DNP1 Re-Exam, February 2024

VIA University College
Software Technology Engineering
Written individual examination in DNP1
(3 hours)

The Danish version comes further down, after the English.
Den danske udgave kommer længere nede, efter den engelske.

You are allowed to use any IDE (Visual Studio, VS Code, Rider, etc) and to browse the Internet for information (**however**, no live communication, e.g. chats, or contact with any other people, no uploading to public git repository, no use of ChatGPT or similar AI).

This is an individual exam!

When finished, hand in your ***entire solution in a zip file*** to WiseFlow.

In general, you must remember to follow the conventions, theory, and best practices taught in class, such as (but not limited to) asynchronous programming, dependency injection, at least minimal error handling, RESTful endpoints, efficient data access. Read the exercise description thoroughly.

Point evaluation

The exam set is split into 3 separate, unrelated exercises. You can do the exercises in any order.

Various parts (component/project, package, class, method, etc.) is worth a number of points. You can score a total maximum of 100 points.

It is not "all or nothing", so if part of something works, but not entirely, you may still get a reduced number of points for that.

The points are awarded per exercise roughly as follows:

1. Blazor: 35%
2. Web API: 35%
3. Entity Framework: 30%

Setup solution

First, you need to create a new solution. Your study number must be included in the solution name, an example could be:

”DnpExam123456”

You will through this exam create a *total of three projects*: Blazor, WebApi, EFC.

Do not create extra projects, if not specifically stated.

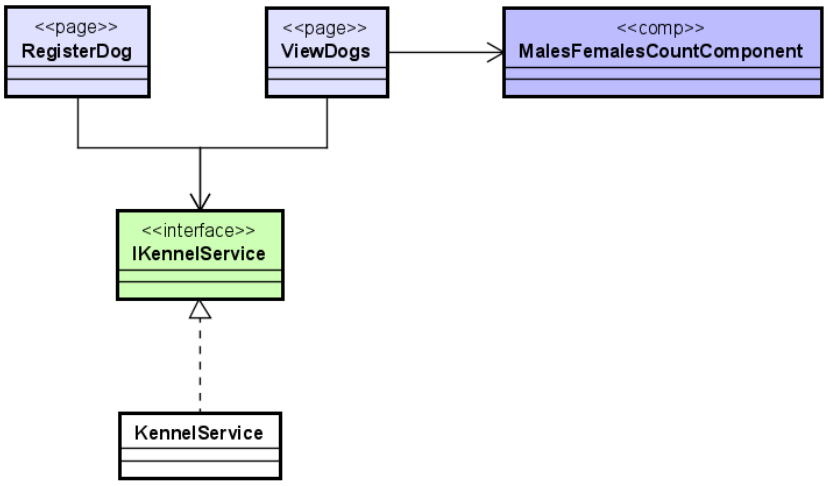
Exercise 1 – Blazor

In this exercise you will create a stand-alone Blazor app. It is a small application for a Dog Adoption Kennel.

You may use Blazor-Server or Blazor-WASM.

Asynchronous programming is optional in this exercise.

Below is a simplified, incomplete class-diagram, which must be followed. You will add more to your system than is shown in the diagram.



1.1 Create Blazor Project

Create a new project, call it "Blazor".

1.2 Entity

Create the following data model class, call it "Dog". It must have the following properties:

- Id
- Name
- Sex
- Breed: E.g. "Labrador Retriever", or "Västgötaspits", or "Poodle", etc.
- ImageUrl: Just the URL to an image
- Description
- ArrivalDate

Use appropriate types. Place the class somewhere reasonable in your Blazor project.

1.3 KennelService with dummy data

Create an interface, IKennelService.

Then create an implementation, KennelService.

Place both interface and class somewhere reasonable.

The KennelService must initially create at least 5 Dogs, with varying details, and keep these dogs in a list.

The service must be registered to be injected in various pages, etc.

1.4 Register dog

Create a new page, call it “RegisterDog”.

This page must contain input for the following Dog properties: Name, Sex, Breed, ImageUrl, Description.

There must be a button, which when clicked will pass a Dog instance to the IKennelService. Define an appropriate method on this interface.

The implementation of this method in KennelService must then:

1) Set the ArrivalDate to the current date.
2) In a reasonable way set the Id to the next available value.
3) Add the Dog to a list of Dogs in the class.

When the dog has been created, display a message for the user.

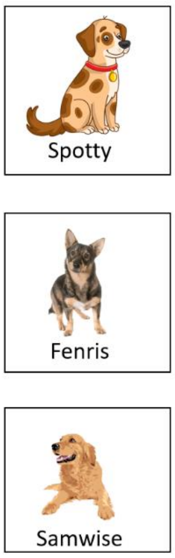
The page must be added to the navigation menu.

1.5 View dogs

Create a new page, call it “ViewDogs”.

This page must show a list of the dogs, ordered by earliest arrival date first. So, the dog which entered the kennel first is shown on top.

The view must initially look something like the following image:



It just shows a list of the Dogs, showing image and name.

You must then be able to click on a dog, and the rest of the details of that specific dog must be shown somewhere on the page.

This page must be added to the navigation menu.

1.6 Component that shows # male/female dogs

In this exercise you must make a *component*, to be used by the page in the previous exercise 1.5.

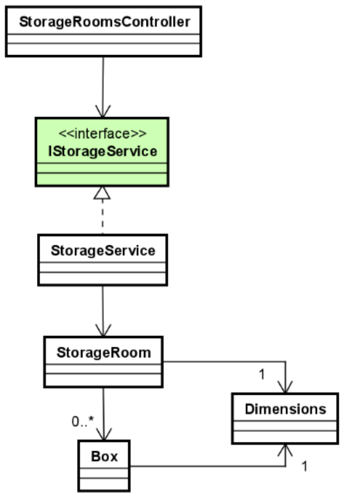
Create a component, it must use the IKennelService. It must display the number of male dogs and female dogs.

Put the component at the top of the ViewDogs page.

Exercise 2 – Web API

In this exercise, you will create a Web API to manage storage rooms and their content.

Below is a simplified, incomplete class diagram of the most relevant parts. This must be followed.



Asynchronous programming is optional in this exercise.

2.1 Create Web API project

Create a new Web API project that uses controllers for routing. Name it "WebApi".

2.2 Entities

Create entities StorageRoom and Box. They both use a third object type, called Dimensions.

They have the following properties:

1. StorageRoom:
    a. Id
    b. Location, e.g. "row 4, aisle 7"
    c. Dimensions, i.e. an instance of this object.
    d. Boxes, i.e. the list of Boxes.
2. Box
    a. Id
    b. Label, e.g. "Kitchen", or "Old books", etc.
    c. Dimensions

Finally, the Dimensions class must have the properties Length, Width, Height.

Place the classes somewhere relevant.

2.3 Storage Service

Create the StorageService class (with interface), which:

1. Contains a List of StorageRooms
2. Initially seeds the list with at least five rooms.
3. Put some boxes in some of the rooms

The StorageService must be set up for dependency injection, so that it can be used in the controller class.

Place the class and interface somewhere relevant.

2.4 Controller

2.4.1 Create rooms controller

Create the controller class. Each action in the controller must delegate the logic handling to relevant methods in the IStorageService interface (implemented in the class.

The controller must have endpoints as described in the following. You must use RESTful conventions for routing.

2.4.2 *Add box to storage room*

You must already have some rooms. This endpoint must take a Box, to be added to an existing StorageRoom.

The StorageService must be responsible for setting the ID of the incoming Box.

You do not have to worry about whether there is enough space in the room. Assume this is handled outside of the system.

You must include relevant error handling.

2.4.3 *See all boxes in a specific room*

This endpoint must return a collection of the boxes in a room, specified by id.

You must include relevant error handling.

2.4.4 *Remove boxes in a specific room*

This endpoint must receive relevant information, so that a specific box can be removed from a specific room.

You must include relevant error handling.

2.4.5 *Query with filter*

This endpoint returns a collection of rooms. It must make it possible to filter the rooms by two criteria:

a) Include rooms with a cubic meter volume greater than a specific number. (disregard any potential boxes in the storage room)
b) Include rooms with less than a specific number of boxes.

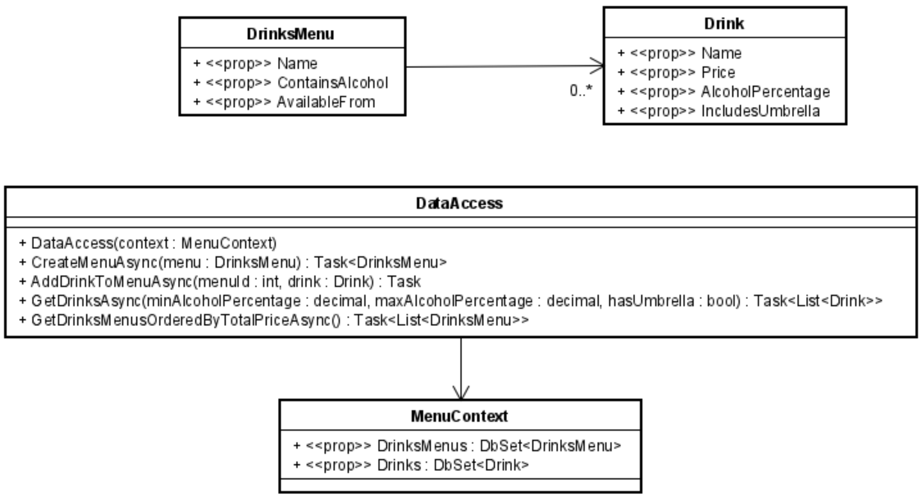
Both filter arguments must be optional, and it must be possible to apply none, either, or both.

Exercise 3 – Entity Framework

In this exercise you will create a database, and write a few methods to exemplify the usage of the database.

You must use Entity Framework Core and SQLite .

Here is a simplified diagram of what the exercise will contain:


DrinksMenu
+ <<prop>> Name
+ <<prop>> ContainsAlcohol
+ <<prop>> AvailableFrom
0..*
Drink
+ <<prop>> Name
+ <<prop>> Price
+ <<prop>> AlcoholPercentage
+ <<prop>> IncludesUmbrella
DataAccess
+ DataAccess(context : MenuContext)
+ CreateMenuAsync(menu : DrinksMenu) : Task<DrinksMenu>
+ AddDrinkToMenuAsync(menuId : int, drink : Drink) : Task
+ GetDrinksAsync(minAlcoholPercentage : decimal, maxAlcoholPercentage : decimal, hasUmbrella : bool) : Task<List<Drink>>
+ GetDrinksMenusOrderedByTotalPriceAsync() : Task<List<DrinksMenu>>
MenuContext
+ <<prop>> DrinksMenus : DbSet<DrinksMenu>
+ <<prop>> Drinks : DbSet<Drink>

Notice some methods are asynchronous, and you must use asynchronous programming in this exercise, where possible.

3.1 Create console project

Create a new Console Application Project, call it “EFC”.

3.2 Entities

Create the DrinksMenu and Drink entities in an appropriate directory.

You must define a good primary key for both entities.

The DrinksMenu has the following properties:

- Name, e.g. “Juices”, “Wines”, “Soft Drinks”, etc.
- ContainsAlcohol. This indicates whether the drinks contain alcohol or not.
- AvailableFrom. This is a time of day, because some drinks are not available until noon or evening.

The Drink has the following properties:

- Name, e.g. “Long Island Ice Tea”, “Orange Juice”, etc.
- Price, e.g. “42.50”, or “17.95”, etc.
- AlcoholPercentage, e.g. “4.6”, or “11.0”, etc.

- IncludesUmbrella, i.e. whether this particular drink comes with a little umbrella.

3.3 DbContext

Add EFC NuGet packages to the project.

Create the MenuContext. It must derive from DbContext. It must define relevant DbSet properties.

Then create the database.

You must use SQLite.

3.4 Data Access

Create the DataAccess class, for now only with its constructor.

3.5 Create DrinksMenu

Implement the CreateDrinksMenu(..) method on the DataAccess class. It receives a DrinksMenu as parameter and inserts it into the database, using the MenuContext.

In the Program.cs file, give an example of how this method is used. Create at least three menus.

3.6 Add Drink to DrinksMenu

Implement the AddDrinkToDrinksMenu(..) method on the DataAccess class. The method must add a new Drink to an existing DrinksMenu.

In the Program.cs file, give an example of how this method is used. Add at least two Drinks to each menu.

3.7 Get Drinks with filter

Implement the GetDrinks(..) method on the DataAccess class.

It must receive three optional parameters. Two of the parameters filter Drinks which have an alcohol percentage larger than minimumAlcoholPercentage and less than maximumAlcoholPercentage, and the last filter filters whether or not the drink has an umbrella. Return drinks which match the parameters.

The filter parameters are all optional and must be able to be applied with none, some, or all of them.

In the Program.cs file, give an example of how this method is used.

3.8 Get DrinksMenus ordered by price

Implement the GetDrinksMenusOrderedByTotalPrice(..) method on the DataAccess class.

It must return all the DrinksMenus in the database, but ordered by their total price. The most expensive DrinksMenu comes first.

The total price of a DrinksMenu is defined as the sum of the price of all Drinks in the DrinksMenu.

In the Program.cs file, give an example of how this method is used.

This is the end of the exam-exercise. Good luck.

DNP1 Reeksamen, Februar 2024

VIA University College
Software Ingeniør
Skriftlig individual eksamen iDNP1
(3 timer)

Det er tilladt at bruge en vilkårlig IDE (Visual Studio, VS Code, Rider, etc) og at søge på internettet efter information (**men!** Ingen kommunikation med andre, som fx chats, eller anden form for live kontakt med andre. Du må ikke upload din kode til et public git repository. Du må ikke bruge ChatGPT eller lignende AI).

Dette er en individuel eksamen!

Når du er færdig, skal du aflevere ***hele din solution i en zip fil*** to WiseFlow.

Generelt skal du huske at følge konventioner, teori, bedste praksis undervist i timerne, så som (men ikke begrænset til) asynkron programmering, afhængighedsindskydning, i hvert fald minimum fejlhåndtering, RESTful endpoints, effektiv data håndtering. Læs opgavebeskrivelsen grundigt.

Point evaluering

Dette eksamenssæt er inddelt i 3 separate, urelaterede opgaver. Du kan vælge at løse de 3 opgaver i vilkårlig rækkefølge.

Forskellige dele (f.eks. komponenter, projekter, pakker, klasser, metoder, osv.) kan give et antal point. Du kan score maksimalt 100 point.

Det er ikke "alt eller intet", så hvis noget af en delopgave virker, men ikke helt, så kan du måske stadig få et antal reducerede point for denne halve løsning.

Points for opgaverne er fordelt ca. som følger:

1. Blazor: 35%
2. Web API: 35%
3. Entity Framework: 30%

Opsætning af solution

Du skal oprette en ny solution. Dit studie-nummer skal være en del af solution-navnet, det kunne f.eks. være:

"DnpExam123456"

Du kommer gennem denne eksamen til at oprette i alt tre projekter: Blazor, WebApi, EFC.

Du skal ikke oprette projekter, der ikke specifikt er efterspurgt.

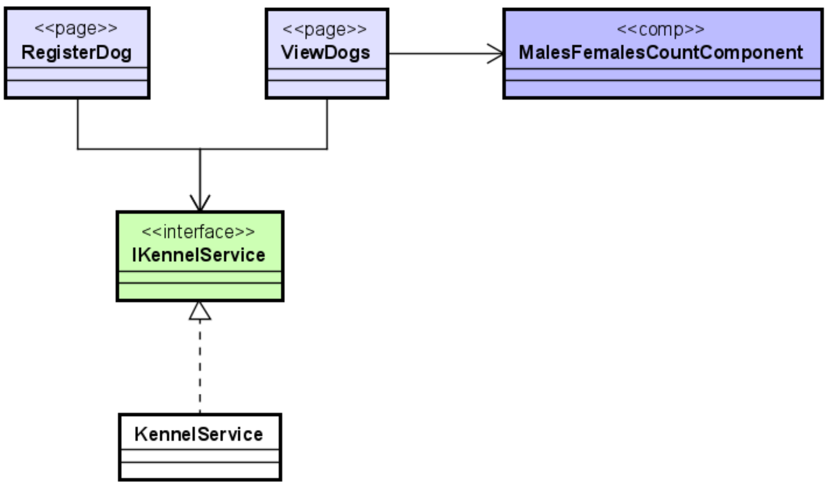
Opgave 1 – Blazor

I denne opgave skal du oprette en lille selvstænding Blazor applikation. Opgaven handler om et hjemmeside for en Kennel, hvor man kan adoptere hunde fra.

Du må benytte Blazor-Server eller Blazor-WASM.

Asynkron programmering er valgfrit i denne ogpave.

Nedenfor finder du et simplificeret, ukomplet klasse-diagram, som skal følges. Du skal tilføje mere til dit system end hvad der vises herunder:



1.1 Opret Blazor Project

Opret et nyt Blazor project, kald det "Blazor".

1.2 Entiteter

Opret følgende model klasse, kald den "Dog". Den skal have følgende properties:

- Id
- Name
- Sex
- Breed: f.eks. "Labrador Retriever", eller "Västgötaspits", eller "Poodle", etc.
- ImageUrl: Dette er bare en URL til et online billed.
- Description
- ArrivalDate

Anvend passende typer.

Placér klassen et fornuftigt sted i dit Blazor projekt.

1.3 KennelService med dummy data

Opret et interface, IKennelService.

Opret dernest en implementation, KennelService.

Placer både interface og klasse et passende sted.

KennelService Klassen skal initialisere en liste med mindst 5 Dogs, med varierende detaljer.

Service klassen skal registreres til dependency injection, så den kan bruges i diverse pages.

1.4 Registrer dog

Opret en ny page, kald den "RegisterDog".

Denne page skal indeholde input for følgende Dog properties: Name, Sex, Breed, ImageUrl, Description.

Der skal være en knap, som når trykket på, skal sende en Dog instans til IKennelService. Definer en passende metode på dette interface.

Implementationen af metoden skal så:

1) Sætte ArrivalDate til nuværende dato.
2) På fornuftig måde sætte Id til næste tilgængelige værdi.
3) Tilføje Dog instansen til listen af Dogs i klassen.

Når Dog instansen er oprettet og håndteret, så skal der vises en besked til brugeren.

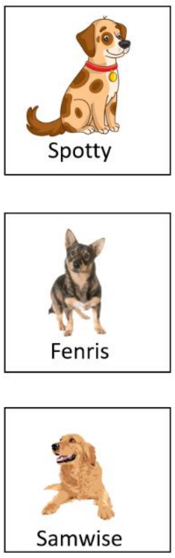
Tilføj denne page til navigationsmenuen.

1.5 Se Dogs

Opret en ny page, kald den "ViewDogs".

Denne page skal vise en liste af Dogs, sorteret efter tidligst ankomste først. Dvs, hunden der først kom ind i kennelen skal vises øverst/først.

Siden skal som udgangspunkt se ud nogenlunde som følger:



Der vises blot en liste af Dogs, med deres billede og navn.

Det skal være muligt at klikke på en hund og dermed se de resterende detaljer for den specifikke hund, et eller andet sted på siden.

Denne page skal tilføjes til navigationsmenuen.

1.6 Component der viser # male/female hunde

I denne opgave skal du lave en *component*, der kan bruges på siden fra foregående opgave 1.5.

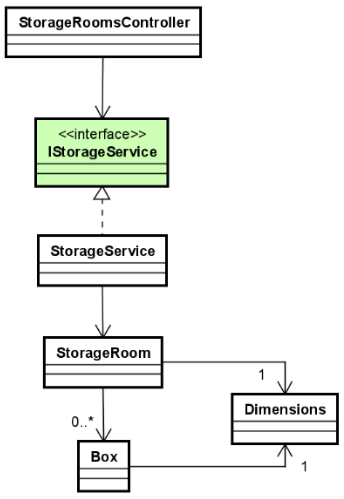
Opret denne component, den skal bruge IKennelService interfacet. Den skal vise antallet af han-hund og hun-hunde.

Put component i toppen af ViewDogs siden.

Opgave 2 – Web API

I denne opgave skal du oprette et Web API til at håndtere opbevaringsrum og deres indhold, eller: StorageRoom og Box.

Nedenfor ses et simplificeret, ukomplet klasse diagram over de mest relevante dele af systemet. Dette skal følges.



Asynkron programmering er valgfrit i denne opgave.

2.1 Opret Web API projekt

Opret et nyt Web API projekt, der bruger controllers til routing. Kald projektet "WebApi".

2.2 Entiteter

Opret entiterne StorageRoom og Box. De bruger begge en tredje objekttype, kaldet Dimensions.

De skal have følgende properties:

1. StorageRoom:
    a. Id
    b. Location, f.eks. "row 4, aisle 7"
    c. Dimensions, dvs en instans af denne objekttype.
    d. Boxes, dvs listen af Boxes.
2. Box
    a. Id
    b. Label, f.eks. "Kitchen", or "Old books", etc.
    c. Dimensions

Til sidst skal Dimensions klassens have properties som følger: Length, Width, Height.

Placer disse klasser et passende sted.

2.3 Storage Service

Opret klassen StorageService (med tilhørende interface), som:

1. Indeholder en liste af StorageRooms.
2. I starten opretter listen med mindst 5 StorageRooms.
3. Putter nogle Boxes i nogle af disse StorageRooms.

StorageService-klassen skal sættes op til dependency injection, så den kan bruges i controller-klassen.

Placer klasse og interface et passende sted.

2.4 Controller

2.4.1 *StorageRooms controller*

Opret controller-klassen. Hver action i denne controller skal delegere logik-håndteringen til relevant metoder i IStorageService-interfacet (implementeret i klassen).

Controller-klassen skal have endpoints som beskrevet nedenfor. Brug RESTful konventioner for routing.

2.4.2 *Tilføj kasse til rum*

Du har, jvf opgave 2.3, allerede nogle StorageRooms. Dette endpoint skal modtage en Box, som skal tilføjes til et eksisterende StorageRoom.

StorageService-klassen skal være ansvarlig for at sætte ID på den indgående Box.

Du behøver ikke tage hensyn til, om der er plads nok i rummet. Antag dette kontrolleres udenfor systemet.

Du skal inkludere relevant fejlhåndtering.

2.4.3 *Se alle kasser i et angivet rum*

Dette endpoint skal returnere en liste af kasser i et specifikt rum, angivet med id.

Du skal inkludere relevant fejlhåndtering.

2.4.4 *Fjern kasser i et angivet rum*

Dette endpoint skal modtage relevant information, så der kan blive fjernet en angivet kasse fra et angivet rum.

Du skal inkludere relevant fejlhåndtering.

2.4.5 *Query med filter*

Dette endpoint skal returnere en liste af rum. Det skal være muligt at filtrere disse rum ud fra to kriterier:

a) Inkluder rum med et kubikmeter-volumen større end et angivet tal (se bort fra potentielle kasser i rummet)
b) Inkluder rum med færre end et angivet antal kasser

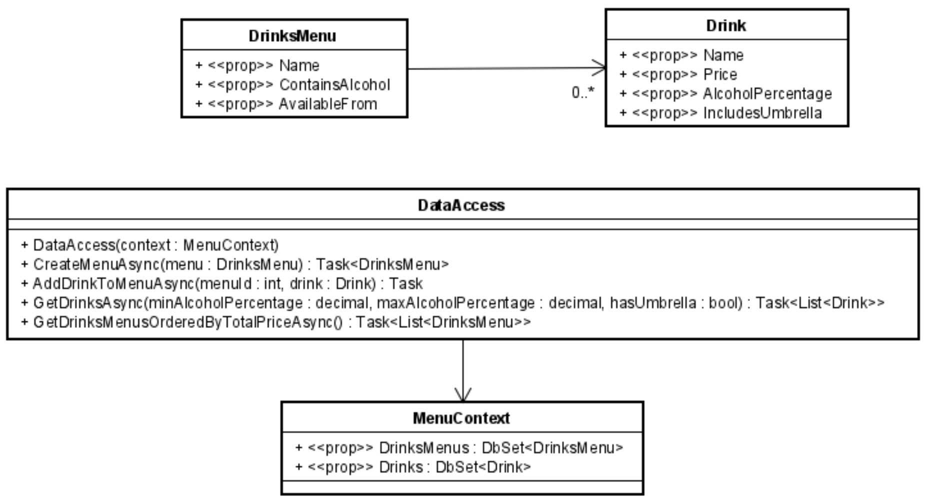
Begge filter-argumenter er valgfri, og det skal være muligt at anvende ingen, én af dem, eller begge.

Opgave 3 – Entity Framework

I denne opgave skal du oprette en database, og skrive et par metoder til at eksemplificerer brugen af database.

Du skal bruge Entity Framework Core og SQLite.

Nedenfor finder du et simplificeret diagram over hvad denne opgave vil indeholde:


DrinksMenu
+ <<prop>> Name
+ <<prop>> ContainsAlcohol
+ <<prop>> AvailableFrom
0..*
Drink
+ <<prop>> Name
+ <<prop>> Price
+ <<prop>> AlcoholPercentage
+ <<prop>> IncludesUmbrella
DataAccess
+ DataAccess(context : MenuContext)
+ CreateMenuAsync(menu : DrinksMenu) : Task<DrinksMenu>
+ AddDrinkToMenuAsync(menuId : int, drink : Drink) : Task
+ GetDrinksAsync(minAlcoholPercentage : decimal, maxAlcoholPercentage : decimal, hasUmbrella : bool) : Task<List<Drink>>
+ GetDrinksMenusOrderedByTotalPriceAsync() : Task<List<DrinksMenu>>
MenuContext
+ <<prop>> DrinksMenus : DbSet<DrinksMenu>
+ <<prop>> Drinks : DbSet<Drink>

Bemærk at nogle metoder er asynkrone, og du skal bruge asynkron programmering i denne opgave, hvor det er muligt.

3.1 Opret konsol projekt

Opret et nyt Consoler Application Project, kald det ”EFC”.

3.2 Entiteter

Opret entiteterne DrinksMenu og Drink. Put dem et passende sted.

Du skal definere gode primærnøgler for begge entiteter.

DrinksMenu entiteten har følgende properties:

- Name, f.eks “Juices”, “Wines”, “Soft Drinks”, etc.
- ContainsAlcohol. Denne indikerer om en drink indeholder alkohol eller ej
- AvailableFrom. Dette er et tidspunkt på dagen, fordi nogle drinks først et tilgængelig til f.eks. middag eller aften.

Drink entiteten har følgende properties:

- Name, f.eks. “Long Island Ice Tea”, “Orange Juice”, etc.
- Price, f.eks. “42.50”, or “17.95”, etc.
- AlcoholPercentage, f.eks. “4.6”, or “11.0”, or ”0.0”, etc.

- IncludesUmbrella, altså om denne drink inkluderer en lille paraply eller ej..

3.3 DbContext

Tilføj relevant EFC NuGet pakker til projektet.

Opret MenuContext klasse. Den skal nedarve fra DbContext. Du skal definere relevant DbSet-properties.

Add EFC NuGet packages to the project.

Create the MenuContext. It must derive from DbContext. It must define relevant DbSets.

Opret dernest database.

Du skal bruge SQLite.

3.4 Data Access

Opret DataAccess klassen, du kan indtil videre nøjes med dens konstruktør.

3.5 Opret DrinksMenu

Implementer metoden CreateDrinksMenu(..) i DataAccess klasse. Den skal modtage en DrinksMenu instans som parameter, og insætte denne i database, ved brug af MenuContext klassen.

I Program.cs filen skal du give et eksempel på, hvordan denne metode bruges. Indsæt mindst tre menuer.

3.6 Tilføj Drink til DrinksMenu

Implementer metoden AddDrinkToDrinksMenu(..) i DataAccess klassen. Metoden skal modtage en ny Drink, og tilføje til en eksisterende DrinksMenu.

I Program.cs filen skal du vise et eksempel på hvordan denne metode kan bruges. Tilføje mindst to Drinks til hver DrinksMenu.

3.7 Hent Drinks med filter

Implementer GetDrinks(..) metoden i DataAccess klassen.

Den skal modtage tre valgfrie (optional) parametre. To af parametrene filtrer Drinks som har alcoholprocemt større end minimumAlcoholPercentage og mindre end maximumAlcoholPercentage, og det sidste filter er til om en Drink indeholder paraply (umbrella) eller ej. Returner drinks som matcher parametrene.

Alle filtre er valgfri, og metoden skal kunne anvendes uden nogen filtre, med nogle af filtrene, eller med alle filtrene.

I Program.cs filen skal du give et eksempel på hvordan denne metode kan bruges.

3.8 Hent DrinksMenus sorteret efter pris

Implementer metoden GetDrinksMenusOrderedByTotalPrice(..) i DataAccess klassen.

Denne metode skal returnere alle DrinksMenus i database, sorteret efter deres totale pris. Den dyreste DrinksMenu skal være først.

Den totale pris for én DrinksMenu er defineret som summen af priserne for alle Drinks i denne ene DrinksMenu.

I Program.cs filen skal du give et eksempel på hvordan denne metode kan bruges.

Dette er slutningen af eksamenssættet. Held og lykke.