FEEDBACK APP

Eksamensprojekt til kurset *Webudvikling – Backend*

App og Rapport er udarbejdet af Louise Høpfner

Aflevering: 30/5 2018

INDHOLD

INDLEDNING

Denne rapport beskriver funktionaliteten, koden og de kodemæssige valg der er truffet i udviklingen af FeedbackApp. Rapporten er bygget op, så det første afsnit *Overordnet beskrivelse af appen* er en præsentation af FeedbackApp fra et bruger perspektiv. I afsnittet *Arkitektur* bliver den bagvedliggende kode gennemgået, samt de udviklingsmæssige valg der er taget. Til sidst vil jeg gennemgå nogle fremtidige udviklingsperspektiver i afsnittet *Videreudvikling*.

FeedbackApp er en .NET Core 2.0 MVC webapplikation. Den er udviklet ved brug af Microsoft Visual Studio Community 2017 Version 15.5.7. Den primære kilde brugt i udviklingen af FeedbackApp er bogen Learning ASP.NET Core 2.0 [1].

Appen er udviklet til at gemme data i en SQL database. Det er muligvis nødvendigt at køre kommandoen Update-Database fra Package Manager Console i Visual Studio før appen virker. Det er for at oprette databasen lokalt med de relevante tables.

OVERORDNET BESKRIVELSE AF APPEN

I dette afsnit vil jeg beskrive FeedbackApp set fra et bruger perspektiv. Altså, hvad kan appen og hvordan anvendes den. Det vil jeg beskrive ved først at gennemgå den overordnede funktionalitet og derefter de user stories app'en er udviklet ud fra.

Funktionalitet

FeedbackApp er en webapplikation, hvor en bruger (manager) kan oprette undersøgelser (surveys) med specialiseret feedback. Det vil sige, den feedback en bruger (user) får, når de har besvaret undersøgelsen, afhænger af, hvilke svar de har givet.

User stories

Dette afsnit beskriver de overordnede user stories [2] som funktionaliteten i denne app dækker over. Det er ikke alle user stories der er implementeret endnu. Det vil fremgå tydeligt hvilke der er.

Der er defineret to former for brugere af app'en. En manager, der opretter og administrere surveys, og en user, der besvarer surveys. I Tabel 1 kan man se en oversigt over alle overordnede user stories.

Nr.	*Bruger*	*Story*
1	Manager	Som en manager vil jeg gerne oprette en survey. #
2	Manager	Som en manager vil jeg gerne redigere en survey. #
3	Manager	Som en manager vil jeg gerne åbne eller lukke en survey. #
4	Manager	Som en manager vil jeg gerne prøve en survey. #
5	User	Som en user vil jeg gerne besvare en survey. # #

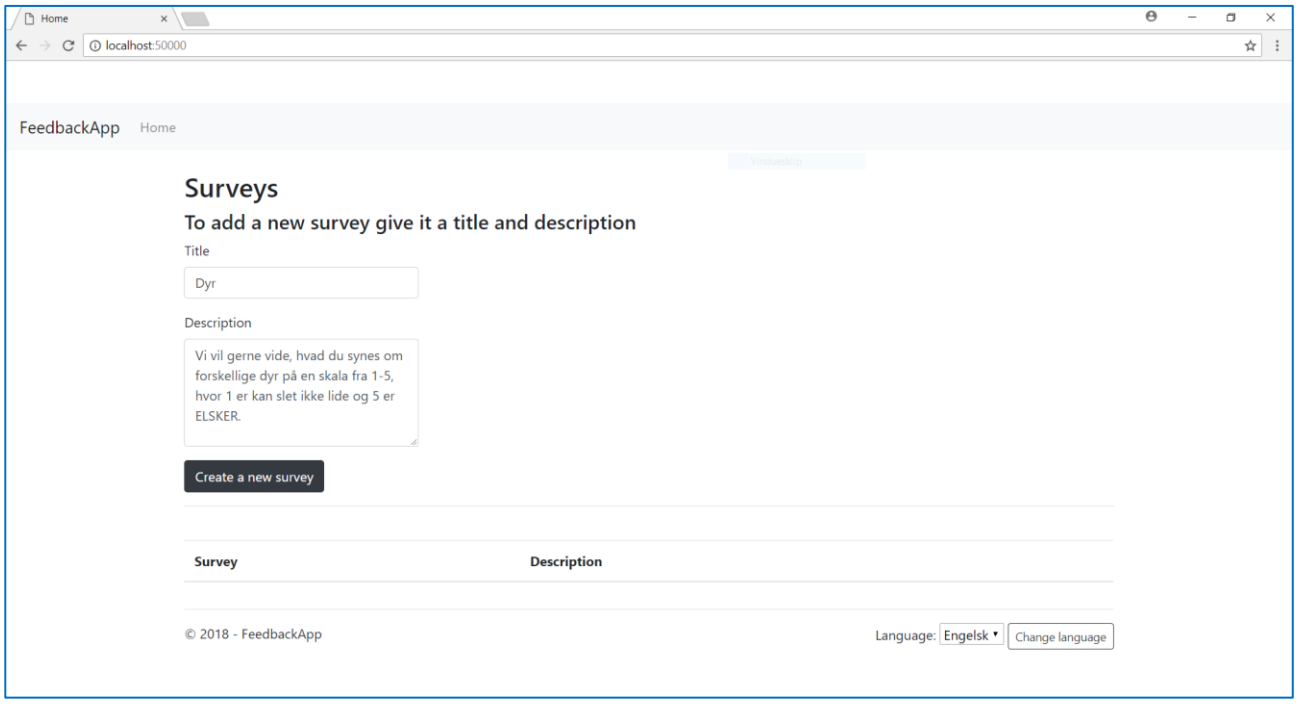
Tabel 1: Oversigt over user storie i app'en FeedbackApp. Stories markeret med grøn # er implementeret. Stories markeret med rød # er ikke implementeret.

USER STORY 1

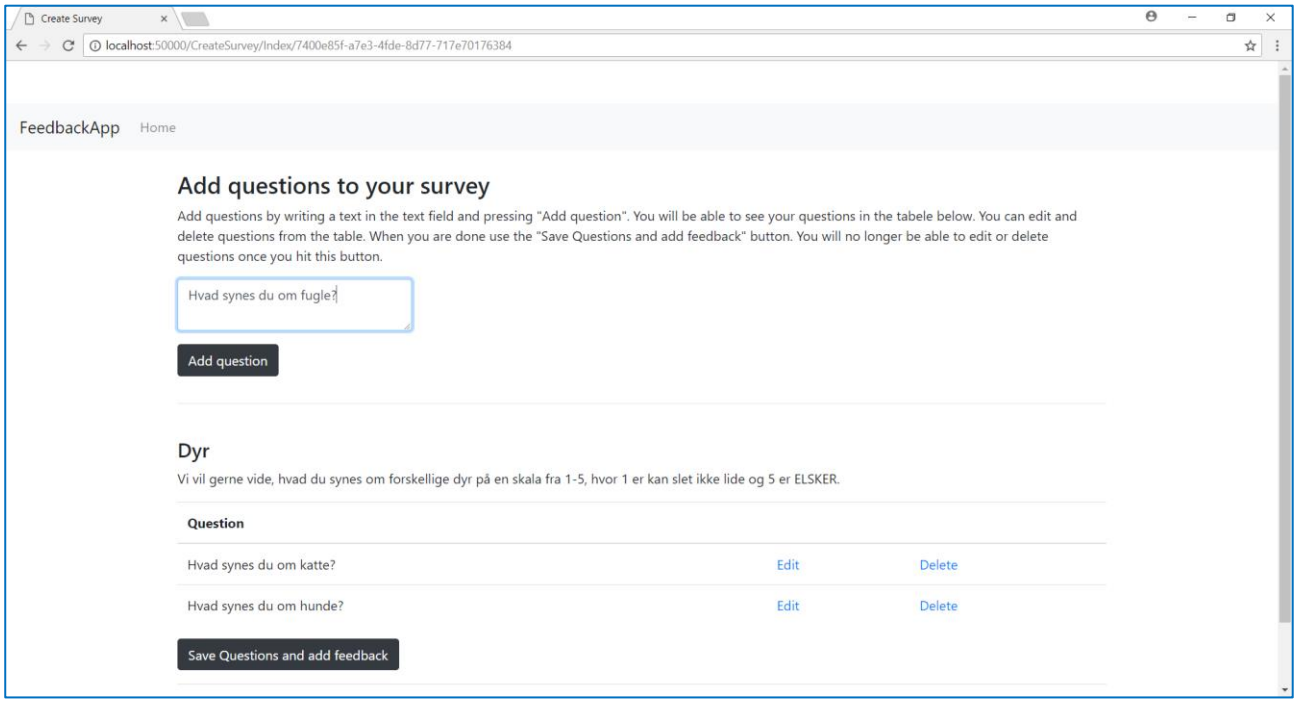
Denne user story dækker over den meste funktionalitet i app'en. I dette afsnit gennemgår jeg, hvordan denne user story er implementeret fra et manager perspektiv. Altså, ikke den bagvedliggende kode.

På de næste billeder ses flowet igennem denne user story. Den starter på forsiden som ses på Figur 1, hvor en manager kan indtaste en titel og beskrivelse og med en knap oprette en survey og begynde at tilføje spørgsmål. På de næste to sider kan der tilføjes først spørgsmål, som vist på Figur 2, og derefter feedback, som vist på Figur 3 og Figur 4. Til sidst sendes manageren tilbage til forsiden, hvor den oprettede survey nu vil fremgå i en tabel, som det ses på Figur 5.

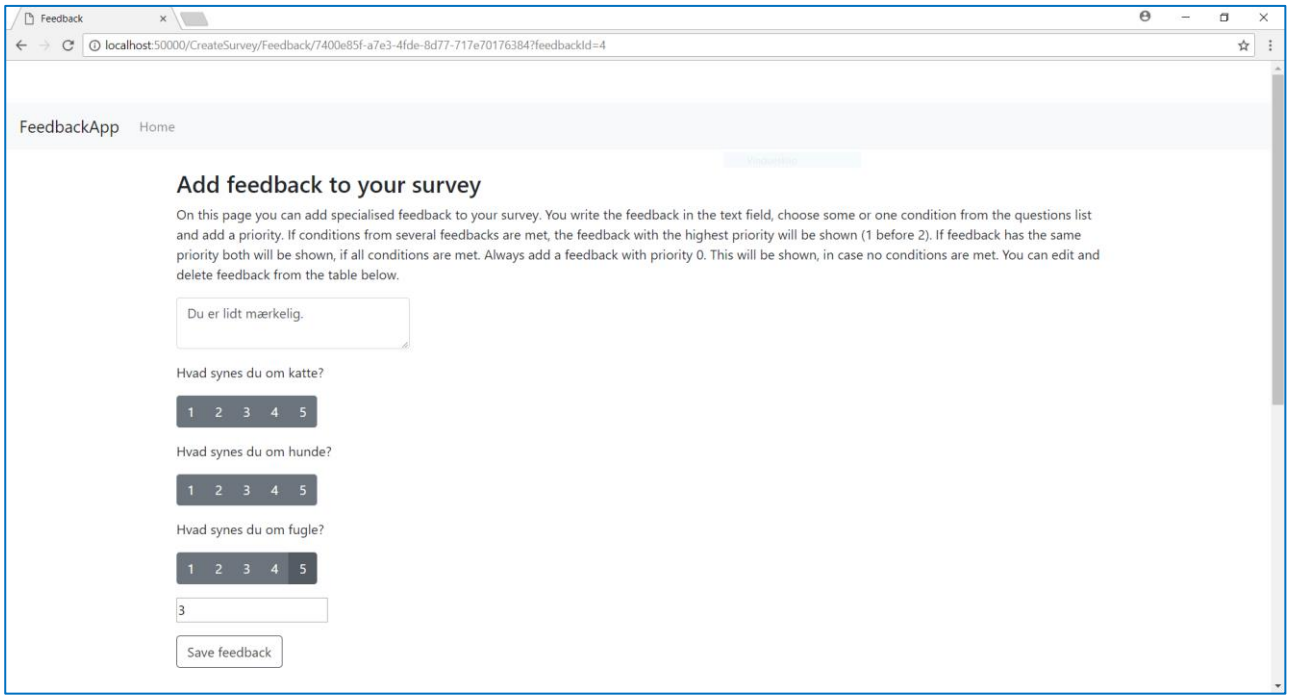
Spørgsmål og feedback kan tilføjes, redigeres og slettes. Når feedback tilføjes, kan man vælge en eller flere betingelser, der styrer hvornår feedback vises. En betingelse består af svaret på et spørgsmål. Prioriteten styrer om en feedback vises, hvis betingelser for flere feedback elementer er opfyldt. Har to elementer samme prioritet vises de begge to i fald betingelserne for begge er opfyldt. Prioritet 0 vises i tilfælde af at ingen betingelser for feedback er opfyldt. På Figur 4 er vist noget feedback med betingelser og prioritet. Kan en bruger ingen dyr lide, vil feedbacken være "Tak for dine svar. Vi sætter stor pris på dem". Kan man lide både katte og hunde vil man få "Du er nok et kattemenneske!" og "Du er nok et hundemenneske". Svarer man at man kan lide alle tre dyr, vil man kun få vist "Du er bare vil med ALLE dyr!". Er svaret kun helt positivt til fugle vil feedback være "Du er lidt mærkelig".



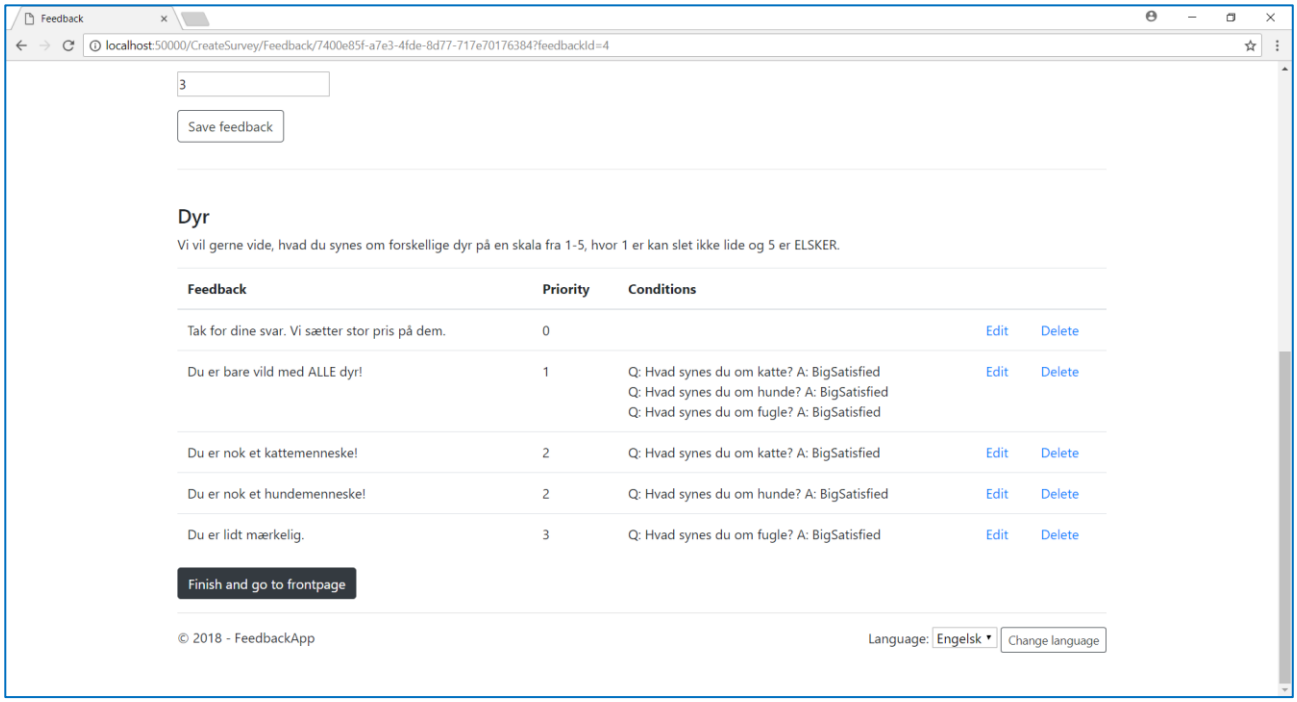
Figur 1: Forsiden (Home) for en manager, der bruger FeedbackApp. Her er tilføjet en titel og beskrivelse. Derefter kan knappen "Create a new survey" anvendes til at komme i gang med at tilføje spørgsmål, som vist på Figur 2.



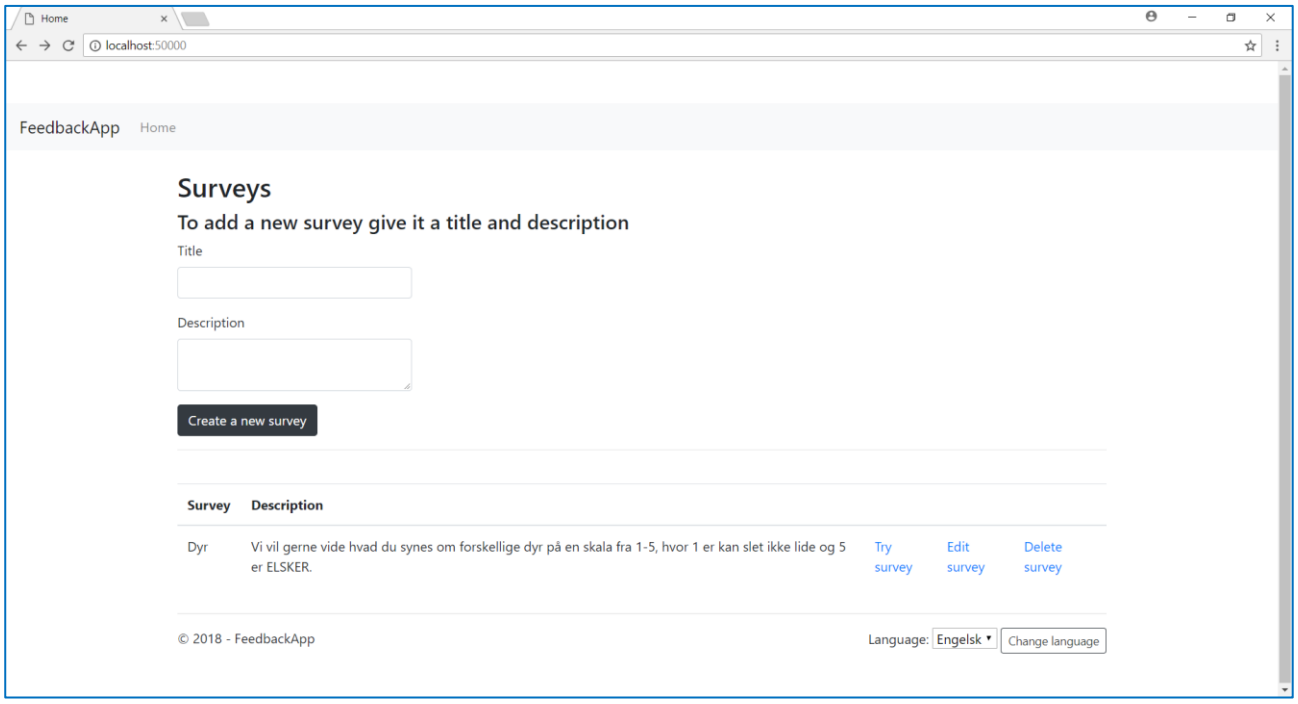
Figur 2: PÅ denne side kan en manager tilføje spørgsmål til den survey hun er ved at oprette. Her er der tilføjet to spørgsmål, og ved brug af knappen "Add question" vil det tredje blive tilføjet. Alle spørgsmål kan slettes eller redigeres. Knappen "Save Questions and add feedback" sender brugeren videre til feedback siden som vist på Figur 3 og Figur 4.



Figur 3: Den øverste del af feedback siden. Her kan der tilføjes en feedback tekst, betingelser for at feedback vises samt en prioritet. På Figur 4 ses den nederste del af siden. Her vises den feedback der allerede er tilføjet.



Figur 4: Nederste del af feedback siden. Her vises en tabel med en oversigt over feedback der er tilføjet. Hver feedback kan redigeres og slettes. Knappen "Finish and go to frontpage". Afslutter oprettelsen af en survey og sender manageren tilbage til forsiden.



Figur 5: Forsiden af FeedbackApp for en manager når der er tilføjet en survey. Her kan manageren bruge linket "Try survey" til at afprøve den valgte survey.

USER STORY 2 # OG 3

En manager vil gerne redigere en survey (2) eller åbne/lukke en survey (3). Den første af disse user stories kan gennemføres ved at bruge linket "Edit Survey", som vist på Figur 5. Dette starter samme flow, som når man opretter en survey, bare med spørgsmål og feedback tilføjet fra starten. En forbedring af denne user story ville være, at det muligt også at redigere titel og beskrivelse.

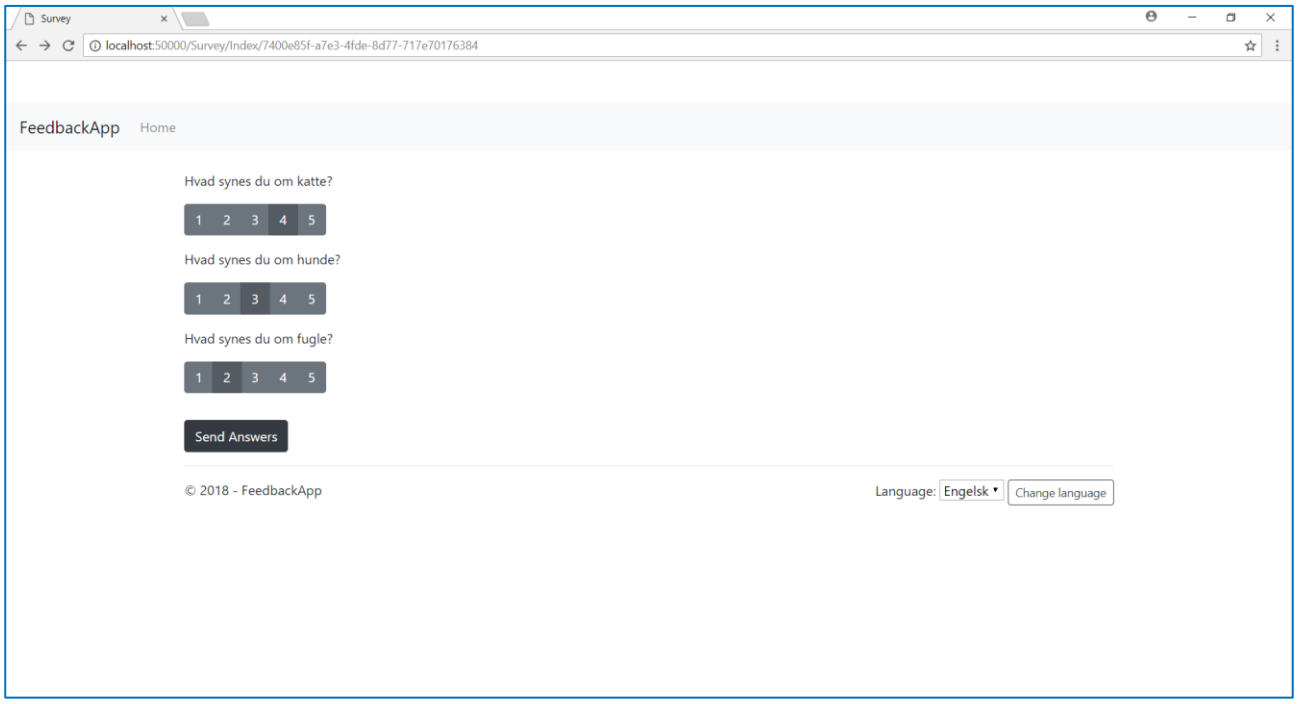
Åbne/lukke vil kunne åbne og lukke en survey for besvarelser fra almindelige users. Dette er ikke implementeret.

USER STORY 4

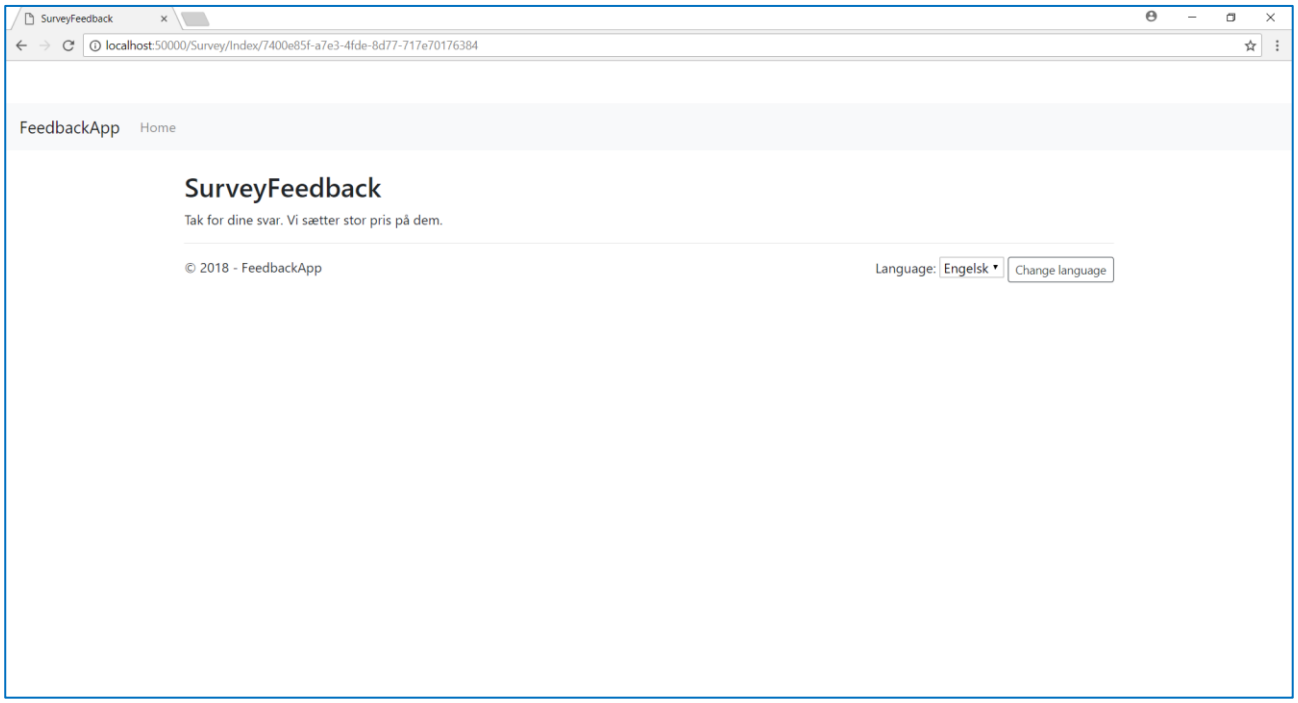
Som vist på Figur 5 er det muligt for en manager at afprøve en survey med linket "Try survey". Dette sender manageren videre til en side, hvor spørgsmål kan besvares og derefter kan den relevante feedback ses. Dette er vist på Figur 6 og Figur 7.

USER STORY 5

Denne user story er halvt implementeret. Som FeedbackApp er nu, er det muligt at besvare en survey. Det gøres af en manager. En user vil skulle gøre det på helt samme måde. De vil få et link til den samme side, som når en manager afprøver en survey. For en user skal besvarelsen bare gemmes, og denne del er ikke implementeret.



Figur 6: PÅ denne side er en manager ved at afprøve en survey.



Figur 7: På denne side vises den feedback en bruger vil få, hvis spørgsmålene er besvaret som på Figur 6.

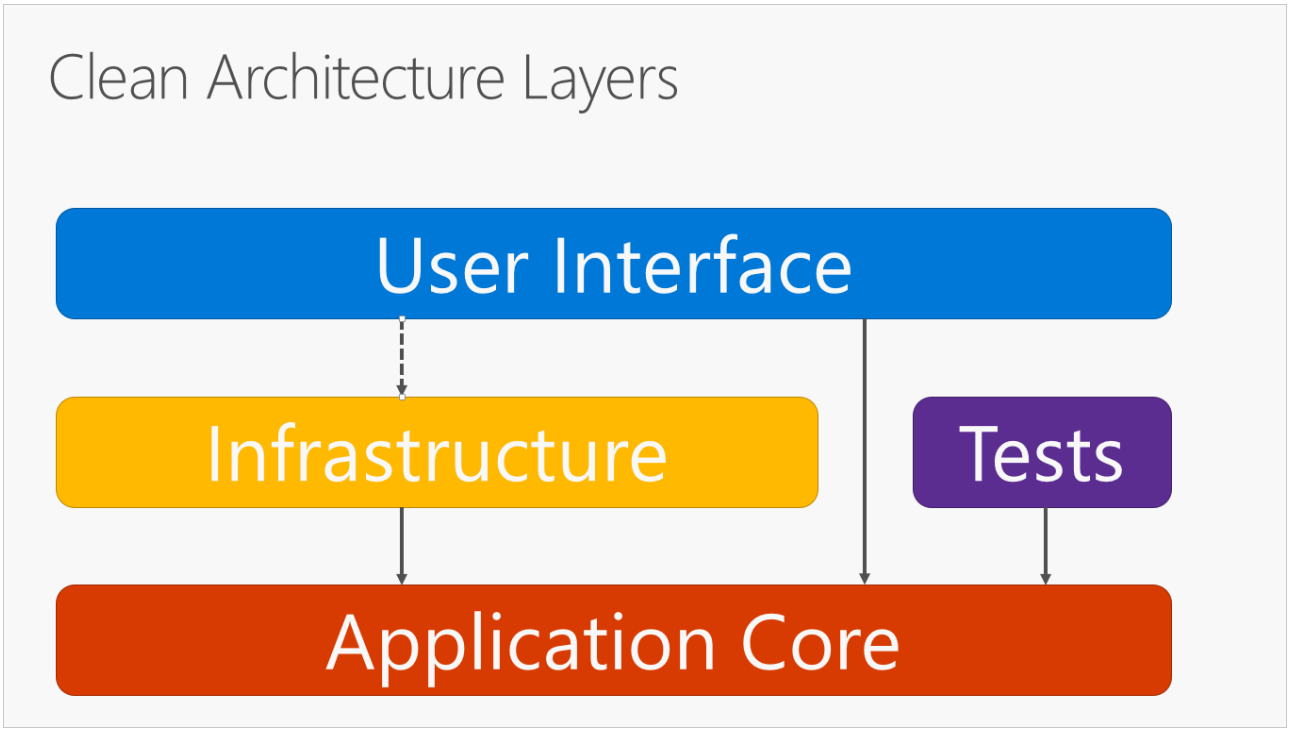
ARKITEKTUR

Dette afsnit indeholder en gennemgang af, hvordan FeedbackApp er bygget op fra et udviklingsperspektiv, altså koden. Det er ikke alle dele af koden, der vil blive remset op, men i stedet vil dele, der er relevante for arkitektur, design eller vigtig funktionalitet blive gennemgået.

FeedbackApp er bygget op som en .NET Core 2.0 MVC applikation. Den er delt i forskellige lag, med hver deres ansvarsområde, som man kan se på Figur 8. Applikation Core er applikationens kerne, der indeholder models og interfaces, samt evt. et service lag, der implementerer nogle af disse interfaces. UI laget indeholder alle MVC Views, Controllers og ViewModels. Infrastructure laget indeholder adgang til data, som f.eks. Repositorys, der igen implementerer Interfaces fra Application Core [3].

Jeg har ikke valgt at dele koden op i forskellige projekter, der repræsenterer disse lag, da projektet er forholdsvist lille. Inddelingen i lag eksisterer dog stadig, da den handler om, hvilke afhængigheder der er mellem de forskellige klasser. Pointen er, at alle compile time afhængigheder skal pege mod Application core laget.

Ved hjælp af interfaces og dependency injection kan man holde en løs kobling mellem sine klasser og skabe en Clean [3] arkitektur.



Figur 8: Diagram over lag og afhængigheder i Clean arkitekturen. De stiplede pile repræsenterer runtime afhængigheder og de solide pile repræsenterer compiletime afhængigheder. Figuren er taget direkte fra Learning ASP.NET Core 2.0 [1].

APPLICATION CORE

I dette projekt består Application Core laget af alle models; Answers, Feedback, Question, Condition og Survey, samt interfaces; IFeedbackService og ISurveyService.

Service laget indeholder de to services SurveyService og FeedbackService. Disse bliver injected ind i de relevante controllers ved hjælp af .NET Core's indbyggede dependency injection (DI). Disse services skal hjælpe med at holde applikationens komponenter løst koblet [1] og DI bliver muligt med brugen af Interfaces og dependency inversion princippet [3].

INFRASTRUCTURE

Dette lag består af de klasser, der har noget at gøre med adgang til data. Det vil sige EF Core DBContext (FeedbackContext), migrations og repositories, der f.eks. henter data ind fra andre kilder.

For at have et eksempel på data fra andre kilder, har jeg tilføjet et CatRepository, der implementerer interfacet ICatRepository. Disse tilhører henholdsvis infrastructure laget og application core laget.

Man vil så fra UI laget kunne hente en kat ved hjælp af CatRepository, men fordi den er implementeret gennem ICatRepository, som hører til Application Core bliver der ikke skabt en compile time afhængighed mellem UI laget og infrastructure laget. Dette er implementeret som et eksempel i TagHelperen CatTagHelper, der indsætter et billede af en tilfældig kat. Vil man se det virke, kan man prøve at oprette en undersøgelse med titlen "kat".

UI

Dette lag indeholder alle controllers, views og viewmodels. Jeg har valgt at bruge viewmodels, da det hjælper med at holde UI og forretningslogik adskilt, hvilket er et godt design princip, der er i overensstemmelse med Separation of Concerns [3]. Det betyder, at ændres der i et View på en måde, så det kræver ændringer i den model, som det View bruger, så skal man kun lave ændringer i den tilhørende viewmodel og ikke i en model, som hører til i application core laget. Det betyder også, at man kan holde sine application core models fri for properties, der kun er relevante i en UI sammenhæng.

I dette projekt har jeg valgt at have f.eks. tilføjelse af spørgsmål og redigering af spørgsmål på samme side, da jeg synes det gav den bedste brugeroplevelse. Det kræver noget UI logik, og derfor fik jeg hurtigt brug for viewmodels frem for models.

ViewComponents og partial views

På forsiden af FeedbackApp kan en manager se sine oprettede surveys i en tabel. Denne tabel har jeg valgt at lave som en ViewComponent og ikke f.eks. et partial view. Det har jeg, fordi denne tabel så kan bruges andre steder, hvis det skulle blive nødvendigt, uden at man behøver

tilføje kode i en controller, der henter alle surveys. Det står denne ViewComponent for. Når der senere skal implementeres login for managers, så er det kun i denne ViewComponent, at der skal ændres kode, så man kun henter surveys, der er oprettet at den enkelte manager.

Projektet indeholder også nogle partial views. Ud over _Menu og _Layout drejer det sig om _SelectLanguagePartial og _Questions partial. Ingen af disse kræver noget controller logik, og det var derfor ikke nødvendigt at oprette dem som ViewComponents. Deres funktion er mere at opdele koden, for at gøre den mere overskuelig, og undgå gentagelse af den samme kode i overensstemmelse med DRY princippet [3].

Unit Testing

Løsningen indeholder et test projekt, der er et xUnit Test Project. Dette projekt indeholder kun en enkelt klasse; FeedbackServiceTests. Denne klasse tester specifikt funktionen FeedbackService.GetFeedback(). Funktionen indeholder en del logik for at finde den helt rigtige feedback, på baggrund af de svar en bruger har givet på en survey. Jeg valgte derfor at bruge Test Driven Development (TDD) [1] til udviklingen af denne funktion. Den fremgangsmåde har flere fordele. Man bliver tvunget til tidligt at gennemtænke logikken som funktionen skal udføre, da man starter med at skrive sine tests fremfor selve funktionen. Man har nogle grundige tests, der kan køres igen, hvis der laves ændringer i funktionen og til sidst sikrer det selvfølgelig også at funktionen virker som forventet.

De første tre tests er lavet som Fact tests. Den fjerde er en Theory, der henter data fra funktionen `GetData()`. Med denne fremgangsmåde kan man bruge den samme testkode, men med forskellige data scenarier.

Localization

Der er tilføjet localization til applicationen, så den nemt og hurtigt kan oversættes til andre sprog. Localization er konfigureret med både services og middleware. For at tilføje localization har jeg tilføjet `.AddLocalization()`, `.AddViewLocalizatio()` og `.AddDataAnnotationsLocalization()` i `ConfigureServices()`. Dette gør det muligt at injecte `IStringLocalizer` ind i alle klasser og `IViewLocalizer` ind i alle views. For at en gøre det muligt at skifte sprog har jeg brugt et middleware der hedder `UseRequestLocalization()`. Sproget kan vælges af en bruger i det partial view, der hedder _SelectLanguagePartial og bliver sat i HomeControlleren i `SetLanguage()`, der sætter sproget ved hjælp af provideren `CookieRequestCultureProvider` [4].

Jeg har dog ikke prioriteret at oprette ressource filer for alle views, og det er derfor kun forsiden (Home), hvor localization er implementeret.

Database

I FeedbackApp er persistering af data indført ved hjælp af en SQL database. Dette er den en god og sikker måde at gemme data frem for f.eks. at gemme det i tekstfiler [1]. Det er også muligt at vælge en NoSQL løsning, som MongoDB. NoSQL er en måde at gemme ustruktureret data, hvor data kan gemmes på mange måder. Det gør det meget anvendeligt i store agile projekter, hvor data hurtigt kan ændre sig [5]. I tilfældet af FeedbackApp er det ikke et stort og agilt projekt, og der ud over er udvikleren ikke erfaren med NoSQL. Derfor var det umiddelbare valg SQL.

Til databasen og kommunikation med denne, er brugt SQL Server og Entity Framework Core 2.

For at åbne kommunikation til databasen og opdatere entiteter er der defineret en `DbContext`, der hedder `FeedbackContext`. I denne er der implementeret en DbSet property for hver model/entitet i databasen. Dernæst er DbContext registreret som en service med DI i startup klassen, med connection string og database provider som parametre. Til sidst er der brugt Migrations til at lave databasen [1] [6] .

I SurveyService kan man se, hvordan kommunikation med databasen foregår. Her får servicen `dbContextOptions` ved DI i constructoren, som så bruges når new FeedbackContext kaldes.

VIDEREUDVIKLING

I dette afsnit vil jeg gennemgå den vigtigste ting der mangler for at udviklingen af FeedbackApp er færdig. Det drejer sig om login. Så flere managers kan bruge appen og almindelige users kun kan besvare surveys.

Login

Identity configuration

Login funktionalitet kan tilføjes forholdsvist enkelt ved at bruge ASP.NET Core Identity. Jeg vil her kort gennemgå de skridt der mangler for at implementere login med Identity.

Først skal der defineres en bruger klasse og denne skal nedarve fra IdentityUser. Det er muligt at undvære denne klasse og bare bruge IdentityUser direkte, men på denne måde er det muligt at tilføje flere properties brugeren, hvis vi får brug for det.

Herefter skal Identity registreres i vores services med `AddIdentity`(), `AddEntityFrameworkStores`() og `AddDefaultTokenProviders`(). De to sidste fortæller at vi vil bruge Entity Framework til at gemme vores data og at vi vil bruge default token providers. I Configure metoden skal

middlewaren `UseIdentity()` tilføjes. Til sidst skal FeedbackContext nedarve fra IdentityDbContext.

Det er nu muligt at bruge Authorize attributten i en controller til at kræve login.

Login side

Efter configurationen vil man blive sendt videre til en login side /Account/Login, hvis man rammer en controller action der er dekoreret med [Authorize]. Der mangler altså at blive lavet en login side og noget der kan styre login og logout.

Til dette formål kan man bruge Identitys to services UserManager og SignInManager. UserManager understøtter ting som at oprette en bruger og sætte password. SignInManager styrer ting forbundet med login, som f.eks. authentication cookies.

Her fra skal der laves en AccountController med login action og register action, en LoginModel og et View til login siden og et View til register siden. AccountControlleren skal gøre brug af de to services UserManager og SignInManager til at oprette brugere og til at styre login [7].

Refaktorering og fejlhåndtering

Til sidst vil jeg nævne et par steder jeg synes kunne forbedres. Hele CreateSurveyController er blevet en smule lang og kompliceret, og den kunne trænge til en refaktorering. Den kunne evt. deles i to controllers, der håndterer spørgsmål og feedback hver for sig. Der ud over kunne man ligge noget logik omkring dekorering af viewmodels ud i private funktioner eller sågar UI services.

Jeg kunne også tænke mig at lave noget mere custom fejlhåndtering. F.eks. bliver fejl fra databasen nærmest ikke håndteret lige nu. Jeg bruger middleware som `UseExceptionHandler()` og `UseStatusCodePages()`, men det er ikke noget jeg har arbejdet målrettet med i dette projekt. Prøver man f.eks. at slette et spørgsmål i en survey, som er indeholdt i en Condition, så vil databasen smide en fejl. Herefter, bliver brugeren sendt videre til en 404 side. Det kunne håndteres meget bedre og mere brugervenligt. Det kunne være en god udfordring at lave et middleware selv, der kunne håndtere de forskellige fejl og vise brugeren af siden nogle mere brugbare beskeder eller fejlsider.

REFERENCER

[1] J. D. Oliveira og M. Bruchet, Learning ASP.NET Core 2.0, Packt Publishing, 2017.

[2] M. Cohn, »User Stories,« [Online]. Available: https://www.mountaingoatsoftware.com/agile/user-stories. [Senest hentet eller vist den 22 maj 2018].

[3] S. Smith, Architecting Modern Web applications with ASP.NET Core and Microsoft Azure, DevDiv, .NET and Visual Studio product teams, 2017.

[4] A. Lock, »Adding Localisation to an ASP.NET Core application,« .NET Escapades, [Online]. Available: https://andrewlock.net/adding-localisation-to-an-asp-net-core-application/. [Senest hentet eller vist den 24 maj 2018].

[5] XPlentyBlog, »The SQL vs NoSQL Difference: MySQL vs MongoDB,« [Online]. Available: https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2. [Senest hentet eller vist den 26 maj 2018].

[6] Microsoft documentation, »Getting Started with EF Core on ASP.NET Core with a New database,« [Online]. Available: https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/new-db. [Senest hentet eller vist den 25 maj 2018].

[7] A. Hanson, »ADDING ASP.NET CORE IDENTITY TO AN EXISTING PROJECT,« [Online]. Available: https://hanson.io/bootstrapping-asp-net-core-week-4/. [Senest hentet eller vist den 25 maj 2018].