### Migrering af adgangskontrol fra SQLserver til MySQL

Med det projekt, der bliver genereret af Visual Studio, kommer også et sikkerhedsframework. Da det foregår i en Microsoftverden, er det naturligt, at der anvendes Microsoftprodukter. Således bruges SQLserver, men da det erMySQL, som skal holde alle data i Voresjazzklub, skal der foretages nogle ændringer i koden.

Der er nogle klasser i Microsoft.AspNet.Identity.EntityFramework, der tilsammen håndterer brugeradgangen. Forventningen var, at der skulle skiftes ud i nogle metoder og andre skulle overrides, og at connection string skulle ændres fra SQLserver til MySQL.

Efter at have boret i problematikken et par timer, måtte det konstateres, at det nok kunne lade sig gøre, men at det samtidigt ville være en større operation.

Hvis projektopgaven var fokuseret på sikkerhed, ville det ganske givet være en interessant øvelse at ændre i frameworket. Her anvendes så i stedet dele fra en opskrift fundet på nettet. Det, der ikke anvendes, er hosting i Azure.

[MySQLMigrering]

https://docs.microsoft.com/en-us/aspnet/identity/overview/extensibility/implementing-a-custom-mysql-aspnet-identity-storage-provider

Der skal hentes et projekt bestående af ti klasser. Det skal bygges og i Voresjazzklub skal der så være en reference til det. Derudover et par mindre justeringer i Voresjazzklub.

Download AspNet.Identity.MySQL, og build AspNet.Identity.MySQL.sln

Højre klik Voresjazzklub | Add | Reference

Browse til AspNet.Identity.MySQL..dll

Der står ganske vist noget helt andet i opskriften, men den er datostemplet 22/5/2015 – og det er lang tid i den verden.

Der skal ændres i tre filer. Microsoft.AspNet.Identity.EntityFramework skal skiftes ud med AspNet.Identity.MySQL + et par småjusteringer i to metoder. I web.config skal DefaultConnection flyttes til at pege på MySQL

**IdentityModels.cs:**

//using Microsoft.AspNet.Identity.EntityFramework;

using AspNet.Identity.MySQL;

public class ApplicationDbContext : MySQLDatabase {

public ApplicationDbContext(string connectionName)

: base(connectionName) {

}

public static ApplicationDbContext Create() {

return new ApplicationDbContext("DefaultConnection");

}

}

**IdentityConfig.cs:**

//using Microsoft.AspNet.Identity.EntityFramework;

using AspNet.Identity.MySQL;

var manager = new ApplicationUserManager(

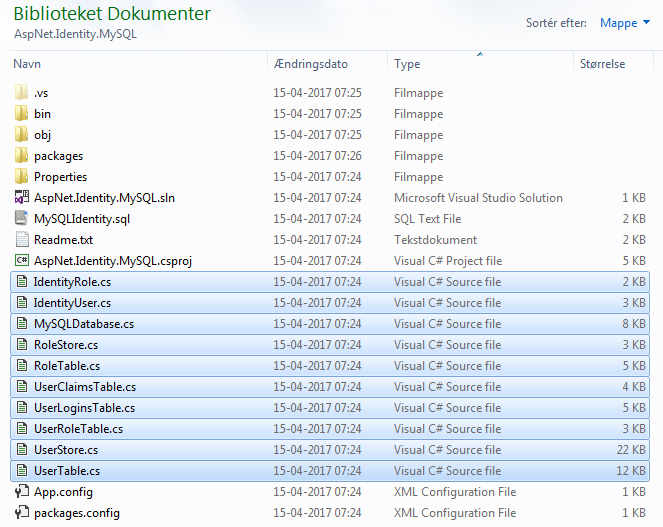
new UserStore<ApplicationUser>(

context.Get<ApplicationDbContext>() as MySQLDatabase));

**web.config:**

Connection string change to MySQL

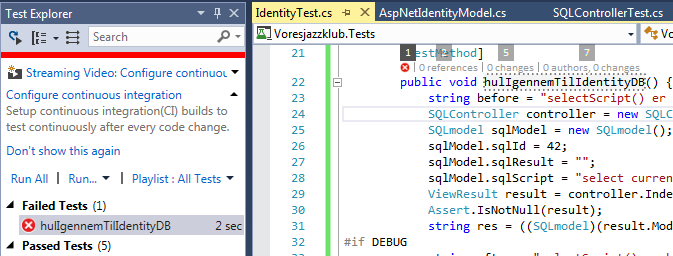
Disse klasser kommer der ud af at hente AspNet.Identity.MySQL:

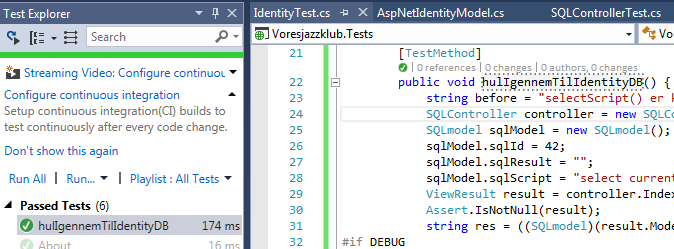


En sidste lille ting, der skal gøres før Voresjazzklub igen kan komme op at køre, er, at der skal oprettes fem tabeller i MySQL (se Appendix: ”Script til migrering af adgangskontrol”). Da tabellerne allerede ligger separeret fra de øvrige tabeller, er det valgt, at fortsætte sådan.

De tabeller, der er oprettet specifikt til Voresjazzklub ligger i databasen Voresjazzklub, og tabeller til adgangskontrol ligger i databasen MySQLIdentity. Når tiden er moden til en ændring af login mv. så det ligner de oprindelige tanker, er det nemmere at skille tingene ad, fordi der skal fjerne en database og en reference til et andet projekt.

Når der bliver lavet noget nyt, skal der også være kørt en test. Første gang den køres, forventes det, at den fejler, og når der er kodet tilpas meget, og testen køres igen, skal den blive grøn. Det er naturligvis også tilfælde her, hvor det tjekkes om der er forbindelse til den nye database.





Herefter startes applikationen og en bruger registreres. I dette tilfælde er det brugeren [b@b.dk](mailto:b@b.dk), som det kan anes oppe i højre hjørne i billedet.

Ved at gå ind på siden SQL, hvor et vilkårligt SQL-script kan køres fra, hvilket muligvis kan opfattes som en sikkerhedstrussel, vælges alle registrerede brugere. Her genfindes så den nyligt registrerede bruger.

