Hovedopgave: Projekt RateMyDebate

Indholdsfortegnelse

1. Introduktion

a. Glossary

ASP.NET - Web application framework

Client - ID/bruger forbundet til Hub.

Cross-domain support - Tillader udveksling af resurser fra et domain til et andet.

CRUD-operationer - Create, Read, Update, Delete. Anvendes i henhold til database operationer. Altså indsættelse, læsning, redigering og sletning af database entries.

CSHTML - Filtype anvendt af Razor View Engine i Views

Domain - Unikt ID for en internet resurse.

Hub - Del af SignalR library. Giver metoder, som tillader at kommunikere med forbundne clients.
Microsoft Entity Framework - Object/Relational Mapping framework, som sparer developeren meget arbejde ved at give automatiserede måder at tilgå og gemme data i en database.

- Database Context - Lag mellem bruger og database. Kan queue ændringer og gemme dem til den underliggende database
- Code First - Fremgangsmåde, som tillader developeren at skabe databaserved at skrive klasser i C# og migrere dem som datatables.

IDE - Integrated development environment - Software anvendt til softwarekonstruktion i.e. source code editing, debugging, testing, building, etc...

Library/bibliotek - Samling af resurser importeret til udviklingen af softwaren.

MVC - *Model-View-Controller* - Core software pattern anvendt til at fordele ansvar og information i applikationen.

- Model - Indeholder tilstande for et objekt i applikationen
- View - Visuel repræsentation for brugeren
- Controller - Mellemand for bruger og applikationen. Kan ændre modellens tilstand, f.eks. ved at tage input fra viewet.

NuGet Package Manager - Feature i Visual Studio anvendt til import og organisering af værktøjer til brug i projektet.

Package Manager Console - Feature i Visual Studio, som tillader brugen af PowerShell kommandoer.

Partial View - View som kan vises inde i et hvert andet view.

PowerShell - task automation og configuration management framework anvendt via kommandoer i Package Manager Console.

Pipeline - Applikationen's livscyklus.

Real-time - Den aktuelle tid for en handling.

Repository - Class anvendt til CRUD-operationer.

Session state variabel - Globalt unikt ID baseret på cookies, som gives til brugere. Anvendes i autentifikationsprocesser.

- Cookie - Data opbevaret i brugerens browser

View engine - Har til ansvar at synliggøre HTML fra views til browseren.

ViewModel - Samling af flere forskellige objekter og data i en enkelt model til, f.eks., at parse videre til view.

b. Anvendte værktøjer og biblioteker

Microsoft Visual Studio

Visual Studio er IDE'et, som jeg har anvendt til at skabe selve web applikationen og den dertilhørende database.
Al kodning er foregået i Visual Studio. Tilmed er importeringen af SignalR gennemført via. NuGet Package Manageren i Visual Studio. Hele databasen er kodet i C# klasser i Visual Studio og konverteret til en database vha. Package Manager Console featuren.

ASP.NET MVC 5

JQuery

Javascript library. Anvendt til store dele af debat funktionaliteten, f.eks. vha. AJAX calls.

SignalR 2.0

Officielt open-source Microsoft bibliotek, som baserer sig på websockets. Anvendes til at give real-time funktionalitet til applikationen ved brug af "Hub"-konceptet.

Hubs er grupper af forbindelser, som forbinder flere brugere. Ved at anvende dette bibliotek er det muligt at skabe de ønskede chatrum nødvendige for at holde debatterne.

Entity Framework

.NET framework anvendt til bl.a. at simplificere alle CRUD operationer på databasen.

Ninject

IoC-container(Inversion of Control) anvendt til dependency injection.

Moq

Mocking library til unit testing.

Razor view engine

En af fire mest populære view engines for ASP.NET MVC. Anvender CSHTML og @{ } code blocks.

Owin

Web specifikation for decoupling af web server og applikation. Muliggør hosting af SignalR og clients i andre domains og dermed giver cross-domain support.

SimpleCrypto

Kryptografi library anvendt i krypteringen af passwords vha. hashing algoritmer.

GitHub

Version control værktøj. GitHub tillader mig at gemme mine opdateringer i et online repository. Skulle fejl forekomme eller data blive tabt, kan jeg reverte til en tidligere version af applikationen.

KanBanFlow

Online "whiteboard" værktøj anvendt til planlægning af arbejdsprocessen.

Visual Paradigm

CAD software anvendt til at designe UML diagrammer for projektet.

c. Koncept

Projektet's mål er at skabe en web application, som tillader brugere at skabe, deltage i og tilskue debatter i real-time. Hver debat skabes af en bruger, som opretter denne inden for en af en store vifte a kategorier. Brugeren skal tildele debatten et topic samt en case, altså et emne, samt en nærmere beskrivelse af, hvad brugeren ønsker at debattere. Ydermere sætter brugeren selv en tidsbegrænsning på debatten.
Herefter vil en anden bruger kunne udfordre skaberen af debatten, og andre brugere kan tilskue

chatten samt tildele en enkelt stemme til deres foretrukne debattør.
Når tidsbegrænsningen nås vil debatten slutte, point tælles op, en besked om vinderen gives til alle deltagere og resultatet gemmes i en database.

Kode

C#

Programmeringssprog.
Primært anvendt til at skabe Model og Controller classes

JavaScript

Programmeringssprog.

Anvendt til at give applikationen dynamisk funktionalitet sammen med JQuery.

HTML

Markup language.
Anvendt til layout af web pages.

CSS

Style sheet language.
Avendt til design og formattering af web pages.

d. Problemformulering

"Jeg vil skabe en live debat hjemmeside i ASP.NET frameworket.

Hjemmesiden skal være kompatibel med og køre i de mest populære browsere; Internet Explorer, Mozilla Firefox og Google Chrome.

Som IDE vil jeg bruge Microsoft Visual Studio. ASP.NET kombineret med Visual Studio kan spare enormt meget tid i forhold til nødvendig kodeskrivning, hvilket vil være praktisk i en enmandsgruppe.

For at nå mit mål vil jeg lægge en tidsplan for, hvornår hvilke dele af projektet skal være færdige på hvilke dage. Til dette formål vil jeg bruge værktøjet KanBanFlow, en gratis webapp, som tidsplanen

kan skrives på med stort overblik ved at anvende dens kolonneværktøjer til at sortere opgaverne i "to-do", "do today", "in progress" og "done".

For at reducere bugs og sikre, at koden kører som forventet, vil jeg anvende unit testing i Visual Studio og følge andre software konventioner efter bedste evne. Disse inkluderer f.eks. diagrammer skabt i CAD software såsom Visual Paradigm og normalisering af databasen.

Da jeg er alene om projektet, og ikke har nogen kunde at arbejde med, vil Kanbanflow agere som en stor del af den software designs process, hvor ønskerne for produktet hives frem. Derfor vil Kanbanflow fungere som overblik for user stories, som jeg selv skriver, giver tidsramme og prioritet. Disse krav, som jeg stiller mig selv, vil jeg så herefter selv forsøge at nå i sprints efter SCRUM udviklingsmetoden.

Jeg vil primært fokusere på kodning og software designet, og ikke så meget på business aspektet, da jeg føler et større behov for at fokusere på de to førstnævnte, og ikke føler mig særligt klar til at lave en fuld analyse af markedet, som jeg vil publicere på.

Som en del af tidsplanen, vil jeg følge en regelmæssig "Nine-to-five"-arbejdsdag, og opdatere tidsplanen og Kanbanflow ved enden af hver dag."

Begrundelse for valg af dette projekt

Jeg har længe ønsket at forsøge at skabe en form for online samfund, og da det var tid til at skrive opgaven, gik jeg med dette koncept i tankerne.
Projektet ville give mig mulighed for tre områder, som jeg gerne ville forbedre og forsøge mig i, hvilke inkluderede ASP.NET, en applikation med login og adskillige områder, som anvender bruger autentifikation og til sidst en form for live del af applikationen med mere kompleks logik.

Jeg har altid været en del af flere forskellige online samfund og været interesseret i, hvordan en sådan side skabes, vedligeholdes og kommercialiseres.
Kommercialiseringen er ikke implementerer i produktet, men jeg vil komme ind på forskellige tilgange til at monetarisere produktetet.

2. Arbejdsmetoder og planlægning

a. Fundamentet for projektet

På trods af at have gået med idéen om projektet i en længere periode, var mit første mål ved projektet's start, at udlægge så mange krav for projektet som muligt, og planlægge mit forløb derefter.

Derfor anvendte jeg Use Case Requirements teknikken til at finde de grundlæggende, nødvendige funktioner i projektet.

I forbindelse med nødvendige at finde ud af, hvilke funktioner ville være nødvendige, lavede jeg adskillige papir prototyper. Disse tjente ikke blot til, at analysere krav, men blev primært skabt til at angive et grundlæggende design for projektet, da jeg på ingen mulig måde er designer af natur.

Efterfølgende fortsatte jeg til at forsøge at designe en forside for web applikationen, som i projektet's formål tjente til at finde et passende tema for siden, hvilket involverede en masse CSS kodning.

Herefter ville jeg etablere et tidligt database design. Denne gang brugte jeg samme metode som design delen, og udlagde data table classesne på papir med deres tilhørende variabler og relationer til hinanden.

Den samlede proces tog omkring to uger, og var stor hjælp i at skabe et solidt fundament for projektet. Jeg fandt samtlige nødvendige krav for at gøre projektet funktionelt, som gjorde det nemt at rykke videre til konstruktionen af applikationen.

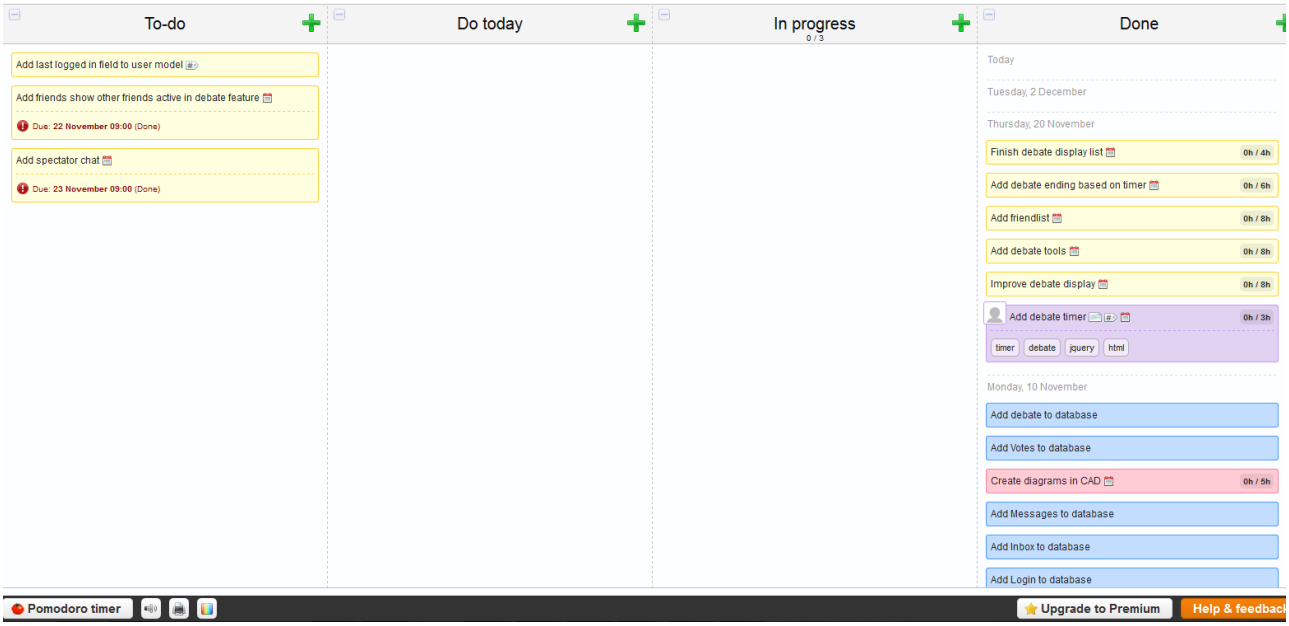
b. Videre organisering og problemer

Hofstadters law: "Everything takes longer than planned even when you take hofstadters law into account."

(Hofstadter, 1979)

Jeg blev bekendt med KanBanFlow et stykke inde i denne fase. Efter at have forsøgt mig med adskillige projekter, uden nogen nedskreven rutine, følte jeg, at det var nødvendigt med en bedre planlægning. Da jeg havde problemer med at finde en arbejdsrutine i dette projekt valgte jeg at forsøge mig med KanBanFlow.

KanBanFlow havde til formål at give et klarere overblik for, hvilke features som skulle implementeres og hvornår.



Figur 1 - KanBanFlow

Jeg nævnte Hofstadter's lov i starten af sektionen, da jeg netop ønskede at undgå at fejlestimere, hvor lang tid hver task ville tage.

På mange måder fejlestimerede jeg stadigvæk i høj grad, hvor meget tid ville være nødvendig. På trods af applikationens funktionalitet er der stadigvæk store mangler, og det føles ikke som et færdigtpoleret og fuldstændigt intuitivt produkt.

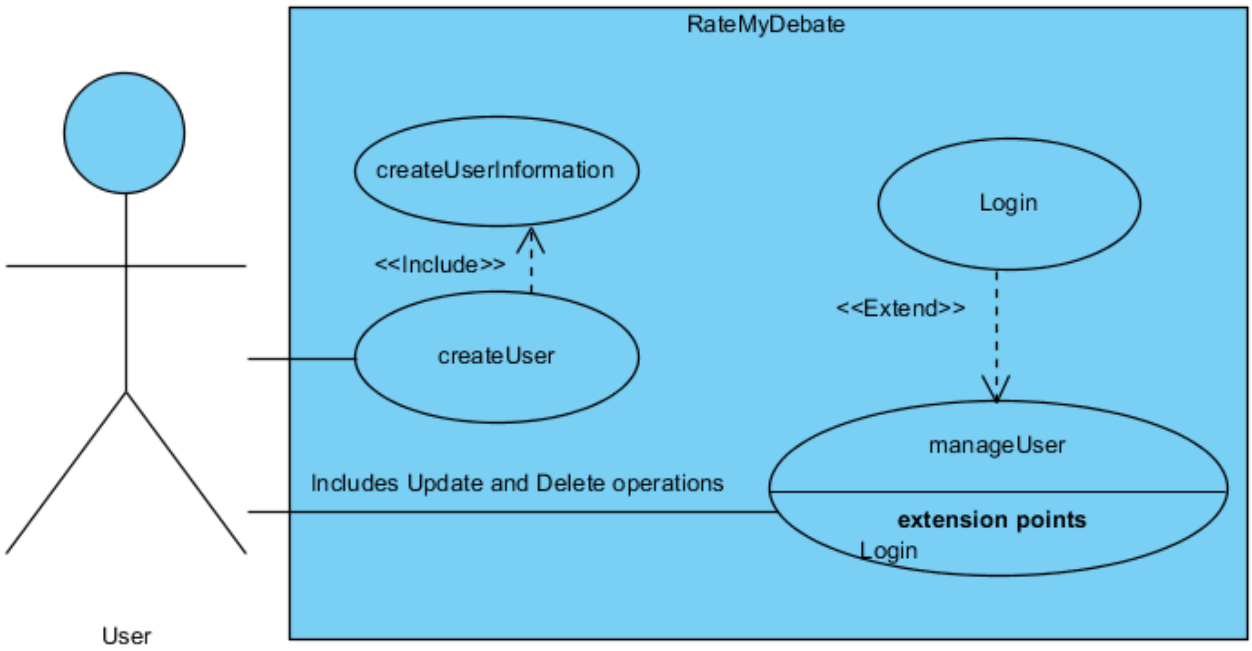
Jeg afsatte tid til at færdiggøre så mange features som muligt indtil d. 1. December.
Debat funktionen var naturligvis den højest prioriterede, men færdiggørelsen af denne var langt sværere end forventet. Derfor har der ikke været meget tid til at finpolere og færdiggøre de sidste touches, som vil gøre applikationen fuldkommen brugervenlig.

Jeg har sat de sidste to uger af til at skrive resten af rapporten, og forsøge at færdiggøre så meget af projektet som muligt. I næste sektion vil jeg komme ind på, hvordan jeg føler denne tidshåndtering har fungeret.

3. Marketing

a. Requirements analyse

Use case based requirements:



Figur 2 - User Requirements

Use Case: CreateUser

Primary Actor:

Brugeren

Main flow:

Brugeren registrerer sine ønskede login credentialer.

Sub flow:

Når login credentialer er validerede som gyldige, vil brugeren sendes videre til at udfylde brugerinformation(use case CreateUserInformation).

Alternate flow:

Ugyldige login credentialer indtastes og brugeren sendes tilbage til at forsøge igen.

Preconditions:

Ingen

Postconditions:

Brugerens login information er klar til registrering.

Use Case: CreateUserInformation

Main flow:

Brugeren registrerer sine ønskede brugerinformationer. Bruger login information samt brugerinformation gemmes i databasen, og brugeren er registreret.

Alternate flow:

Ingen

Preconditions:

Bruger har indtastet ønsket login information.

Postconditions:

Brugerens login og brugerinformation registreres i databasen.

Use Case: Login

Primary Actor:

Brugeren

Main flow:

Brugeren indtaster sin login information succesfuldt og logges ind på sin konto.

Alternate flow:

Ugyldige login credentialer indtastes og brugeren sendes tilbage til at forsøge igen.

Preconditions:

Ingen.

Postconditions:

Brugeren logges ind.

Use Case: Edit User

Primary Actor:

Brugeren

Main flow:

Brugeren indtaster og registrerer for at ændre sin login- eller brugerinformation.

Alternate flow:

Ugyldig login- eller brugerinformation indtastes. Brugeren sendes tilbage for at forsøge igen.

Preconditions:

Brugeren skal være logget ind.

Postconditions:

Brugerens nyindtastede information registreres i databasen.

Use Case: DeleteUser

Primary Actor:

Brugeren

Main flow:

Brugeren bekræfter at slette sin konto og associerede bruger information, som herefter fjernes fra databasen.

Alternate flow:

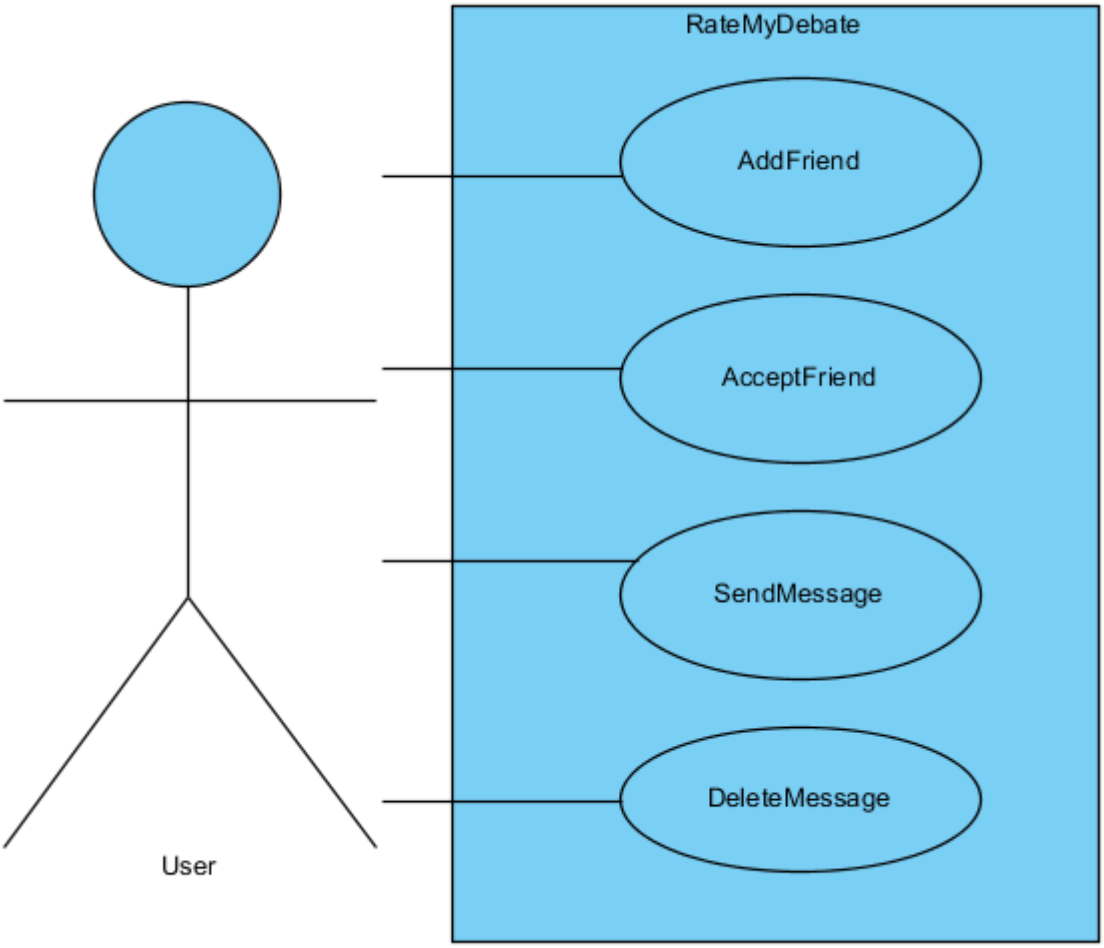
Ingen

Preconditions:

Brugeren er logget ind

Postconditions:

Brugeren er slettet fra databasen.



Figur 3 - Friend Requirements

Use Case: AddFriend

Primary Actor:

Brugeren

Main flow:

Brugeren tilføjer en anden bruger, enten ved indtastning af navn eller valg fra liste, og afventer brugerens accept.

Alternate flow:

Ugyldig venneinvitationsmodtager. Brugeren bedes om at forsøge igen.

Preconditions:

Brugeren er logget ind

Postconditions:

Invitation om venskab afventer svar fra modtager.

Use Case: AcceptFriendship

Primary Actor:

Brugeren

Main flow:

Bruger godkender modtaget venneinvitation, og venskabet registreres i databasen.

Alternate flow:

Ingen

Preconditions:

Brugeren er logget ind

Postconditions:

Venskab er registreret i databasen

Use Case: DenyFriendship

Primary Actor:

Brugeren

Main flow:

Brugeren nægter venneinvitation, og invitationen fjernes.

Alternate flow:

Ingen

Preconditions:

Brugeren er logget ind

Postconditions:

Venskabsinvitation fjernes og venskab registreres ikke.

Use Case: SendMessage

Primary Actor:

Brugeren

Main flow:

Brugeren indtaster en besked, som sendes til modtagerens inbox.

Alternate flow:

Ugyldig modtager indtastes. Bruger bedes om at checke og forsøge igen.

Preconditions:

Brugeren er logget ind

Postconditions:

Besked er registreret i modtagers inbox

Use Case: DeleteMessage

Primary Actor:

Brugeren

Main flow:

Bruger vælger en besked til sletning og bekræfter denne. Beskeden fjernes fra brugerens inbox.

Alternate flow:

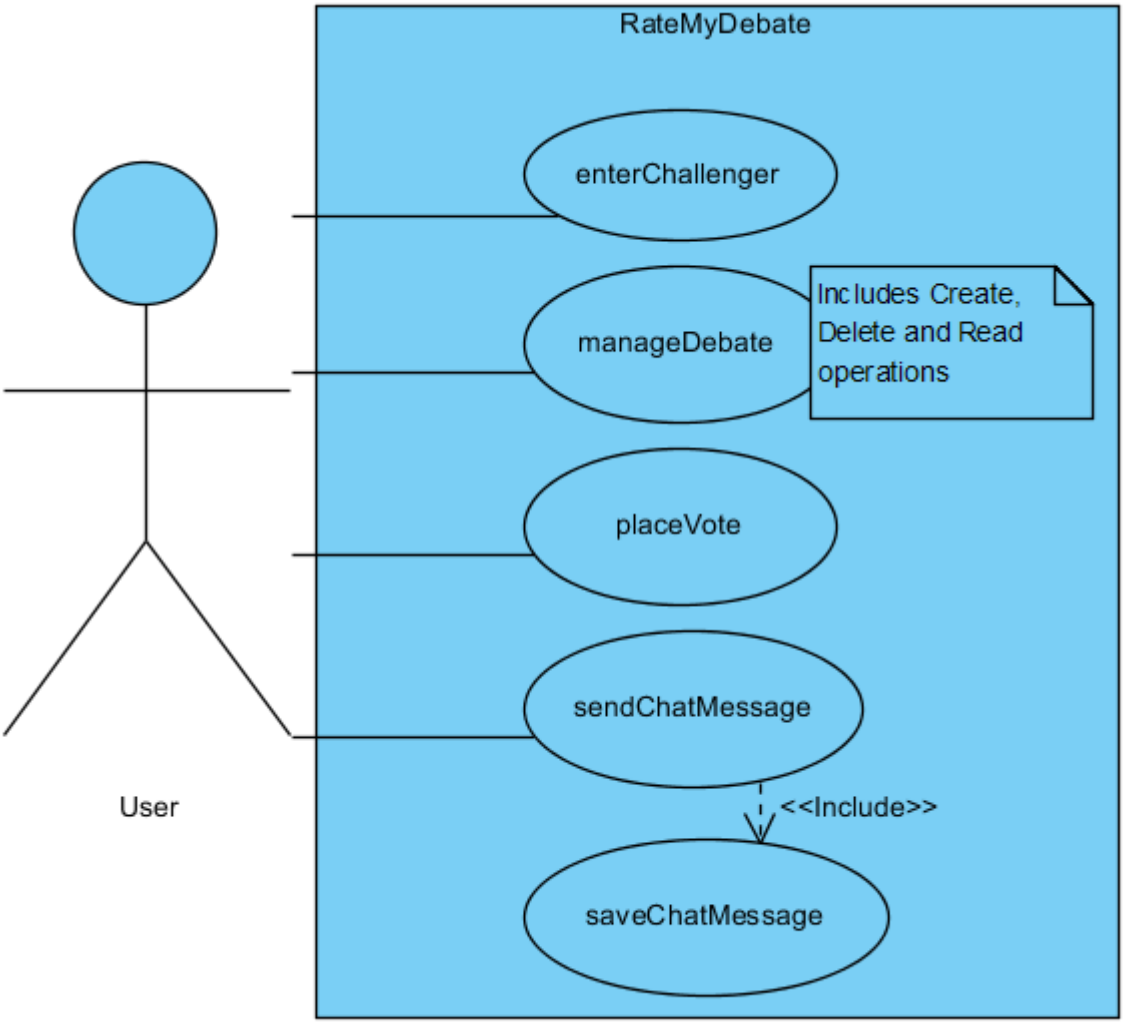
Ingen

Preconditions:

Brugeren er logget ind

Postconditions:

Besked er fjernet fra brugerens inbox.



Figur 4 - Debate Requirements

Use Case: CreateDebate

Primary Actor:

Brugeren

Main flow:

Brugeren udfylder den nødvendige information og starter en debat. Debatten aktiveres og brugeren sendes videre til denne som debattør.

Alternate flow:

Ingen

Preconditions:

Bruger er logget ind

Postconditions:

En debat startes med brugeren som skaber og debattør.

Use Case: DisplayDebate

Primary Actor:

Brugeren

Main flow:

Brugeren trykker på den ønskede debat og sendes videre til at tilskue debatten.

Alternate flow:

Debatten er sat som inaktiv. Arkiveret version vises i stedet.

Preconditions:

Bruger er logget ind

Postconditions:

Brugeren tilskuer aktiv debat.

Use Case: SurrenderDebate

Primary Actor:

Brugeren

Main flow:

Brugeren trykker på en knap, som omgående stopper debatten og tillægger et nederlag til denne bruger.

Alternate flow:

Bruger bedes bekræfte valg, og kan fortryde for at vende tilbage til debat.

Preconditions:

Bruger er logget ind og er en af to debattørere.

Postconditions:

Bruger modtager nederlag. Anden bruger modtager sejr. Debat sat som inaktiv og arkiveret.

Use Case: PlaceVote

Primary Actor:

Brugeren

Main flow:

Tilskueren trykker på en af to knapper for at vælge en debattør at stemme på.

Alternate flow:

Debatten er sat som inaktiv. Arkiveret version vises i stedet.

Preconditions:

Bruger er logget ind

Postconditions:

Brugeren tilskuer aktiv debat.

Use Case: SendChatMessage

Primary Actor:

Brugeren

Main flow:

Debattøren indtaster en besked i et tekstfelt, og trykker på en knap for at broadcaste beskeden til alle brugere forbundet til chatten.

Sub flow:

SaveChatMessage metoden kaldes for at gemme den sendte metode til databasen.

Alternate flow:

Beskeden fanges i et filter for bande- eller trusselsord. Brugeren bedes om at omskrive beskeden.

Preconditions:

Bruger er logget ind og debattør i debatten.

Postconditions:

Chat besked er broadcastet til alle tilskuere.

Use Case: SaveChatMessage

Main flow:

Metoden modtager en chatbesked fra SendChatMessage metoden, og gemmer beskeden til den underliggende database.

Alternate flow:

Ingen

Preconditions:

Chatbesked er godkendt af filte

Postconditions:

Chatbesked er gemt til databasen

b. Interessent identifikation

Primær interessent identifikation

Brugerne - har en stor interesse i at føle sig trygge ved at anvende applikationen samt få den forventede værdi ud af produktet(underholdning, udfordring etc.).

Investorere og reklamerende selskaber - Forventer returnering af investering.

Developers - Enhver person, som arbejder med applikationen, vil naturligvis have højeste interesse i dens profitabilitet.

Sekundær interessent identifikation

Lovgivende institutioner - databeskyttelse af bl.a. passwords, betalinger samt private samtaler.

Politi - i tilfælde af vold- og dødstrusler eller andre ulovligheder, som kan forekomme.

Regering - Beskatning, legitimisering(hvis selskab oprettes for applikationen).

c. Feasibility study

Marked

Der er uden tvivl et marked for applikationer, hvor mennesker kan diskutere og interagere med hinanden. Kombineret med et stødt stigende behov for intensitet og mindre ventetid, har en real-time diskussions applikation en plads på markedet.

Adskillige andre hjemmesider, som tilbyder lignende services, f.eks. Reddit, Facebook og adskillige andre social networking sites, genererer ufatteligt høje aktivitetsstatisikker.

Facebook alene havde 864 millioner unikke dagligt aktive brugere i september 2014[1], og kan på mange måder anses for at være et pseudo-real-time social network.

Reddit havde i samme måned 174 millioner unikke dagligt aktive brugere.[2]

På trods af at ovennævnte eksempler ikke har samme format som deres konkurrenter, så deler de mange træk, og er dermed konkurrenter. Dette projekt ville ikke være en undtagelse.

Dette betyder også i høj grad en delt demografi med ovennævnte konkurrenter. Primært vil målgruppen være bevendte computer-brugere. Evt. ved implementeringen af chat regulationer, som giver de ældre målgrupper bedre mulighed for at konkurrere aktivt i en real-time chat vil applikationen kunne sprede sig til en bredere demografi.

I tilføjelse til dette kan det med tryghed siges at markedet er stødt voksende i takt med at computere er blevet en del af nyere generationers hverdag, og dermed øger størrelsen af den primære målgruppe.

Marketing strategi

Et vigtigt aspekt ved en applikation som denne er, at den baserer sig på og afhænger af mennesker. Ved en fejlagtig lancering vil applikationen uden tvivl føles som en spøgelsesby. Derfor vil det være vigtigt at sikre adskillige brugere ved lancering samt foretage regulationer i applikationens kapabiliteter indtil en større brugerbase er nået, for at undgå spøgelseseffekten.

Strategien vil derfor basere sig meget kraftigt på følgende principper:

[1] http://newsroom.fb.com/company-info/ 02-12-2014
[2] http://www.reddit.com/about/

- Sikre at produktet er meget funktionelt og intuitivt ved lancering

- Gratis brug ved lancering

- Opbyg "hype"/forventning for siden ved lanceringstid gennem reklamering.

Grundet de meget lave finansielle resurser bag projektet er der tre klare muligheder for at reklamere for applikationen. En fundraiser af en art, sandsynligvis online, er en gratis og nem mulighed for at fremskaffe resurser til reklameringen af produktet. Der bør dog fremstå stor ærlighed om projektet's fremtidige marketingsplaner for at beskytte både donorere samt projektets omdømme på længere sigt. Anden del af strategien vil være manuel, gratis reklamering i kredse befærdet af store dele af den primære målgruppe. Dette ville især være online debat forums og social networks. Endda venner og familie; enhver bruger tæller. Denne form for marketing skaber også muligheden for at interagere med brugerbasen og sikre loyalitet i projektet's begyndelse. Reddit tilbyder tilmed et gratis sponsored link, som vil blive vist til hele sidens users, dog meget sjældent.[3] Sidste del er risikabel og bør overvejes kraftigt. Et lån kan foretages til at betale for reklamer. Finanserne bør bruges udelukkende på reklamer, som rammer den primære målgruppe, for at sikre et optimalt return på investeringen.
Facebook tilbyder ad-services enten ved at skabe en side for produktet eller ved at pushe reklamer ud til venner og venners forbindelser på Facebook. Tilmed tilbydes det at sætte et budget for reklameringen, hvilket er meget nyttet for et produkt af denne størrelse. Jeg kunne forestille mig at afsætte en større procentdel af finanserne til Facebook reklamering, således at applikationen bliver et genkendt emne i min større sociale kreds samt skabe profiler på større social medier, inkluderende Facebook, for at skabe et følge.

I samme åndedrag vil jeg nævne Google Adsense. Adsense vil ikke nødvendigvis ramme produktet's primære målgruppe udelukkende, men da der udelukkende betales for clicks på reklamer, kan jeg forvente et godt bytte, såfremt at applikationen forekommer intuitiv for alle målgruppe som måtte finde den.

Til sidst må det dog siges, at den største og bedste reklame, som applikationen kan få, vil være at spredes via "word-of-mouth". Hvis det færdige produkt er tilfredsstillende for brugerne vil det unægteligt sprede sig til brugernes videre sociale kredse, og stige i popularitet som en følge af dette.

[3] http://www.reddit.com/advertising

For at følge op på strategien for reklameringen samt de finansielle aspekter, vil det være nødvendigt at undersøge andre områder, som kan have negativ indflydelse på populariteten. Det vigtigste aspekt vil højst sandsynligt være at begrænse antallet af kategorier til de mest populære tænkelige emner. Dette nævnes, da for mange specifikke kategorier kan skabe en vis form for diaspora. Ved at holde enkelte, men meget brede kategorier åbne, vil aktiviteten på siden føles mere livlig, da det samles for brugerne til enkelte steder. Dette vil forbedre bruger oplevelsen, og fastholde en core brugerbase, hvilket vil hjælpe til at fastholde nytilkommende brugere.

Strategien på længere sigt vil forsøge at gøre applikationen mere kommerciel. Efter en stabil brugerbase er etableret vil det være tid for at sætte enkelte købsmuligheder op, som gør applikationen mere underholdende at bruge. Det skal fortsat være gratis at oprette sig som bruger, skabe og følge med i debatter, da produktet's liv i allerhøjeste grad afhænger af at disse feaures anvendes.

Nye features, som kunne tilføjes til betaling kunne inkludere:

Betaling for promovering af sin egen eller en andens debat.
Premium medlemskab: Ingen reklamer, større venneliste, mulighed for at tracke andre brugere's debatter
Komplimentære premium medlemskaber, som kan købes til andre brugere.
Premium abonnering på specifikke kategorier, som giver notifikationer om meget hotte topics m.m.

Teknologi

Teknologi kravene for et projekt af denne størrelse er meget små.
Da produktet udelukkende anvendes i en browser, er de mest nødvendige stykker teknologi følgende:

Et Integrated Development Environment (Visual Studio - gratis i små selskaber eller som studerende)

Server hosting med understøttelse af .NET frameworket samt SQL databaser.

Microsoft har arbejdet imod at samle og integrere deres produkter med hinanden. Microsoft Azure tilbyder pålidelig hosting samt meget fleksible og billige løsninger for mindre websites.

Tilmed er det muligt at deploye web applikationer direkte fra Visual Studio til Azure, og derfor falder mit valg på Azure for hosting.

Personale

Skulle applikationen blive populær ville ansættelsen af mere personale med en skarpere arbejdsdeling være idéel. Hvis traffiken vokser til at klassificere siden som værende et mindre website kunne et hold bestå af op til følgende tre personer i mere eller mindre fleksible roller:

Developer: Front-end og Back-end. Flekisbel med kundskaber i ASP.NET. Udover JQuery kunne Node.js anvendes for sine realtime kapabiliteter.

Community manager: Håndterer support tickets samt interagerer med communitiet via social networking.

Server manager og database- og data analyst: Sørger for at holde sidens nedetid minimal samt analysere data relateret til folks yndlingsprodukter. Står tilmed for at optimere database design og queries.

Konklusion

Der eksisterer uden tvivl et marked for denne type applikation.
Markedet er tilmed stødt stigende idet at målgruppen hurtigt vokser i takt med at mere computerbevendte generationer kommer til. Potentielle tab og risikoer ved et projekt af denne størrelse er minimale, og størstedelen af funktionaliteten kan opnås med en enkelt person.
Den nødvendige teknologi til at udvikle produktet er gratis, og er tilmed et meget populært produkt, som på længere sigt vil have store fordele i tilfælde af tilføjelser af flere developers.
Publiseringen af produktet er meget fleksibel, og kan tilpasses til et meget lavt budget.
Det samme gælder for reklamering og produktet's publicitet, da der findes mange muligheder for gratis og lavbudget reklamer.
Dermed kan projektet være en absolut minimal finansiel risiko, men have stort potentialle for udbytte.
Grundet applikationen's simple koncept er det tilmed muligt at opretholde og advancere den med et mindre team på 2-3 personer skulle den vokse i popularitet.

Med dette sagt er applikationen afgjort feasible, og afhænger i højeste grad af brugere, som er interesserede i konceptet.

d. SWOT-analyse

Strengths

1. Legale aspekter

Der er mange legale fordele ved et projekt af størrelse, skulle der vælges at registreres et selskab omkring det. Der eksisterer adskillige lovlige forpligtelser og beskatninger for større selskab, men grundet projektet's størrelse, kan det registreres som en enkeltmandsvirksomhed.

En enkeltmandsvirksomhed har store fordele i, at den ikke rammes af ovennævnte beskattelser og forpligtelser, hvilket i høj grad simplificerer kommercialiseringen af produktet, og betyder at der sjældent, hvis nogensinde, vil være behov for legal rådgivning eller komplikationer i dette henseende, så længe at virksomheden drives som enkeltmandvirksomhed.
Dette vil sige at, at så længe jeg sørger for at opfylde mine forpligtelser ikke blot overfor ansatte, men også i mit personlige liv(husleje m.m.), vil det være muligt at have projektet registreret som en virksomhed, og på lovlig vis have en indkomst fra denne. Dette er en følge af, at jeg som ejer hæfter for at opfylde disse forpligtelser med hele min formue, da virksomheden som en følge af denne registrering er identisk med ejeren(mig). Fordelen ved at registrere selskabet på denne måde er, at det giver mulighed for at beskatte og lovliggøre indkomsten under en af tre ordninger, som jeg som ejer selv kan vælge.
Disse tre ordninger inkluderer personskattereglerne, virksomhedsordningen og kapitalafkastordningen. På nuværende tidspunkt ville det være optimalt at registrere virksomheden under personskattereglerne. Dog bør dette valg ændres til en af de to andre i tilfælde af ansættelser, da kapitalafkast- og virksomhedsordningen begger tillader en ophobning en ophobning af indkomst som følge af renteudgifter trukket fra den personlige indkomst imod at betale en senere skat. I tilfælde af ansættelser og skiftet til en af disse ordningen, som begge har mange af de samme fordele, vil der være en foreløbig kapital at arbejde med, som vil give mulighed for at investere i virksomheden. Dette kunne inkludere f.eks. bedre hardware eller stærkere hosting.

For at konkludere, så er der adskillige fordele i tillæg til de legale forpligtelser en virksomhed vil have så længe de økonomiske resurser forvaltes varsomt og fornuftigt.

2. Finansielt risikofrit

Grundet projektet's lave behov for finansielle resurser, er der meget få risici ved at udføre projektet. En enkelt erfaren udvikler vil kunne udvikle produktet billigt, da teknologi behovene er meget lave. De nødvendige værktøjer til at udvikle projektet kan fås gratis, og den største udgift vil være publiceringen af produktet, som primært vil have omkostninger såfremt at projektet opnår success.

Dermed er projektet i realiteten risikofrit i henhold til finanser, og vil ikke være en stor satsning.

3. Unikt koncept

Naturligvis er der intet unikt eller nyt ved real-time chatte eller social networking. Der findes mange produkter på markedet, som er utroligt populære. Det unikke ved konceptet er kombinationen af konkurrence elementet ved stemme-systemet samt intensiteten ved, at det hele foregår i real-time.

Efter mange undersøgelser har jeg endnu ikke fundet noget, som har præcist det samme koncept, og det er en stor styrke, da konkurrence niveauet er nedsat på denne måde. Det vil sige, at der naturligvis stadigvæk findes mange konkurrenter på markedet, da mange af de sociale krav, som siden's brugere ønsker opfyldt, ligesåvel kan blive opfyldt af at snakke med venner på facebook.

Konceptet har en fordel i at appelere til grundlæggende menneskelig natur.
På YouTube, Facebook, online forums, politiske debatter, daglige samtaler osv., findes der konstant mennesker, som deler deres meninger omkring emner de er meget lidenskabelige om. I mange af disse sociale arenaer modtager de ofte ikke den samme lidenskabelige response, hvis noget overhovedet eller samtalen tager en uciviliseret drejning. Tilmed er der en større social fare ved at gå imod den populære mening eller svare en ven imod. Man risikerer at blive druknet i hånende kommentarer eller skade sine forhold. I dette format ved begge deltagere, at de direkte går ind for at argumentere på en civilseret måde overfor en fremmed person, hvor begge er isolerede fra populær mening og sociale stigma mens debatten foregår.
Af denne grund er stemmerne skjulte indtil debatten er overstået. I realiteten kunne stemme-aspektet undværes, og funktionaliteten ville være den samme. Stemmerne er en anden styrke, som har til formål at appellere til en anden del af menneskelig natur, nemlig den dybtliggende lyst mange mennesker har til at konkurrere og opnå noget.

En stor del af målgruppen har en eller flere interesser, som de går højt op i og som involverer konkurrenceaspeketet. Her tales der om fritidshobbier som fodbold eller videospil til studier og ønsket om at klare sig godt i forhold til sine medstuderende eller arbejdet med presset for at klare sig godt for at undgå fyring eller få den forfremmelse.

Til slut om konceptet følger det i høj grad med den udvikling i moderne produkter, som sigter imod, at alting skal foregå hurtigt, være spændende og have så lidt ventetid som muligt. Moderne teknologi har nået et punkt, hvor sekunder kan føles som en evighed, og fokus hurtigt kan flyttes til noget andet, som hurtigt kan opfylde vores voksende krav om omgående tilfredsstillelse af vores underholdnings- og spændingsbehov. Den største konkurrent i dette henseende vil være Omegle.com, som tilbyder valg af topic til debat. Projektet opfylder alle disse krav baseret på, at debatterne foregår i real-time med en tidsbegrænsning. Dette betyder, at begge parter ved, at der vil være hurtige svar og forventes det samme for at få delt sine synspunkter så hurtigt som muligt før tiden løber ud. Dette tjener til at give et intensivt, spændende og underholdende debatmiljø, som tilfredsstiller behovet for morskab samtidigt med at begge debattører kommer ud, forhåbenligt, mere oplyste end før.

Weaknesses

1. Store krav til brugere

For at kunne holde en civiliseret, længere debat kræves det, at begge debattørere har en større forståelse for emnet, som debateres, samt modenheden til at holde sig høflige og fokuserede på debatten skulle den blive meget intens. Derfor er projektet's største svaghed, at det ikke kan vides om der sættes for store krav til brugerne, eller om det på succesrig vis formår at tiltrække brugere, som er værd at tilskue.

2. Afhængigt af en større brugerbase

Et stort problem for en applikation, som udelukkende baserer sig på brugeres villighed til at deltage er, at hvis der ikke er nok brugere til at deltage, så mister applikationen størstedelen af sin funktionalitet. Dermed afhænger projektet som helhed af at kunne samle en større brugerbase uden forud at have etableret sig selv, og derfor vil meget reklame være nødvendig til at kickstarte projektet.

Opportunities

1. Udvidelse af features

Der eksisterer altid mange muligheder for koncept orienterede social networking sites for at udvide projektet med nye features på et senere tidspunkt. Politik, f.eks., er en rig resurse for meninger. Med det sagt kan der etableres nye features, som f.eks. polling, uforpligtende spørgsmål som inviterer kommentarer fra alle bruger osv.
På denne måde kan sitet etablere sig som et større samfund, der på alle måde omhandler at debattere og lære nye holdninger/perspektiver.

2. Android/IOS App

En app til Android eller IOS devices ville være en stor mulighed for at holde et livligt samfund på siden. Det ville muligvis ikke være den mest optimale løsning til at deltage i debatter, men mange mennesker har en eller anden form for ventetid i løbet af deres dagligdag, som betyder at RateMyDebate kan etablere sig som et underholdningsmedium i de ventetider. F.eks. på bussen eller i ventestuen hos lægen. Hvis ovennævnte features i del 1 af denne sektion blev implementeres, kunne appen anvendes til at still uforpligtende spørgsmål, hvis brugeren pludselig står overfor et emne, som de ønsker input på i deres hverdag.

Threats

Svært marked at komme ind på

Som tidligere nævnt, er mange dele af konceptet ikke unikt eller nyt på nogen som helst måde. Mange selskaber, som tilbyder disse services, dog i et andet format, er allerede etablerede som giganter på markedet, og breder sig konstant til nye områder.
Tilmed vil disse giganter integrere deres produkter med hinanden, for at gøre det svært for brugere at forlade disse services grundet deres belejlighed.
Selskaber som Facebook er nået et punkt, hvor nye applikationer frivilligt arbejder imod at integrere funktionalitet i deres applikationer, således at det er muligt at logge ind med brugeres Facebook-login i stedet for at skulle registrere en ny bruger. Markedet er på mange måder enormt serviceorienteret for brugerne, og det er ubelejligt for brugere at skulle registrere en ny bruger på hver side de besøger. Det er for meget arbejde, og nemmere at holde sig til de systemer, man allerede er integreret i.

Derfor eksisterer der på denne måde en mentalitet blandt nykommere om "if you can't beat them, join them".
Selskaber som Google med købekraft til at opkøbe f.eks., YouTube, forsøger at komme ind på social networking markedet ved at integrere samtlige af deres services i deres social networking service, Google Plus, således at YouTube kontoer, Google docs og anden cloud storage etc., alle omhandler netop at have et Google Plus login. På trods af giganten Google's ekstreme resurser og magt på det digitale marked ansås Google Plus for en fiasko i lang tid, og endda stadigvæk af mange på trods af at Google i oktober 2013 kunne tælle 540 million aktive brugere.[4]
Derfor er det et benhårdt marked, især hvis ens produkt mangler en større variation af features, og ikke gør det belejligt nok for brugere at give ens produkt et forsøg.

[4] http://thenextweb.com/google/2013/10/29/two-years-later-google-growing-540m-active-users-worldwide-1-5b-photos-uploaded/

4. Softwaredesign

Dette afsnit sigter mod at giver en nærmere forståelse af...

- hvordan applikationen er bygget op.
- hvordan ansvaret for forskellige funktionaliteter er fordelt i applikationen.
- hvilke software patterns, som er anvendt og hvordan.

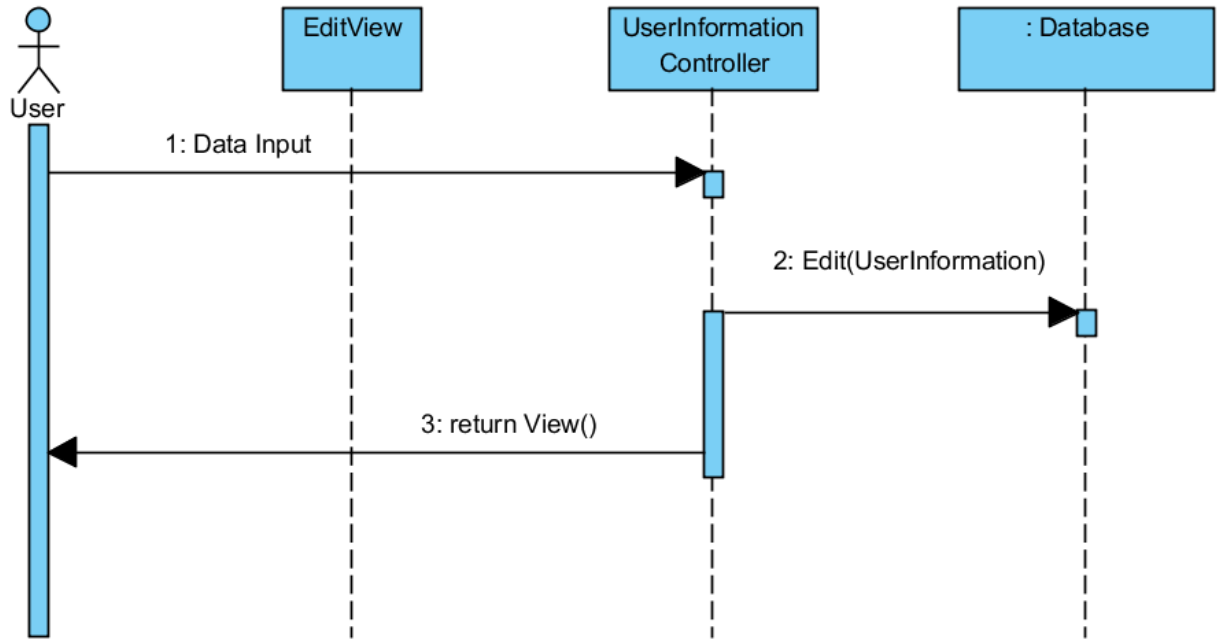
a. Model-View-Controller(MVC)

Dette er hovedsagligt en lægmand's forklaring. Hvis du er indviet i, hvordan MVC fungerer kan dette afsnit springes over.

Applikationen's struktur er baseret omkring det meget populære arkitetur pattern, MVC.
MVC står i store træk for hvor tilstande, visuelle repræsentatior samt logik skal fordeles.

Models er klasser, som indeholder information om objekter's tilstande.
Views er den visuelle repræsentation samt brugergrænseflade, som tager imod input fra brugeren.
Controlleren er logik delen, som brugerinput sendes videre til for eksempel for at ændre i en model's tilstand.

Et simpelt eksempel kunne være applikationen's metode til at redigere en bruger's information.
Eksemplet er her illustreret i et Sequence Diagram:



Figur 5 - Sequence Diagram for Edit

Brugeren indtaster data input i viewet, som sendes videre til controlleren, som udfører "Edit" metoden, hvor modellen parses videre til databasen.

Herefter returneres brugeren til Index viewet.

En kode beskrivelse af handlingen følger herunder.

Her ses model-classen for vores bruger information. Det er intet mere end en klasse med variabler, som anvender DataAnnotations til at sætte begrænsninger for hvilke værdier, som må indtastes:

```
  public class UserInformation
    {
        [Key]
        public int userInformationId { get; set; }

        [Required]
        [Display(Name = "First name")]
        public String fName { get; set; }

        [Display(Name = "Last name")]
        [Required]
        public String lName { get; set; }

        [Required]
        [Display(Name = "Username")]
        [Remote("ValidateNickName", "UserInformation", HttpMethod = "POST", ErrorMessage =
"User name already exists. Please enter a different user name.")]
        public String nickName { get; set; }

        [Display(Name = "Age")]
        [Required]
        public int age { get; set; }

        [Display(Name = "Autobiography")]
        [Required]
        public String autobiography { get; set; }

        [Display(Name = "Email")]
        [Required]
        public String Email { get; set; }

        [ForeignKey("accountId")]
        public int userId { get; set; }
        public UserModel accountId { get; set; }
    }
```

Her ses body-delen af koden, som dækker redigeringen for Edit viewet:

```
@model RateMyDebate.Models.UserInformation


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.HiddenFor(model => model.userId)
    @Html.HiddenFor(model => model.nickName)

    <div class="form-horizontal">
        <h4>UserInformation</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.userInformationId)

        <div class="form-group">
            @Html.LabelFor(model => model.fName, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.fName)
                @Html.ValidationMessageFor(model => model.fName)
            </div>
        </div>

..... adskillige flere form-groups som ovenstående findes her. Alle er relaterede til model-
classen's non-key variabler, som kan ses i UserInformation model-class koden.

       <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>

}
```

Viewet her benytter sig af Razor syntax (@{ } blokkene) samt HTML.Helpers til at binde information, som skal sendes videre til den relaterede controller metode, når brugeren trykker på "Save"-knappen.

`@model RateMyDebate.Models.UserInformation` angiver hvilken model, som anvendes i viewet og som de nedenstående form-grupper referer til i deres lambda expresssions (model => mode.userInformationId for eksempel).

`@Html.EditorFor` opretter en tekstbox, som brugeren kan skrive en værdi ind i, som vil blive sendt til den bundne variabel i modellen.

`@Html.EditorFor(model => model.fName)` vil for eksempel tage en værdi, som vil sættes til at være fornavn, når controller metoden kaldes og foretager ændringerne i modellen.

Dette er koden for controller metoden Edit, som kaldes fra af brugeren via Edit viewet:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit( UserInformation userinformation)
{

    if (ModelState.IsValid)
    {
        db.Entry(userinformation).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(userinformation);
}
```

Som kan ses her, modtager Edit metoden alle variabler fra Viewet samlet i et UserInformation objekt.

If-sætningen checker hvorvidt objektet's informationer er gyldige.

Hvis samtlige værdier er indtastede gyldigt i overensstemmelse med model-classens DataAnnotations vil controlleren kunne udføre logikken.
F.eks. er det nødvendigt at have et unikt Nickname, da Nickname har følgende DataAnnotation:

```
[Remote("ValidateNickName", "UserInformation", HttpMethod = "POST", ErrorMessage = "User name already exists. Please enter a different user name.")]
        public String nickName { get; set; }
```

Hvis brugeren forsøger at sætte sit Nickname til et allerede eksisterende i databasen, vil brugeren modtage en fejlbesked på grund af DataAnnotationen. Denne DataAnnotation anvendes dog kun, når brugeren opretter sin konto, da det ikke er muligt at redigere sit Nickname senere hen.

Hvis checket giver true, vil if-sætningen udføres.

`db.Entry(userinformation).State = EntityState.Modified;` angiver, at der er foretaget ændringer i databasen.
db refererer til vores Database Context variabel øverst i Controller classen:

```
private RateMyDebateContext db = new RateMyDebateContext();
```

Context classen er en del af Entity Framework, og fungerer som et lag imellem applikationen og databasen, som håndterer en stor del af logikken nødvendig, for at kunne udføre, f.eks. CRUD(Create. Read. Update, Delete) operationer på databasen. Context classen kan tracke det rigtige database entry, og tilstanden sættes som modificeret, hvorefter linjerne

```
db.SaveChanges();
return RedirectToAction("Index");
```

kalder ContextClassens metode til at gemme ændringerne i den underliggende database. Dernæst returneres vi til "Index" viewet, og MVC handlingen er overstået.

b. Model designs

Et vigtigt koncept inden for MVC, er hvorledes databasen bygges op. I modsætning til de vanlige SQL queries vi ofte ser, så foretages størstedelen af database håndteringen ved hjælp af Entity Framework Code First og mit anvendte IDE, Microsoft Visual Studio.

Code First baserer sig på at database tabellerne kodes først som klasser, og senere migrere og oversætte dem til database tabeller.

Min del af arbejdet er at skrive klasserne, som også fungerer som model delen af MVC, oprette dem i DatabaseContexten og udføre kommandoerne, som migrerer og opretter eller opdaterer tablesne. Her vises en forklaring på en typisk database opdatering for at give indsigt i, hvordan EF Code First og Visual Studio fungere sammen.
Forklaringen har til formål, ikke at demonstrere værktøjerne, men sammenhængen mellem databasen og MVC samt hvilken rolle Entity Frameworket spiller i denne sammenhæng.

Dette er Category modellen, som også agerer som klassen for samme data table.

```
public class Category
{
    [Key]
    public int CategoryId { get; set; }

    [Display(Name = "Category")]
    public String CategoryName { get; set; }
}
```

De er altså fuldstændigt bundne og kompatible med hinande på alle tidspunkter, så længe data tablen er opdateret med model designet. Af samme grund er det muligt at anvende model objekterne, når jeg foretager og gemmer ændringer til databasen som det sås i "MVC" sektionen.

Da jeg ønsker at sætte en maximum String længde på CategoryName, så reglen også gælder for databasen, kan jeg anvende DataAnnotations til at sætte begrænsningen. Tilmed kan jeg sætte en fejlmeddelelse, som kan sendes, hvis der forsøges at oprette en kategori med for mange karakterer.

Nu ser CategoryName således ud:

```
[Display(Name = "Category")]
[StringLength(30, ErrorMessage = "Not allowed to use more than 30 characters in a
                                                                    category name")]
public String CategoryName { get; set; }
```

For at sikre at Category modellen oprettes som en data table i databasen, er det nødvendigt at oprette den som et "DbSet" i mine DatabaseContext fil.
Herunder ses den fulde RateMyDebateContext:

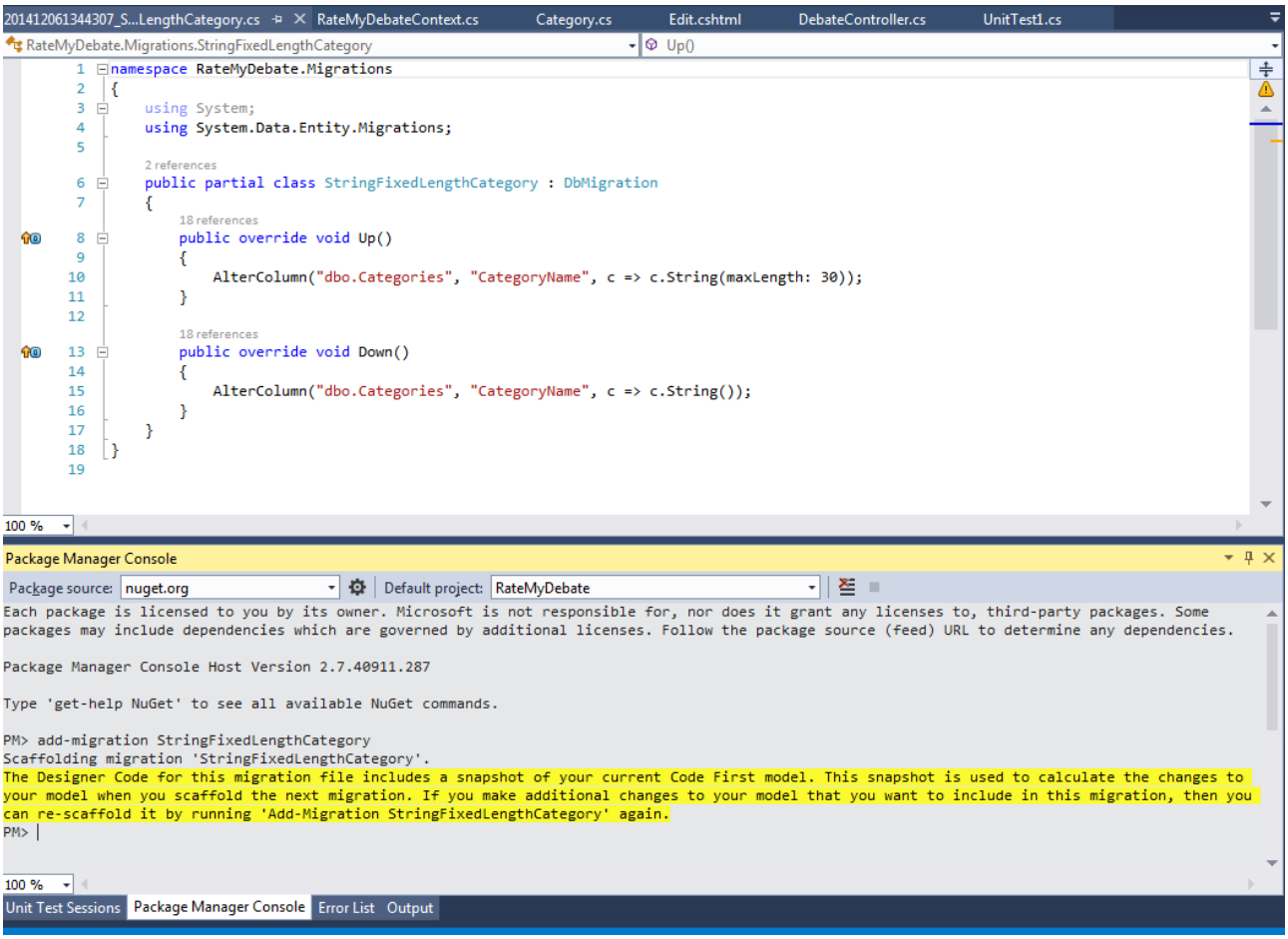
```
public DbSet<UserModel> UserModel { get; set; }
public DbSet<UserInformation> UserInformation { get; set; }

public DbSet<Inbox> Inbox { get; set; }

public DbSet<Message> Message { get; set; }
public DbSet<Debate> Debate { get; set; }
public DbSet<Category> Categories { get; set; }
public DbSet<Vote> Votes { get; set; }
public DbSet<MessageMap> MessageMaps { get; set; }

public DbSet<Result> Results { get; set; }
public DbSet<Friendship> Friendships { get; set; }
```

Nu er det muligt at fortsætte til at lave migrationen, som vil opdateres til databasen.
I Visual Studio kan jeg anvende "Package Manager Console"-værktøjet til at auto-generere min migration. I billedet nedenunder ses resultatet af "add-migration" kommandoen, som jeg har givet navnet StringFixedLengthCategory.

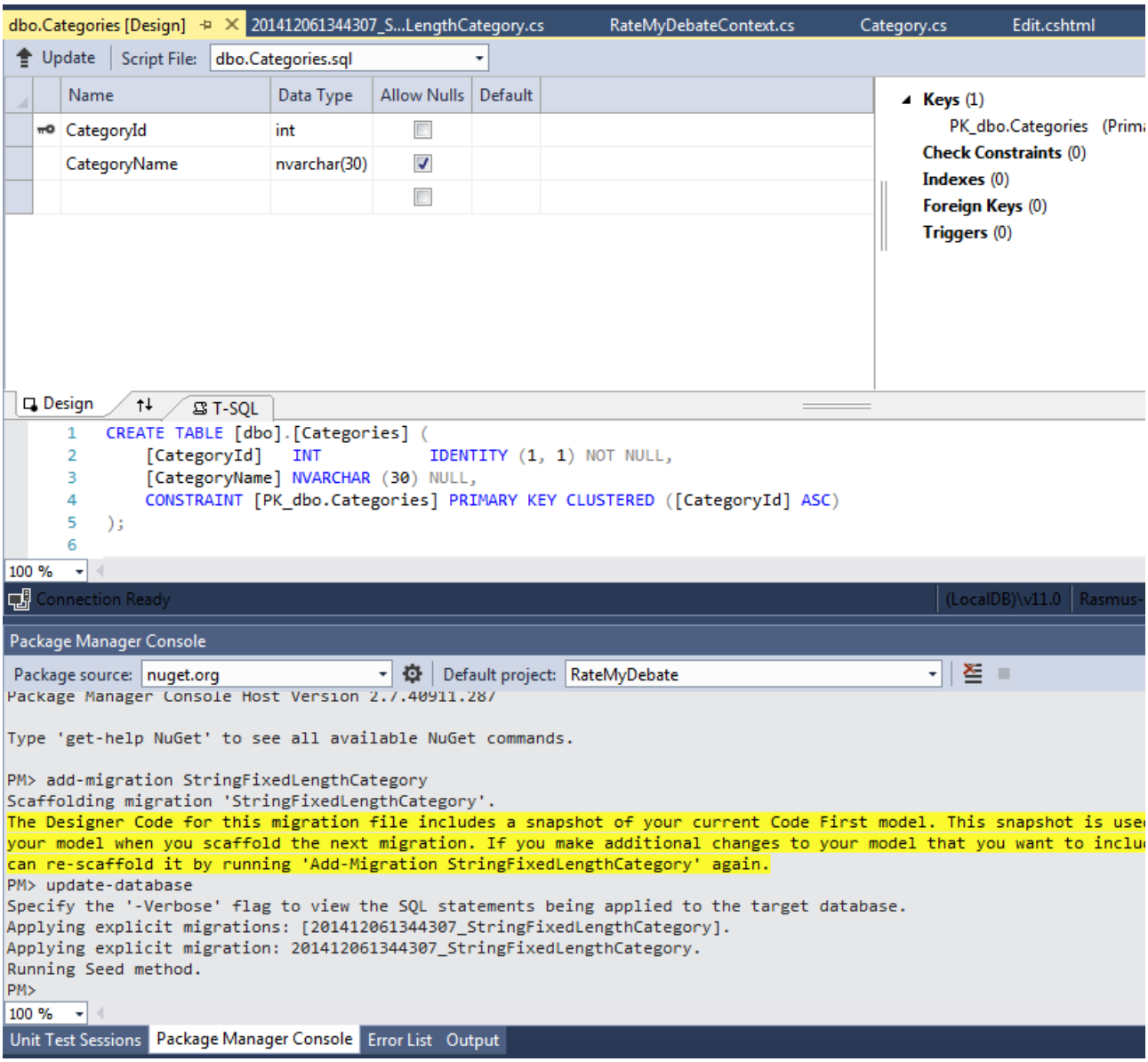


Figur 6 - Package Manager Console

Koden i midten af billedet er den autogenererede DbMigration class.
I Up() metoden kan det ses at den ønskede ændring i data table kolonnen er klar:

```
AlterColumn("dbo.Categories", "CategoryName", c => c.String(maxLength: 30));
```

For at fuldføre denne ændring til databasen, behøver jeg nu at fuldføre update-database kommandoen i Packet Manager Console.



Figur 7 - Database Constraints

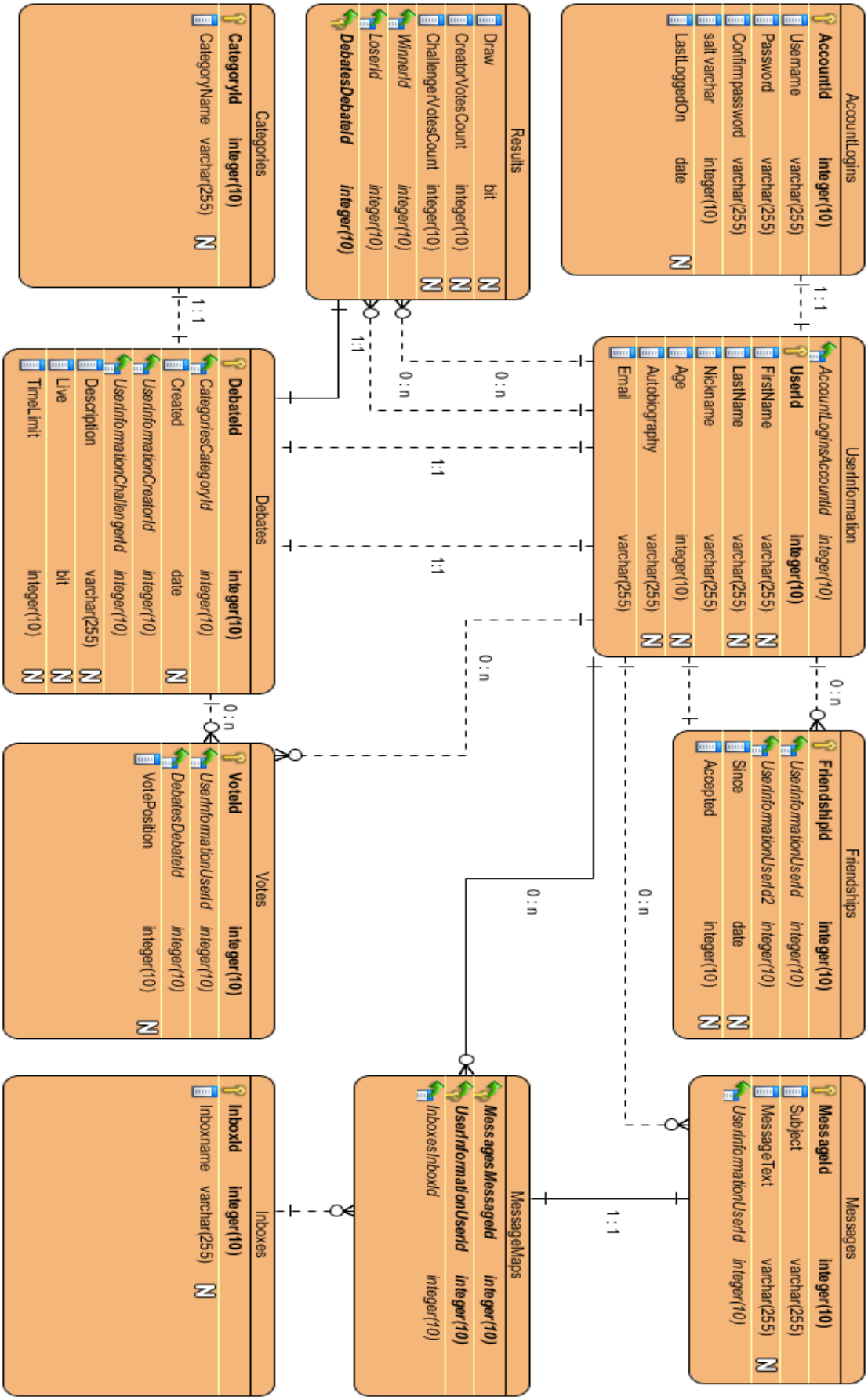
Som det kan ses er CategoryName nu datatype nvarchar(30), så database restriktionen er succesfuldt opdateret.

c. Database design

Oversigt

Databasen indeholder på nuværende tidspunkt 10 data tables.
Den primært anvendte data table er UserInformation tablen, som indeholder al information, der kan vises om brugeren. Den primary key(PK) i denne table anvendes til at referere til alle tables, som har noget med brugeren at gøre. Stemmer som brugeren har afgivet, debatter den har vundet og beskeder, som den har sendt eller modtaget.

I dette afsnit om database designet vil jeg komme ind på bestemte valg for database designet samt forslag til, hvordan det kan forbedres.


AccountLogins
AccountId integer(10)
Username varchar(255)
Password varchar(255)
Confirmpassword varchar(255)
salt varchar integer(10)
LastLoggedOn date N
UserInformation
AccountLoginsAccountId integer(10)
UserId integer(10)
FirstName varchar(255) N
LastName varchar(255) N
Nickname varchar(255)
Age integer(10) N
Autobiography varchar(255) N
Email varchar(255)
Results
Draw bit N
CreatorVotesCount integer(10) N
ChallengerVotesCount integer(10) N
WinnerId integer(10)
LoserId integer(10)
DebatesDebateId integer(10)
Categories
CategoryId integer(10)
CategoryName varchar(255) N
Debates
DebateId integer(10)
CategoriesCategoryId integer(10)
Created date N
UserInformationCreatorId integer(10)
UserInformationChallengerId integer(10)
Description varchar(255) N
Live bit N
TimeLimit integer(10) N
Votes
VoteId integer(10)
UserInformationUserId integer(10)
DebatesDebateId integer(10)
VotePosition integer(10) N
Friendships
FriendshipId integer(10)
UserInformationUserId integer(10)
UserInformationUserId2 integer(10)
Since date N
Accepted integer(10) N
Messages
MessageId integer(10)
Subject varchar(255)
MessageText varchar(255) N
UserInformationUserId integer(10)
MessageMaps
MessagesMessageId integer(10)
UserInformationUserId integer(10)
InboxesInboxId integer(10)
Inboxes
InboxId integer(10)
Inboxname varchar(255) N
1:1
0:n

Figur 8 - Database Diagram

Adskillelse af AccountLogin og UserInformation

Der kan argumenteres for(ikke nødvendigvis gode argumenter), at AccountLogin og UserInformation hører under en samlet table kaldet Account, da de begge indeholder information relevant til brugeren's konto og har et 1:1 forhold, altså et login kan udelukkende anvendes til at skaffe adgang til denne brugerinformation.

Jeg har på følgende grundlag valgt at separere de to:

1) Klarere overblik
2) Hurtigere queries
3) Sikkerhed

Det klarere overblik kommer naturligvis af, at adskillelsen giver et bedre overblik over, hvilken table tjener hvilket formål samtidigt med, at de er normaliserede. En forbedring, som ville gøre det til et endnu cleanere design ville være, at anvende AccountLogin's PK som PK i UserInformation, da de har et 1:1 forhold. Databasen ville bevare samme funktionalitet og være cleanere i design, hvis denne fejl var set i tid.

De hurtigere queries kommer fra, at da programmet konstant anvender UserInformation tablen, vil det være meget lettere queries. Da Login tablen udelukkende anvendes ved Login/Logoff vil det være spild at have den samlet med UserInformation tablen, da queriesne unødvendigt ville hente samtlig information fra AccountLogin tablen. Det eneste dette ville kontribuere til ville være øget traffik, langsommere queries og det vigtigeste punkt: sikkerhed.

Da UserInformation tablen/modellen konstant er i brug i applikationen, blandt andet i session state variabler, poserer det en stor sikkerhedsrisiko, at have brugerens sensitive login informationer liggende i en variabel på denne måde.

Den mest sensitive information, som kunne opsnappes om en bruger, hvis adgang fandtes til Userinformation tablen ville være brugeren's e-mail. At placere e-mail'en i denne table anser jeg som en fejl, da e-mailen spiller en større rolle i AccountLogin tablen end UserInformation tablen, især i tilfælde af et nødvendigt account recovery, hvis password og brugernavn glemmes.

Forslag til bedre arkivering og mere effektive debat-queries

På nuværende tidspunkt baserer arkiveringen af afsluttede debatter sig på følgende variabel, som er en del af Debate-modellen/data tablen:

```
public Boolean Live { get; set; }
```

Når en debat instantieres markeres den som aktiv ved at Live ændres til "true".
Når en debat rammer sit "TimeLimit", køres en række metoder hvoraf én ændrer Live til "false".
Live variablen anvendes bl.a. til queries, som skal vise nuværende aktive debatter. Er en debat afsluttet, vil variablen forhindre queriet i at hente table entriet, som kan ses i Index metoden af DebateController, som henter alle aktiver debatter:

```
        public ActionResult Index(String category, String creator, String challenger)
        {
            List<Debate> myList = db.Debate.ToList().Where(x =>
x.Live.Equals(true)).ToList();
DebateUser VM = new DebateUser();

            var CategoryQry = from d in db.Categories
                              orderby d.CategoryName
                              select d.CategoryName;

            VM.User = db.UserInformation.ToList();
            VM.Debate = db.Debate.ToList();
            VM.Categories = db.Categories.ToList();


            if (!String.IsNullOrEmpty(category))
            {
                myList = myList.Where(x =>
x.CategoryId.CategoryName.Contains(category)).ToList();
            }

            if (!String.IsNullOrEmpty(creator))
            {
                myList = myList.Where(x => x.CreatorId.nickName.Contains(creator)).ToList();
            }

            if (!String.IsNullOrEmpty(challenger))
            {
                myList = myList.ToList().Where(x =>
x.ChallengerId.nickName.Contains(challenger)).ToList();
            }

            VM.Debate = myList;

            ViewBag.category = new SelectList(CategoryQry);


            return View(VM);
        }
```

Metoden tager mod tre parametre, som anvendes i et søgefilter, hvis brugeren opfylder disse. Først instantieres en List, som tager imod Debate objekter. Queriet, som fylder listen op henter udelukkende Live entries.

Efterfølgende hentes andre data, som er nødvendige for at præsentere siden i en ViewModel(DebateUser VM), og en række if-sætninger filtrer søgeresultaterne. Den endelige debat-liste tilføjes til ViewModellen's debat-liste, og ViewModellen sendes videre til viewet.

Min anvendte ViewModel class ser således ud:

```
public class DebateUser
{

    public List<UserInformation> User { get; set; }
    public List<Debate> Debate { get; set; }

    public List<Category> Categories { get; set; }


}
```

ViewModellen sørger for, at jeg kan sende al nødvendig information videre til viewet. Da Debates ikke selv kan query til deres foreign keys, er det nødvendigt at parse informationen med i en ViewModel, som også tillader, at jeg kan kontrollere, at det kun er nødvendig data som sendes igennem

Nu da metoden er forklaret, vil jeg udlægge hvilke effektiviseringen, som bør udføres for at forbedre denne process.

På længere sigt er det et problem, at samtlige debatter, som nogensinde har været i systemet vil blive itereret over, når queriet udføres. Selv med en Live variabel til at indikere inaktivering og arkivering, ville det være bedre at have en data tabel, som er tilegnet arkiverede debatter.
Dvs., når en debat ender kan den omgående flyttes til denne tabel. Dermed vil der udelukkende findes aktive debatter, som bliver itereret over, når queriet kører, og det kan hente samtlige debatter uden at skulle anvende filtre. Dette kan på længere sigt have stor effekt.

Det samme gælder for denne linje kode:

```
VM.User = db.UserInformation.ToList();
```

Koden henter på meget ineffektiv vis samtlige brugerinformationer ind for at sikre, at enhver bruger, som det vil være nødvendig at præsentere i Index viewet(enten som skaber af en debat eller udfordrer) ikke er manglende fra liste.

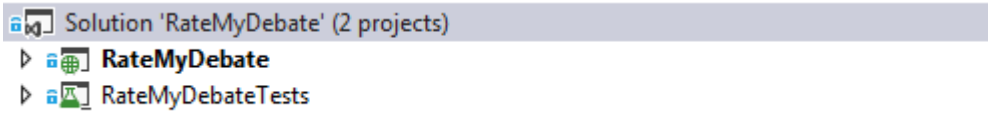
Dette kan undgås ved at omskrive koden således, at den færdigt filtrerede debat liste itereres over, og brugerinformation for hvert debat entry i forhold til creator og challenger hentes:

```
for(alle debatter i listen) {
  Hent Creator- og ChallengerUserInformation i debat...
  Tilføj disse til ViewModel's UserInformation liste.
}
```

På denne måde gøres queriet mere effektivt, mere clean og delvist mere sikkert, da samtlige UserInformations nu ikke hentes fra databasen ved hver indexering af aktive debatter.

d. Testing, repositories og mocking

I min Visual Studio "Solution", som er samlingen af flere projekter, har jeg oprettet et projekt dedikeret til både unit og integration testning.



I unit testningen anvender jeg Moq til at skabe mocks for at undgå testning på min database's data, og på grundigere vis verificere udførelsen af CRUD operationer.

I integration testningen har jeg blandt andet udført tests på mere komplexe metoder, som invokeres i programmet ved timed events og anvender flere komponenter af applikationen.

Disse tests har alle til formål at hjælpe til fejlfindingen i individuelle komponenter og større komplekse sammenhænge på kort og lang sigt. Fremtidige ændringer i applikationen kan have indflydelse på en eller flere komponenter. At have testsne skrevet og klare til at debugge betyder, at det kan tage få minutter at opdage fejl, som ellers kunne have været svære at finde ved at teste ved runtime.
Dermed medvirker tests også til at øge hastigheden for softwarekonstruktionen, da testning ved runtime er meget langsommere, da der i mange af situationerne skal bruges længere tid på at sætte

test området klar til testing, f.eks. ved at logge ind eller timede events.
Ved brug af unit tests er det muligt at teste samtlige komponenter i en test class, og de 20 sekunder det tager er et glimrende vindue til at starte en ny kop kaffe.

Repository og dependency injection

Repository patternet er et pattern, som separerer business logic og CRUD operationer ved at flytte CRUD operationer fra Controller classes over i et seperat repository.

En af begrundelserne for brugen af patternet er, at det giver et meget cleanere og simplere design ved flytte koden over i en anden class, og stadigvæk bevare den samme funktionalitet.

Tilmed følger patternet på mange måder Data Access Object patternet idet at et repository netop virker som et data access objekt, da dets funktion netop er at udføre CRUD operationer.
Samtidigt betyder det også, at fleksibiliteten forhøjes signifikant, da et repository kan anvendes i andre controllers. Dermed forbliver data adgangen konsistent og solid i modsætning til at genskrive samtlige CRUD operationer i andre controllers, som anvender disse objekter.

Interfacet for debat repositoriet ser således ud:

```
public interface IDebateRepository : IDisposable
{
    IQueryable<Debate> GetDebates { get; }
    Debate FindDebate(int debateId);
    void AddDebate(Debate debate);
    void UpdateDebate(Debate debate);
    void DeleteDebate(int debateId);
    void Save();
}
```

IDebateRepository implementeres i DebateRepository classen, hvor logikken for metoderne skrives. Herefter anvendes repositoriet i en controller ved dependency injection:

```
public class DebateController : Controller
{
    private RateMyDebateContext db = new RateMyDebateContext();
    private readonly IDebateRepository _debateRepository;


    public DebateController(IDebateRepository debateRepository)
```

```
        {
            _debateRepository = debateRepository;
        }

}
```

Det vil sige, at et objekt "injectes" i et andet objekt, som er "dependent" på dette.
For at fuldføre dependency injectionen skal vores IDebateRepository bindes til DebateRepository. Til dette formål anvendes den importerede Ninject package. I applikationen laves en ControllerFactory class, som foretager bindingen af repository interface og class samt en linje til at starte bindingen ved Application_start i global.asax

Controller factory:

```
   public class ControllerFactory : DefaultControllerFactory
    {

        private IKernel ninjectKernel;
        public ControllerFactory()
        {
            ninjectKernel = new StandardKernel();
            AddBindings();
        }
        protected override IController GetControllerInstance(RequestContext requestContext,
Type controllerType)
        {
            return controllerType == null ? null :
(IController)ninjectKernel.Get(controllerType);
        }
        private void AddBindings()
        {
            ninjectKernel.Bind<IDebateRepository>().To<DebateRepository>();
        }
    }
```

I classen's constructor initialiseres ninjectKernel og AddBindings metoden kaldes.
I AddBindings metoden udføres selve bindingen af ninjectKernel.

GetControllerInstance metoden anvendes af MVC's routing, når den har fundet af, hvilken type Controller som anvendes. Dermed er den ansvarlig for at returnere en instans af controlleren, som herefter anvendes.

Global.asax:

```
ControllerBuilder.Current.SetControllerFactory(new ControllerFactory());
```

Denne kode køres ved applikationen's start, og sætter den nuværende controllerbuilder til at anvende en ny instans af mit ControllerFactory som sit controller factory.

Unit testing

Jeg har anvendt unit testing med mocks primært, da mocking tillader mig at teste mine metoder uden at ændre på data i applikationen's database.

Her ses en unit test af Create metoden:

```
[TestMethod]
public void TestMethod1()
{
    var mockRepo = new Mock<IDebateRepository>();

    DebateController controller = new DebateController(mockRepo.Object);

    var debate = new Debate
    {
        DebateId = 0, CategoryIdId = 4, CreatorIdId = 3, Live = true,
        Subject = "Emne", Description = "Beskrivelse",
        DateTime = DateTime.Now, TimeLimit = 50
    };

    var result = controller.Create(debate) as RedirectResult;

    mockRepo.Verify(d => d.AddDebate(debate), Times.Once);
    mockRepo.Verify(d => d.Save(), Times.Once);

    Assert.IsNotNull(result);
}
```

Som kan ses her, anvendes et Mock af IDebateRepository. Dette repository indsættes i en ny instans af DebateControlleren. Ved at anvende et mock repository tilgår jeg altså ikke applikationen's database.

Herefter instantieres et ny debat objekt til testen, og Create metoden kaldes, mens en variabel venter et RedirectResult, som metoden vil returnere, hvis den er succesfuld.

Herefter anvendes Moq's Verify funktioner til at verificere at debat objektet gemmes til databasen, men ikke bliver kaldt mere end én gang, og Assert.IsNotNull(result) checker, at vi på succesfuld vis kaldte Create metoden.

Integration testing

I integration tests tester jeg den egentlige database for applikationen. Disse tests kan altså have indflydelse på data i databasen, og derfor er det vigtigt at være varsom i testningen ved Create, Update og Delete operationer. Disse tests har være essentielle i sikringen af testning af mere komplekse metoder, som foretager adskillige database kald, blandt andet ProcessDebateResult metoden, som er en metode, der kaldes efter en fuldendt debat, og er meget besværlig at teste ved runtime.

For at give et simplere eksempel på en integration test ses her en test af DebateControlleren's Index metode:

```
// Testing to make sure the debate list is NOT empty,
// when parsing through a search keyword that we know to exist in the database.
[TestMethod]
public void DebateIndexTest2()
{
    DebateController controller = new DebateController(debateRepository);

    var result = controller.Index("Atheism", null, null) as ViewResult;

    var model = result.ViewData.Model as DebateUser;

    var list = model.Debate;

    Assert.AreNotEqual(0, list.Count);
}
```

Metoden har til formål at teste søgefilteret på Index viewet, som blandt andet kan filtrere debatter baseret på kategori, hvilket denne tests formål at undersøge funktionaliteten af.
Søgefilter værdierne parses ind i metoden's parametre således:

```
public ActionResult Index(String category, String creator, String challenger)
```

I controlleren's Index metode findes disse linjer kode, som behandler listen af debatter baseret på den parsede Category parameter:

```
if (!String.IsNullOrEmpty(category))
{
myList = myList.Where(x => x.CategoryId.CategoryName.Equals(category)).ToList();
}
```

myList er den fulde liste af aktive debatter. Søgefilter metoderne vil reducere denne liste baseret på de parsede værdier. Som det kan ses i denne lambda expression reduceres myList til udelukkende at indeholde debatter, som har det parsede category parameter set i metoden's parameter definition.

Når index metoden har returneret result variablen hentes ViewModellen fra ViewDate'en i result variablen således:

```
var model = result.ViewData.Model as DebateUser;

var list = model.Debate;

Assert.AreNotEqual(0, list.Count);
```

Debat listen overføres herefter til en liste variabel.
En assertion af, at debat listen ikke er tom, når vi ved, at der eksisterer debatter, som anvender "Atheism" kategorien i databasen fuldføres.

Ved at anvende integration tests er det dermed let og hurtigt at undersøge funktionaliteten af disse komponenter og sikre, at database operationer foretages korrekt.

e. Gennemgang af debat process med kodeforklaring

Dette afsnit søger at give en bedre indsigt i, hvordan en debat i applikationen forløber med forklaring af den underliggende kode, da dette er den mest komplicerede del af applikationen.

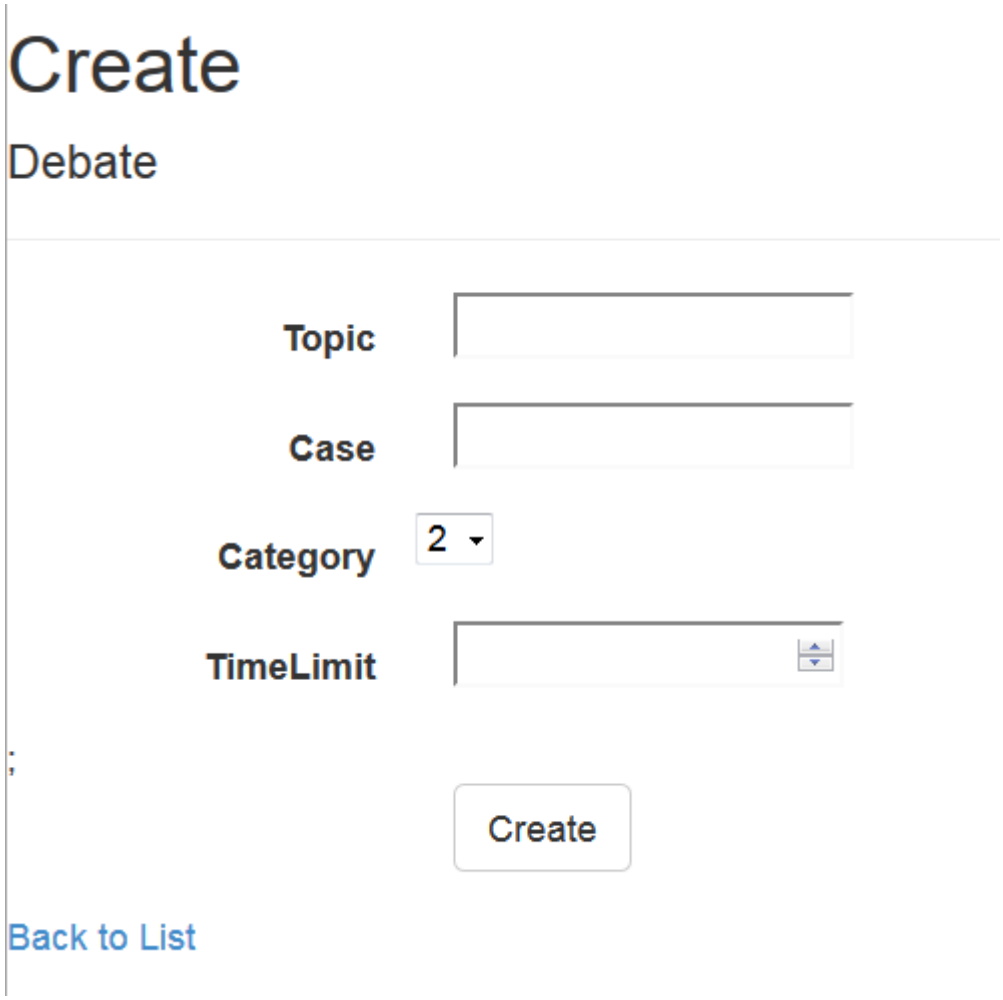
Afsnittet deles op i forklaringer af...

- et standard debat forløb.
- SignalR Hub opsættelse.
- View kode forklaring.
- Controller kode forklaring.
- forbedringer.

Debat forløb

Deltagelse som skaber af debat:

En debat begyndes ved, at en logget ind bruger via hjemmesiden navigerer til Create viewet, og skaber en ny debat, hvor følgende information udfyldes af brugeren:



Figur 9 - Create View

Topic - en kort linje, som opsummerer debatten's område. F.eks. "Kirken bør separeres fra staten".
Case - en uddybende beskrivelse af topiccet. F.eks. "Jeg føler ikke, at kirken er separeret nok fra staten. Hvorfor skal jeg betale skat, som blandt andet går til Kirkeministeriet, når jeg på ingen mulig måde vil støtte kirken. Kirken er ikke central for, hvordan landet regeres længere, og bør udelukkende håndteres af kirken selv uden et ministerie. Hvad er det næste? Facebookministeriet?"

Category - En kategori vælges, som er relevant for emnet. F.eks. 'politics' til dette emne.

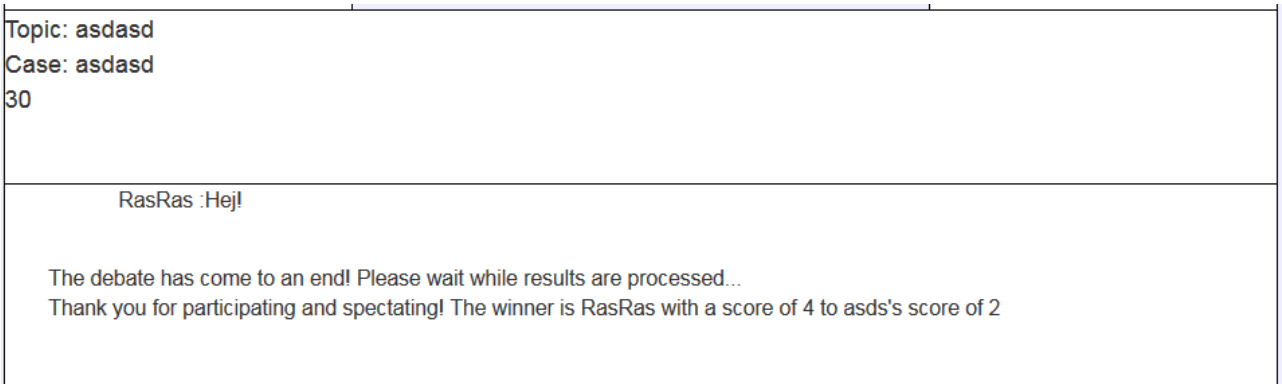
TimeLimit - Angives i sekunder. Tidsramme for hvor længe debatten skal vare.

Herefter trykkes på Create knappen, hvorefter brugeren sendes videre til LiveChat Viewet:

Figur 10 - LiveChat View - Creator perspektiv

Her ser vi et eksempel, hvor der allerede er en udfordrer.
Det eneste værktøj debatørerne har at arbejde med, er besked boxen, som kan anvendes til at sende beskeder, som broadcastes til alle deltagere inklusive tilskuere.

Som det kan ses er ovenstående debat på 3 ud af 30 sekunder. Når tælleren når op på 30, vil debatten ende og en besked vil fremstå i debatten med point optælling og kåring af vinder og taber:

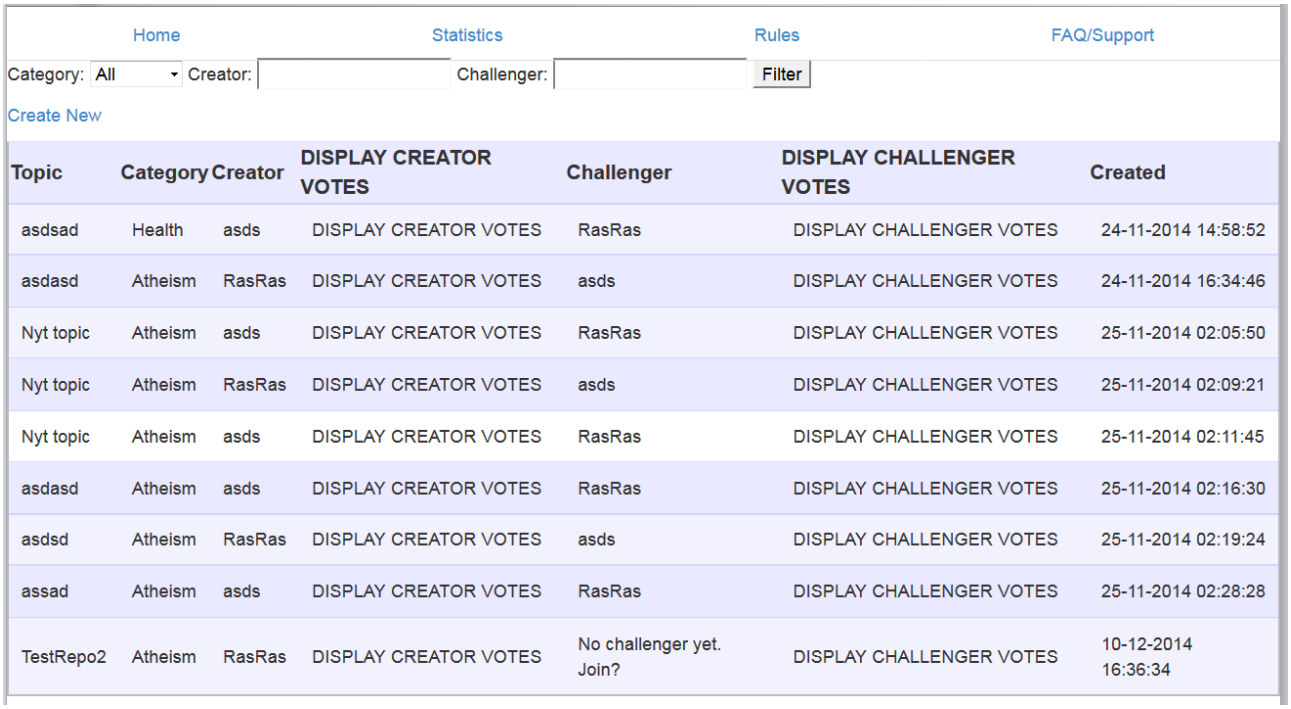


Figur 11 - Debat Afslutning

Deltagelse som udfordrer:

Det er meget simpelt at udfordre. Der er ikke de nødvendige indbyggede restriktioner nødvendige for at tillade en debat skaber at godkende en udfordrer osv., så derfor er denne proces meget simpel.

Er du logget ind som en bruger andet en skaberen af debatten, vil du ved visningen af Index viewet se muligheden for at deltage ved at trykke på "No challenger yet. Join?" som vist i dette billede i nederste debat listet:



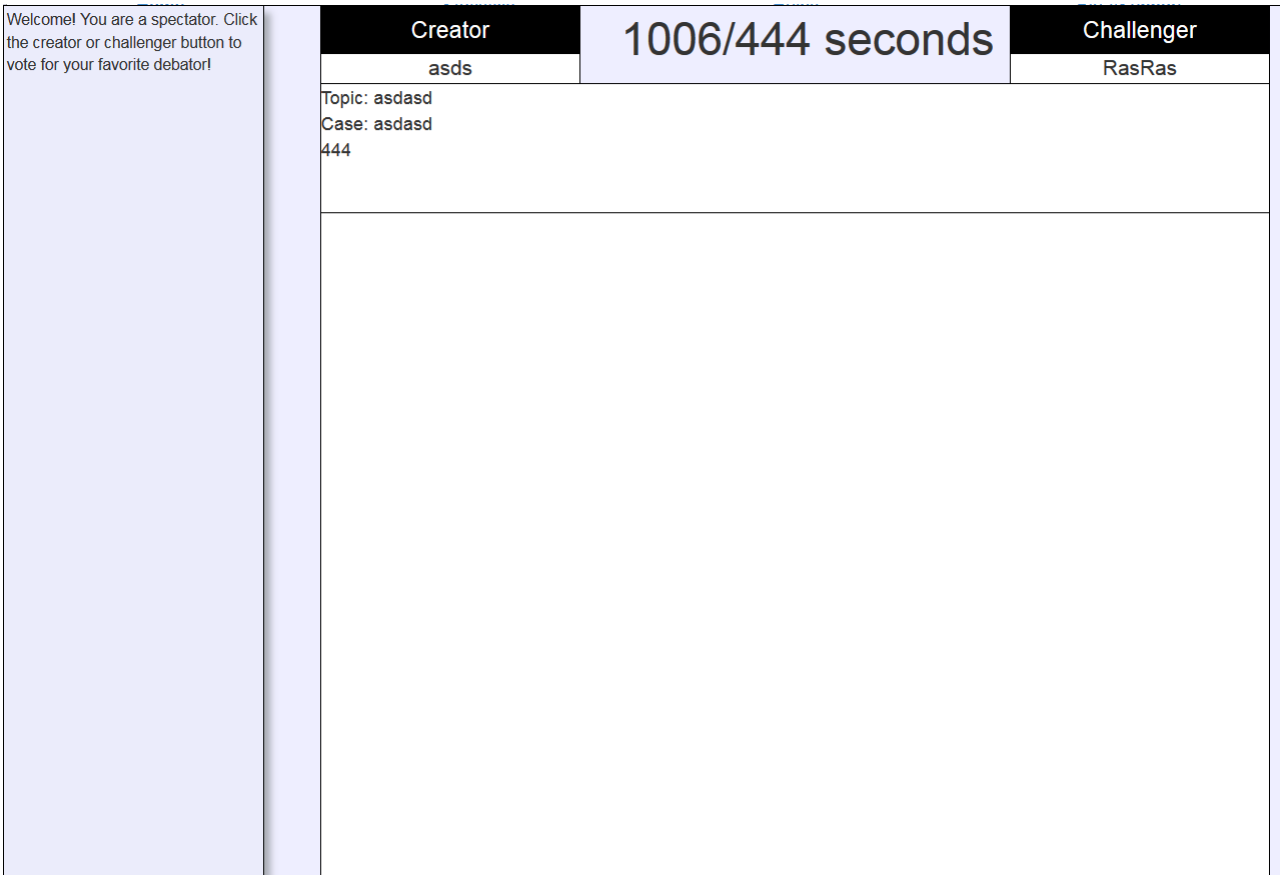
Topic	Category	Creator	DISPLAY CREATOR VOTES	Challenger	DISPLAY CHALLENGER VOTES	Created
asdsad	Health	asds	DISPLAY CREATOR VOTES	RasRas	DISPLAY CHALLENGER VOTES	24-11-2014 14:58:52
asdasd	Atheism	RasRas	DISPLAY CREATOR VOTES	asds	DISPLAY CHALLENGER VOTES	24-11-2014 16:34:46
Nyt topic	Atheism	asds	DISPLAY CREATOR VOTES	RasRas	DISPLAY CHALLENGER VOTES	25-11-2014 02:05:50
Nyt topic	Atheism	RasRas	DISPLAY CREATOR VOTES	asds	DISPLAY CHALLENGER VOTES	25-11-2014 02:09:21
Nyt topic	Atheism	asds	DISPLAY CREATOR VOTES	RasRas	DISPLAY CHALLENGER VOTES	25-11-2014 02:11:45
asdasd	Atheism	asds	DISPLAY CREATOR VOTES	RasRas	DISPLAY CHALLENGER VOTES	25-11-2014 02:16:30
asdsd	Atheism	RasRas	DISPLAY CREATOR VOTES	asds	DISPLAY CHALLENGER VOTES	25-11-2014 02:19:24
assad	Atheism	asds	DISPLAY CREATOR VOTES	RasRas	DISPLAY CHALLENGER VOTES	25-11-2014 02:28:28
TestRepo2	Atheism	RasRas	DISPLAY CREATOR VOTES	No challenger yet. Join?	DISPLAY CHALLENGER VOTES	10-12-2014 16:36:34

Figur 12 - Debate Index

Efterfølgende vil denne person sendes ind til samme view som set i "Deltagelse som skaber" sektionen.

Deltagelse som tilskuer:

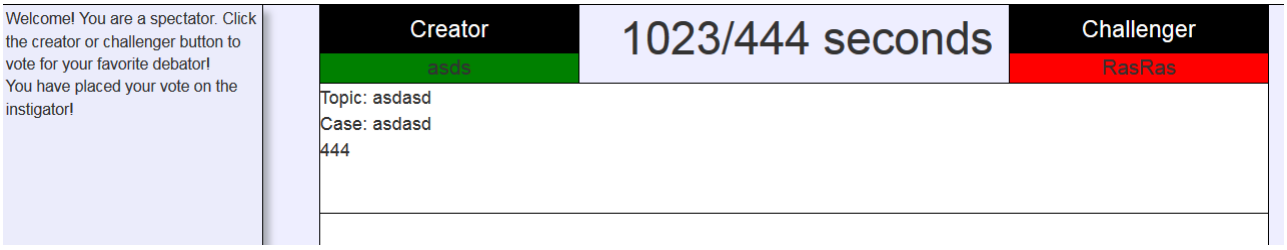
Det er muligt at deltage debatter som tilskuer ved at anvende samme Index view, som set i "Deltagelse som udfordrer". Ved at trykke på den ønskede debat række, vil brugeren sendes videre til følgende View:



Figur 13 - Livechat View - Tilskuer perspektiv

Venstre side af skærmen anvendes til notifikationer til brugeren.

Brugeren kan placere sin stemme på en af debatørerne ved at følge instruktionerne som anvist i notifikationsbaren, og trykke på enten "Creator" eller "Challenger". Ved at trykke på en af de to debatører vil viewet ændre farven på den stemte debatør til grøn, og den anden til rød tilmed gives brugeren besked om, at stemmen er modtaget:



Tilskueren kan til enhver tid ændre mening så længe debatten er live.

f. SignalR Hub

SignalR Hubben er hvor alle forbindelser samles som Clients. Enhver bruger som forbinder til Hubben anses for at være en Client, og bliver dermed del af en større gruppe, som kan modtage broadcasts.

For at kunne anvende SignalR, skal applikationen først sættes op. Dette gøres ved at anvende en OWin Startup.cs class:

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartupAttribute(typeof(RateMyDebate.Startup))]
namespace RateMyDebate
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();

            ConfigureAuth(app);
        }
    }
}
```

Startup classen er en fil, som kaldes ved applikationen's start, og dens rolle i denne sammenhæng, er at give SignalR funktionalitet ved at OWIN adskiller server og applikation, og gør applikationen selvhostende.
Startup classen kaldes af OWin for at initialisere applikationen's pipeline, og dermed skabes grundlaget for meget routing og mapping her.
Ved eksamen vil jeg går mere i dybden med dette for ikke at gøre denne forklaring uoverskuelig.

Når Startup classen er færdig, vil det være muligt at self-hoste applikationen, og dermed benytte SignalR. Dette bringer os videre til Hubben, som er en almindelig class, som implementerer Hub fra SignalR biblioteket. Hub classen er ikke mere end en samling server-side metoder, hvor visse af

dem kan kaldes af clients. Det er vigtigt i de næste sektioner, at der er en vigtig distinktion mellem client og server metode kald, da mange af metoderne har ens navne. Client metoder defineres i Views via. JQuery mens Server metoderne befinder sig i Hub classen. Clients kan kalde Serveren's metoder fra LiveChat Viewet og Serveren kan kalde Client metoder fra sine egne definerede metoder i Hubben. Her følger en gennemgang af Hub classen:

```
namespace RateMyDebate.Hubs
{

    public class ChatHub : Hub
    {
        static readonly HashSet<int> DebateGroups = new HashSet<int>();
        private Timer _timer;
        private int time;

        public void JoinDebateGroup(int debateId)
        {
            String Id = debateId.ToString();
            if (!DebateGroups.Contains(debateId))
            {
                CreateDebateGroup(debateId);
            }
            else
            {
                Groups.Add(Context.ConnectionId, Id);
            }
        }

        public void CreateDebateGroup(int debateId)
        {
            String Id = debateId.ToString();
            if(DebateGroups.Contains(debateId)) return;
            DebateGroups.Add(debateId);
            JoinDebateGroup(debateId);
            Clients.All.debateGroups(new[] {debateId});
        }

        public void StartTimer(int debateId)
        {
            time = 0;
            _timer = new Timer(UpdateTimer, debateId, TimeSpan.FromSeconds(0),
TimeSpan.FromSeconds(1));
        }

        private void UpdateTimer(Object state)
        {
            time += 1;
            BroadcastTimer(state);
        }

        private void BroadcastTimer(Object state)
        {
            String debateId = state.ToString();

            Clients.Group(debateId).broadcastTime(time);
        }

        public void Send(string name, string message, int debateId)
        {
            String debate = debateId.ToString();

            Clients.Group(debate).broadcastMessage(name, message);
        }
    }

}
```

HashSettet DebateGroups er en samling af Id's. Hvert Id korresponderer med en Debat's Id, og vil derfor sikres det, at alle rum er unikke, da DebateId er en primary key. HashSettet er tilmed statisk, da det skal være ens for samtlige forbindelser til Hubben. HashSettet's funktionalitet kommer vi ind på om lidt.

Timer objektet anvendes til at broadcaste 'time' hvert sekund til sin respektive debat. Også dette vil vi komme ind på senere.

JoinDebateGroup og CreateDebateGroup er to essentielle metoder i applikationen. Når en klient forbinder til en debat, vil et metode kald laves til JoinDebateGroup. Hvis den nødvendige gruppe for debatten med debatten's Id ikke eksiterer kaldes CreateDebateGroup, som tilføjer denne gruppe og opdaterer hashsettet. Ellers anvendes clienten's ConnectionId til at placere clienten i den korrekte gruppe.

Metodernes vigtighed stemmer fra den funktionalitet de giver til at placere Clients i seperate rum. Hvis ikke Clients placeres i seperate rum, vil alle debatter broadcaste tid og beskeder til hinanden, da Hubben i realiteten så agerer som en enkelt debate group.

De næste tre metoder anvendes i forhold til at oprette en timer for en seperat debat eller opdatere og broadcaste hver debat's timer.

StartTimer er en metode, som kan kaldes af en skaber af en debat ved at anvende en knap i LiveChat viewet, når vedkommende føler sig klar til at starte tidsfristen. Hver timer er indstillet til at kalde UpdateTimer metoden og parse debatId parametret med videre til metoden hvert sekund. På denne måde opdateres time variablen med +1 hvert sekund. For at holde det simpelt er timeren skabt i sekunder for nu.

Når UpdateTimer metoden kaldes hvert sekund lægges der altså et sekund til, og BroadcastTimer metoden kaldes.

BroadcastTimer kaldes for endelig at pushe time ud til dens respektive debat og opdatere denne med den nye værdi. Objekt parametret indeholder det nødvendige debat id. Debat id'et parses ind i et kald til en client metode i denne linje:

```
Clients.Group(debateId).broadcastTime(time);
```

'broadcastTime' er faktisk navnet på en af de definerede Client metoder i Livechat Viewet, som vi vil komme til senere.

Til sidst ses Send metoden, en metode udelukkende anvendes af debattørerne, som også anvender et call til en Client metode, nemlig 'broadcastMessage'.

g. LiveChat View

LiveChat Viewet sås på billeder i afsnittet om gennemgangen af en almindelig debat i afsnit 4.e. Dette View står altså for præsentationen af det interface, som møder brugere, når de forbinder til en debat, uanset om det er en debattør eller tilskuer.

Viewet anvender Razor syntax, C#, HTML/CSS og JQuery. Jeg har valgt at anvende Razor syntax her (@{ } kode blokke), da det tillader mig at anvende C# inde i en .cshtml side. Dette anvendes til at sikre, at de rigtige partial views vises til brugerne.

HTML/CSS har ansvaret for siden's layout og design samt form-groups, som inkluderer text fields og knapper.

JQuery anvendes til dynamisk funktionalitet. JQuery anvendes her til både at etablere en del af kommunikationen mellem hub og view samt gemme al nødvendig data.

Da View filen er lidt mere kompliceret end filer vi har undersøgt tidligere i rapporten, vil koden blive præsenteret i stykker fra top til bund med forklaringer efter hvert stykke.

Her begyndes med toppen af filen:

```
@model RateMyDebate.ViewModels.DebateDisplayViewModel


@{
    ViewBag.Title = "LiveChat";
    Layout = "";
    @Styles.Render("~/Content/css")
    @Styles.Render("~/Content/RMD.css")
    @Scripts.Render("~/bundles/modernizr")
    var user = Session["UserInfoSession"] as UserInformation;

}
```

Her indikeres i første linje, at denne fil anvender DebateDisplayViewModel, som sin model. Dette betyder, at jeg kan referere til den model, som parses igennem til Viewet via controlleren, og til enhver tid hente information fra denne. DebateDisplayViewModellen er en class, som ikke indeholder andet end disse fire felter:

```
        public Debate Debate { get; set; }
        public Category Category { get; set; }
        public UserInformation CreatorInformation { get; set; }
        public UserInformation ChallengerInformation { get; set; }
```

ViewModellen fyldes af Controlleren med den nødvendige information, således at al nødvendig information om debatten kommer med over. Her angives nogle detaljer for Viewet og CSS filer renderes til brug. Den vigtigeste linje er Session variablen, som sikrer at vi har information om den nuværende bruger.

Efter dette stykke følger det layout, som alle forbundne brugere vil have adgang til, som inkluderer visning af deltagernes navne samt information om debatten. Da jeg ikke føler dette stykke ikke behøver forklaring fortsætter jeg til dette stykke:

```
@if (user == null)
{
    @Html.Partial("NonUser")
}
else if (user.userInformationId == Model.CreatorInformation.userInformationId)
{
    @Html.Partial("_Creator")
}
else if (user.userInformationId == Model.ChallengerInformation.userInformationId)
{
    @Html.Partial("_Participant")
}
else
{
    @Html.Partial("Spectator")
}
```

Denne Razor kode blok spiller en meget vigtig rolle i visningen af de korrekte partial views. Hver 'if' iteration har til formål at tjekke 'user' Session variablen, som vi så det forrige stykke. Ved at undersøge om brugeren's id matcher med det id, som er sat til at være 'Creator' for debatten vil den tolke dette som, at denne bruger har skabt debatten, og derfor vil den vise partial viewet for Creator osv.

Herefter følger JQuery delen, som indeholder visse af de metoder, som nævnes i afsnit 4.f omkring Hubben:

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/jquery.signalR-2.1.1.min.js"></script>
<script src="~/Scripts/Timer.js"></script>
<script src="/signalr/hubs"></script>
<script type="text/javascript">

        $(function() {
            var chat = $.connection.chatHub;
            var client = chat.client;
            var server = chat.server;
            var debateConnectionId = '@Model.Debate.DebateId';

            $.connection.hub.start().done(function() {
                server.joinDebateGroup(debateConnectionId);
            });

            chat.client.broadcastMessage = function (name, message) {
                var formattedName;
                var formattedMessage;


                formattedName = '<div /> ' + name;
                formattedMessage = $('<div />').text(message).html();

                $('#discussion').append(formattedName + ' :' + formattedMessage);
            };

            chat.client.broadcastTime = function (time) {
                var formattedTime;
                var timeLimit = '@Model.Debate.TimeLimit';

           formattedTime = $('<div />').text(time).html() + '/' + timeLimit + ' seconds';

                $('.timerDiv').text(formattedTime).html();

                if (time == timeLimit) {
                    $('.timerDiv').text('Time is up!');
                    $('.chatDiv').append('<br /> The debate has come to an end! Please wait
while results are processed...');

                    var debateId = '@Model.Debate.DebateId';

                    $.ajax({
                        url: "/Debate/ProcessDebateResult",
                        type: "POST",
                        async: false,
                        datatype: "json",
                        data: {
                            debateId: debateId
                        },
                        success: function (data) {
                            $('.chatDiv').append('<br />' + data);
                        }
                    });
                }
            };
        });

</script>
```

JQuery delen her indeholder hvad betegnes som Client metoder. Først initialiseres fire vigtige variabler:

```
var chat = $.connection.chatHub;
var client = chat.client;
var server = chat.server;
var debateConnectionId = '@Model.Debate.DebateId';
```

'chat' variablen forbereder en forbindelse til hubben.
'client' anvendes til definitionen af client side metoder.
'server' anvendes til kald af server side metoder, dvs. de metoder vi så i afsnittet om hubben.
'debateConnectionId' læser DebatId'et, som anvendes i adskillige metodekald, for at sikre broadcasting tilbage til samme debat.

Herefter følger forbindelsen til serveren ved at bruge chat variablen. Ved at anvende JQuery er det muligt for mig at at angive et sæt instruktioner, når forbindelsen til Hubben er fuldendt:

```
$.connection.hub.start().done(function() {
    server.joinDebateGroup(debateConnectionId);
});
```

Når start() er fuldendt og forbindelsen er etableret, vil der foretages et kald til en server metode, hvilket er JoinDebateGroup, som blev forklaret i sidste afsnit. Det er her debat id'et parses igennem, således at brugeren bliver meldt ind i denne debat's forbindelses gruppe, og nu har adgang til alle broadcasts for denne debat.

I JQuery scriptet findes to metoder/functions: broadcastTime og broadcastMessage. Teknisk set anvendes begge metoder ikke til at broadcaste som sådan, men til at behandle og viderføre modtagne broadcasts til brugeren.

```
chat.client.broadcastMessage = function (name, message) {
    var formattedName;
    var formattedMessage;


    formattedName = '<div /> ' + name;
    formattedMessage = $('<div />').text(message).html();

    $('#discussion').append(formattedName + ' :' + formattedMessage);
};
```

Denne metode kaldes fra Hubben hver gang en debatør sender en besked. Her vil de to formatteringsvariabler formatere det modtagne navn og besked. Herefter indsættes dette i 'discussion', som er en HTML hvori chatteksten forholder sig.

Den næste metode er broadcastTime. Denne metode er mere kompliceret, da den benytter et AJAX call ved timed events. AJAX callet anvendes her til at sende en række data til en controller metode, som vil processere det endelige debat resultat.

```
            chat.client.broadcastTime = function (time) {
                var formattedTime;
                var timeLimit = '@Model.Debate.TimeLimit';

            formattedTime = $('<div />').text(time).html() + '/' + timeLimit + ' seconds';

                $('.timerDiv').text(formattedTime).html();

                if (time == timeLimit) {
                    $('.timerDiv').text('Time is up!');
                    $('.chatDiv').append('<br /> The debate has come to an end! Please wait
while results are processed...');

                    var debateId = '@Model.Debate.DebateId';

                    $.ajax({
                        url: "/Debate/ProcessDebateResult",
                        type: "POST",
                        async: false,
                        datatype: "json",
                        data: {
                            debateId: debateId
                        },
                        success: function (data) {
                            $('.chatDiv').append('<br />' + data);
                        }
                    });
                }
            };
```

Denne metode har til en vis grad samme funktion som 'broadcastMessage' idet at den modtager en variabel, og sender denne videre ind til brugeren i formateret form. Den modtager tiden i sekunder

og opdaterer 'timerDiv', som er en anden HTML . Når applikationen kører vil denne metode blive kaldt hvert sekund hos alle clients af Hubben. Tilmed indeholder metoden en if-sætning, som checker om den nuværende tid debatten har kørt er ens med 'timeLimit', som er den tidsbegrænsning, som er sat for debatten. Det vil sige, at når tidsbegrænsningen er nået, vil if-sætningen være true og eksekveres. Inde i if-sætningen udstedes en række beskeder til brugerne om debatten's afslutning. Herefter udføres et AJAX call, som kalder en metode, 'ProcessDebateResult', hvilken befinder sig i DebateController classen, som vi kommer til senere. 'data' feltet, som ses i AJAX callet modtager return data fra ProcessDebateResult, og giver besked om debatten's resultat.

h. Creator og Participant Views

Creator og Participant viewsne er de views, som vises til henholdsvis skaberen af en debat og udfordreren. Disse holdes i seperate views, da en debat skaber vil have flere funktioner til råde end en udfordrer vil. F.eks. evnen til at starte debatten's timer.

Begge views indeholder denne metode til at sende beskeder:

```
$('#sendmessage').click(function () {

    var sender = '@user.nickName';
    var message = $('#message').val();

    chat.server.send(sender, message debateId);
    $('#message').val('').focus();


    $.ajax({
        url: "/Debate/SaveMessage",
        type: "POST",
        data: {
            sender: sender,
            message: message,
            debateId: debateId
        }
    });
});
```

Denne metode aktiveres ved et tryk på en knap med id'et 'sendmessage'. Når denne knap aktiveres, vil metoden hente afsenderen's navn fra Session variablen og beskeden's tekst findes i et text field med id'et 'message'.

Herefter kaldes server/hub metoden 'Send', som sås omtalt i afsnittet om hubben. Metoden sætter fokus tilbage i tekstfeltet, klar til at skrive en ny besked og et AJAX call udføres til DebateController med samme data, som sendes til hub metoden. Metoden som kaldes er SaveMessage, som gemmer teksten i en database således at debattens gemmes til senere læsning, og nyforbunde tilskuere vil modtage hele chatteksten for den igangværende debat ved forbindelse.

En metode som udelukkende findes i Creator partial viewet er muligheden for at starte debatten's timer:

```
$('#startTimer').click(function() {
    server.startTimer(debateId);
});
```

Denne metode tillader skaberen af debatten at kalde hubben's StartTimer metode, og aktivere debatten's timer. Det er min intention at skabe mange forbedringer omkring denne metode. For eksempel bør det ikke være muligt at chatte før timeren faktisk begynder. Tilmed vil jeg sætte to flags, et for hver debattør, som når begge sat til true, vil angive, at begge debatørere er klare til at begynde debatten. Det vil bidrage meget til et mere professionelt og kontrolleret brugerinterface.

i. DebateController

Flere dele af DebateControlleren er omtalt i andre områder af rapporten, blandt andet i forhold til dependency injection. For at holde denne sektion i tråd med de forrige to afsnit, vil jeg udelukkende udlægge metoder, som er relevante for en aktiv debat.

LiveChat

Dette er metoden som kaldes, når en bruger klikker sig ind på viewet.

```
public ActionResult LiveChat(int? id)
{
    DebateDisplayViewModel DDVM = FindDebateDisplayViewModel(id);

    ViewBag.Title = DDVM.Debate.Subject;

    return View(DDVM);
```

```
}
```

Metoden er et actionresult og returnerer f.eks. et View som ses her. Metoden anvender en anden metode, FillDebateDisplayViewModel, til at fylde DebateDisplayViewModellen med nødvendig information, som vi så i afsnittet om Livechat Viewet.
FillDebateDisplayViewModel metoden ser således ud:

```
public DebateDisplayViewModel FindDebateDisplayViewModel(int? id)
{
    DebateDisplayViewModel DDVM = new DebateDisplayViewModel();

    DDVM.Debate = db.Debate.Find(id);
    UserInformation creator = db.UserInformation.Find(DDVM.Debate.CreatorIdId);
    DDVM.CreatorInformation = creator;

    UserInformation challenger =
     db.UserInformation.Find(DDVM.Debate.ChallengerIdId);
    DDVM.ChallengerInformation = challenger;

    Category category = db.Categories.Find(DDVM.Debate.CategoryIdId);
    DDVM.Category = category;

    return DDVM;
}
```

Laver en række reads på databasen og sætter værdierne i ViewModellens variabler.

Herefter sættes Viewet's titel til at være debatten's emne, og metoden returnerer et view til brugeren med DebateDisplayViewModellen som Viewet's model, som også sås i afsnittet om LiveChat Viewet.

EnterChallenger

Denne metode anvendes, når en bruger forsøger at deltage som udfordrer.

```
[HttpPost]
public void EnterChallenger(int debateId)
{
    var user = Session["UserInfoSession"] as UserInformation;
    Debate debate = _debateRepository.FindDebate(debateId);

    if (debate.ChallengerIdId == null)
```

```
        {
            debate.ChallengerIdId = user.userInformationId;
            _debateRepository.UpdateDebate(debate);
            _debateRepository.Save();
        }
    }
```

Session variablen henter vedkommende bruger's information. Herefter undersøger metoden om debatten fortsat er tom(for at forhindre udfordrere i at overskrive hinanden i databasen). Hvis den er tom, vil brugeren indsættes i debatten som udfordrer og tages videre til debatten via JQuery scriptet, som kalder metoden.

SaveMessage

Denne metode kaldes, når en besked sendes til chatten.

```
[HttpPost]
public void SaveMessage(String sender, String message, int debateId)
{
       message = message.Replace("\n", "");
String formattedMessage = "\n" + "[" + DateTime.Now + "] " + sender + " : " + message;
      Debate debate = _debateRepository.FindDebate(debateId);
      debate.ChatText += formattedMessage;
      _debateRepository.UpdateDebate(debate);
      _debateRepository.Save();
}
```

En vigtig del af denne metode er formateringen af beskeden som modtages. Ved at gemme "\n" i beskeden kan disse senere konverteres til linjebrud i LiveChat viewet med følgende linje i chat 'en, hvor beskederne indsættes:

```
@Html.Raw(Html.Encode(Model.Debate.ChatText).Replace("\n", "<br />"))
```

På denne måde bevares formatteringen.
For at forebygge linebreak injections fjernes alle "\n" i beskeden først, så debatørere ikke kan forstyrre chatten ved at spamme linebreaks.

ProcessDebateResult

Dette er en tungere metode, som håndterer processen for at sikre, at debatter og resultater bliver gemt ved afslutning af en debat.

```
        public String ProcessDebateResult(int debateId)
        {
            Debate debate = _debateRepository.FindDebate(debateId);
            int creatorId = debate.CreatorIdId;
            int? challengerId = debate.ChallengerIdId;

            UserInformation creator = db.UserInformation.Find(creatorId);
            UserInformation challenger = db.UserInformation.Find(challengerId);

            String creatorNick = creator.nickName;
            String challengerNick = challenger.nickName;

            if (challengerId == null) return "As no challenger was found, the debate will
end without a conclusion. Sorry folks!";

            Result result = new Result();
            result.DebateId = debateId;
            String endingSentence = "Not assigned";

            List<Vote> creatorVoteCount =
db.Votes.ToList().Where(x => x.DebateId == debateId & x.VotePos == 1).ToList();
            List<Vote> challengerVoteCount =
db.Votes.ToList().Where(x => x.DebateId == debateId & x.VotePos == 2).ToList();

            int creatorVotesNum = creatorVoteCount.Count();
            int challengerVotesNum = challengerVoteCount.Count();

            if (creatorVotesNum > challengerVotesNum)
            {
                result.WinnerId = creatorId;
                result.LoserId = challengerId;

                endingSentence = "Thank you for participating and spectating! The winner is
" +
                 creatorNick + " with a score of " + result.CreatorVotesCount + " to " +
                 challengerNick + "'s score of " + result.ChallengerVotesCount;
            }
            else if (creatorVotesNum < challengerVotesNum)
            {
                result.WinnerId = challengerId;
                result.LoserId = creatorId;

                endingSentence = "Thank you for participating and spectating! The winner is
" +
                 challengerNick + " with a score of " + result.ChallengerVotesCount + " to "
+
                 creatorNick + "'s score of " + result.CreatorVotesCount;
            }
            else if (creatorVotesNum == challengerVotesNum)
            {
                result.Draw = true;

                endingSentence = "Thank you for participating and spectating! Unfortunately
a winner could not be found as both participants had a vote count of " + creatorVotesNum;
            }

            result.CreatorVotesCount = creatorVotesNum;
            result.ChallengerVotesCount = challengerVotesNum;

            SaveResult(result);
```

```
            SetDebateInactive(debateId);

            return endingSentence;
        }
```

Metoden henter alle stemmer, som debatørerne har modtaget i den gældende debat, og stemmerne sættes i to liste respektive for hver debatør. I den første række if-sætninger måles de to lister's stemmeantal mod hinanden. Den debatør med flest stemmer sættes som vinderen i result objektet, den med færrest som taber. Hvis begge debatørere har lige mange stemmer anses debatten for et draw. Tilmed indstilles 'endSentence' til at matche med resultatet.
Efter følgende kaldes disse metoder:

```
            SaveResult(result);
            SetDebateInactive(debateId);
```

SaveResult gemmer resultatet i en data tabel for resultater.
SetDebateInactive sætter debatten inaktiv, så den nu ikke vil fremstå i Index viewet.

Til sidst returneres 'endsentence', som modtages af AJAX callet, som kalder på metoden i første omgang, og 'endsentence' broadcastes til alle i debatten for at informere om resultatet.

j. Forbedringer

Der er umådeligt mange forbedringer, som kunne foretages for at forbedre denne del af programmet. Det er svært at skrive dem alle ned i et afsnit, og mange vil sikkert blive udeladt. Jeg skal forsøge at nævne flere ved eksaminationen.

Fra brugeren's perspektiv vil jeg anse det som værende langt fra færdigt. Det vil sige, at selv hvis al konstruktion bag software fulgte samtlige konventioner for god kode, vil det stadigvæk se ud som en halvfærdig applikation, da flere features mangler.

Fra softwaren's perspektiv føler jeg, at koden er virkeligt mangelfuld på mange måder. Jeg har påskyndt processen lidt for at gøre det til et produkt så meget som muligt, men der mangler adskillige

Afsluttende konklusion

Produktet er ikke lancerbart på nuværende tidspunkt, men jeg føler, at jeg kan færdiggøre det inden for en rimelig tidsramme. Med dette projekt har jeg formået mine bedste arbejdsrutiner samt lært meget om planlægningen af et større projekt.

Desværre var flere elementer af projektet sværere at gennemføre end forventet, og dermed fik jeg ikke lavet det projekt, som jeg havde ønsket det skulle blive inden afleveringen.

Mine tidsestimeringer fejlede primært pga. mit ubekendtskab med de anvendte programmeringssprog og dertilhørende koncepter. Da jeg undervurderede applikationen's samlede komplexitet kom jeg ud for store tidskonsumerende udfordringer. Blandt andet havde jeg ikke forventet, hvor stor en udfordring det ville være at synkronisere en timer blandt flere brugere, og jeg kom ud for flere bugs i brugen af Visual Studio, som tog længere tid at opdage.

Litteraturliste

Bøger:

Freeman, Adam
Pro ASP.NET MVC 5
Fifth edition 2013

Aguilar, José
SignalR Programming in Microsoft ASP.NET
First printing 2014

Aguilar, José
ASP.NET SignalR
First printing 2013

Galloway, Jon & Wilson, Brad & Allen, Scott & Matson, David
Professional ASP.NET MVC 5
First edition 2014

Vespa, Roberto
SignalR Real-time Application Cookbook
First edition 2014

Ingebrigtsen, Einar
SignalR: Real-time Application Development
First edition 2013

Artikler:

Bubblog(ukendt forfatter) - 10/11/2014
Controller Factory and Ninject
https://bubblogging.wordpress.com/2012/06/04/mvc-controller-factory-ninject/

Dykstra, Tom - 10/9/2014
Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application
http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

Fletcher, Patrick - 10/22/2014
Getting started with SignalR 2
http://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr

Fletcher, Patrick - 10/22/2014
Supported Platforms(SignalR 2)
http://www.asp.net/signalr/overview/getting-started/supported-platforms

FitzMacken, Tom & Fletcher, Patrick - 02/12/2014
Working with Groups in SignalR
http://www.asp.net/signalr/overview/guide-to-the-api/working-with-groups

Dykstra, Tom & FitzMacken, Tom - 10/22/2014
Server Broadcast with SignalR 2
http://www.asp.net/signalr/overview/getting-started/tutorial-server-broadcast-with-signalr

Medie:

TechEd North America 2014

SignalR: Building Real-Time Applications with ASP.NET SignalR

https://www.youtube.com/watch?v=us-Q3do-N7M

Citerede Værker

Hofstadter, D. (1979). *Gödel, Escher, Bach.* Basic Books.

Appendix