DNP1 Re-Exam, August 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, JetBrains 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, upload 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.

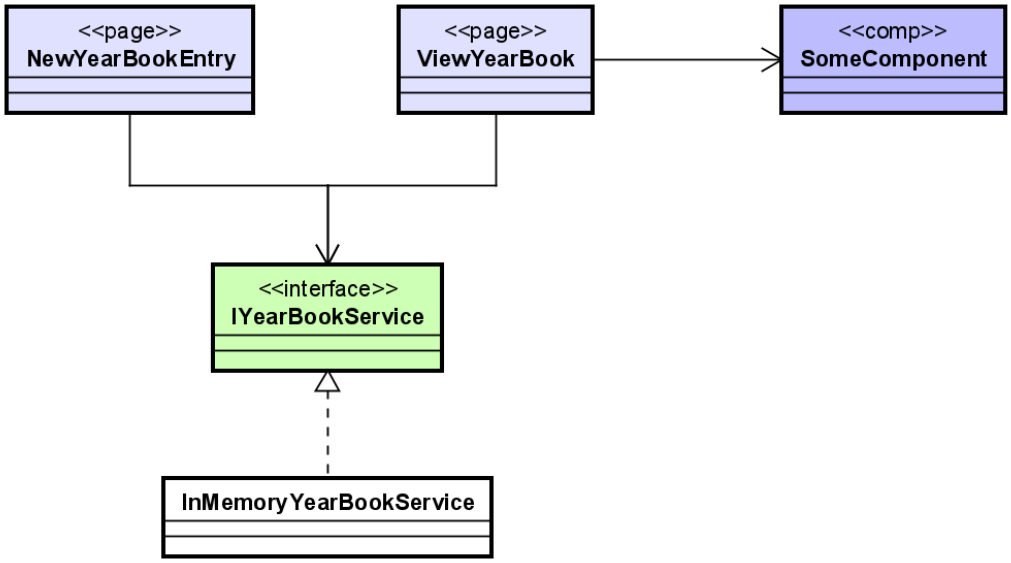
Exercise 1 – Blazor

In this exercise you will create a stand-alone Blazor app. It is a small application for a Year Book.

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 "YearBookEntry". It must have the following properties:

- Id
- Name
- Pronouns
- Fun fact
- Image Url
- Year (i.e. the year this person started)

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

1.3 YearBookService with dummy data

Create an interface, IYearBookService.

Then create an implementation, InMemoryYearBookService.

Place both interface and class somewhere reasonable.

The InMemoryYearBookService must initially create at least 5 entries, with varying details, and keep these entries in a list.

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

1.4 Register year book entry

Create a new page, call it "NewYearBookEntry".

This page must contain input for the following entry properties: Name, Pronouns, Fun fact, Image Url, Year.

Add a button to the page. When you click the button, a new instance of YearBookEntry will be created and handed over to the IYearBookService. Define an appropriate method on this interface.

The implementation of this method in InMemoryYearBookService must then:

1) In a reasonable way set the Id to the next available value.
2) Add the entry to the list of entries in the class.

When an entry is successfully created, display a message to the user.

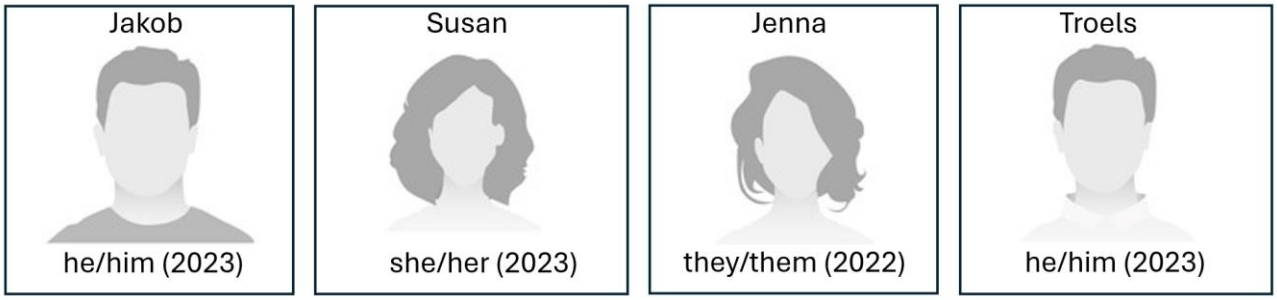
The page must be added to the navigation menu.

1.5 View year book

Create a new page, call it "ViewYearBook".

This page must show a list of the year book entries, ordered by earliest year first. So, the entries with the lowest year number are shown on top.

The view must display the details for the entries, and should look something like the following image:



It just shows a list of the year book entries, showing image, name, and pronouns.

You must then be able to click on an entry, and then the fun fact must be displayed somehow.

This page must be added to the navigation menu.

1.6 Component that shows # students per year

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 IYearBookService. It must display the number of students per year. E.g.:

2020: 87

2021: 68

2022: 103

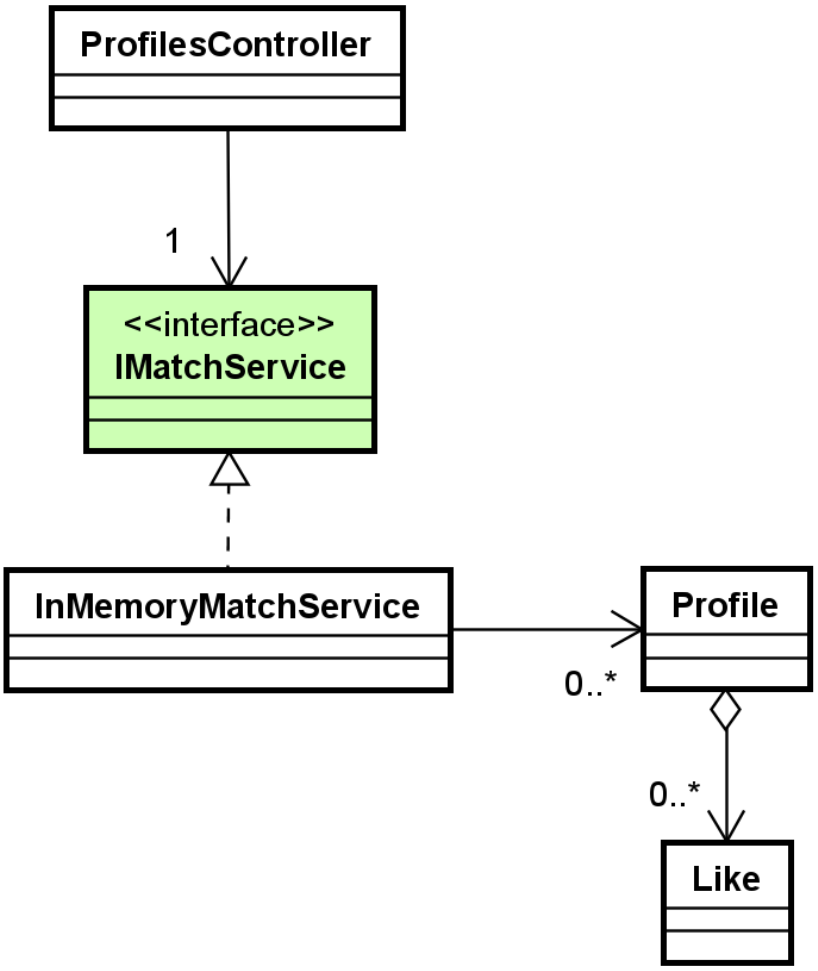
2023: 81

etc

Put the component at the top of the ViewYearBook page.

Exercise 2 – Web API

In this exercise, you will create a Web API backend for a dating app. The API consists of profiles, who have a list of liked profiles. Below is a simplified, incomplete class diagram of the most relevant types:



Note:

Asynchronous programming is optional in this exercise.

You must include error handling, where relevant.

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 Profile, and Like. The entities must contain the following properties:

1. Profile: Id, Name, Age, Gender, Likes

2. Like: An Id referencing the other profile, which was liked.

Choose reasonable types for each property.

Place the classes somewhere relevant.

2.3 - InMemoryMatchService

Create a class (which implements the IMatchService interface) that:

1. Contains a list of profiles.
2. Seeds dummy data in the constructor (i.e. it populates the list of profiles with a few profiles, each profile having some likes).

The MatchService interface and class must be set up for dependency injection, so that it can be injected and used in the controller class.

Place the class and interface somewhere relevant.

2.4 - Controller and endpoints

2.4.1 Create ProfilesController

Create a controller class. Each action in the controller must delegate the logic handling to associated methods in the InMemoryMatchService class (you will have to add these methods to the interface as well).

The controller must have the following endpoints, following RESTful conventions for routing:

2.4.2 Create profile

Create an endpoint which receives a profile. The InMemoryMatchService class must be responsible for setting the profile Id and adding the profile to the list of profiles.

Ensure that all properties have a value.

2.4.3 Add like

Create an endpoint responsible for adding a like to profile, i.e. creating a like referring to another profile.

Ensure that the Id of the like references an existing profile.

2.4.4. View all profiles

Create an endpoint responsible for returning profiles, where it must be possible to filter the list of profiles with an optional preferences filter.

The possible preferences are Gender, MinAge, MaxAge. It must be possible to apply all filters at the same time, only some of them, or none.

2.4.5. View matches

Create an endpoint responsible for returning the matches of a specific profile.

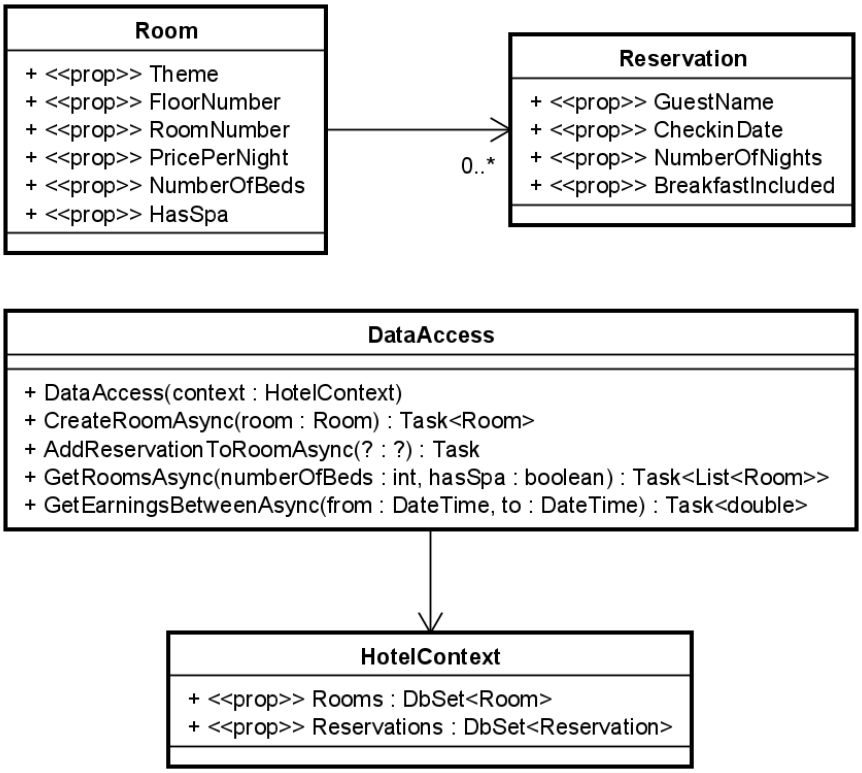
A match is defined as two profiles who have mutually liked each other's profiles.

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, not-entirely-complete diagram of what the exercise will contain:



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 Room and Reservation entities in an appropriate directory.

You must define a good primary key for both entities.

The Room has the following properties:

- Theme (e.g. jungle, savannah, castle, clouds, etc.)
- Floor number

- Room number
- Price per night
- Number of beds
- Has spa

The Reservation has the following properties:

- Guest name
- CheckinDate
- NumberOfNights
- Include breakfast.

I.e. a guest checks in to the hotel at a given date, and then they stay for a number of nights.

3.3 DbContext

Add EFC NuGet packages to the project.

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

Then create the database.

You must use SQLite.

3.4 Data access class

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

3.5 Create Room

Implement the CreateRoomAsync(..) method on the DataAccess class. It receives a Room as parameter and inserts it into the database, using the HotelContext.

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

3.6 Add reservation to room

Implement the AddReservationToRoomAsync(..) method on the DataAccess class. The method must add a new Reservation to an existing Room. In the Program.cs file, give an example of how this method is used. Add at least two Reservations to each Room.

3.7 Get rooms with filter

Implement the GetRoomsAsync(..) method on the DataAccess class. It must receive two optional parameters. The first parameter filters rooms by number of beds and the second filters whether or not the room has a spa.

Return rooms 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 earnings for date interval

Implement the GetEarningsBetweenAsync(..) method on the DataAccess class. It must calculate and return the total earnings from reservations between the two provided dates.

Notice a reservation can span multiple nights.

If a reservation includes breakfast, add 20% to the total price of this reservation.

Include reservations with check-in between the two provided dates.

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 uploade 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, 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.

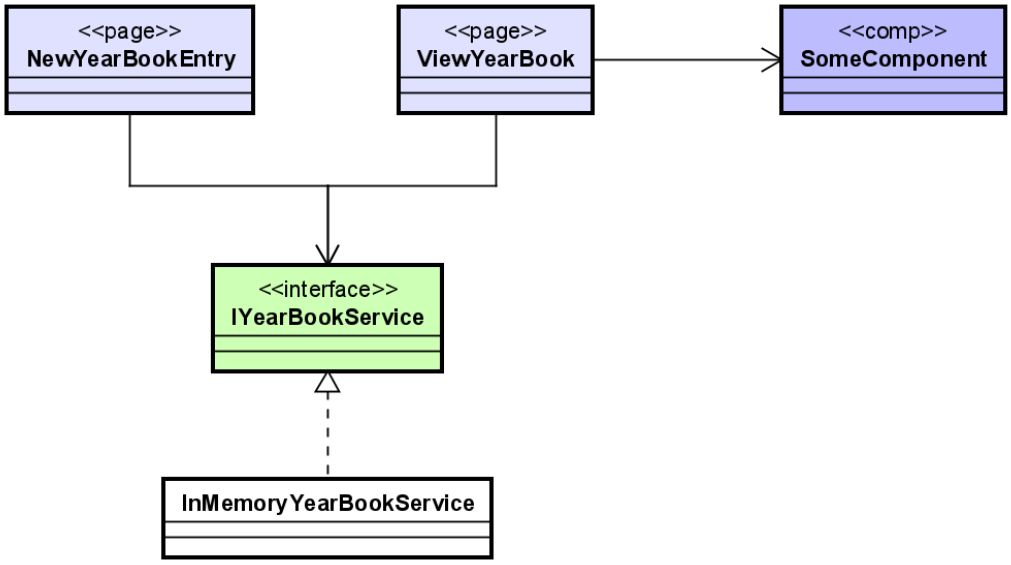
Opgave 1 – Blazor

I denne opgave skal du udvikle en lille selvstændig Blazor applikation. Opgaven handler om en hjemmeside for en årsbog.

Du må selv vælge mellem Blazor-Server eller Blazor-WASM.

Asynkron programmering er valgfrit i denne opgave.

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 entitet klasse, kald den "YearBookEntry". Den skal have følgende properties:

- Id
- Name
- Pronouns
- Fun fact
- Image Url
- Year (dvs det år vedkommende startede)

Anvend passende typer.

Placér klassen et fornuftigt sted i dit Blazor projekt.

1.3 YearBookService med dummy data

Opret et interface, IYearBookService.

Opret dernest en implementation, InMemoryYearBookService.

Placér både interface og klasse et passende sted.

InMemoeryYearBookService klassen skal initialisere en liste med mindst 5 YearBookEntries, med forskellige detaljer. Disse YearBookEntries skal gemmes i en liste.

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

1.4 Registrer year book entry

Opret en ny page, kald den "NewYearBookEntry".

Denne side skal indeholde input felter til følgende properties: Name, Pronouns, Fun fact, Image Url, year.

Tilføj en knap til siden. Når man klikker på knappen skal en ny instans af YearBookEntry oprettes og overleveres til IYearBookService. Definer en passende metode på interfacet.

Implementationen af denne metode i InMemoryYearBookService skal dernæst:

1) På fornuftig made sætte Id til næste tilgængelige værdi.
2) Tilføje entry'en til listen af entry'er i klassen.

Når en entry er succesfuldt oprettet skal der vises en besked til brugeren.

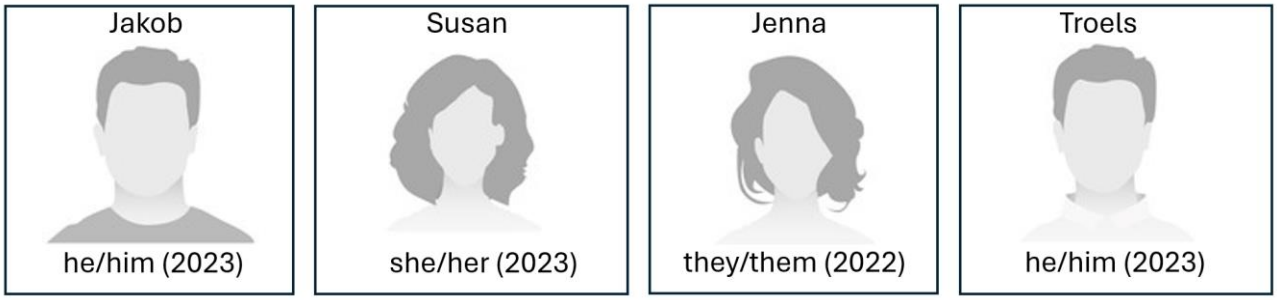
Denne side skal tilføjes til navigation menu'en.

1.5 Vis year book

Opret en ny page, kald den "ViewYearBook".

Denne page skal vise en liste af year book entries, sorteret efter tidligste år først. Dvs. de entries med laveste year vises øverst.

View'et skal vise nogle detaljer for entries, og skal se nogenlunde sådan her ud:



Det viser en liste af year book entries, hvor der for hver entry vises billede, navn, og pronouns.

Det skal være muligt at klikke på en entry, og se fun fact blive vist et sted på siden.

Denne side skal tilføjes til navigation menu'en.

1.6 Komponent, der viser # studerende per år

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

Opret en component, den skal bruge IYearBookService. Den skal vise antallet af studerende per år, f.eks.:

2020: 87

2021: 68

2022: 103

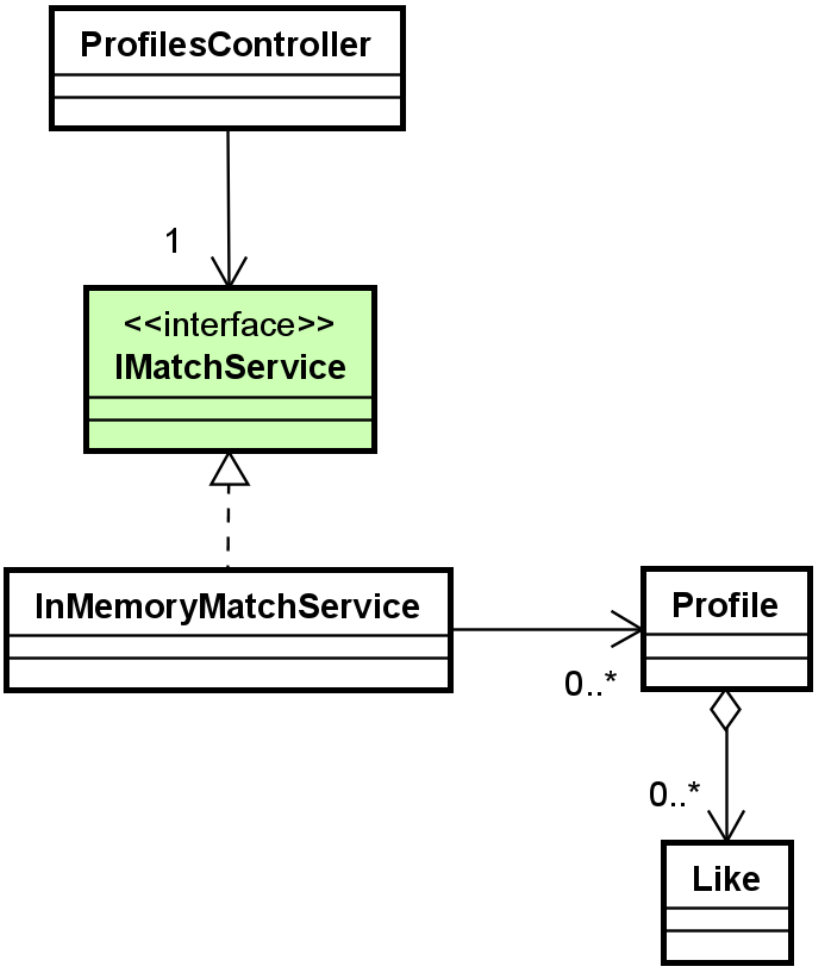
2023: 81

etc.

Indsæt komponenten øverst i ViewYearBook siden.

Opgave 2 – Web API

I denne opgave skal du udvikle en Web API for en dating app. API'en består af profiler, som har en liste af valgte (liked) profiler. Nedenfor ses et simplificeret, ukomplet klassediagram over de mest relevante dele af systemet. Dette skal følges:



Note:

Asynkron programmering er valgfrit i denne opgave.

Du skal implementere fejlhåndtering, hvor det er relevant.

2.1 Opret Web API projekt

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

2.2 Entiteter

Opret entiteterne Profile, og Like. Entiteterne skal indeholde følgende properties:

- Profile: Id, Name, Age, Gender, Likes
- Like: Et Id, som refererer den anden profil, som er blevet liked.

Vælg passende typer til hver property, og placer klasserne et passende sted.

2.3 - InMemoryMatchService

Opret en klasse (som implementerer IMatchService interface) som:

1) Indeholder en liste af profiler
2) Opretter dummy data i constructoren (dvs den indsætter et antal profiler, hver med et par likes)

MatchService interface og klasse skal sættes op til dependency injection, så de kan blive injected og brugt i controller klassen.

Placer interface og klasse et passende sted.

2.4 - Controller og end-points

2.4.1 Opret ProfilesController

Opret en controller klasse. Hver action i controller'en skal delegere logic-håndteringen til passende metoder i InMemoryMatchService klassen (metoderne skal også tilføjes til interfacet).

Controlleren skal have nedenstående endpoints, som følger RESTful konventioner for routing:

2.4.2 Opret profile

Opret et endpoint som modtager en profile. InMemoryMatchService-klassen skal være ansvarlig for at sætte Id, og tilføje profilen til listen af profiler.

Du skal sikre at alle properties har en værdi.

2.4.3 Tilføj like

Opret et endpoint som er ansvarlig for at tilføje et like til en profil, dvs oprette et like, der refererer en anden profil.

Du skal sikre at Id'et i like-objektet refererer en eksisterende profil.

2.4.4. Se alle profiler

Opret et endpoint, som er ansvarlig for at returnere profiler, hvor det skal være muligt at filtrere listen af profiler med et valgfrit præference-filter.

De mulige præferencer er: Gender, MinAge, MaxAge. Det skal være muligt at anvende alle filtre, kun nogle af dem, eller ingen.

2.4.5. View matches

Opret et endpoint, som er ansvarlig for at returnere matches for en specific profil.

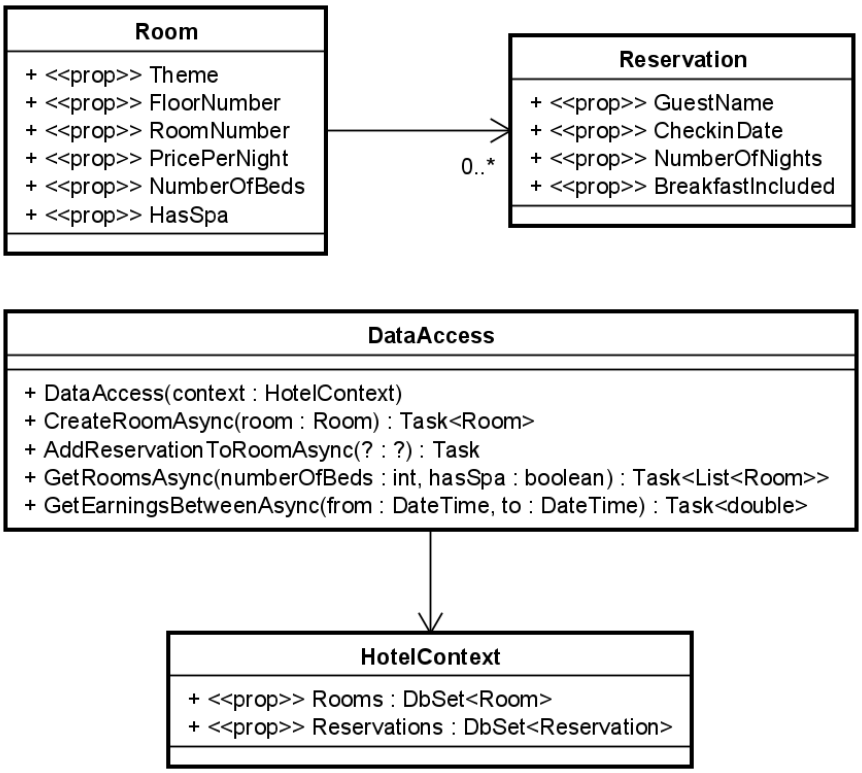
Et match er defineret som to profiler, der gensidigt har et like til den andens profil.

Opgave 3 – Entity Framework

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

Du skal bruge Entity Framework Core og SQLite.

Nedenfor ses et simplificeret, ikke-helt-færdigt diagram af hvad opgaven indeholder:



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

3.1 Opret konsol projekt

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

3.2 Entiteter

Opret Room og Reservation entiteter i en passende folder.

Du skal definere en god primær nøgle for begge entiteter.

Room har følgende properties:

- Theme (f.eks. jungle, savannah, castle, clouds, etc.)
- Floor number

- Room number
- Price per night
- Number of beds
- Has spa

The Reservation has the following properties:

- Guest name
- CheckinDate
- NumberOfNights
- Include breakfast.

Dvs. en gæst tjekker ind i hotellet på en given dato, og bliver der for et antal nætter.

3.3 DbContext

Tilføj EFC NuGet pakker til projektet.

Opret HotelContext klassen. Den skal nedarvede fra DbContext. Den skal definere relevante DbSet properties.

Opret dernæst database.

Du skal bruge SQLite.

3.4 Data access klasse

Opret DataAccess klassen, i første omgang kan du nøjes med dens constructor.

3.5 Opret Room

Implementer metoden CreateRoomAsync(..) i DataAccess klassen. Den modtager et Room som parameter, og indsætter det i databasen, ved brug af HotelContext.

I Program.cs filen skal du give et eksempel på hvordan denne metode kan bruges. Opret mindst 3 room objekter.

3.6 Tilføj reservation til room

Implementer metoden AddReservationToRoomAsync(..) i DataAccess klassen. Metoden skal tilføje en ny reservation til et eksisterende room.

I Program.cs filen skal du give et eksempel på hvordan denne metode kan bruges. Indsæt mindst to reservationer til hvert værelse.

3.7 Get rooms med filter

Implementer metoden GetRoomsAsync(..) i DataAccess klassen. Den skal modtage to valgfri parametre. Den første parameter filtrerer rooms ud fra antal senge, og det andet filtrerer ud fra om rummet har spa.

Returner de rum, der matcher parametrene.

Parametrene er valgfri, og det skal være muligt at anvende ingen, nogle, eller alle filter-parametre.

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

3.8 Get fortjeneste for et dato interval

Implementer metoden GetEarningsBetweenAsync(..) i DataAccess klassen. Metoden skal udregne og returnere den totale fortjeneste fra reservationer mellem de to givne datoer.

Bemærk at en reservation kan spænde over flere natter.

Hvis en reservation inkluderer morgenmad, tilføj da 20% til den totale pris for denne reservation.

Inkluder reservationer, hvor check-in er mellem de to givne datoer.

I Program.cs filen skal du give et eksempel på, hvordan metoden kan bruges.

Dette er slutningen af eksamenssættet. Held og lykke.