Feedback App

Eksamensprojekt til kurset *Webudvikling – Backend*

App og Rapport er udarbejdet af Louise Høpfner

Aflevering: 30/5 2018

Indhold

[Indledning 3](#_Toc514960964)

[Overordnet beskrivelse af appen 3](#_Toc514960965)

[Funktionalitet 3](#_Toc514960966)

[User stories 3](#_Toc514960967)

[User story 1 # 4](#_Toc514960968)

[User story 2 og 3 # 7](#_Toc514960969)

[User story 4 # 7](#_Toc514960970)

[User story 5 ## 8](#_Toc514960971)

[Arkitektur 8](#_Toc514960972)

[Application Core 9](#_Toc514960973)

[Infrastructure 9](#_Toc514960974)

[UI 10](#_Toc514960975)

[ViewComponents og partial views 10](#_Toc514960976)

[Unit Testing 11](#_Toc514960977)

[Localization 11](#_Toc514960978)

[Logging 11](#_Toc514960979)

[Frontend 11](#_Toc514960980)

[Videreudvikling 11](#_Toc514960981)

[Database 11](#_Toc514960982)

[Login 11](#_Toc514960983)

[Referencer 12](#_Toc514960984)

# 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].

# 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. **# #** |

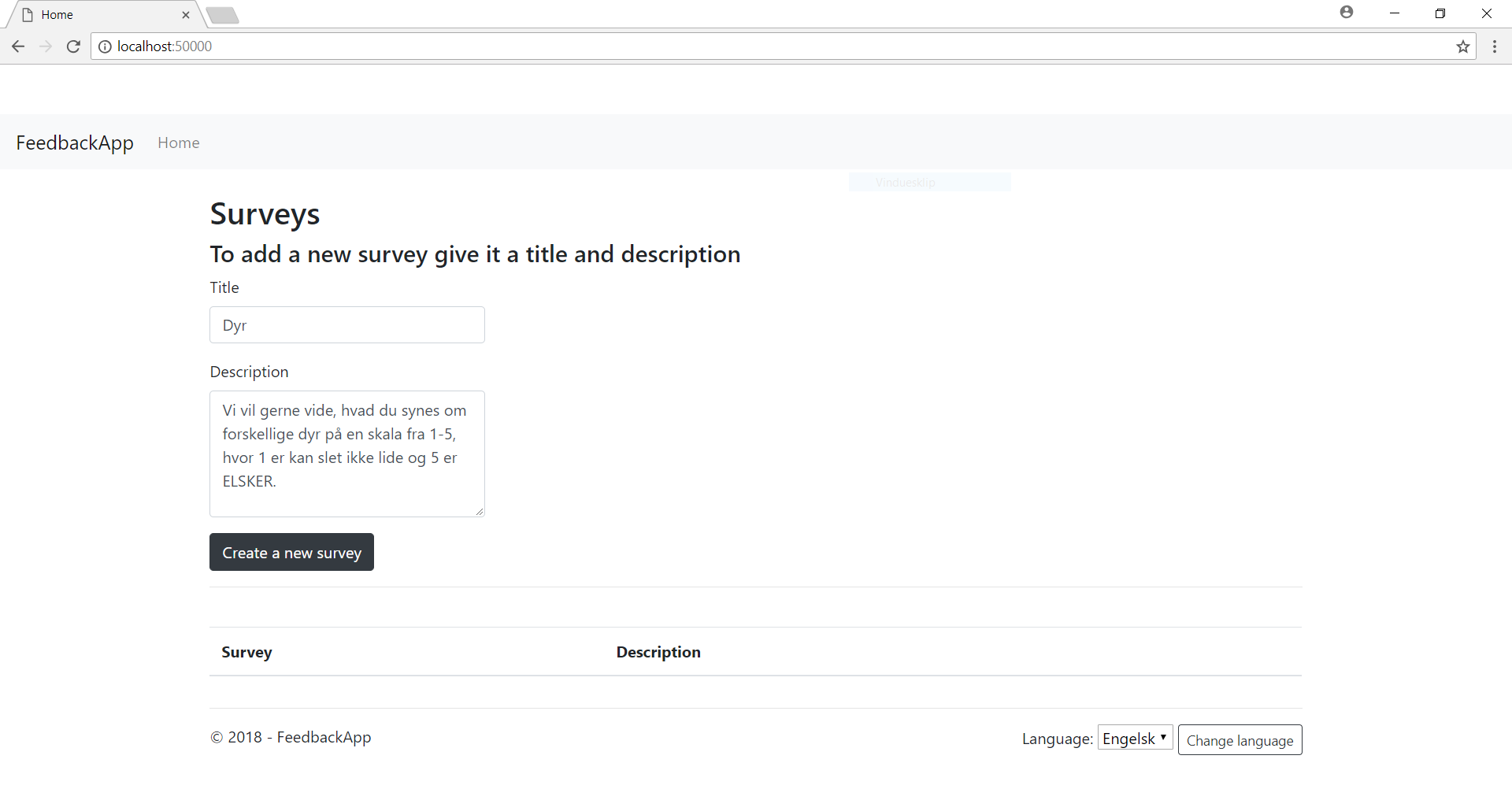
Tabel 1: Oversigt over user storie i app'en FeedbackApp. Stories markeret med grøn # er implementeret. Stories markeret med rød # er ikke implementeret.

### User story 1 **#**

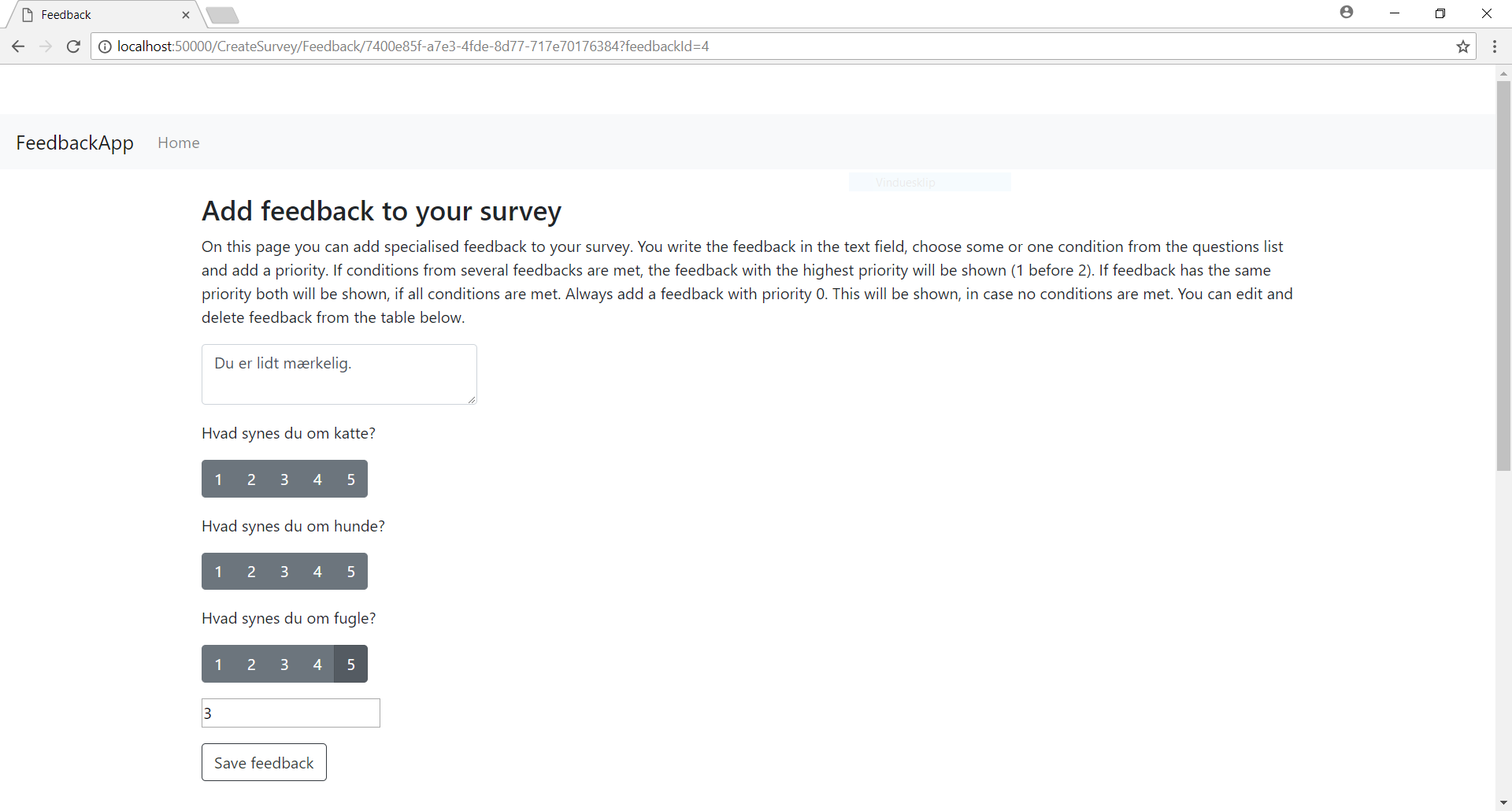
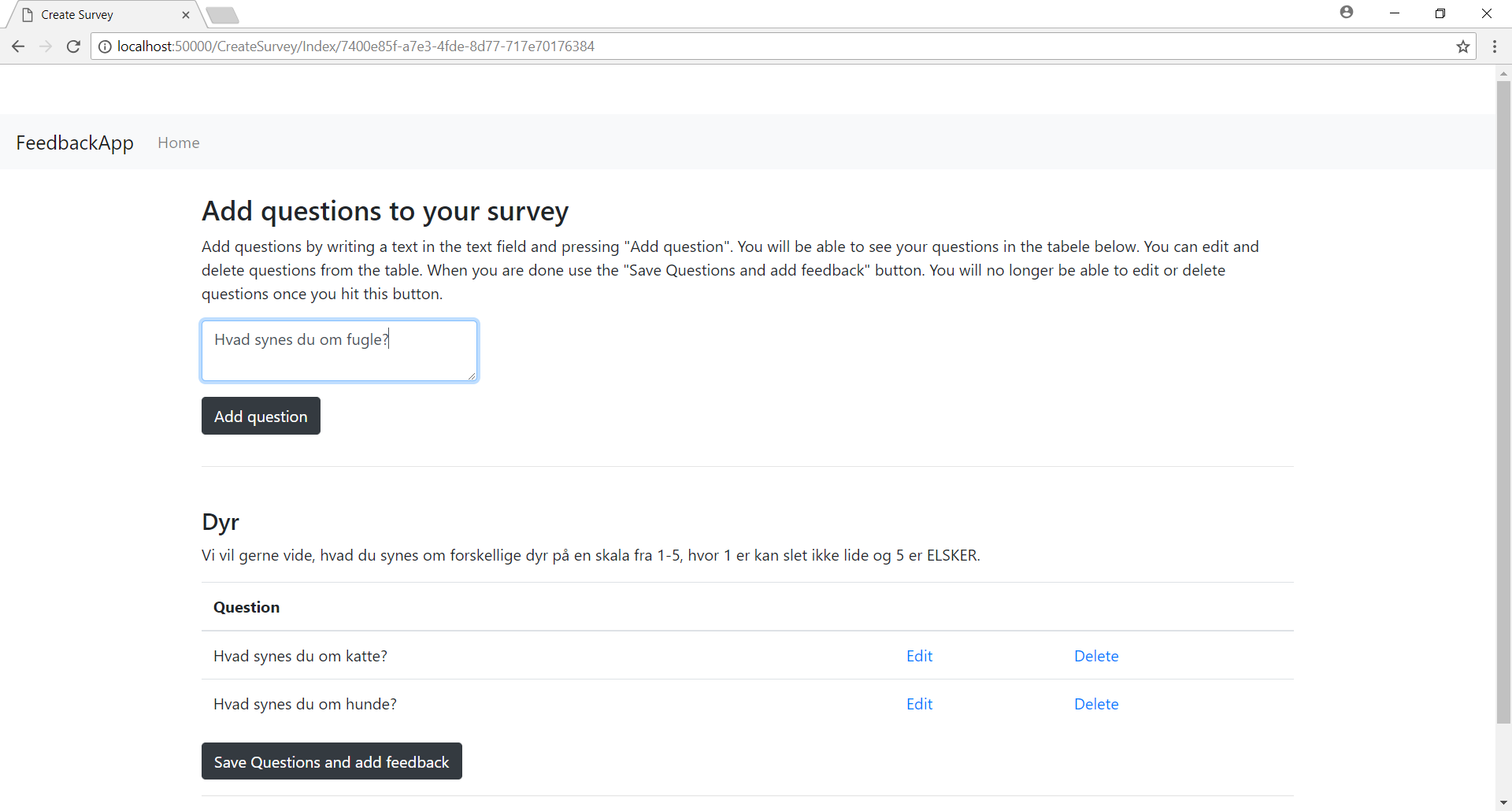
Denne user story dækker over den meste funktionalitet i app’en. I dette afsnit gennemgår jeg, hvordan denne user story er implementeret fra et manager perspektiv. Altså, ikke den bagvedliggende kode.

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 er opfyldt.

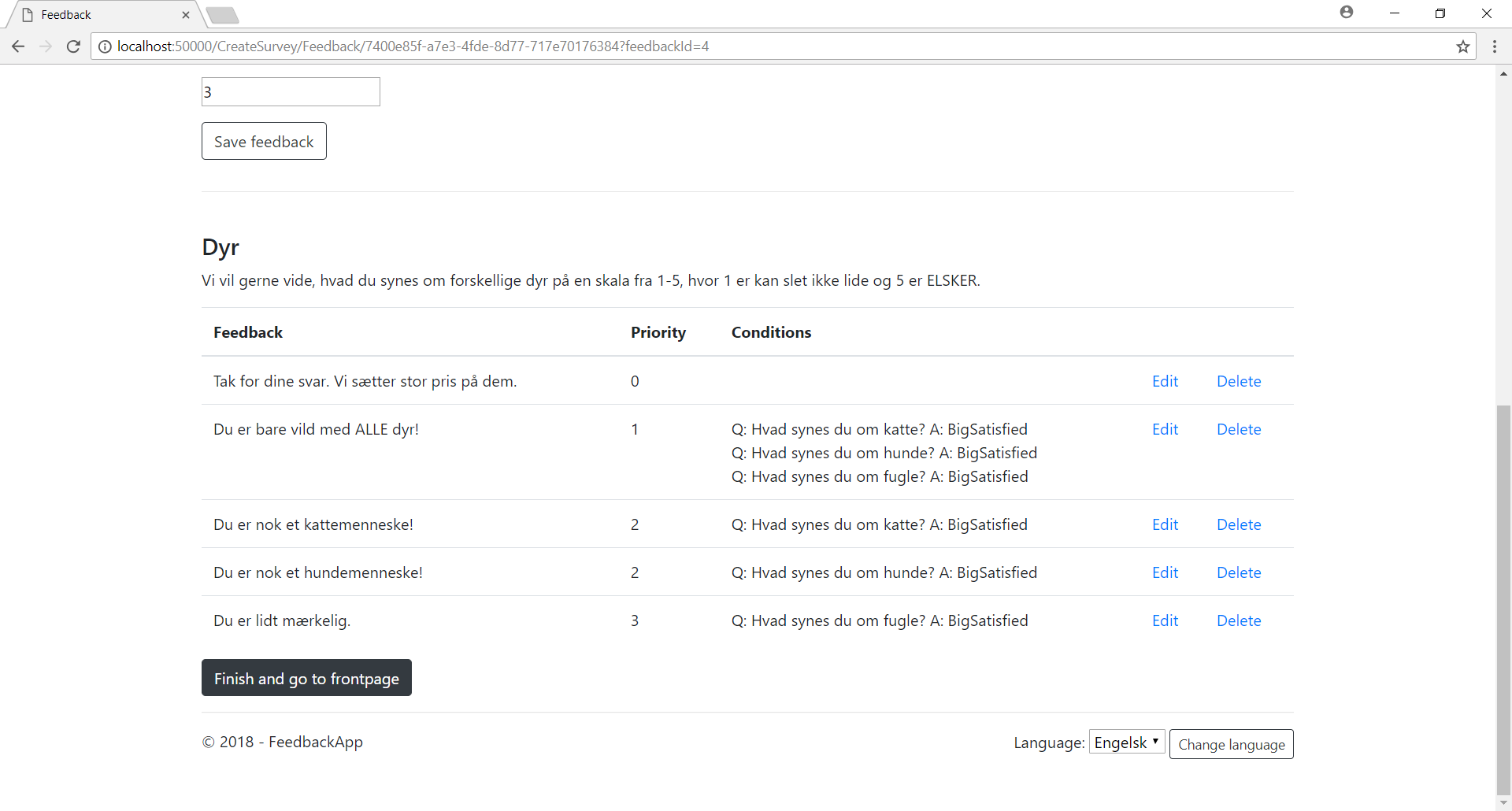
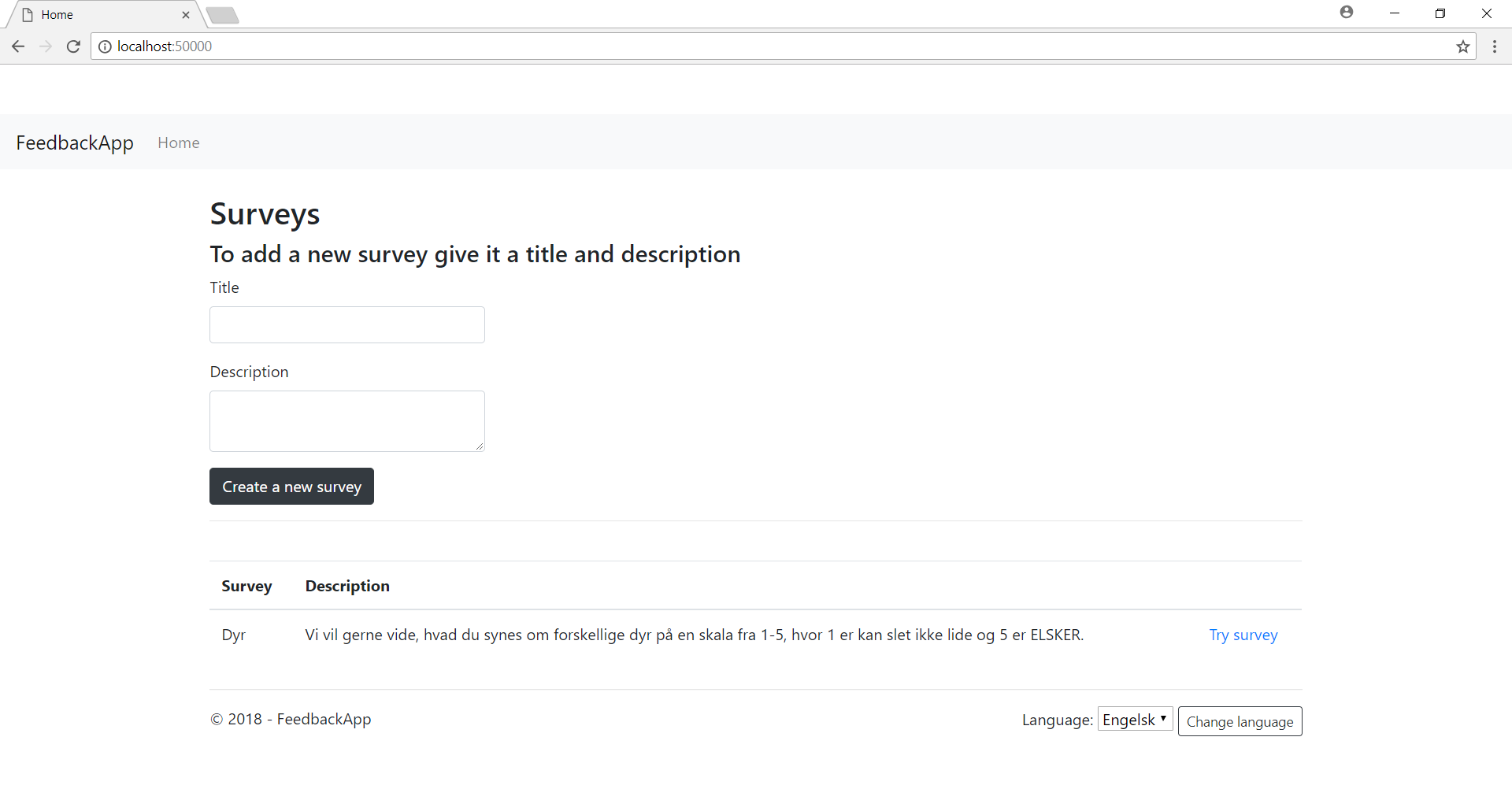
På Figur 4 er vist noget feedback med betingelser og prioritet. Kan en bruger ingen dyr lide, vil feedbacken være ”Tak for dine svar. Vi sætter stor pris på dem”. Kan man lide både katte og hunde vil man få ”Du er nok et kattemenneske!” og ”Du er nok et hundemenneske”. Svarer man at man kan lide alle 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: 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.

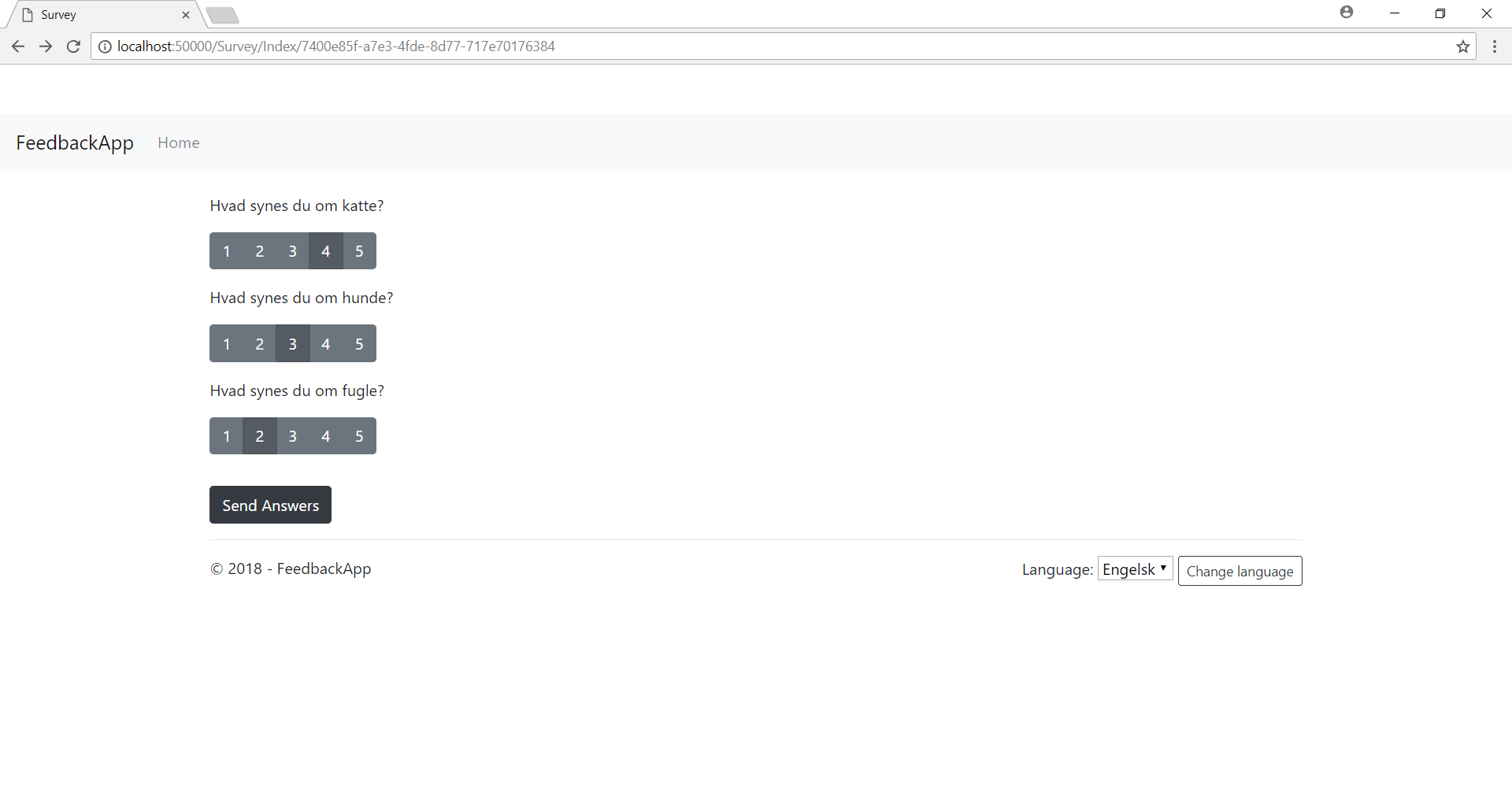
Figur 5: Nederste del af feedback siden. Her vises en tabel med en oversigt over feedback der er tilføjet. Hver feedback kan redigeres og slettes. Knappen ”Finish and go to frontpage”. Afslutter oprettelsen af en survey og sender manageren tilbage til forsiden, som vist på Figur 5.

### User story 2 og 3 **#**

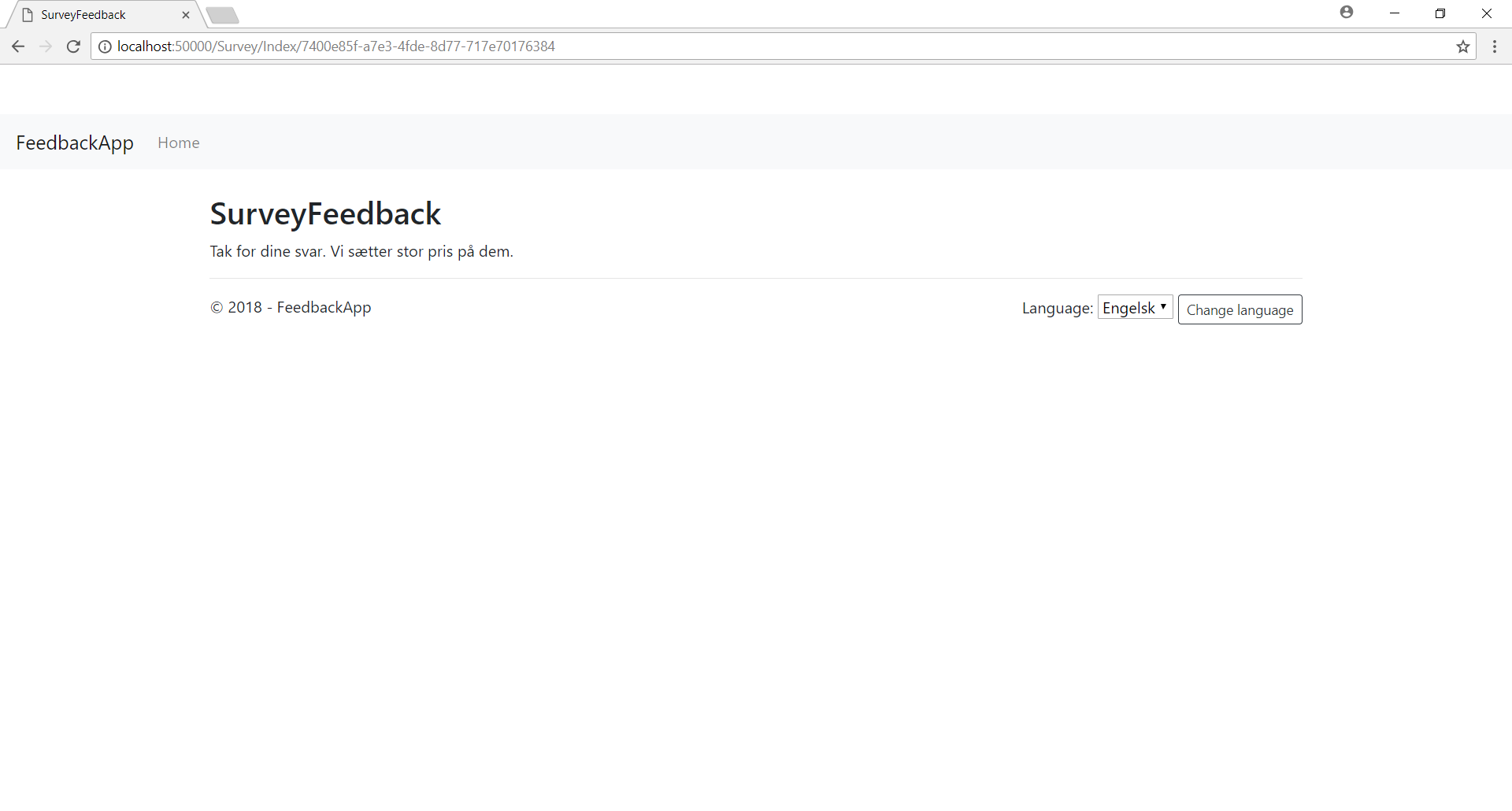
En manager vil gerne redigere en survey (2) eller åbne/lukke en survey (3). Disse user stories er ikke implementeret. De ville begge bestå i en knap i tabellen med surveys på forsiden. Rediger ville sende en manager ind i flowet fra user story 1, men med allerede eksisterende spørgsmål og feedback. Åbne/lukke vil kunne åbne og lukke en survey for besvarelser fra almindelige users.

### 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.



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.

### 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.

# 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 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

Da data persistence ikke er implementeret endnu, i form af f.eks. kald mod en database og Entity Framework, er der ikke noget i dette lag endnu.

For at have et eksempel på noget i dette lag, har jeg dog 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(). Denne funktion 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. Denne 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.

## Localization

Der er tilføjet localization til applicationen, så den nemt og hurtigt kan oversættes til andre sprog. Localization er konfigureret, både services og middleware, og en bruger kan vælge sprog nederst på alle sider. Jeg har dog ikke prioriteret at oprette ressource filer for alle views, og det er derfor kun forsiden (Home), hvor localization er implementeret. Til konfigurationen har jeg fulgt guiden [Adding Localisation to an ASP.NET Core application](https://andrewlock.net/adding-localisation-to-an-asp-net-core-application/) [4].

# Videreudvikling

## Database

## Login

# 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]. |