiets of lopen) en als tweede item een CyclingSession of RunningSession (als algemeen type Object).

```
public IList<Tuple<SessionType, Object>> TimeLine { get; set; }
    = new List<Tuple<SessionType, Object>>();
```

Data layer

Voor de datalaag maken we gebruik van het 'repository pattern' en het 'unit of work pattern', een duidelijke uitleg met voorbeeld vinden we terug in het volgende YouTube filmpje https://www.youtube.com/watch?v=rtXpYpZdOzM en op https://programmingwithmosh.com/net/common-mistakes-with-the-repository-pattern/

Het klassenmodel ziet er als volgt uit :



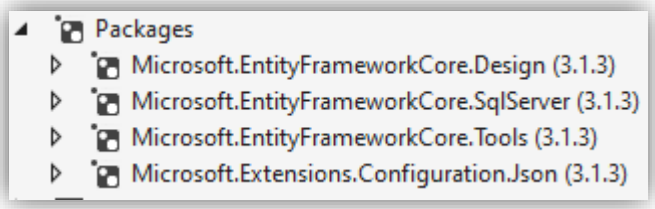
We hebben de twee repositories die respectievelijk de ICyclingRepository en de IRunningRepository implementeren. Daarnaast hebben we ook de klasse UnitOfWork die de IUnitOfWork interface implementeert met verwijzingen naar de repository interfaces. En tenslotte hebben we ook nog de TrainingContext klasse die de link maakt naar de databank met behulp van Entity Framework Core.

We gaan even dieper in op de klasse TrainingContext. In deze klasse voorzien we een parameter connectionstring die we uit een configuratiebestand gaan halen in plaats van deze hard coded op te nemen.

```
public class TrainingContext : DbContext
{
    private string connectionString;

    0 references
    public TrainingContext()...

    4 references | 1/1 passing
    public TrainingContext(string db="Production") ...
    2 references
    private void SetConnectionString(string db = "Production")...

    9 references
    public DbSet<CyclingSession> CyclingSessions { get; set; }
    8 references
    public DbSet<RunningSession> RunningSessions { get; set; }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)...
}
```

De volgende methode zorgt ervoor dat het configuratiebestand appsettings.json kan worden ingelezen. We moet daarbij wel de nodige NuGet packages installeren en een referentie leggen naar de DomainLibrary.

```
private void SetConnectionString(string db = "Production")
{
    var builder = new ConfigurationBuilder();
    builder.AddJsonFile("appsettings.json", optional: false);

    var configuration = builder.Build();
    switch (db)
    {
        case "Production":
            connectionString = configuration.GetConnectionString("ProdSQLconnection").ToString();
            break;
        case "Test":
            connectionString = configuration.GetConnectionString("TestSQLconnection").ToString();
            break;
    }
}
```




```
{
  "ConnectionStrings": {
    "ProdSQLconnection": "Data Source=lenovo-pc\\sqlexpress;Initial Catalog=trainingDB;Integrated Security=True",
    "TestSQLconnection": "Data Source=lenovo-pc\\sqlexpress;Initial Catalog=trainingDBtest;Integrated Security=True"
  }
}
```

De klasse bevat ook twee constructors, een lege constructor (nodig voor de migration tools) en een constructor waarin we kunnen aangeven met welke databank we wensen te werken. We onderscheiden voorlopig enkel een productie- en testdatabank, waarbij we default voor productie kiezen.

```
public TrainingContext()
{
}

4 references | 1/1 passing
public TrainingContext(string db="Production") : base()
{
    SetConnectionString(db);
}
```

De OnConfiguration methode zet de optie dat we met een SQL Server databank gaan werken. Om de migration tools die gebruik maken van de lege constructor gevolg door de OnConfiguration methode te kunnen gebruiken, moeten we ervoor zorgen dat de parameter connectionString zeker is ingevuld, vandaar dat we dit ook toevoegen in deze methode.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if(connectionString==null)
    {
        SetConnectionString();
    }
    optionsBuilder.UseSqlServer(connectionString);
}
```

Presentation layer

[het ontwerp van de schermen is vrij te kiezen – maar de user stories uit de paragraaf 'context' moeten wel uitvoerbaar zijn]

Testen

Voor de functionaliteit van de businesslaag te testen kunnen we gebruik maken van Unit Testen. Daarvoor kan een project worden toegevoegd aan de solution.

Wensen we ook de repositories in combinatie met de businesslaag te testen (of een volledige use case te simuleren) dan gebruiken we geen unit testen maar integration tests. Ongeacht het type tests dat we wensen te doen kunnen we in Visual Studio kiezen om een MS Test Project te gebruiken. Voor de rest blijft de aanpak en opzet hetzelfde. Om vlot te weken kan je gebruik maken van een extra klasse TrainingContextTest die overerft van TrainingContext waarin we er telkens voor zorgen dat we met een lege test databank kunnen werken. Op die manier ben je niet afhankelijk van de toestand waarin je de test databank hebt achtergelaten.

```
public class TrainingContextTest : TrainingContext
{
    2 references | 2/2 passing
    public TrainingContextTest(bool keepExistingDB = false) : base("Test")
    {
        if (keepExistingDB)
        {
            Database.EnsureCreated();
        }
        else
        {
            Database.EnsureDeleted();
            Database.EnsureCreated();
        }
    }
}
```

(https://en.wikipedia.org/wiki/Software_testing)

De broncode kan gevonden worden op https://github.com/tvdewiel/TrainingManagerProject