Övning 17 - Användarhantering för passbokning

I denna övning kommer vi att bygga en liten webbapplikation för att administrera och boka gympass. Övningens fokus är att träna på användarhantering genom att använda oss av Identity-ramverket. Under övningens gång kommer även repetera många av de moment vi lärt oss tidigare under kursen. Om du känner dig rostig så får du gärna titta tillbaka på tidigare övningar.

Vi bygger ut vår applikation steg för steg genom nya versions iterationer och försöker fokusera på ett problem i taget. Diskutera gärna moment och lösningar med era klasskamrater, men skriv er kod individuellt. Det är mycket viktigt att ni själva skriver er kod för att övningen skall bli ett effektivt läromoment.

Applikationen blir mer komplex för varje version och beskrivningen lägger mer och mer ansvar på dig att själv hitta lösningar. Det är inte självklart att ni hinner klart med alla versioner, men då är det resan snarare än målet som räknas. Många av de viktigare momenten ingår dock i de första versionerna.

Applikationsbeskrivning

Applikationen har två huvudsakliga användargrupper: administratörer och gymmedlemmar.

Det som ska implementeras är följande:

- Som besökare vill jag kunna logga in.
- Som besökare vill jag **enbart** kunna se en lista av gympass. Endast inloggade besökare kan interagera med dem.

- Som medlem vill jag kunna boka ett gympass.
- Som medlem vill jag kunna avboka mig från ett gympass
- Som medlem vill jag kunna se vilka som bokat upp sig på ett gympass.

- Som administratör vill jag kunna göra allt som medlemmar kan.
- Som administratör vill jag kunna lägga till nya gympass.
- Som administratör vill jag kunna redigera ett befintligt gympass.

Databas:

- Ett gympass kan ha många medlemmar bokade på sig.
- En medlem kan vara bokade på flera gympass

Databasen innehåller således en många-till-många relation mellan medlem och pass. Ni kommer därför behöva skapa en ”kopplingstabell” mellan dessa entiteter. Vi gjorde det tillsammans i https://github.com/LexiconNC19/LUniversiyNC19

Entiteter:

- Ett gympass har ett namn, en starttid, en längd (i tid) och en beskrivning.
- En användare har ett namn, en tidsstämpel då man startade medlemskapet, en e-postadress och ett lösenord.

Bygg applikationen

Gymbokning v0.1 - Ett skalprojekt

1. Skapa ett nytt MVC-projekt välj "*Authentication: Individual User Accounts*"

2. Skapa en ny klass i Models-mappen: GymClass.cs med de publika "egenskaperna":

- `int Id  { get; set; }`
- `string Name { get; set; }`
- `DateTime StartTime { get; set; }`
- `TimeSpan Duration { get; set; }`
- `DateTime EndTime { get { return StartTime + Duration} }`
- `String Description { get; set; }`

Observera hur `EndTime` endast har en "get-metod" och alltså inte explicit kommer att lagras i databasen utan beräknas utifrån passets starttid och tidsspann.
Vi väntar till senare med att utöka vår `ApplicationUser`-klass med dess nya "egenskaper".

3. För att identity filerna ska kunna visas i solution explorer behöver vi generera dessa. Det gör vi genom att Scaffolda fram dem. Add > New Scaffolded Item välj Identity i vänstra kolumnen, sen Add.
Vi sätter våran layout i Viewstart så vi lämnar tomt i första fältet. Sen väljer vi att overrida de filer vi behöver. En bra start kan vara Register, Login samt Index. Om vi senare behöver ändra i flera filer kan vi bara köra Wizarden igen.
Sedan väljer vi vårt ApplicationDbContext som redan finns i Data mappen. Vi väljer inte att skapa ett nytt context som vi brukar göra.

Dokumentation om något ändå skulle vara oklart:
https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-2.2&tabs=visual-studio#scaffold-identity-into-an-empty-project

4. Lägg till ett DbSet för GymClass-objekten i ApplicationDbContext-klassen. Var noga så att det inte hamnar inne i konstruktorn.

5. Skapa en ny klass i Models mappen: ApplicationUser. Vi låter klassen ärva från IdentityUser.
Vi lämnar den sen som den är.

Nu är vi nästan redo att bygga vårt projekt och ta det för en första testrunda, men först måste vi etablera våra *entitetsrelationer* genom *navigation properties* och *migrera* vår databas.

6. Vi börjar med att skapa en ny klass ApplicationUserGymClass för själva kopplingstabellen. Det är en namnkonvention att klassen namnsätts via de andra klassnamnen. Vi följer även de andra namn konventionerna när vi lägger till propertys. Klassnamn + id för foreign key. Klassnamn på navigationproperty.

7. Skapa en navigation property i GymClass-klassen. Ett pass kan ju bokas av flera medlemmar, så den måste innehålla en kollektion av dessa medlemmar:
`public virtual ICollection<ApplicationUserGymClass> AttendingMembers { get; set; }`
`Observera här att vi anger våran kopplingsklass som  navigationproperty`

8.. Skapa en motsvarande "nav-propp" (*navigation property*) i ApplicationUser-klassen. En medlem kan boka flera pass, alltså behöver klassen en kollektion av dessa pass:

```
public virtual ICollection<ApplicationUserGymClass> AttendedClasses { get; set; }
```

9. Tyvärr finns ännu inte funktionaliteten att definiera upp en komposit nyckel med hjälp av data annotations. Så vi får använda oss av fluent api för att åstadkomma det.
Vi använder oss av en override av OnModelCreating i ApplicationDbContext klassen.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
      base.OnModelCreating(modelBuilder);

      modelBuilder.Entity<ApplicationUserGymClass>()
        .HasKey(t => new { t.ApplicationUserId, t.GymClassId });
    }
```

10.
```
services.AddDefaultIdentity<IdentityUser>()
        .AddRoles<IdentityRole>()                <=  Vi behöver lägga till det här  Startup klassen!
```

Nu är våra grundläggande *POCOs* på plats med nav-proppar kan vi "*scaffolda*" en GymClassController med vyer.

11. Högerklicka på Controllers-mappen och välj Add > Controller. I Wizarden väljer du sedan "MVC Controller with views, using Entity Framework"
Model class skall vara din nya "GymClass" och som context använder du `ApplicationDbContext`.
Resten fylls i automatiskt så det är bara att klicka "*Add*"

När allt är på plats skall vi bara migrera våra entiteter och relationer till databasen så är vi klara med skalapplikationen version 0.1.

12. Add-Migration, Update.Database

Eftersom vi har bytt ut IdentityUser mot våran egen implementation ApplicationUser behöver vi ersätta alla instanser av IdentityUser uppe i vårat Scaffoldade projekt!
Nu är vi redo att bygga projektet och ta det för en test-tur! När du har webbapplikationen framför dig så testar du att registrera en medlem, besök localhost:#portnr#/GymClasses och skapa ett gympass! Funkar det? Strålande!
Passa även på att öppna er *serverexplorer* och observera korskopplingstabellen som EntityFramwork generat utifrån era navigation properties och den många-till-många relationen de antydde.

Gymbokning v0.2 - Passbokning

Nu när vi kan vi registrera medlemmar och skapa gympass så är det naturliga nästa steget i funktionaliteten att låta medlemmar boka in sig på pass.

I denna version kommer vi att (1) skriva en metod i vår `GymClassController` som bokar på eller av den inloggade medlemmen till ett pass. Vi kommer sedan att (2) skapa en länk i index-vyn till denna action och slutligen (3) bygga ut passens detalj-vy för att lista de medlemmar som bokat sig på passet. I alla dessa steg finns det elegantare lösningar, men nu vill vi hålla det enkelt och effektivt.

1.
Vi börjar med att skapa en *BookingToggle-metod* vår GymClassController*:*

Skapa en public metod enligt följande:

```
public async Task<IActionResult> BookingToogle(int? id)
{
    if (id == null) return NotFound();
```

Vi behöver först och främst ta reda på vilket pass som ska bokas. Sen behöver vi även ta reda på den nu inloggade medlemmens id för att antagligen bokas eller avbokas från passet.

I metoddeklarationen ser vi att metoden är publik och har returtypen "ActionResult", vi ser också att den tar en inparameter Id som är ett heltal. Detta är Id:t på det gympass vi vill boka.

Med gympasset vi redan tagit fram kan vi titta på om medlemmen finns med bland dess AttendingMembers.
Beroende på resultatet bokar vi eller bokar av medlemmen från det aktuella passet.

2.
För att nu kunna använda vår nya action-metod behöver vi lägga till en länk i vår index-vy.
Vi öppnar GymClass:ens Indexvy (/views/GymClasses/Index.cshtml). Om vi tittar på de redan existerande actionlänkarna till details/edit/delete så skall vi göra precis likadant fast med länktexten "book" och som länkar till *actionen* "BookingToggle". Precis som i de andra länkarna så vill vi skicka med passets id som en inparameter.

3.
Även om det inte syns i applikationen så kan inloggade medlemmar nu boka på och av sig från pass. Det sista tillägget i denna version är en möjlighet att se detta.
Vi öppar GymClass:ens Details-vy (`/views/GymClasses/Details.cshtml`)
Längst ned i tabellen lägger vi till ett utskrift med hjälp av en foreach-loop som skriver ut e-posten på alla medlemmar som bokat sig på passet. I en senare version kommer vi byta e-posten mot ett namn, men än så länge har våra användarobjekt ingen namn-property.

Version 0.2 är klar. Bygg applikationen och testa att klicka runt!

Gymbokning v0.3 - Inloggad eller inte inloggad, det är frågan

I uppgiftsbeskrivningen står det att inloggade och icke-inloggade användare skall ha olika åtkomst till vyer och funktionalitet på sidan, och nu när vi äntligen börjar få lite funktionalitet kan det var dags att låsa av delar av sidan.

I denna version (1) städar vi upp lite i applikationen genom att ta bort `HomeControllern` och dess vyer, istället låter vi (2) *routingen* till att direkt ta oss till listan med pass. I samband med detta måste vi även (3) rensa de nu döda länkarna från vår nav-bar.
Sedan kommer vi att (4) låsa av de actions som icke-inloggade användare inte skall ha tillgång till. Vi passar även på att (5) gömma undan länkar som de icke-inloggade saknar åtkomst till.
Vi väntar dock med roller och administratörer till en senare version.

1.
Radera filen `/Controllers/HomeController.cs` och hela mappen `/views/Home`.

2.
Öppna filen `Startup.cs` och andra default controller från "`Home`" till "`GymClasses`"

3.
Öppna filen `/Views/Shared/_Layout.cshtml`
Radera listan med länkar i nav-baren.
Byt länktext på *sidrubriken* från "`Application name`" till "`Gymbokning`" eller annat passande namn.
Peka även om den från home-kontrollern till `GymClasses`-kontrollern.

4
Gå till `GymClassesController` och titta på vilka action metoder som finns där. Vilka skall endast en inloggad användare ha tillgång till? Använd `[Authorize]`-annotationen för att blockera dessa.

5.
Öppna `Index.cshtml`
Ur `User`-klassen kan ni få svar på huruvida en besökare är inloggad eller inte. Genom att använda `User.Identity.IsAuthenticated` returneras ett boolskt värde som är sant om besökaren är inloggad, annars är det falskt. Använd detta i en *`if-sats`* för att endast generera länkarna i vyer för inloggade användare.
Observera att ni i kodblock inte kan skriva "lösa strängar" utan att innesluta dessa i html-taggar. Detta gäller t.ex. de pipe-tecken ("|") som separerar vissa länkar i de genererade vyerna. Använd då `<span> | </span>` för dessa.

Gymbokning v0.4 - The All-Mighty Admin

Vi kommer i denna version att fortsätta arbetet med roller. Vi kommer att seeda in rollen "`Admin`" och dessutom en användare som erhåller denna. I ett senare skede kan man föreställa sig en kontrollpanel där det är möjligt att uppgradera vanliga användare till Admin-rollen, men just nu nöjer vi oss med en enda.

Användarobjekten och rollobjekten är något mer komplexa än de vanliga POCOs vi är vana att arbeta med sedan tidigare i kursen. Då goda exempel på hur ni skall koda denna version gavs under tidigare code-a -long kommer jag inte gå in på detaljer här i övningsbeskrivningen utan hänvisar till era egna anteckningar och repositoryt på GitHub.

Glöm inte att lägga till IdentityRole i Startup.cs och i ApplicationDbConntext.cs.

Seed()-metoden:
Seeda en roll som heter "`admin`".
Seeda en användare med användarnamnet "`admin@Gymbokning.se`" och valfritt hemligt lösenord som läses ut från User Secrets.
Lägg till användaren "`admin@Gymbokning.se`" i rollen "`admin`".

GymClassesController:
Blockera Edit-, Create- och Delete-actions från alla besökare som ej tillhör rollen "`admin`"

Index-vyn:
Göm Länkar som endast admin har tillgång till. Använd `User.IsInRole("admin")` för era if-satser.

Gymbokning v0.5 - Användaren i fokus?

I version 0.5 har det blivit hög tid att bygga ut vår `ApplicationUser`-klass (`/Models/IdentityModels.cs`)

1.
Lägg till properties:

```
string FirstName { get; set; };
string LastName { get; set; };
string FullName  { get { return FirstName + “ “ + LastName; } };
DateTime TimeOfRegistration  { get; set; };
```

2.
Gör samma tillägg i `Register`-klassen (Areas/Identity/Pages/`Account/Register.cshtml`)

3.
Lägg till dessa fält i registreringsvyn (`/Views/Account/Register.cshtml`). `TimeOfRegistration` skall dock ej fyllas i av användaren utan sätts i controllen på serversidan till den faktiskta registreringstiden.

4.
Lägg till dessa i register-action i `AccountControllern` (`/Controllers/AccountController.cs`) då ni skapar `ApplicationUser`-Objektet.

5.
Uppdatera `GymClasses` index-vy (`/Views/GymClasses/Index.cshtml`) så att den visar `FullName` istället för `Email`.

6.
Uppdatera seeden av ert användarobjekt med de nya egenskaperna. Detta gäller framför allt de egenskaper som inte är nullable.

6.
Migrera modellerna och uppdatera databasen

Gymbokning v beta 1.0 - Användaren i fokus!

Vi gör oss redo att lansera vår applikation som version 1.0 för beta-testare. För första gången skall applikationen användas av potentiella slutanvändare och nu gäller det att polera!

1.
"Boka" länken skall säga "Boka av" om användaren redan är bokad på passet.

2.
Gamla pass vars datum passerat skall ej visas passlistan om man inte specifikt ber om detta (knapp, checkbox eller något liknande, ev. en separat vy för "Historik")

3.
En inloggad användare skall se två nya länkar i nav-bar: "Bokade Pass" och "Historia".
"Bokade Pass" visar en filtrerad passlista med endast de pass användaren anmält sig till.
"Historia" visar gamla pass som användaren gått på och vars datum passerat.

4.
Inputvalidering i alla fält

5.
Uppdatera färgtemat

6.
Övrigt småfix du upptäcker

Vi är redo att släppa version 1.0 för beta-testning! Bra jobbat :)