

INGENIØRHØJSKOLEN AARHUS

SEMESTERPROJEKT 4

IKT

The screenshot shows a web-based application for advertising items. At the top, there is a search bar with a magnifying glass icon and the placeholder text "Search...". To the right of the search bar are links for "BrugerA" and "Log ud". Below the search bar, the word "Annoncer" is displayed in bold. On the left, there is a listing for a "Kop byttes" (cup exchange) item by "Jacob Jensen". The listing includes a large image of a black coffee cup, the location "Århus C" marked on a map, and a short description in Latin. At the bottom of the listing is a link "Vis annonce". On the right side of the screen, there is a sidebar titled "Indsnævring af annoncer" (Filtering advertisements). It features a "Radius" slider, a "Kategorier" (Categories) section with a list of items like "Apps", "Games", "Movies" (which is currently selected), "Books", and "Newspapers".

Dokumentation: BargainBarter

Udarbejdet af:

Simon Thrane Hansen	201500150
Lars Hjerrild	201409555
Jeppe Skødt Benjaminsen	201500154
Kasper Lauge Madsen	201409873
Mikkel Cramer Poulsen	20112893
Søren Kruse Holm	201409556
Morten Christensen	201500162

Vejleder:

Poul Ejnar Rovsing

15. december 2016

Indhold

1 Projektbeskrivelse	4
2 MoSCoW analyse	5
3 Gherkin Userstories	6
3.1 Aktør kontekst diagram	6
3.1.1 Aktørbeskrivelse	6
3.2 Userstories	7
3.2.1 Oprette en bytteannonce	7
3.2.2 Opret en brugerprofil	7
3.2.3 Se geografisk lokation for bytteannonce	7
3.2.4 Glemt password eller brugernavn	8
3.2.5 Kommentering en annonce	8
3.2.6 Søg efter bytteannoncer	8
3.2.7 Se bytthistorik	9
3.2.8 Kontakt en bruger direkte	9
3.2.9 Anmeldelse af en bytthandel	9
4 Ikke-funktionelle krav	11
5 Accepttestspecifikation	12
6 Systembeskrivelse	15
7 System arkitektur	17
7.1 Domæneanalyse	17
7.2 Overordnede sekvensdiagrammer	18
8 Software-arkitektur	22
8.1 MVC	22
8.2 3-lags-modellen	23
8.2.1 Præsentationslaget	25
8.2.2 Businesslag	25
8.2.3 Database lag	25
8.3 Repository Pattern	26
8.4 Unit Of Work	27
8.5 Valg af Teknologier	27
8.5.1 ASP.NET	27
8.5.2 SignalR	27
8.6 Generelle overvejelser i forbindelse med Arkitektur	27

9 Design og Implementering	29
9.1 Controllere	29
9.1.1 Home	29
9.1.2 BarterAd	29
9.1.3 Search	30
9.1.4 UserProfile	31
9.1.5 Chat	31
9.2 Generel controller-funktionalitet	31
9.2.1 Eksemplificering af controller-funktionalitet	31
9.3 Modellen	32
9.4 Data fra Database	33
9.4.1 Create	34
9.4.2 Read	34
9.4.3 Update	35
9.4.4 Delete	36
9.5 Chat Room	37
9.5.1 Chat Controller	37
9.5.2 Chat Hub	37
9.5.3 Sekvens diagram af Chat room	38
9.5.4 Chat view	38
9.6 Databasestruktur	39
9.6.1 Persistent data	39
9.7 ER-diagram	40
9.8 Repository Pattern	42
9.9 Map og distance	46
9.9.1 Konvertering til koordinater	47
9.9.2 Beregning af afstand	47
9.9.3 Kortvisning	47
9.10 Rating af bruger	48
9.10.1 Forundersøgelse	48
9.10.2 Virkemåde	48
9.11 Byttehandel	49
9.12 Kommentarer	51
9.13 Brugervenlighed	52
10 Test	53
10.1 Continuous Integration	53
10.2 Unit Tests af controllerne	54
10.2.1 Unit test igennem DAL	56
10.3 Integrationstest	57
10.4 Manuelle tests	57
10.4.1 Test af byttehandel	58
10.4.2 Test af Chat	58
10.4.3 Opret annonce	59

11 Accepttest	60
11.1 Funktionelle krav	60
11.1.1 Oprette En Bytteannonce	60
11.1.2 Opret En Brugerprofil	60
11.1.3 Se Geografisk Lokation For Bytteannonce	61
11.1.4 Glemt Password Eller Brugernavn	61
11.1.5 Kommenter En Annonce	62
11.1.6 Søg Efter Bytteannoncer	62
11.1.7 Se Byttestrøm	62
11.1.8 Kontakt en bruger direkte	63
11.1.9 Anmeldelse af en byttestrøm	63
11.2 Ikke funktionelle krav	64
12 Litteraturliste	67

1. Projektbeskrivelse

I dette projekt udvikles en valutafri platform til at bytte brugte såvel som ubrugte ejendele. Platformen skal kunne tilgås via internettet, hvor brugeren skal være i stand til at lave en annonce med et billede samt en beskrivelse af en artikel¹, som brugeren ønsker at bytte. Brugeren skal have mulighed for at kunne søge efter artikler, som andre brugere har lagt op, via platformens indbyggede søgefunktion. Når brugeren har fundet en artikel, som han ønsker at eje, kan han anmode om en byttehandel med en af sine egne artikler.

Systemet skal kunne vise den geografiske placering af de forskellige artikler, og det skal være muligt at søge på artikler på baggrund af forskellige kriterier, herunder afstand og kategori.

Der skal desuden være understøttelse for, at brugerne kan have kontakt med hinanden via platformen. Både via et kommentarfelt under de forskellige annoncer der er på platformen, men også via en real-time chat funktion.

Denne platform skal præsenteres for brugeren som en hjemmeside, der skal være intuitiv og brugervenlig.

¹Artikel skal her forstås som den ting man ønsker at bytte i annoncen

2. MoSCoW analyse

Dette afsnit viser MoSCoW-analysen for BargainBarter systemet.

Must have:

- Mulighed for oprettelse af annoncer.
- Mulighed for oprettelse af brugerprofiler.
- En grafisk brugergrænseflade, der tilpasser sig til enhedstypen.
- Tilknyttelse til en ekstern (server side) databaseenhed til opbevaring af brugerprofiler, opslag og bytteannoncer.

Should have:

- Kortintegration¹, hvor forskellige bytteannoncer vises geografisk.
- Et chat-modul, hvor brugere kan kommunikere omkring byttehandler.
- Historik for en given brugers bytte-historik.
- Mulighed for, at man kan rate andre brugere, man har byttet med.

Could have:

- Et indbygget søgemodul således brugeren kan søge efter ting på baggrund af selvvalgte kriterier.
- E-mail notifikationer.
- En "glemt password" feature.

Won't have:

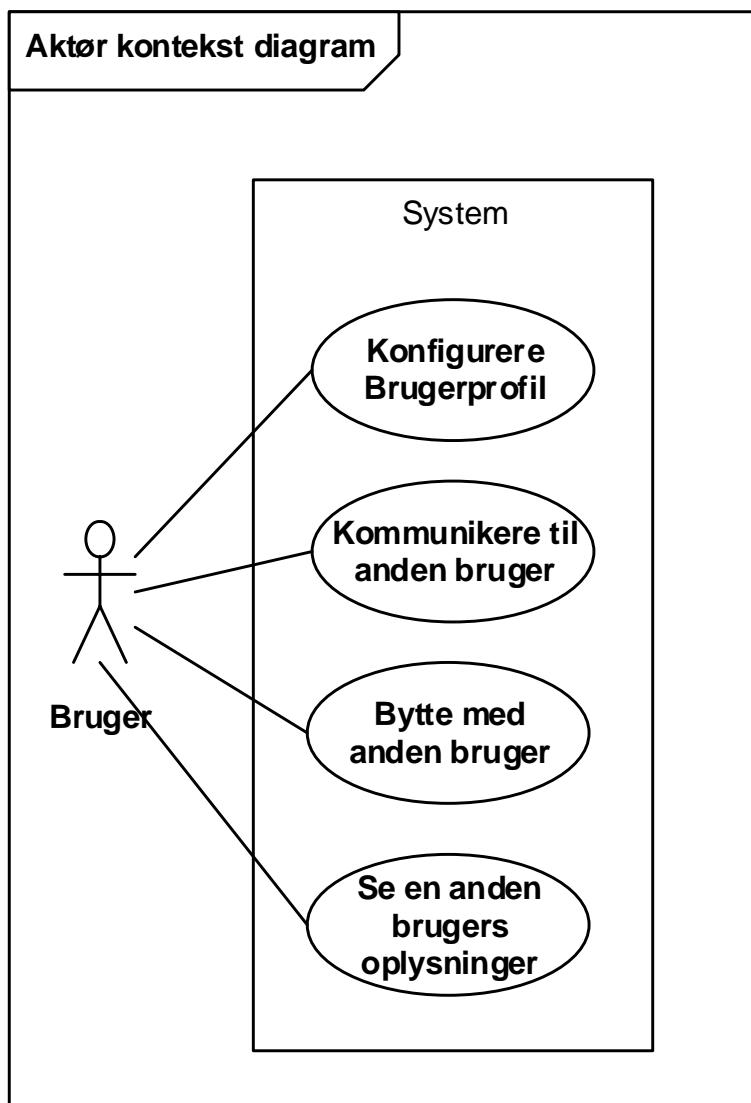
- En tilhørende mobil applikation med GPS integration.
- SMS notifikation.

¹Geografisk landkort

3. Gherkin Userstories

3.1 Aktør kontekst diagram

Diagrammet på 3.1 viser de epic userstories som systemet består af. Hver epic bliver delt op i mere specifikke userstories, for at skabe et bedre overblik



Figur 3.1: Aktør kontekst diagram

3.1.1 Aktørbeskrivelse

Den primære aktør er brugeren. Brugeren kan via en web-side oprette en brugerprofil. Derefter er det muligt at oprette bytteannoncer, søge efter ønskede genstande og kommunikere med andre brugere for at få færdiggjort en bytvehandel.

3.2 Userstories

3.2.1 Oprette en bytteannonce

EGENSKAB: Oprette en bytteannonce

Som bruger

Ønsker jeg at kunne oprette en bytteannonce

For at kunne bytte med andre brugere af systemet.

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Oprette en bytteannonce

Når bruger ønsker at oprette en bytteannonce i systemet

Så nавигerer han til menupunktet "Opret annonce"

Så udfylder han bytteannonce-skabelonen

Og trykker på "Opret annonce"-knappen

3.2.2 Opret en brugerprofil

EGENSKAB: Opret en brugerprofil

Som bruger

Ønsker jeg at kunne oprette en brugerprofil på websiden

For at få adgang til websidens funktionalitet

BAGGRUND

Givet at bruger ønsker en brugerprofil

SCENARIE: Opret en brugerprofil

Når bruger ønsker at oprette sig i systemet

Så nавигerer han til Menupunktet "Ny bruger"

Så indtaster han brugernavn, e-mail, password mm.

Så verificerer brugeren sig på tilknyttede e-mail adresse

Og brugeren kan nu logge ind med sin nye brugerprofil

3.2.3 Se geografisk lokation for bytteannonce

EGENSKAB: Se geografisk lokation for bytteannonce

Som bruger

Ønsker jeg at kunne se hvor den ønskede byttevare er

For at kunne se hvor en evt. byttehandel kunne finde sted

BAGGRUND

Givet at bruger Jakob er logget ind

Og han er inde på et annonceopslag lavet af personen Gitte

SCENARIE: Jakob vil se Gittes geografiske lokation

Når Jakob ønsker at se Gittes lokation

Så kigger han til højre for annoncebilledet
Og ser et kort, som angiver Gittes lokation

3.2.4 Glemt password eller brugernavn

EGENSKAB: Glemt password eller brugernavn

Som bruger

Ønsker jeg at kunne få tilsendt mit brugernavn og et nyt password

For at kunne komme ind på min bruger på trods af at have glemt mit password eller brugernavn

BAGGRUND

Givet at bruger er oprettet i systemet

Og har glemt sit password eller brugernavn

SCENARIE: Glemt password eller brugernavn

Når bruger ønsker at få tilsendt sit brugernavn med et nyt password

Så navigerer han til websidepunktet "Glemt password eller brugernavn"

Så skriver brugeren sin e-mailadresse ind

Så finder brugeren den tilsendte e-mail

Og brugeren kan nu logge ind med det tilsendte password

3.2.5 Kommentering en annonce

EGENSKAB: Kommentering af annoncer

Som bruger

Ønsker jeg at kunne kommentere annoncer

For at kunne afklaret spørgsmål etc. om annoncen

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Kommentere en annonce

Når bruger ønsker at kommentere en annonce

Så skal bruger navigere ned til en kommentarboks under annoncen

Så skal bruger kunne skrive en kommentar i kommentarboksen

Så skal bruger trykke "send" ved siden af kommentarboksen

Og kommentaren vises da under kommentarboksen

3.2.6 Søg efter bytteannoncer

EGENSKAB: Søg efter bytteannoncer

Som bruger

Ønsker jeg at kunne søge efter bytteannoncer

For at kunne finde en bestemt type vare

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Søg efter bytteannoncer

Når bruger ønsker at søge efter bytteannoncer i systemet

Så nавигerer brugeren til menupunktet "Søg"

Så indtaster brugeren den varer som brugeren ønsker

Og trykker på "søg"-knappen

3.2.7 Se bytnehistorik

EGENSKAB: Se bytnehistorik

Som bruger

Ønsker jeg at kunne se min bytnehistorik

For at kunne se, hvilke ting jeg har byttet mig til.

BAGGRUND

Givet at Bruger er logget ind

SCENARIE: Se Bytnehistorik – Webapplikationen

Når bruger ønsker at se sin bytnehistorik i systemet

Så nавигerer han til menupunktet "Brugerprofil"

Og trykker på "Bytnehistorik-knappen"

3.2.8 Kontakt en bruger direkte

EGENSKAB: Kontakt en bruger direkte

Som bruger

Ønsker jeg at kunne kontakte en bruger direkte via en chat fra annoncen

BAGGRUND

Givet at en brugeren er logget ind

SCENARIE: Kontakt brugere direkte

Når Brugeren ønsker at kontakte en anden bruger direkte

Så nавигeres der til den anden brugers annonce

Så nавигeres der til knappen "Start chat"

Så trykkes der på "Start chat"

Og så kan der chattes med den anden bruger

3.2.9 Anmeldelse af en bytnehandel

EGENSKAB: Anmeldelse¹ af en bytnehandel

Som bruger

Ønsker jeg at kunne vurdere en bytnehandel

For at angivne, hvor trovædig den anden bruger er.

BAGGRUND

¹Der er her tale om en rating

Givet at en bruger er logget ind
Og har afsluttet en byttehandel med en anden bruger

SCENARIE: Anmeldelse af andre brugere

Når Brugeren ønsker at vurdere byttehandlen en byttehandel med en anden bruger

Så nавигерer brugeren hen til "Afsluttede byttehandler"

Så trykkes "Vurder byttehandel" på den givne byttehandel

Så angives en score mellem 1 og 5

Så skrives der en kommentar (valgfrit)

Så trykkes "Send"

Og hans anmeldelse vises nu på den anmeldte brugers brugerprofil, og tælles med i den anmeldte
brugers samlede brugervurdering

4. Ikke-funktionelle krav

1. Man skal kunne komme ind på alle annoncer med kun 2 klik fra forsiden <http://10.29.0.30/bargainbarter>
2. Hjemmesiden skal kunne håndtere 10 brugere på samme tid
3. Ved brug af søgefeltet under hjemmesiden skal der maksimalt gå 15 sekunder fra man trykker på "søg"-knappen før resultaterne er fundet frem og er blevet vist på skærmen
4. Systemet skal have en online hjemmeside (fast IP)
5. Hvis en bruger søger efter en annonceplacering inden for en vis afstand, må den maksimale afstand til den pågældende annonceplacering højst overskride den valgte afstand med 100 meter
6. Hvis bruger A ønsker at kontakte bruger B gennem den indbyggede chat, så skal der maksimalt gå 10 sekunder fra bruger A trykker "send" til, at beskeden afleveret til bruger B
7. Når en bruger ønsker at vurdere en anden bruger, så skal man maksimalt kunne skrive 500 tegn
8. Når en bruger kommenterer en anden brugers annonce, skal der maksimalt gå 15 sekunder fra, at den kommenterende bruger trykker "Send" til at den anden bruger kan se kommentaren på den pågældende annonce
9. Hjemmesiden skal kunne tilgås fra følgende enheder (PC, mobil (Android: OnePlus3 og iPhone 5s) og tablet)

5. Accepttestspezifikation

I følgende afsnit følger accepttestspezifikationen for de ikke-funktionelle krav.

Krav	Beskrivelse	Forventet resultat	Resultat	✓ / X
Punkt 1	Der klikkes på "Annoncer"	Brugeren kommer ind på en oversigt over alle annoncer		
Punkt 2	Hjemmesiden bliver tilgået af 10 brugere samtidig	Der sker ikke nogen ændring i funktionaliteten for de enkelte brugere på hjemmesiden		
Punkt 3	Der skrives "Stol" i søgefeltet og der trykkes på søg, når der trykkes på søg startes et stop-ur. Uret stoppes når Søgeresultaterne bliver vist på skærmen	Stop-uret viser mindre end 15 sekunder		
Punkt 4	En computer tilkoblet et andet netværk end hjemmesidens server, går til hjemmesidens url-adresse http://10.29.0.30/bargainbarter via en Chrome Browseren	Browseren åbner forsiden til systemets hjemmeside		
Punkt 5	Der søges efter annoncer inden for 10km	Den annonce, der ligger længst væk fra søgelokationen er mindre end 10100m		
Punkt 6	En bruger skriver "Hej" til en anden bruger gennem det indbyggede chat-modul. Når der trykkes på send startes et stop-ur og uret stoppes, når den anden kan se beskeden	Stop-uret står på mindre end 15 sekunder		
Punkt 7	Skriv en brugervurdering på 500 tegn og prøver at skrive videre	Brugeren kan nu ikke tilføje flere tegn		
Punkt 8	Skriv en brugervurdering på under 500 tegn og tryk "Send". Når der trykkes på "Send" startes et stop-ur og uret stoppes, når brugeranmeldelsen offentliggøres	Stop-uret viser mindre end 15 sekunder		

Tabel 5.1: Accepttestspezifikation af ikke-funktionelle krav del 1

Krav	Beskrivelse	Forventet resultat	Resultat	✓ / X
Punkt 9	Tilgå hjemmesiden fra en af de afgivne enheder i afsnit 4 via http://10.29.0.30/bargainbarter , log ind og opret en BarterAd	En BarterAd er blevet opret i systemet		

Tabel 5.2: Accepttestspezifikation af ikke-funktionelle krav del 2

6. Systembeskrivelse

Systemet BargainBarter er en Webapplikation, der hostes på <http://10.29.0.30/BargainBarter> på en af AU's servere på deres netværk. Denne hjemmeside tilbyder besøgende at oprette sig som bruger, oprette og bytteannoncer og chatte med andre brugere af systemet. Kernefunktionaliteten af BargainBarter er at kunne oprette en bytte annonce, med det formål at bytte den med en andens brugers ejendel. Når annoncen er oprettet kan den af andre brugere ses i det samlede overblik over annoncer på forsiden. På forsiden kan der klikkes ind på annoncen for nærmere/flere detaljer. Systemet skal igennem annoncen præsentere brugeren for annnoncens lokation, brugerens rating og evt. kommentarer til annoncen. Systemet skal have et indbygget chat-modul, så brugere kan kommunikere sammen i real-tid om bytthandler mm.

En rigt billede, der illustrerer systemets primære funktionalitet og formål kan ses på figur 6.1 hvor et simpel byttescenarie demonstreres.

**Figur 6.1:** Rigt billede

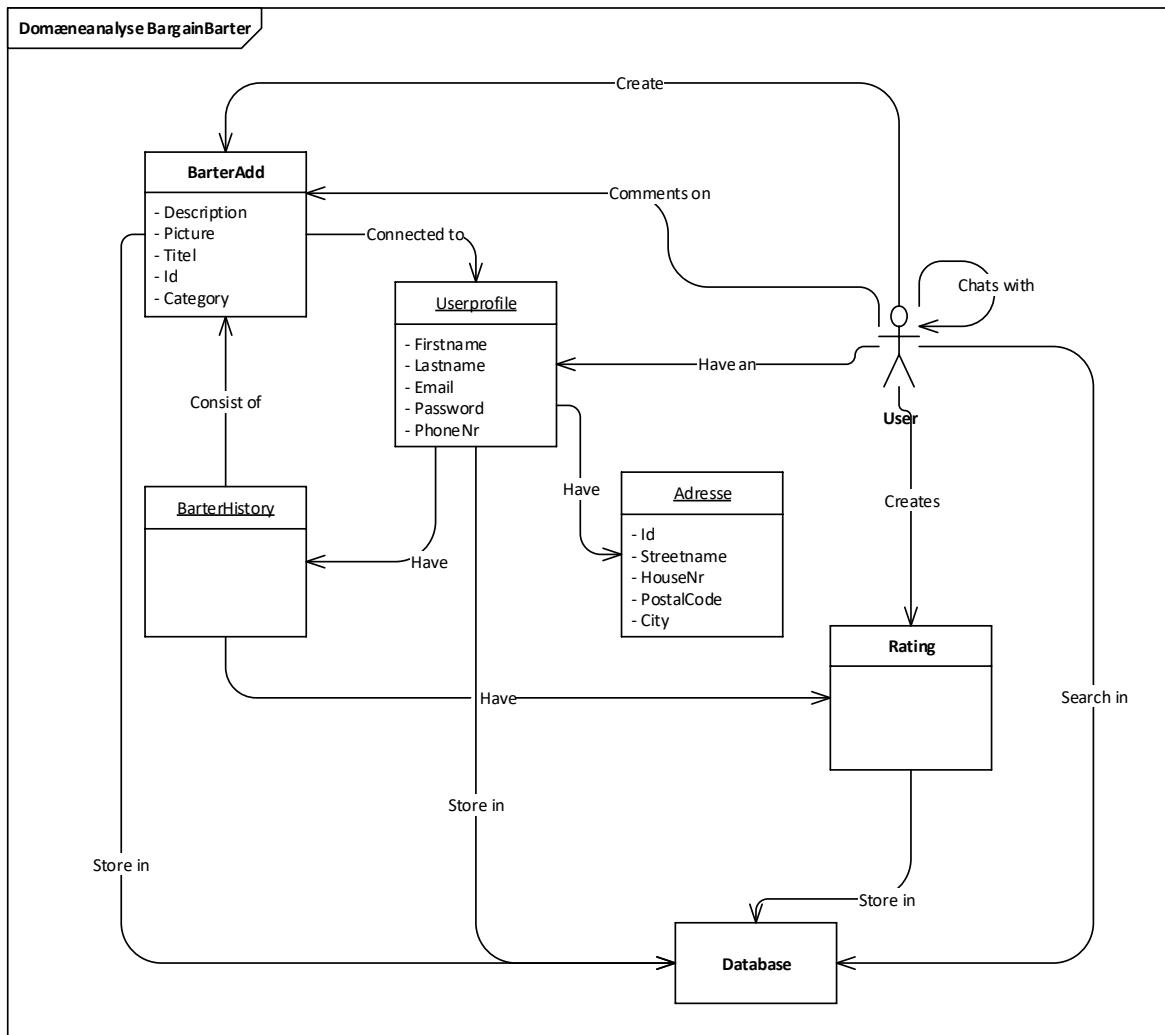
På figur 6.1 ses det, at bytthandlerne enten bekræftes eller afvises direkte i systemet. De fysiske bytthandler foregår dog uden om systemet, og brugeren skal derfor selv udføre handlen for at få værdi af systemet.

7. System arkitektur

I de følgende afsnit er den overordnede arkitektur for BargainBarter beskrevet. Afsnittet starter med en domæneanalyse, der overordnet viser, hvilke konceptuelle klasser web-applikationen skal bestå af, og hvordan disse er forbundet. Efterfølgende er der udviklet en række af sekvensdiagrammer, der viser det grundlæggende forløb/flow af forskellige essentielle scenerier/user stories.

7.1 Domæneanalyse

Der er udarbejdet en domæneanalyse af systemet som ses på figur 7.1. Analysen viser hvordan systemet "BargainBarter" er opbygget, og hvilke relationer der er imellem de forskellige konceptuelle klasser.

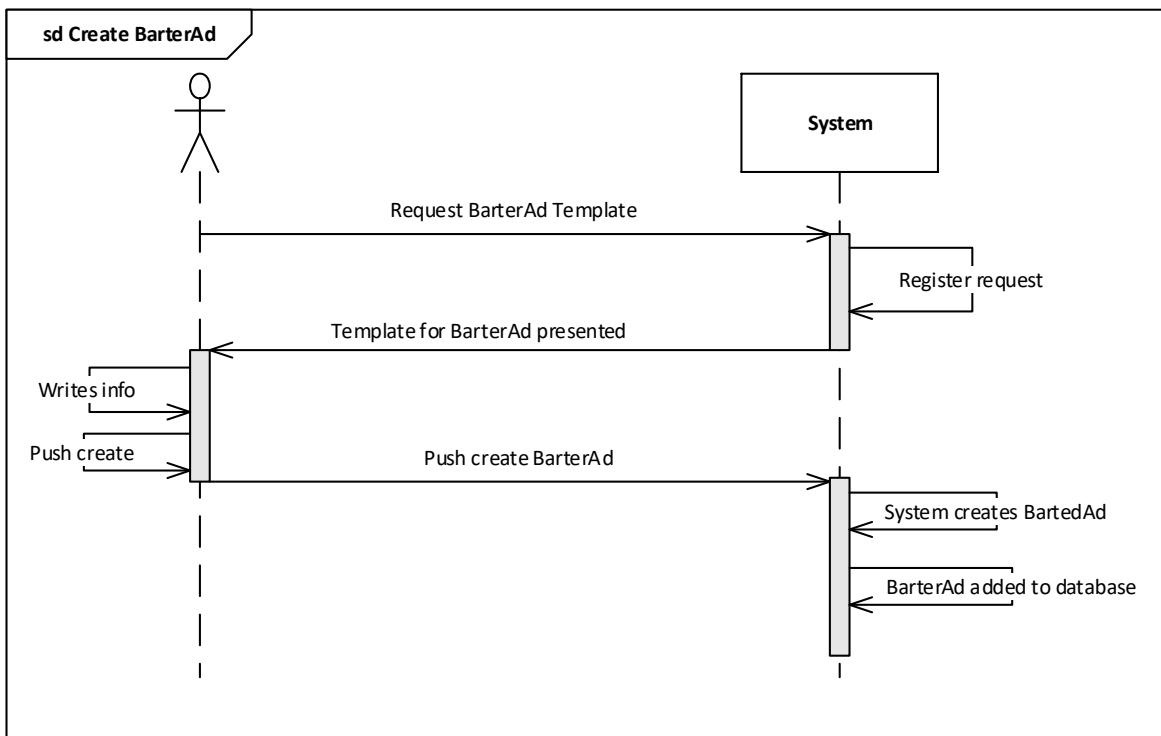


Figur 7.1: Domæneanalyse for BargainBarter

Dette diagram er godt til at illustrere bestanddelene i systemet. Det kan eksempelvis ses, at brugere har profiler, som har et tilhørende sæt af BarterAds. Diagrammet er essentiel ift. udvælgelse af data, der ønskes persisteres/gemmes i databasen. Dette beskrives senere i design og implementering

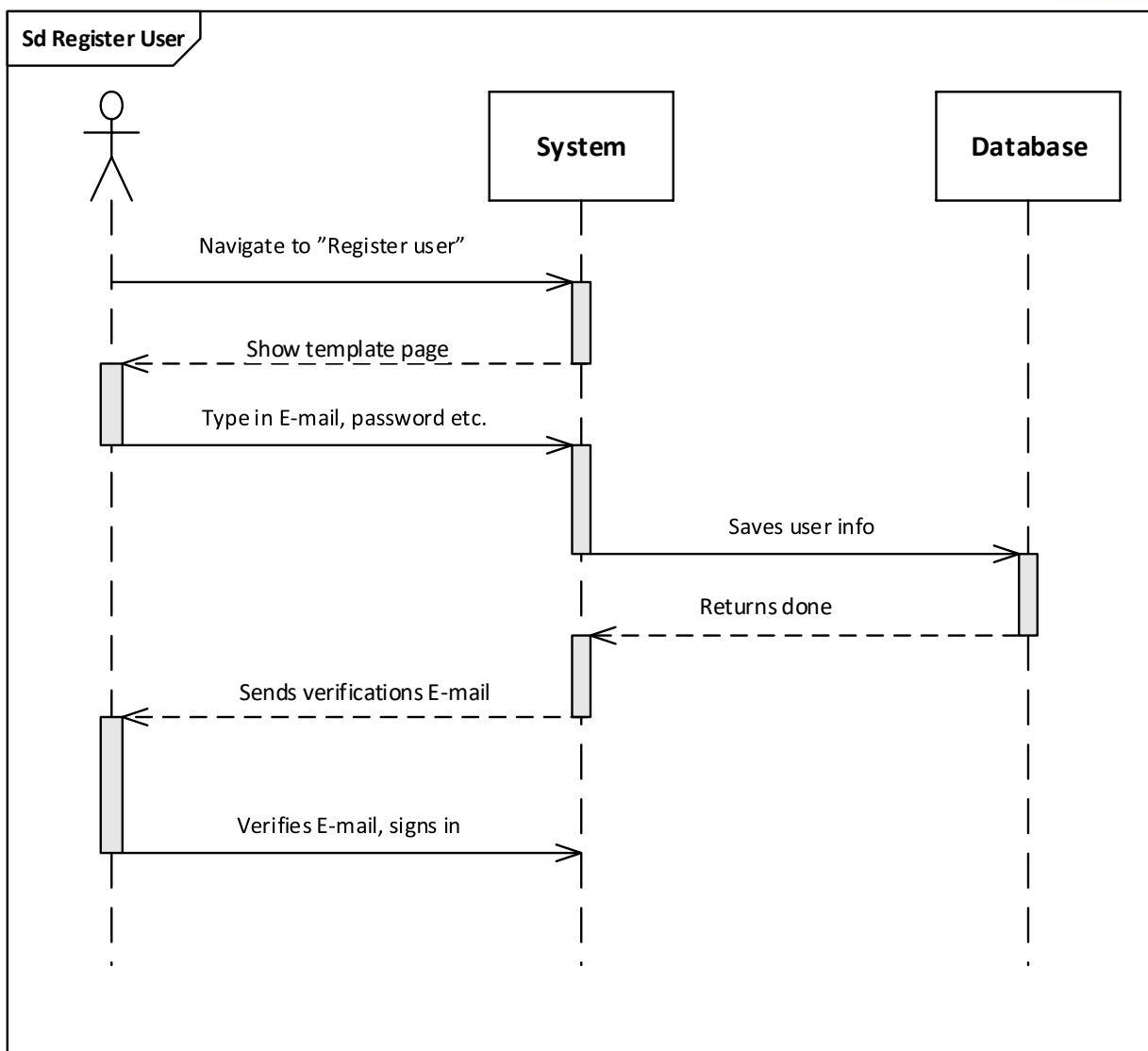
7.2 Overordnede sekvensdiagrammer

Der er lavet enkelte system sekvensdiagrammer, for udvalgte user stories, der har særlig signifikans for systemmet. Figur 7.2 viser hvordan brugeren igennem systemet for oprettet en bytte annonce, på figur 7.3 ses hvordan brugeren kan oprette en brugerprofil og på figur 7.4 vises søgefunktionaliteten.



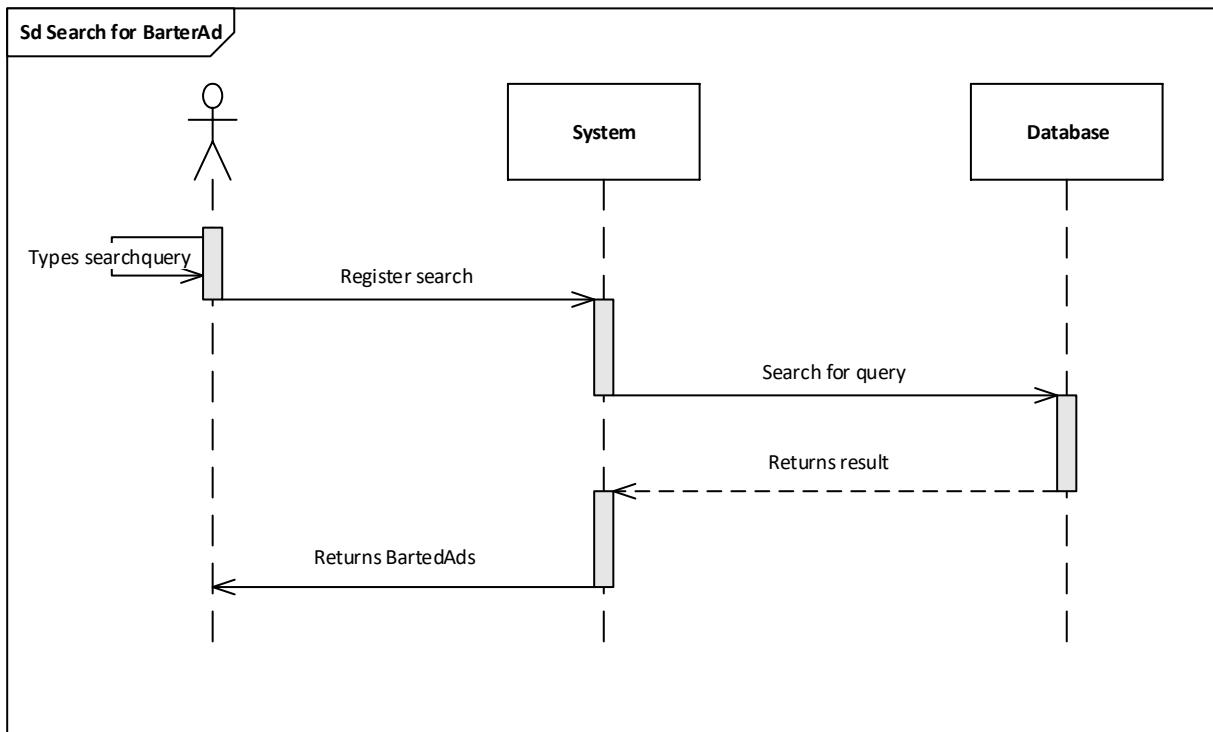
Figur 7.2: Opretbytteannonce, System sekvensdiagram

På figur 7.2 vises der hvordan en bruger skal oprette en annonce (BarterAd). Brugeren trykker på en knap "Opret bytteannonce" på hjemmesiden. Systemet registrerer dette ved så at præsentere en formular, hvorved brugeren kan indtaste relevante oplysninger. Når brugeren så trykker "Opret", så registrerer systemet og opretter derefter en annonce, som bliver lagt på en database.



Figur 7.3: Opretbrugerprofildia, System sekvensdiagram

På figur 7.3 vises der hvordan en nu bruger skal registrere sig som ny bruger på hjemmesiden. Brugeren nавигerer til menupunktet "Registrer bruger" systemet registrere dette og viser en formular for brugeren, hvor der kan indtastes relevante informationer. Informationen gemmes ned p\u00e5 en database og systemet sender en verifikations E-mail til brugeren. Brugeren verificere sin E-mail og kan nu logge ind.



Figur 7.4: Opretbytteannonce, System sekvensdiagram

På figur 7.4 vises der, hvordan en bruger søger efter noget bestemt. Brugeren indtaster et søgeord, som så sendes til systemet. Systemet søger så i databasen, hvor den så returner et resultat til systemet. Systemet præsenterer så de BarterAds, der passer til brugersøgeord.

8. Software-arkitektur

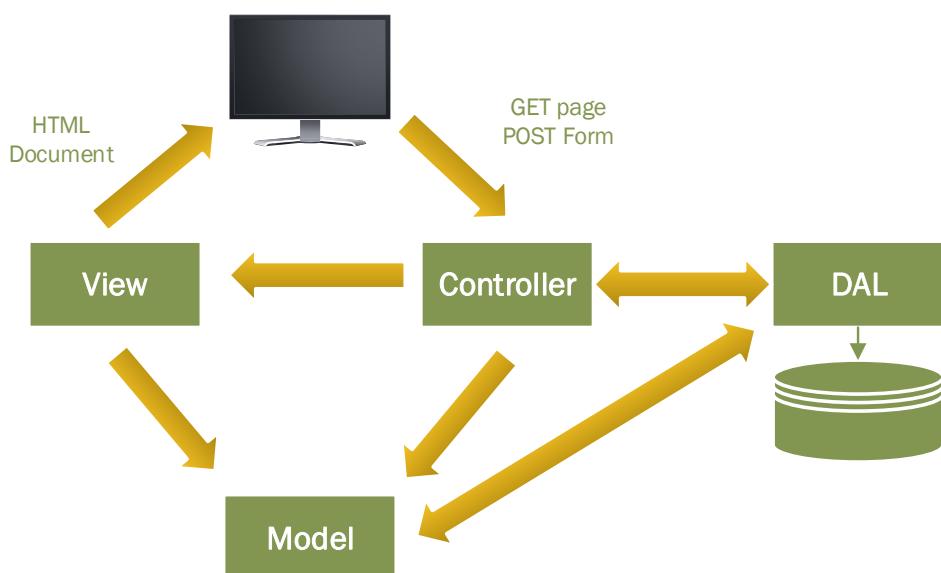
I følgende afsnit redegøres for det systemets arkitektur og de anvendte patterns sammen med en begrundelse for hvorfor de er valgt.

8.1 MVC

Model-View-Controller[9] er et GUI-pattern, der forsøger at adskille den grafiske brugergrænseflade fra resten af systemet. Denne adskillelse har en række af fordele. De væsentligste fordele er, at applikationen bliver mere testbar, skalerbar samt nemmere at vedligeholde. I projektet er der anvendt ASP.NET[4] MVC, hvilket er en afart af den rene form for MVC. ASP.NET MVC består af tre dele:

- Model: er den del der arbejder med den data relaterede logik, dvs. Data Acces Layer. Det er den data der tages fra databasen og bliver manipuleret af Controllers.
- View: er den grafiske del af programmet og indeholder præsentationslogikken. Denne del skal være afkoblet fra modellen.
- Controller: er grænsefladen mellem modellen og viewet. Controllerens opgave er at manipulere data fra modellen og interagere med Views for at vise outputtet og modtage inputs fra viewet

På figur 8.1 kan der ses den overordnet struktur for, hvordan de tre dele af MVC kommunikere indbyrdes.



Figur 8.1: MVC struktur

MVC blev valgt på baggrund af, at det er en integreret del af ASP.NET MVC. ASP.NET MVC er sat op, så applikationens controllere og deres forskellige actions bliver kaldt gennem url'en. Dette bliver sat op i RouteConfig.cs, hvor der som standard er følgende opsætning:

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

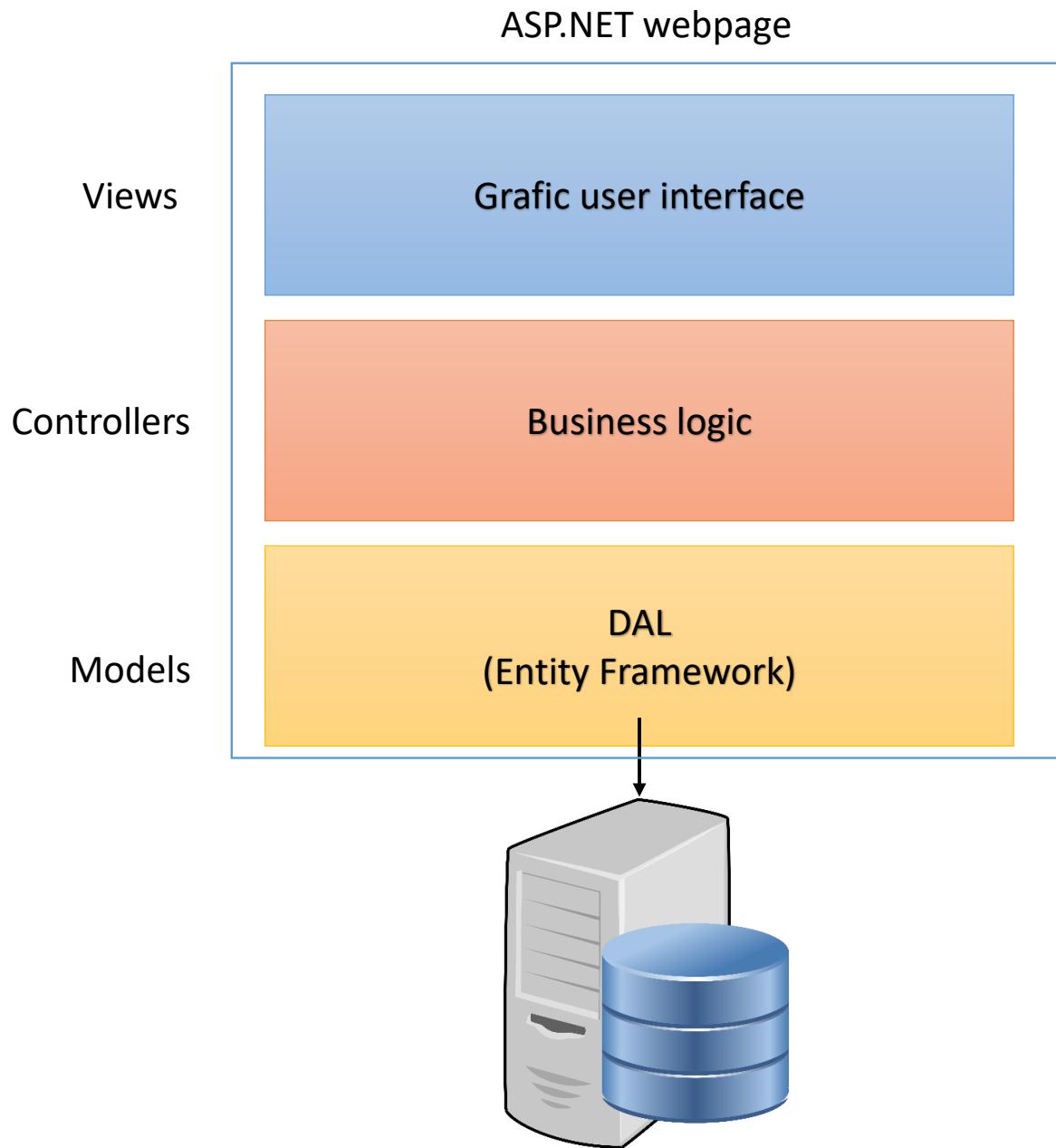
        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
                UrlParameter.Optional })
    }
}
```

Koden oven over viser, hvordan ASP.NET MVC router mellem de forskellige sider på baggrund af url'en. Det første argument i url'en router til en controller. Det efterfølgende argument er navnet på actionen, der skal kaldes i den valgte controller, hvis dette ikke er angivet kaldes "Index"-metoden som default. Det sidste sidste argument er parametrene actionen skal kaldes med. Der er opsat en række af default argumenter, som kaldes, hvis der ikke er angivet nogen. Default argumenterne er følgende:

"defaults: new controller = "Home", action = "Index", id =UrlParameter.Optional ".

8.2 3-lags-modellen

Systemet bygger på trelags-modellen [1], som kan ses i figur 8.2.



Figur 8.2: Lagmodel for systemet

3-lags modellen forsøger som MVC at adskille præsentations-, business- og data-access logikken fra hinanden. 3-lags modellen foreskriver, at disse lag skal være fysisk adskilte fra hinanden. Fordelene ved den fysiske adskille baserer sig primært på "separation of concerns" og "Single Responsibility", hvilket medfører et skalerbart og vedligeholdelses venligt system. ASP.NET MVC baseres på 3-lags modellen, hvilket gør, at systemets arkitektur lægger sig tæt op af 3-lags modellen. Systemets views er præsentationslogikken, controllers og modellen er business-logikken og entity framework samt modellen er systemets Data access lag.

8.2.1 Præsentationslaget

For web-applikationen generelt, og for de opstillede ikke-funktionelle krav til systemet, er brugervenlighed en væsentlig faktor. Dette betyder at præsentationslogikken er ekstremt vigtig for systemet. Da det skal være let for en bruger at navigere rundt på siden og kunne forstå, hvad meningen er med de forskellige handlinger der kan foretages.

Da projektet er en webapplikation præsenteres den grundlæggende i HTML[16] i browseren. HTML'en bliver vha. CSS[15] og Javascript[17] stylet for at opnå et mere overskueligt design. I forbindelse med styling af views er front-end frameworkt Bootstrap[13] anvendt. Bootstrap er blevet anvendt for at give hjemmesiden et mere responsivt design. Således at hjemmesiden kan benyttes fra flere forskellige typer af enheder (Smartphones, PC mm.), der var et af de ikke-funktionelle krav i afsnit 4. Bootstrap er et framework, der arbejder med relative værdier for skærmstørrelsen, således at layoutet skalere og relativt fylder det samme på forskellige typer af enheder/skærmstørrelser.

I praksis bliver præsentationslaget i ASP.NET returneret af controllerne til brugeren, der kan se dette i browseren.

8.2.2 Businesslag

Business-laget i projektet er i høj grad controllerne. Der ligger selvfølgelig også nogle forretnings-specifikke valg i modellen. Forretningslogikken ligger derfor som udgangspunkt i controllerne, hvor der er forsøgt at afkoble så meget som muligt af logikken fra præsentationslaget. Dette er også en fordel, da controlleren har lettere ved at blive testet i modsætningen til præsentationslaget. I ASP.NET MVC sender man ofte data til viewet fra controlleren som et objekt. Dette gør, at man kan arbejde parallelt på controller og view. Da der i objektet ligger et fast defineret interface. Derfor går en del af forretningslogikken udpå at fremstille eller samle et objekt, der har et klart interface til at kunne blive anvendt i viewet.

8.2.3 Database lag

I BargainBarter anvendes ADO.NET Entity Framework[11] som det nederste data access layer. Entity Framework er et framework, der fjerner det impedans mismatch, der ofte opstår mellem databasen og objektmodellen i et projekt. Entity Framework er valgt, da man vha. en objekt orienteret tankegang kan operere på en database. Man slipper derfor for en del bekymringer og ekstraarbejde i form af triviell kode, der skulle skrives i forbindelse med mapningen af database-objekter til kode-objekter. I brugen af Entity Framework har LINQ [7] været et godt hjælpemiddel til at operere på databasen.

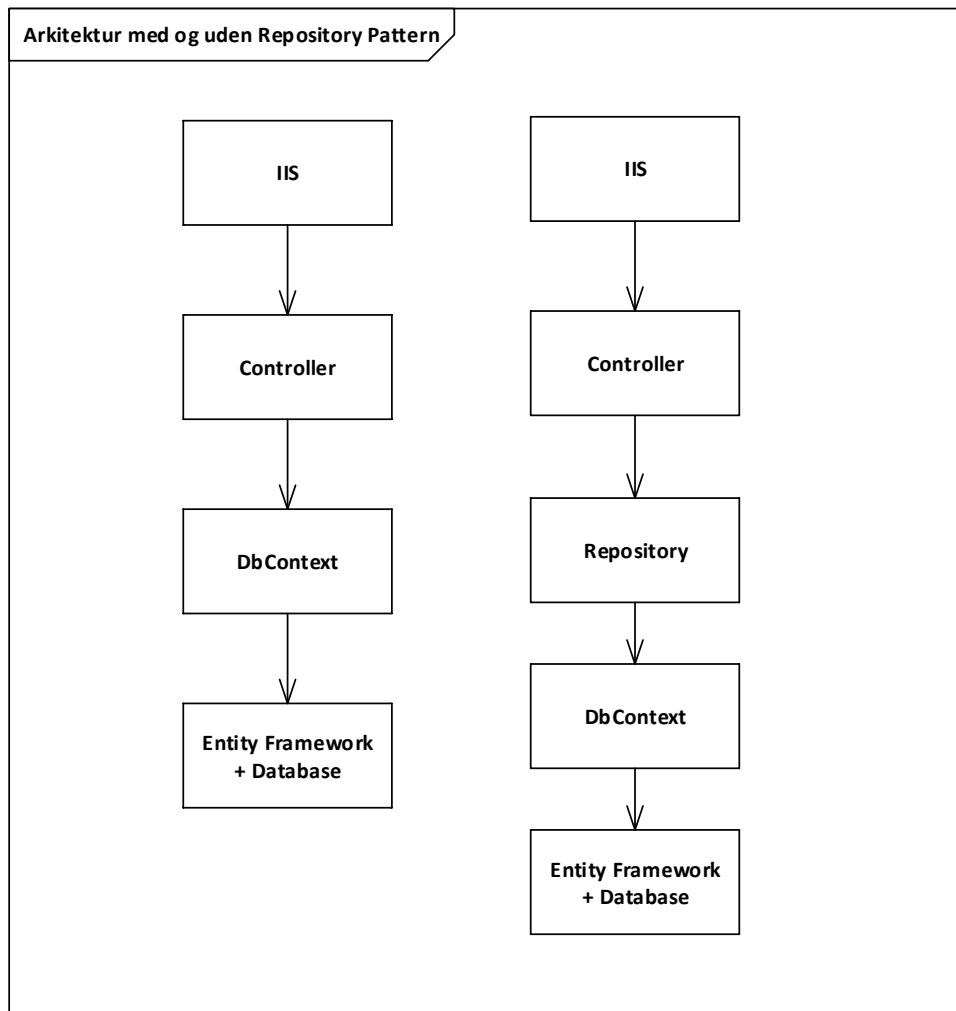
Entity Framework var desuden oplagt, da der i projektet har været stillet en Microsoft SQL Server Database til rådighed af AU. Uddover dette arbejder Entity Framework også godt sammen med ASP.Net MVC frameworkt.

I brugen af Entity Framework, anvendes en Code First New Database[5] approach. På denne måde bliver databasen genereret på baggrund af objektmodellen man opstiller i projektet. Man kan via migrations udvidde objektmodellen og dermed databasen løbende i projektet. Migrations gemmer også historikken over ændringerne på databasen. Dette approach er derfor meget fleksibel, idet man altid kan rulle databasen tilbage til et tidligere stadie. Da man uddelegerer mapningen mellem objektmodellen og entitetsmodellen til et framework, med nogle meget strikte regler kommer disse to forskellige modeller til at ligne hinanden og interfacet fra projektet til databasen bliver derfor meget homogent og let anvendeligt. Gruppen har desuden modtaget undervisning i brug af frameworkt i faget I4DAB.

Dette er nogle af grundene til at beslutningen om at anvende Entity Framework. Denne beslutning blev truffet tidligt i projektet, og har i høj grad har indflydelse på arkitekturen for systemet.

8.3 Repository Pattern

Repository pattern er et software mønster [12], der indfører et ekstra lag (Repository-lag) mellem databasen og business-logikken. Det virker som data-access lag mellem business-logikken og det oprindelige data-access lag - i dette tilfælde Entity Framework. Dette ekstra lag medfører, at business-logikken ikke skriver direkte ned i databasen. Dette sænker i høj grad koblingen i systemet, da business-laget ikke er afhængig af databasen. Business-laget kalder bare ned i repository-laget, der så sørger for selve transaktionen med databasen. På figur 9.10 kan man se, hvor og hvordan det ekstra lag indføres i arkitekturen.



Figur 8.3: Respository Patterns betydning for arkitekturen

Repository pattern er blevet anvendt for at opnå en lavere kobling i systemet mellem databasen

og business-logikken. Den lavere kobling i systemet, gør systemet mere robust for ændringer. Systemet bliver også samtidig mere testbart, da det på denne måde kan benytte dependency inversion, hvor ved de forskellige database afhængigheder kan mockes ud i forbindelse med testningen af systemet.

8.4 Unit Of Work

Unit of Work er et design pattern, der holder styr på in-memory opdateringer og skriver disse opdateringer til databasen. Dette er dermed den eneste klasse, der taler sammen med databasen. Dette er blevet opnået ved at, at Unit Of Work-klassen indeholder en række GenericRepository's af alle de forskellige model-klasser. Det er Unit Of Work, der vedligeholder disse repositories. Unit Of Work er den eneste, der tilføjer eller sletter i databasen. Det er dermed Unit Of Work, der laver de gængse database-transaktioner og sørge for at databasen bliver opdateret, når en transaktion er afsluttet.

8.5 Valg af Teknologier

8.5.1 ASP.NET

ASP.NET er et udviklings framework for web-applikationer. Det er en del af .NET Frameworket, der gør at al kode, der er kompatibel med .NET og CLR også er kompatibelt i ASP.NET. Dette har været en af de væsentligste grunde til, at ASP.NET er blevet benyttet, da det har gjort at projektet kunne skrives i C#, der er blevet undervist i på semesterets andre kurser. Gruppen har desuden også modtaget undervisning i ASP.NET i forbindelse med faget I4GUI.

8.5.2 SignalR

SignalR[8] er et software-bibliotek til ASP.NET miljøet. Biblioteket giver adgang til at benytte real-time funktioner på det website man udvikler på. Det kan benyttes, når man vil have en form for interaktion med brugerne, eller mellem brugere.

Det forgår i browseren ved brug af javascript-kode, men der kan stadig laves kald tilbage til serveren. SignalR er blevet anvendt for til at implementere systemets real-time chat.

8.6 Generelle overvejelser i forbindelse med Arkitektur

I forbindelse med projektet har der været nogle generelle overvejelser, omkring den arkitektur produktet skulle have. Et af de vigtigste key issues har været brugerkvaliteten. Dvs. at brugervenligheden og skalerbarheden til forskellige apparater, har skullet være i top for at produktet har kunne give værdi. Dette er kommet til livs med at anvende frameworket Bootstrap, som er en stor hjælp til at lave pæne designs, der skalerer til forskellige devicestørrelser, uden alt for meget arbejde.

Et andet kriterie der er tænkt ind i arkitekturen er performance. Dette kan bl.a. ses ved implementeringen af chatteren. I chat controlleren er meget af logikken implementeret client side vha. javascript. Dette giver en bedre brugeroplevelse da opdateringerne sker øjeblikkeligt i chatteren, da der kan spares nogle kald til serveren.

Af cross cutting concerns i projektet har der bl.a. været verificering af en brugers log-in på de forskellige sider. Dvs. at visse dele af hjemmesiden ikke skulle være tilgængelig, med mindre

brugeren var logget ind. Til at løse dette problem er ASP.NET identity[6] blevet brugt og udvidet, med de oplysninger der har været nødvendige for at projektet har passet på domænet.

En anden cross cutting concern er validering af oplysninger når en brugerprofil oprettes. I projektet er det valgt at brugeren skal oplyse en adresse, fulde navn, email, password og telefonnummer. Adresse og kontaktinformationer er valgt, da disse oplysninger alle er essentielle for at en byttehandel kan finde sted i virkeligheden. Passwordet er valgt til ikke at have særligt strikse krav, idet oplysningerne ikke er særligt kritiske at miste. Det er derfor vigtigere at brugerens indgangsbarriere er minimal, og at der ikke at skulle bruges for lang tid på at finde på et password, som han/hun alligevel ikke kan huske bagefter.

9. Design og Implementering

Systemet BargainBarter er designet ud fra de krav og analyser der er beskrevet i de ovenstående sektioner. Da web applikation er lavet i ASP.NET MVC beskrives i denne sektion de forskellige dele af MVC'en, hvor der redegøres fo: controllere, views og modeller.

9.1 Controllere

Der er lavet en overordnet rollefordeling af controllernes ansvar:

9.1.1 Home

Denne controller er ansvarlig for at vise de generelle ting for hjemmesiden. Dette vil sige hovedsiden, kontakt og en About section.

Actions

- Index - Denne action returnere et view indeholdene alle barterads i databasen.
- About - Returnere det view der hedder about med information om den BargainBarter generelt.
- Contact - Action der returnere et view med contact info omkring BB.

9.1.2 BarterAd

Denne controller er overordnet ansvarlig for, at brugeren kan oprette og opdatere Barteradds annoncer på hjemmesiden. Sammenfattet: CRUD operationerne på BarterAds. Ud over dette kan denne controller også hjælpe med at vise enkelte Barteradds.

Actions:

- Index - Returne blot til Home controllerens index, dette er fordi routeconfigen sætter index som standard, så man skal ikke kunne kalde barterads controllerens index.
- ShowBarterAdsOnMap - Viser et kort med alle annoncer.
- Viewphoto - Returnere et billede
- Details - Via denne action kan brugeren indspicere en anden brugers BarterAd. I det returnede view kan det meste info fra barterad modellen findes.
- DetailsOwn - Viser en af dine egne barterads
- Create - returnere viewet med mulighed for at CreateBarterAd
- Create(BarterAd) - brugeren opretter annoncer. Det vil sige, at brugeren kan indtaste Annoncenavn, kategori, beskrivelse og uploadet et billede til en annonce.
- Edit(BarterAd) - via denne action kan brugeren opdatere en af sine bytteannoncer.

- Comment - Denne action giver mulighed for at tilægge en kommentar til en barterad, og gemmer den i databasen
- Delete - Returnere delete viewet, der så giver mulighed for at slette ad'et
- DeleteConfirmed - Denne action kan slette barterads fra databasen. Naturligvis kun sine egne.
- Dispose - Lukker databasen
- ManageAds - Returnere et view med alle dine egne ads, og mulighed for at redigere dem. Eller et andet view hvis brugerne ikke har nogen barterads.
- RequestTrade - Giver mulighed for at kunne anmode om en byttehandel med en anden bruger, hvor du tilknytter en af dine egne ads.
- DeclineTrade - Mulighed for at afvise en byttehandel.
- AcceptTrade - Mulighed for at acceptere en byttehandel
- ShowTrades - Gennem actionen vises de aktuelle Trades
- ShowHistory - Hvis historie over tidligere byttehandler.
- ShowTheirTradeHistory - Viser TradeHistory for en anden bruger
- Edit - Returnere edit viewet
- GiveRating - Returnerer et view med de barterads som indgår i byttehandlen, som skal rates.
- ConfirmRating - Lader brugerne bekræfte sin rating, som indeholder en værdi og eventuelt en kommentar, som gemmes i databasen.
- ShowNearest - Brugerne har mulighed for at sætte en ønsket afstand til annoncer. Returnerer et view med alle annoncer inden for given afstand.
- ShowNeedRating - Viser de byttehandler brugerne mangler at vurdere.

9.1.3 Search

Search-controlleren er ansvarlig for søgning i annoncerne.

Actions

- Index(searchstring) skal tage en tekststreng og søge i annoncerne efter match og vise en side med de matchene annoncer.
- CategorySearch(searchstring) - Skal det samme som Index, med den udvidelse at det også søger på katagori

9.1.4 UserProfile

Denne controller har til ansvar at styre ansvar omkring brugerprofiler. Dette er bl.a. at vise brugerprofiler og rette i brugerprofiler.

- Index Redirector blot til forsiden. Denne giver ikke mening, da der som regel skal være et brugerId for at vide hvilken brugerprofil det drejer sig om, men den er her i tilfældet.
- Edit(id) hvis brugerprofilen er den samme som den bruger man tilgår siden med, får man lov at rette i profilen, ellers returneres blot et view hvor man kan se brugerprofilen.
- Edit(postedUser) De rettede oplysninger i brugerprofilen skrives ned i databasen. Brugerprofilen bliver herefter vist.
- ShowUserProfile(id) sørger for at trække data udfra databasen og sende et pænt View til siden der viser de oplysninger en brugerprofil indeholder. Hvis profilen er den samme som den bruger der tilgår den, får brugeren mulighed for at rette i oplysninger vha. en redigerknap.

9.1.5 Chat

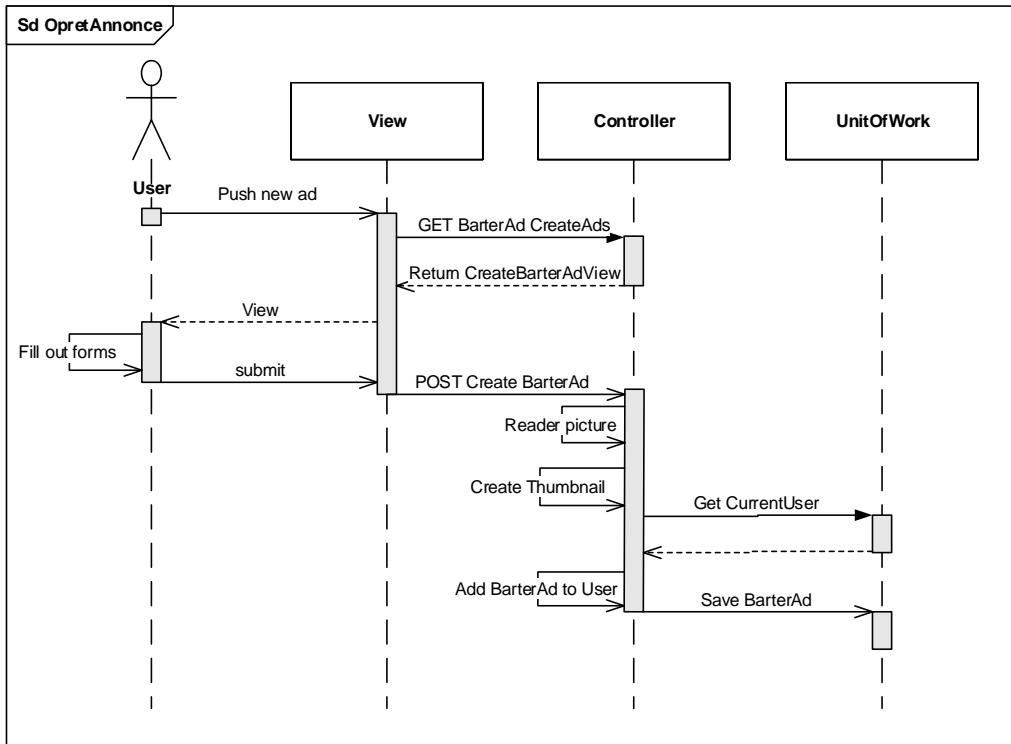
Har en index action der returnerer et view.

9.2 Generel controller-funktionalitet

Generelt er der anvendt forskellige standardmetoder igennem mange af controllerne. Til data acces kalder alle controllere igennem et Unit of Work pattern, som beskrives i sektion 9.8. Til at finde den aktuelle bruger i systemet bruges HttpContext hvor brugeren findes på ID. Controllerne holder selv styr på at kalde fejl hvis de bliver kaldt med ugyldige parametre. Da dette er et meget ensartet mønster igennem alle controllers eksemplificeres dette blot ved en enkelt controller.

9.2.1 Eksemplificering af controller-funktionalitet

Denne sektion omhandler funktionaliteten af en controller, og hvordan den fungerer. Dette er eksemplificeret igennem User storien: Opret en bytteannonce, og tager udgangspunkt i controllerens action til oprettelse af bytteannoncer. Som det kan ses på figur 9.1 bliver actionen kaldt fra viewet. Viewet viser kun denne mulighed hvis brugeren er autentificeret, hvilket viewet, selv holder styr på. Når UI elementet bliver trykket, kalder det et Http request og serveren returnere et view til oprettelsen af BarterAds. Viewet indeholder en række forms hvor informationen om barterads kan indsættes. Denne gruppe af forms gives som parametre til aktionen når brugeren submitter. Under dette submit laver klienten et http POST til serveren, der kalder Create barterad actionen. Actionen i controlleren læser billedet brugeren har uploadet ind i et bytearray, og laver et thumbnail ud fra billedet. Igennem HttpContext findes den aktuelle bruger, og igennem Unit of Work, gemmes annoncen i databasen på brugeren. Selve actionen returnerer et redirect til aktionen ManageAds i BarteradsControlleren.

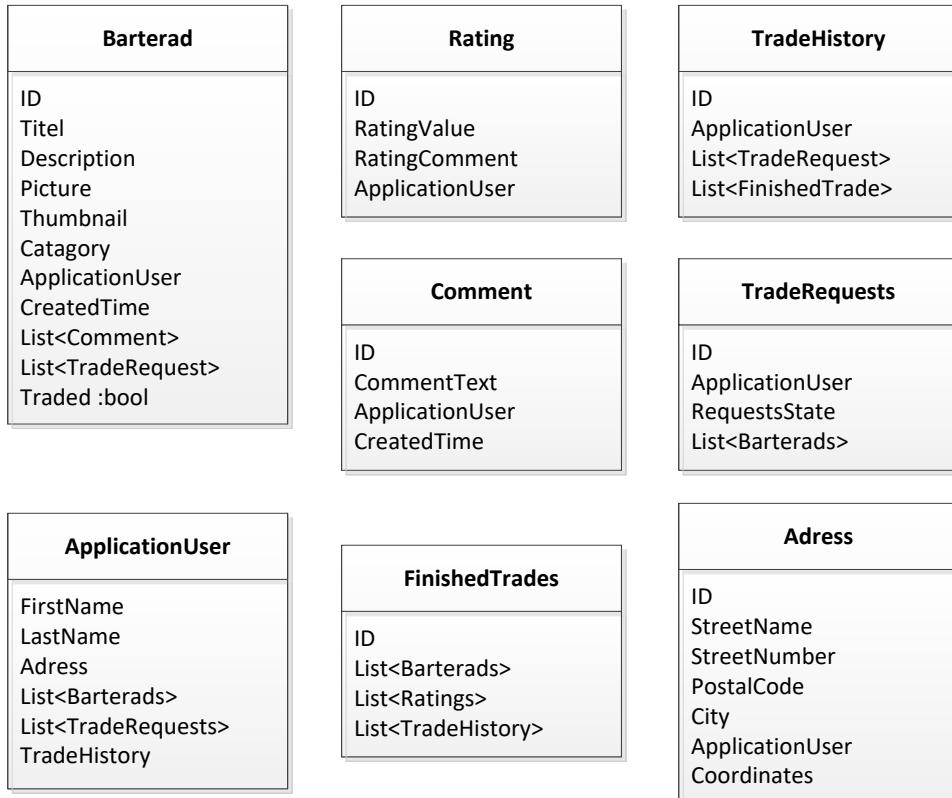


Figur 9.1: Sekvensdiagram for, hvordan barterAds bliver oprettet

Denne beskrivelse er sigende for mange af de interaktioner der foregår på BargainBarter webstedet. De essentielle actions er kaldt igennem UI elementer der laver enten GET eller POST funktioner. Dernæst findes noget data igennem unit of work klassen der evt. behandles, hvorefter der returneres et nyt view.

9.3 Modellen

Den endelige koceptuelle model (med undtagelse af den genererede identitymodel) kan ses på figur. 9.2. Modellen er lavet ud fra domæneanalysen og implementationen af modellen tilgås af controllerne til at hente data.



Figur 9.2: Samlet diagram over alle konceptuelle klasser i BargainBarter

Den implementerede model har nogle begrænsninger i forhold til de definerede krav fra afsnittet omkring "Ikke-funktionelle krav" se afsnit 4. Modellen implementerer på nuværende tidspunkt ikke kravet om, at en kommentar maksimalt kan betå af 500 tegn.

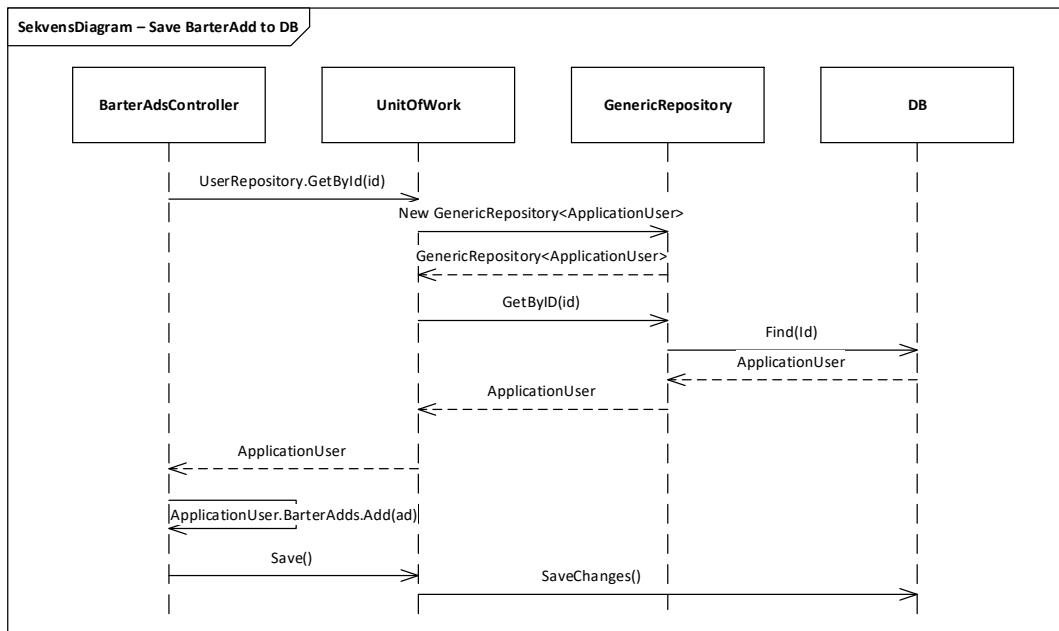
9.4 Data fra Database

Når data fra databasen skal læses, overskrives eller slettes bliver Repository-pattern og UnitOfWork benyttet - mere information omkring disse kan findes i 9.8. I dette afsnit bliver det beskrevet, hvordan implementering af de forskellige CRUD-operationer på databasen er lavet. Da Unit Of Work og Repository-Pattern er blevet benyttet, betyder det, at al tilgang til databasen er stortset ens. Da databasen kun tilgås af UnitOfWork og GenericRepository. Den ens tilgang til databasen betyder, at når data skal hentes i databasen skal der først oprettes et GenericRepository af den benyttede type (f.eks. UserRepository eller BarterAdRepository). Dette genericRepository benyttes som et håndtag til databasen, når der skal hentes data.

Over de følgende sider fremlægges sekvensdiagrammerne for, hvordan CRUD-operationerne er blevet udført for en barterad. Dette er et generelt eksempel og viser, hvordan database-trasaktionerne udføres.

9.4.1 Create

Et sekvensdiagram, for hvordan for eksempel en barterad oprettes og skrives i databasen ved brug af både UnitOfWork og GenericRepository, kan ses på figur 9.3

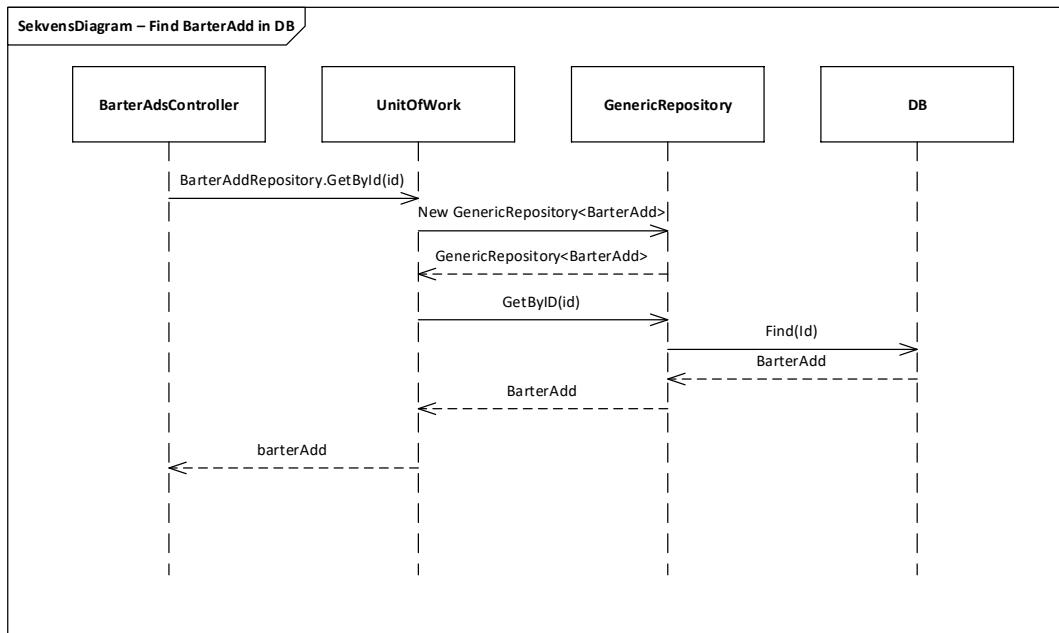


Figur 9.3: Sekvensdiagram for, hvordan en barterAd bliver oprettet og gemt i DB

På sekvensdiagrammet figur 9.3 ses det, hvordan controlleren kalder over i unitofwork, der opretter et nyt genericrepository. Dette genericrepository gemmer barteraden i databasen.

9.4.2 Read

Et sekvensdiagram, for hvordan for eksempel en barterad findes og læses fra databasen ved brug af både UnitOfWork og GenericRepository, kan ses på figur 9.4

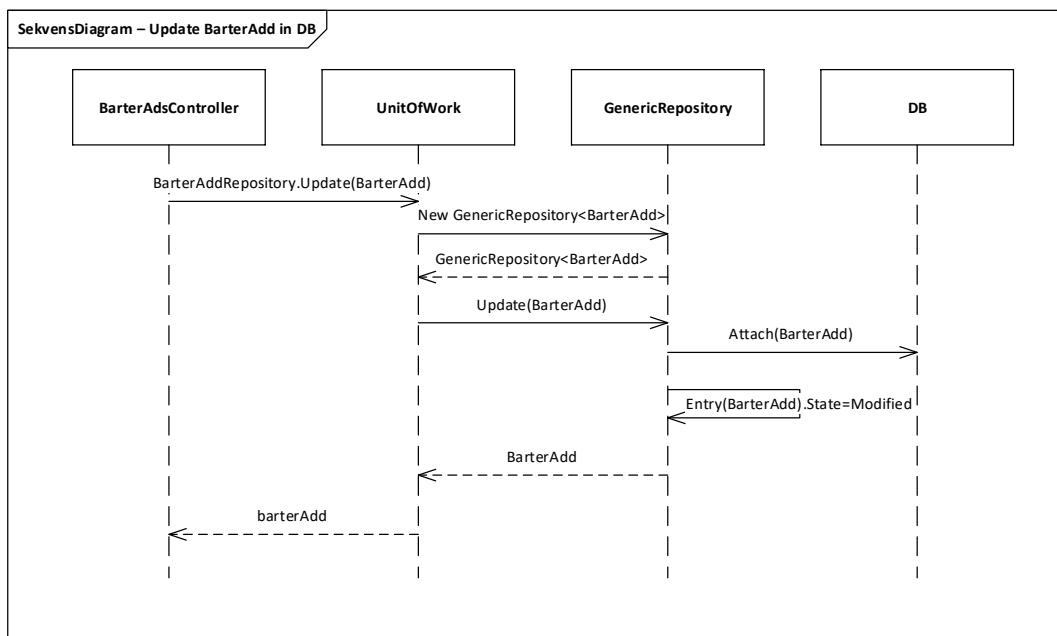


Figur 9.4: Sekvensdiagram for, hvordan en barterAd bliver fundet i DB

På sekvensdiagrammet figur 9.4 ses det, hvordan controlleren kalder over i unitofwork, der opretter et nyt genericrepository. Dette genericrepository finder barteraden i databasen.

9.4.3 Update

Et sekvensdiagram, for hvordan for eksempel en barterad findes og opdateres i databasen ved brug af både UnitOfWork og GenericRepository, kan ses på figur 9.5

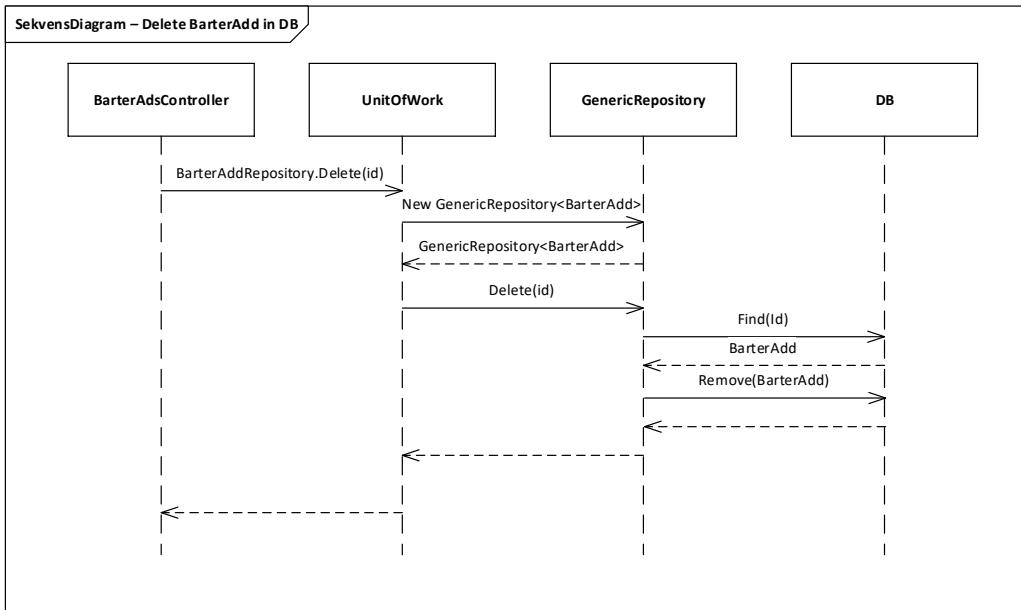


Figur 9.5: Sekvensdiagram for, hvordan en barterAd bliver opdateret i DB

På sekvensdiagrammet figur 9.5 ses det, hvordan controlleren kalder over i unitofwork, der opretter et nyt genericrepository. Dette genericrepository opdatere barteraden i databasen.

9.4.4 Delete

Et sekvensdiagram, for hvordan for eksempel en barterad findes og slettes i databasen ved brug af både UnitOfWork og GenericRepository, kan ses på figur 9.6.



Figur 9.6: Sekvensdiagram for, hvordan en barterAd bliver slettet i DB

På sekvensdiagrammet figur 9.6 ses det, hvordan controlleren kalder over i unitofwork, der opretter et nyt genericrepository. Dette genericrepository sletter barteraden i databasen.

9.5 Chat Room

Der er i projektet blevet lavet et Chat Room, der er udviklet med SignalR pakken fra NuGet. Dette afsnit beskriver hvordan den del af sitet er designet og implementeret. Der i taget udgangspunkt i Microsofts SignalR eksempel. [8]

9.5.1 Chat Controller

Chat controlleren bruger en unit of work til at videregive brugerens navn til chatten, så brugernes navn vil fremgå når brugeren skriver beskeder i chat roomet.

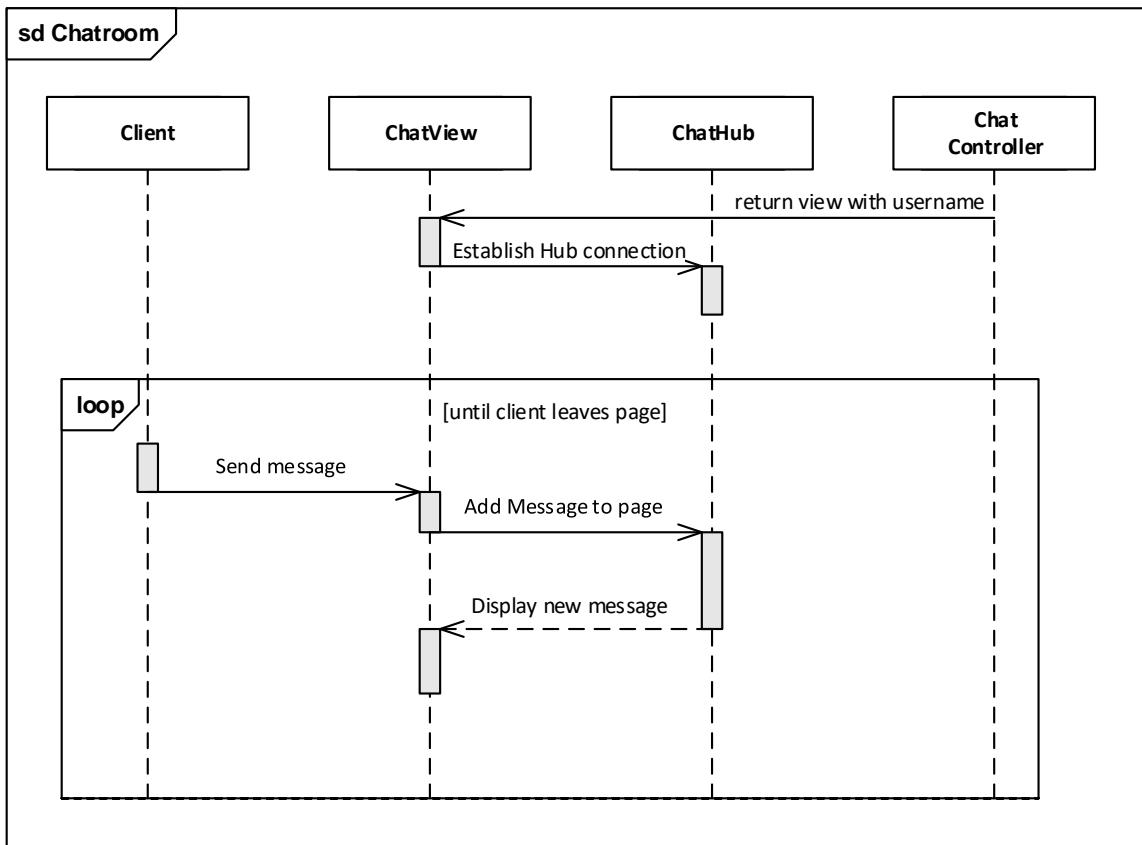
9.5.2 Chat Hub

Chat hubben er den pipeline der kommer til at være imellem client og server. Denne forbindelse gør at SignalR kan kalde tilbage til serveren, med kald direkte i mellem de to. Oprettelsen af denne forbindelse sikrer at brugeren vil opleve ændringerne i real time, da der ikke findes kø ned til serveren. Hubben har desuden funktionen:

```
Clients . All . addNewMessageToPage(name, message);
```

som kalder ned i SignalR pakken. Herefter sørger SignalR for at få denne besked broadcastet til alle clients, i realtime, som sidder på siden.

9.5.3 Sekvensdiagram af Chat room



Figur 9.7: Sekvensdiagram for forløbet af chat afviklingen.

Diagrammet på figur 9.7 viser forløbet når brugeren tilgår chatroomet. Brugerens navn bliver sendt med videre til viewet igennem controlleren gennem Unit of Work, og viewet kan nu bruge navnet i forbindelse med udskrivning af beskeder. Viewet ude ved brugeren opretter derefter en hub connection tilbage til serveren. Brugeren har nu mulighed for at skrive beskeder i chatroomet, så længe han fortsat er på siden.

9.5.4 Chat view

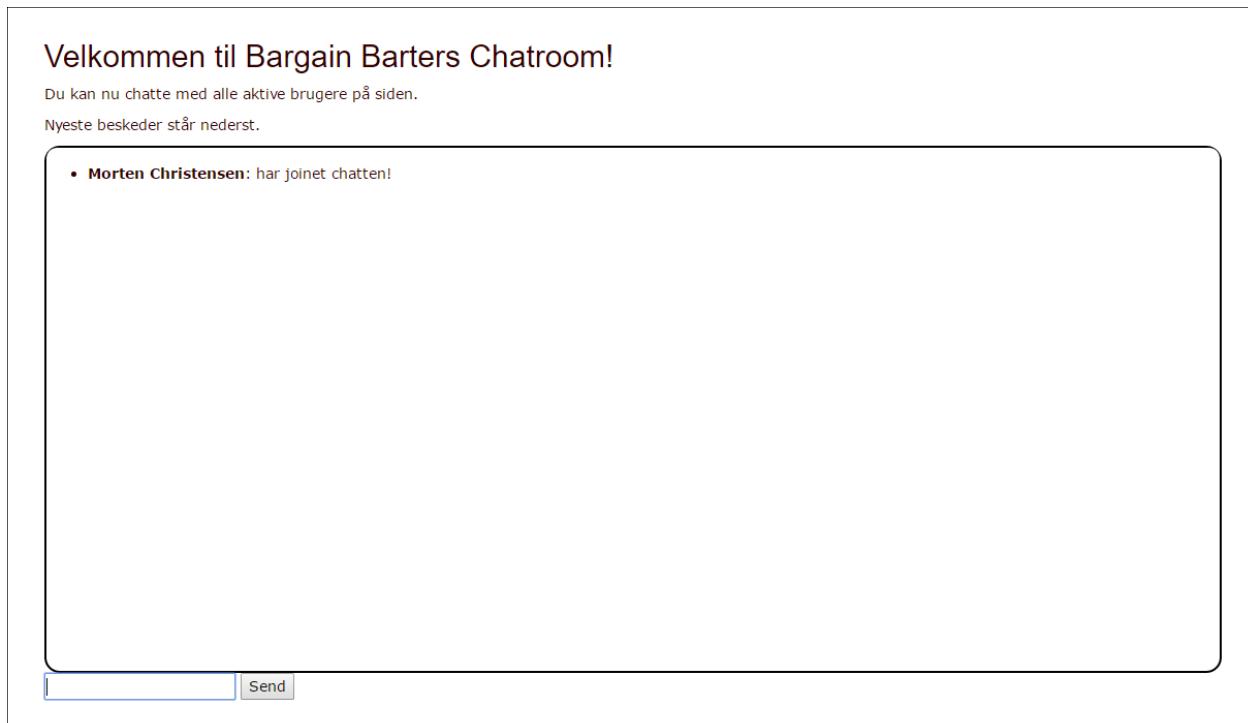
Eftersom det hele skal forgå realtime, bliver der i viewet anvendt javascript for at give brugeren den nødvendige interaktion. Signalr autogenererer et par mapper som holder den nødvendige information i den tid brugeren er på dette view. Viewet binder som det første funktionen fra Hub klassen til den liste, der er container for chat beskederne. Næste punkt er at starte forbindelsen mellem client og server.

```

$.connection.hub.start().done(function () {
if (firsthit === 0) {
chat.server.send($("#displayname").val(), "har joinet chatten!");
firsthit++;
}
  
```

{}

På kodeeksemplet kan vi se at når vi har fået forbindelse til serveren, og dette er gået godt, skrives der i chat boksen at brugeren har tilgået chatteren. Funktionen `chat.server.send` er den samme vi bruger når brugeren skal sende beskeder. I stedet for at der er en foruddefineret værdi, der bliver indsæt som message, er det brugerens eget input fra inputboksen. Efter brugeren har trykket send, sørger `view.setActiveInput` for at sætte inputboksen i fokus, og sætte scrollbaren ned, så man altid kan se det nyeste.



Figur 9.8: Screenshot af det færdige chatroom.

På figur 9.8 ses hvordan chatroommet ser ud på BargainBarter, som det fremvises når man første gang tilgår rummet.

9.6 Databasestruktur

Der er i Barteradds projektet brug for at persistere en del data, og derfor brug for en eller anden form for database, det oplagte valg var at vælge en relationel database, da der er en masse struktur SQL serveren selv holder styr på.

9.6.1 Persistent data

Ud fra domæneanalysen er det klargjort hvad det er for noget data der er nødvendigt at persistere. Der er fundet yderligere data som det er ønsket at gemme i den løbende udvikling, men som ikke er opdateret i den daværende domæneanalyse.

Det valgte data er :

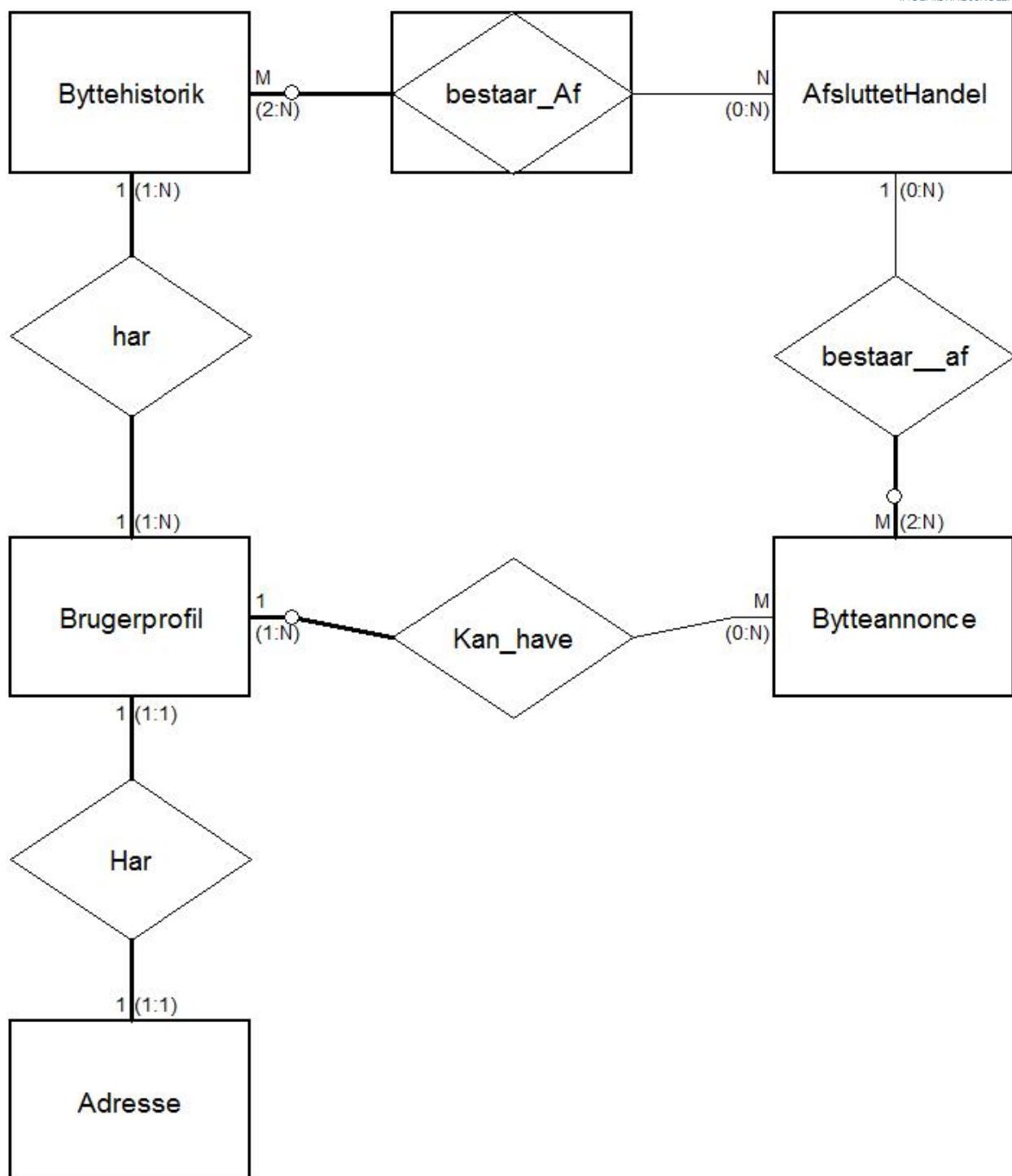
- BytteAnnoncer

- Beskrivelse
- Billeder
- Kategori
- Kommentarer
- Vurdering
- BrugerProfil
 - Brugernavn
 - Password
 - E-mail
 - Lokation
 - BytteHistorik
 - Rating

Det er vigtigt at understrege at der selvfølgelig er en række krav til disse data, der er defineret i systemarkitekturen. Brugeren skal ikke være tilgængelig for alle, og i forhold til almindelige kodeordsstandarder, skal passworded være hashet, således at applikationen ikke gemmer på nogen direkte version af kodeordet.

9.7 ER-diagram

For at opnå større indsigt i database opbygning og problemdomænet blev der oprettet et ER-diagram for at vise sammenhængene mellem de forskellige systemklasser (Entitys) i forhold konnektivitet. Det udarbejde ER-diagram kan ses på figur 9.9.



Figur 9.9: ER-diagram for databasen i systemet

Sammen med ER-diagram blev der også udviklet en række af oneliners, der beskriver, hvordan de forskellige entiteter er forbundet og om et relationship er tvunget eller er mulighed.

- En brugerprofil kan have mange bytteannoncer
- En byttannonce skal have en burgerprofil.
- En brugerprofil skal have en adresse.

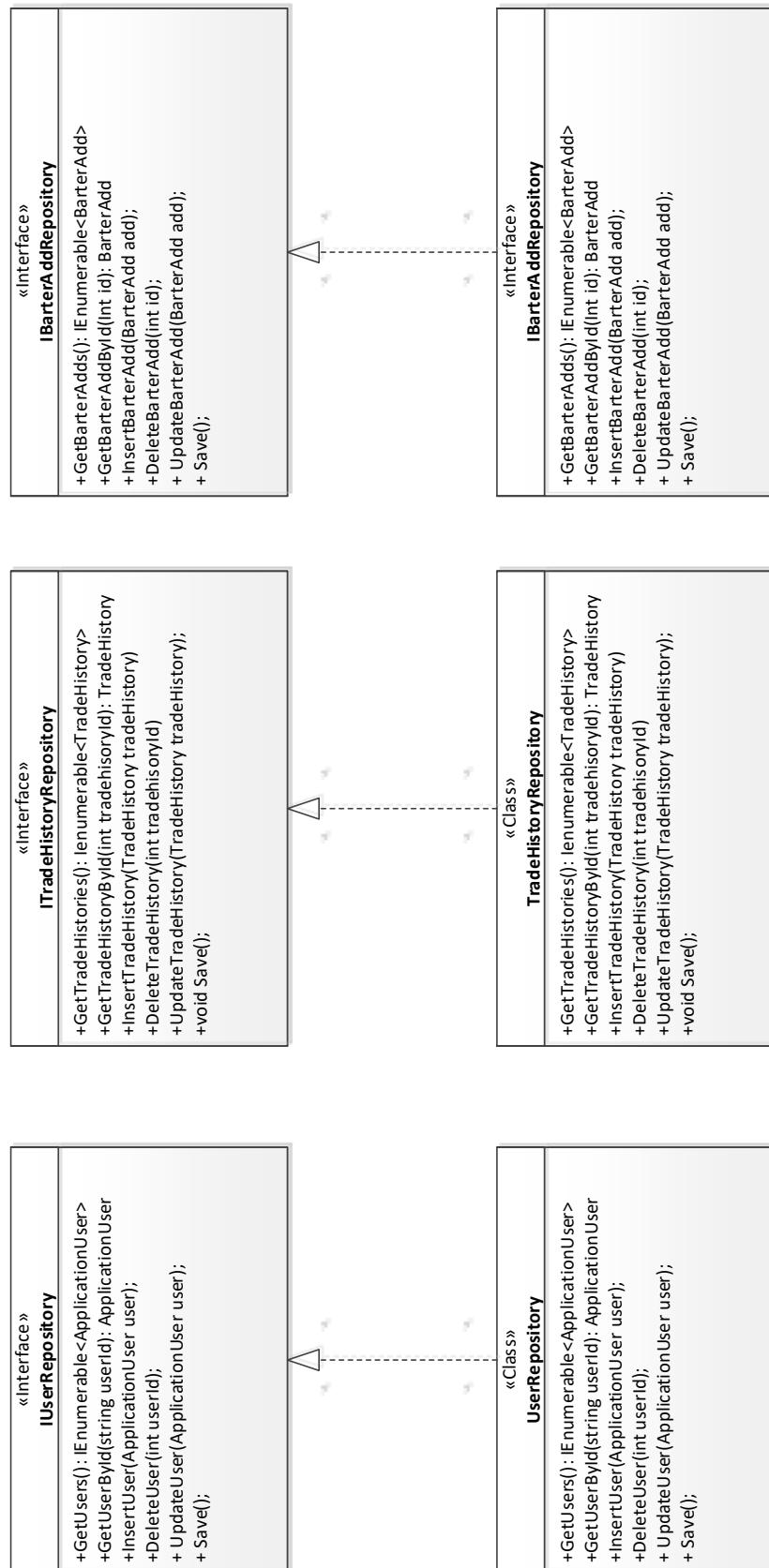
- En adresse skal have en brugerprofil.
- En brugerprofil skal have en Byttehistorik
- En byttehistorik skal have en brugerprofil
- En byttehistorik kan bestå af AfsluttetHandel
- En AfsluttetHandel skal have to Byttehistorik
- En AfsluttetHandel skal bestå af to Bytteannoncer
- En Bytteannonce kan være tilknyttet en AfsluttetHandel

På baggrund af ER-diagrammet kunne databasen være designet af et genereret script. Men det blev i stedet valgt at designe databasen med en code-first database ved brug af Entity Framework.

9.8 Repository Pattern

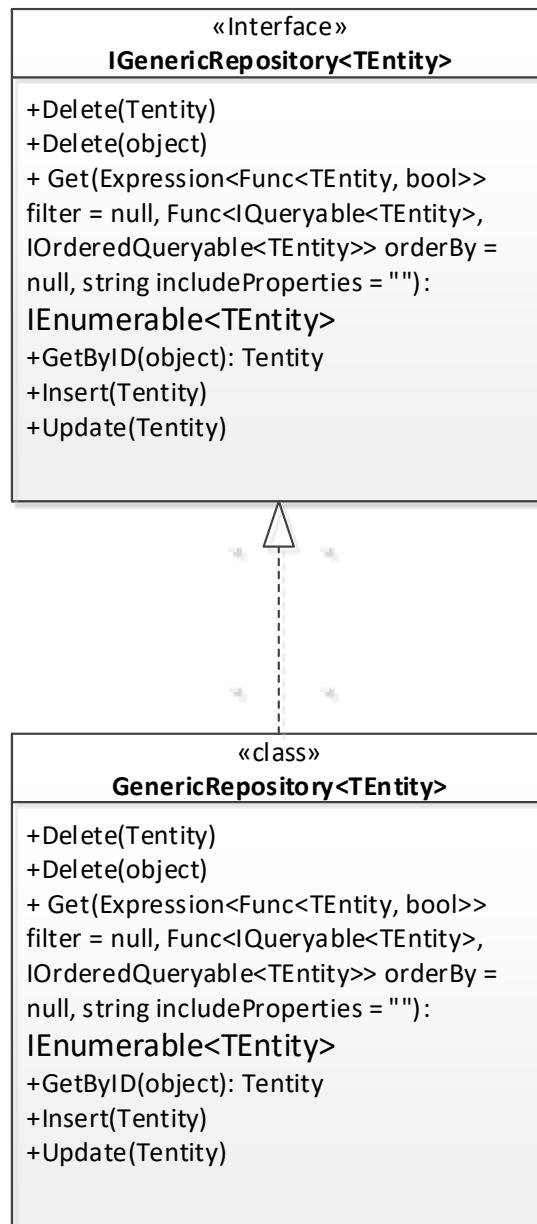
Repository pattern blev implementeret ved brug af forskellige dele og lag. I dette afsnit vil de forskellige dele blive beskrevet. I starten var tanken med repository-pattern, at alle klasserne i model blev tilknyttet et repository, der indeholdt de gængse CRUD-operationer (Create, Read, Update, Delete) på databasen i forbindelse med hver model. Hvert repository blev nedarvet fra et interface, dette er gjort på baggrund ad DIP¹, der gør at i forbindelse med test kan repositoriet mockes ud. Men efter implementering af flere repository blev, det observeret at koden for de forskellige repositories var stort set identisk, hvilket også kan se på en tidligere version af klassediagrammerne på figur 9.10.

¹Dependency Inversion Principle



Figur 9.10: Oversigt over flere næsten identiske repositories

Problemet med den identiske kode i de forskellige repository blev løst ved at lave et generelt template-baseret repository (Generisk repository). Det generiske repository indeholder en template baseret implementering af de enkelte CRUD-operationer. Man kan se metoderne og opbygningen af GenericRepository-klassen på figur 9.11



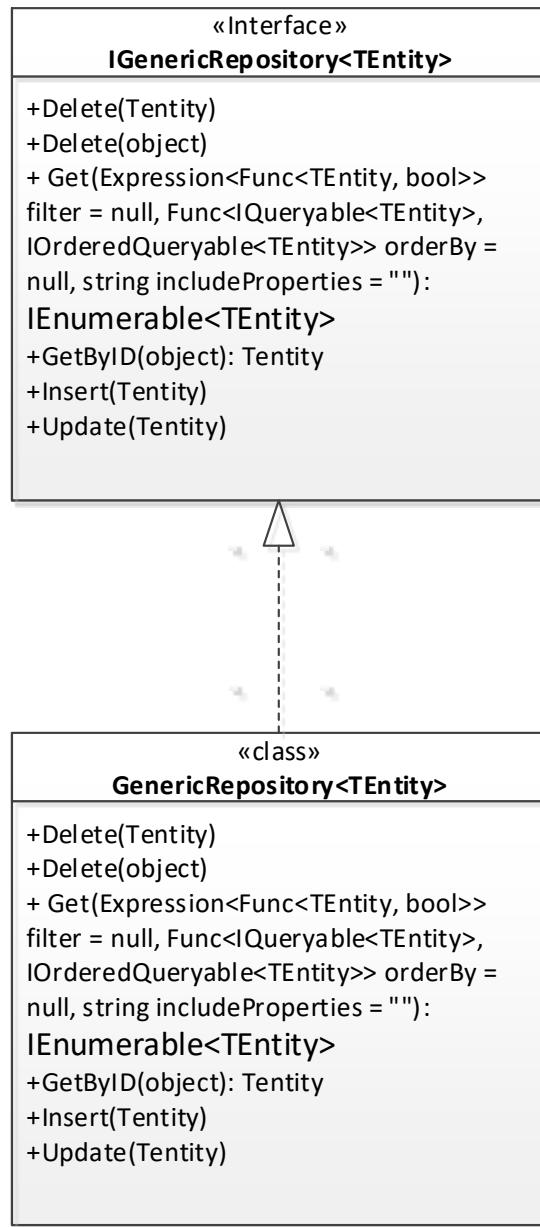
Figur 9.11: GenericRepository opbygning

Af særlige interesaante metoder i GenericRepository kan nævnet Get-metoden, der nok også er den mest anvendte metode i forbindelse med DAL. Denne metode returnere en liste af objekter af en given klasse. Metoden:

```
public virtual IEnumerable<TEntity> Get(Expression<Func<TEntity, bool>>
filter = null, Func<IQueryable<TEntity>,
IOrderedQueryable<TEntity>> orderBy = null, string includeProperties = "")
```

Metoden indeholder en række af filtre, ekspression, orders og includeproperties, der gør, at man kan filtre, inkludere og udvælge, hvilke data, der kommer tilbage fra get-metoden. Man kan bl.a. gennem brug af includeproperties komme rundt om EF Lazy-loading.

GenericRepository's template implementering bliver tilgået igennem unitofwork-klassen. UnitOfWork-klassen indeholder de forskellige GenericRepositories, så hvis der skal arbejdes på de forskellige repositories blev de tilgået igennem denne. UnitOfWork-klassen er udarbejdet på den måde, at første gang de et af de genericRepository bliver tilgået oprettes et nyt repository og næste gang dette repository skal tilgås bliver de tidligere oprettet genericRepository retuneret. UnitOfWork-klassen indeholder desuden også en save-metode og en dispose-metode. Det er disse metoder som comitter til databasen. UnitOfWork-klassen kan ses på nedenstående klasse diagram på figur 9.12

**Figur 9.12:** UnitOfWork opbygning

Repository-pattern med unitofWork er som allerede påpeget valgt for at adskille buisness-logikken fra DAL.

9.9 Map og distance

I forbindelse med implementering af sytemet var en del af annoncen i forhold til projektets rammer, at bytteartiklens position skulle vises på et kort og afstanden til annoncen skulle også vises. Dette blev implementeret ved brug af et Google Maps API.

9.9.1 Konvertering til koordinater

I det følgende beskrives brugen af Google Maps API. API'et er blevet anvendt flere steder i projektet. Det blev anvendt hvert gang en bruger opretter sig med sin adresse på siden. Den indtastede adresse bliver ved brug af API'et omdannet til koordinater, der gemmes i databasen. Koordinaterne er nødvendige for at kunne beregne afstanden til en bytte annonce samt at vise bytteannoncen placering på et kort under den enkelte bytte annonce. Det blev valgt, at brugerens adresse skulle konverteres til koordinater under oprettelsen for at undgå at skulle gøre dette runtime. Hvilket ville have indflydelse på systemets performance i forbindelse med visning af annoncer og beregning af afstand.

9.9.2 Beregning af afstand

Selve beregningen af afstanden gøres ved normal trigonometri og giver derfor afstanden i fugleflugt. Det blev valgt at regne afstanden i fugleflugt, da dette var det mest overskuelige for udviklerne. Afstanden bliver beregnet hver gang denne skal bruges. For at kunne beregne afstanden til annoncen kræves det, at brugerens position er kendt. Det vil sige, at han er oprettet i systemet med en adresse og koordinater. Afstanden bliver som tidligere nævnt udregnet som afstanden mellem brugens lokation og annoncens lokation.

9.9.3 Kortvisning

Bytteannoncernes placering på et kort vises ved brug af Jmeloegui.Mvc.GoogleMap, der en GoogleMap Control udviklet specielt til ASP.NET MVC. Denne kontrol blev tilføjet som en nuget package til projektet og referer til i de konkrete afsnit. Kontrollen virker ved, at man i viewet oprettet et map. I mappet sættes de basale indstillinger som bredde- og længdegrad for centrum af kortet. Zoom-level, størrelse og markers vælges også. I det nedenstående kode-eksempel kan man se, hvordan et map oprettes i viewet. Mappet bliver tilføjet en center-position, zoom-niveau og en marker i midten af korten. Denne kode er taget fra barteradds/detail-viewet.

```

@Html.GoogleMap().Name("map")
.Height(200)
.ApiKey("AIzaSyADSBCkyZfoUjYmTqG5hhcfXFjwGoq7rHU")
.Center(c => c.Latitude((double)@ViewData["Latitude"]))
.Longitude((double)@ViewData["Longitude"])
.Zoom(13)
.Markers
(m => m.Add().Title(Model.Titel))
)

```

Den overstående kode giver følgende kort, der kan ses på figur 9.13. På kortet er en markerør, der viser annoncens position.



Figur 9.13: Eksempel på kort, der viser annonces placering

Som det fremgår af overstående eksempel kan man linke fra modellen til kortet i ligehed med den normale adfærd i MVC. I dette eksempel sættes markørens titel=titlen fra den medgivne model. Dette gøres ved brug af normal MVC-struktur, hvor controlleren for dette view giver en model med til viewet. Kortet kan i viewet tilpasses i størrelse, zoom-level og på mange andre måder. Jmelosegui.Mvc.GoogleMap er også blevet brugt i forbindelse med visning af samtlige annoncer på et kort. Dette er gjort på samme måde som vist i koden ovenover dog er der bare linket til en af liste af modeller i stedet for en enkel model.

9.10 Rating af bruger

9.10.1 Forundersøgelse

Jævnfør Moscow-analysen var der ønske om en funktionalitet, så brugere kan vurdere hinanden efter en udført byttehandel. Mulighederne for at finde et rating-system blev undersøgt, og der blev fundet et JQuery-plugin til bootstrap[14].

Dette plugin gør at man kan vise et antal stjerner, som kan være tomme og fyldte, hvilket vises dynamisk mens brugeren bevæger markøren over. Alt i alt får det stjerne-ratingen til at se æstetisk og lækker ud, og kan både bruges til rating input, og til blot at vise en brugers rating.

Desuden indgik JQuery som pensum på I4GUI, så det var også en mulighed for at få noget erfaring med at arbejde med denne teknologi. Af disse årsager passede dette plugin fint ind i en løsning til rating featuren, og blev derfor valgt til anvendelse af løsningen.

9.10.2 Virkemåde

Når en byttehandel er accepteret fra begge parter, bliver den markeret som en færdiggjort handel, og kan nu rates af begge brugere. Denne rating bidrager til brugernes gennemsnitlige rating, som

er den, som vises sammen med deres annoncer som vist på forsiden, som vist i eksemplet nedenfor.



Arduino Mega

The Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16





T-Shirt

Lækker hvid trøje fra H&M som er i helt vildt god stand, men som er blevet for lille



Figur 9.14: Udsnit af forsiden, hvor annoncerne vises med ejerens rating

Når brugerne skal rate hinanden, skal de give mindst 1 stjerne. Så længe man prøver at give mindre end 1 stjerne vil det ikke være muligt at bekræfte sin rating. Denne feature er vist på figur 9.15. Featuren er implementeret ved hjælp af det anvendte rating plugin, som tilbyder et event, der bliver fyret når ratingens værdi ændrer sig. På den måde kan knappen ændre class mellem disabled og ikke disabled afhængig af ratingens værdi. 9.15

Bedøm med 1-5 stjerner, og skriv eventuelt en kommentar



Skriv din kommentar her

Bekræft

Bedøm med 1-5 stjerner, og skriv eventuelt en kommentar



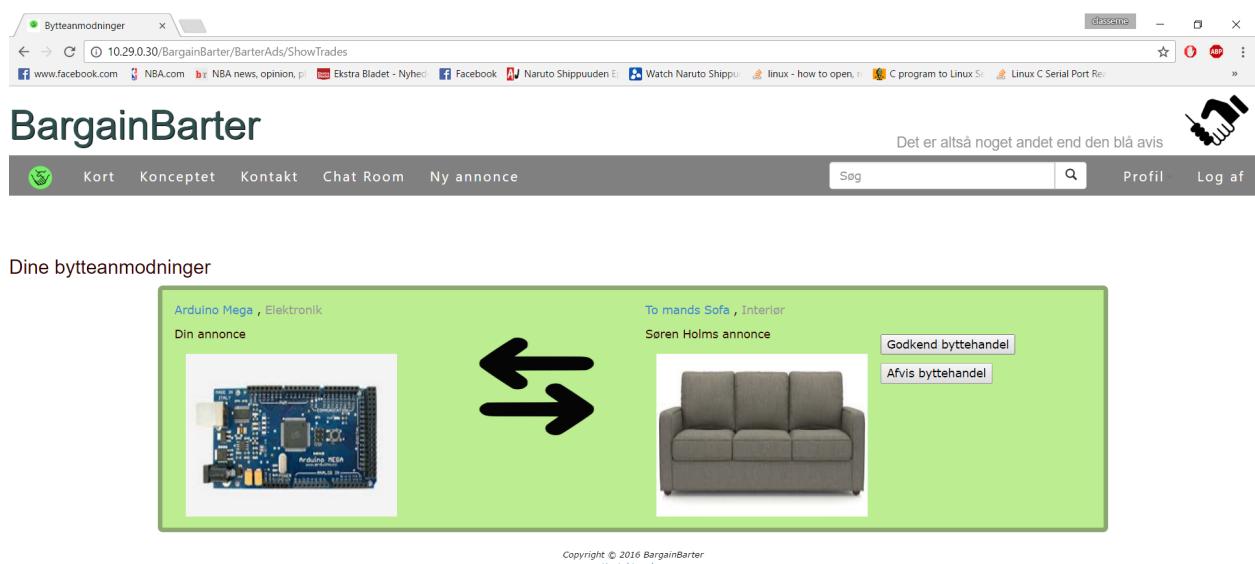
Skriv din kommentar her

Bekræft

Figur 9.15: 'Bekræft'-knap bliver aktiveret når man rater mindst 1 stjerne

9.11 Byttehandel

En af de primære funktioner for projektet er, at brugere skal kunne anmode om en byttehandel, have mulighed for at kontakte hinanden, bytte og til sidst give hinanden en anmeldelse. Før man har mulighed for at bytte, kræves det at man har en bruger, som har oprettet mindst en annonce. En annonce kan kun byttes en gang. Det fungerer ved, at person1 sender en bytteanmodning til person2. Person2 kan derved godkende eller afvise anmodningen se figur 9.16. Ved godkendelse bliver anmodning sendt tilbage til person1, det skyldes at personen kan have ændret mening. Person1 kan derfor også godkende eller afvise. Ved godkendelse bliver byttehandlen færdiggjort, og både person 1 og 2 får byttehandlen ind i deres byttehistorik.

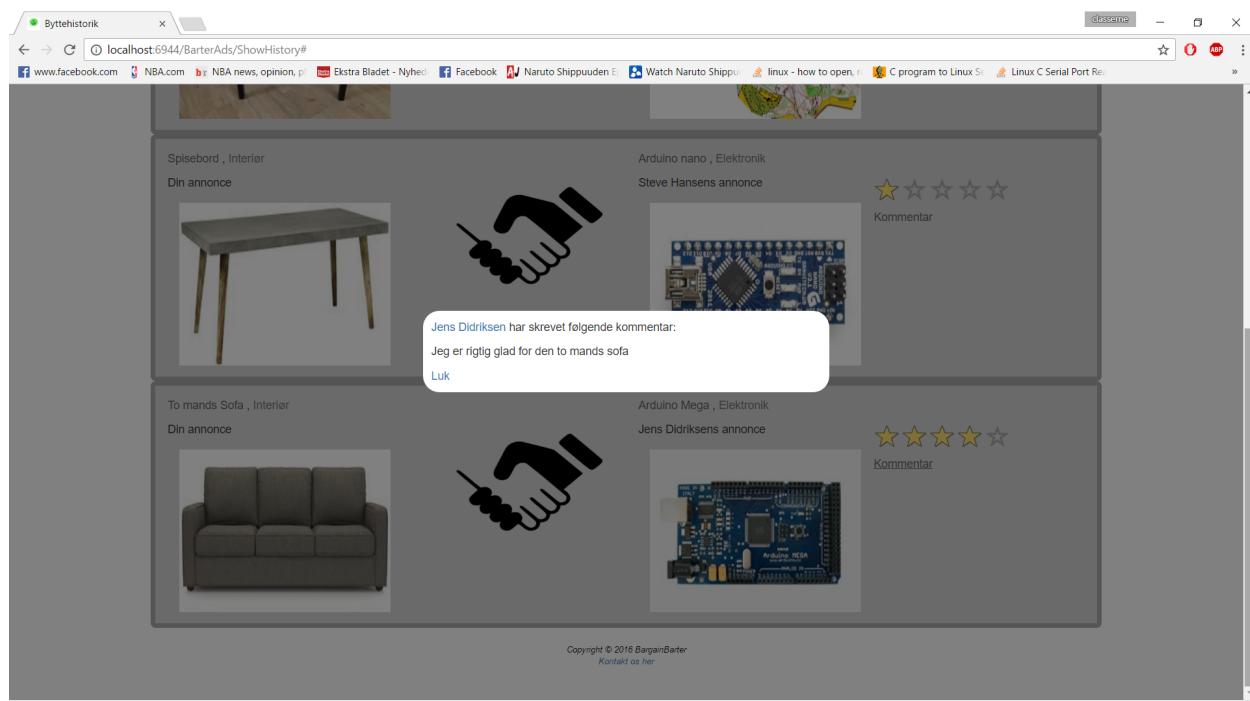


Figur 9.16: View for byteanmodninger

Det er nu muligt for de 2 personer, at give hinanden en anmeldelse med kommentar og rating, alt efter hvordan handlen forløb. De 2 annoncer er nu kun synlige i byttheistorikken, og kan ikke tilgås i andre byttheandler. Afventende anmodninger fra andre brugere, med den byttede annonce, bliver slettet, og kan derved ikke gennemført.

Hver bruger har deres egen byttheistorik, som har en liste af færdiggjorte handler. Andre brugeres byttheistorik kan tilgås, hvor man kan se anmeldelser, og hvis man har byttet med personen, er det muligt lave en anmeldelse af handlen.

Hvis en handel ikke er anmeldt, vil der stå skrevet "Ikke anmeldt" på handel i en byttheistorik. Hvis en handel er anmeldt, kan man se hvilken rating og kommentar som en bruger har givet. Til rating, som er vist i stjerner, bliver der brugt et JQuery-plugin "BootstrapStarRating". Da kommentarer ikke nødvendigvis har samme længde, faldt valget på at lave en "popup"-klasse, se figur 9.17. Med CSS er popup'en blevet styret, så den skalerer alt efter kommentarstørrelsen. Der er blevet brugt JavaScript, så man ved klik kan vise eller skjule kommentaren.



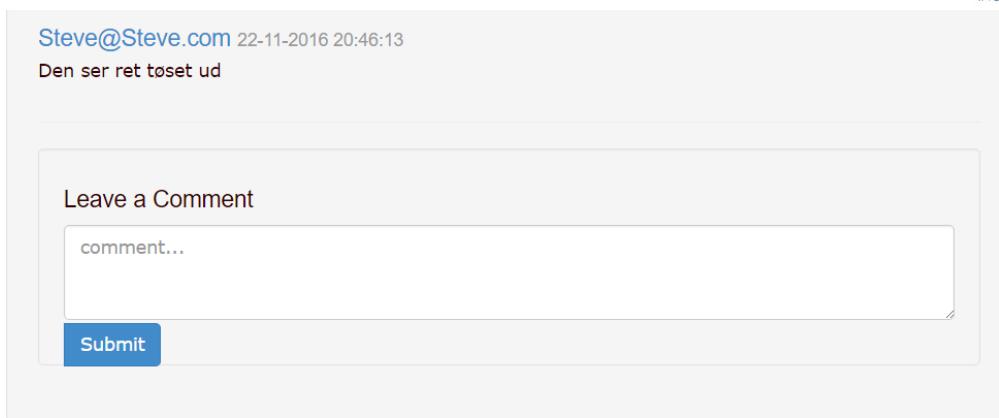
Figur 9.17: Viser rating og popup-kommentar

9.12 Kommentarer

Som nævnt i userstorien: "Kommentering af annonce" i afsnit 3.2.5 skal annoncer i BargainBarter kunne kommenteres. Der er derfor i modellen oprettet en model klasse til kommentarene ,der indeholder:

- Den aktuelle bytteannonce
- Oprettelsestidspunkt
- Kommentarteksten (Maximalt 500 tegn)
- Den kommenterende bruger

På figur 9.18 kan det ses, at brugeren har mulighed for at indtaste og submitte. Controlleren sørger for at der ved et submit, som er en POST action, automatisk bliver fundet alt andet end selve kommentarstrengen og oprettelsestidspunktet. Sidstnævnte bliver oprettet i modellen. Og kommentarstrengen bliver retuneret fra viewet igennem Post-actionen, der kalder metoden på serveren.

**Figur 9.18:** Udseende af kommentarer

Viewet er lavet så det er intuitivt og nemt at overskue. Til viewet er der anvendt en bootstrap template [2]. Actionen til kommentering benytter UnitOfWork som beskrives i 9.8, og virker efter principerne beskrevet i afsnit 9.2. Logikken til kommentarer er delvis genbrugt under ratings. Da denne funktionalitet bygger meget på den samme logik.

9.13 Brugervenlighed

Brugervenlighed har været en afgørende faktor i forbindelse med udviklingen af systemet. Da BargainBarter er en webapplikation, skal den gerne appellerer til en bred målgruppe. BargainBarter er forsøgt at lave brugervenligt ved at benytte frameworket: Bootstrap, så webapplikationen bliver skalerbar på tværs af forskellige enheder og skærmstørrelser. Når der nævnes brugervenlighed i forbindelse med BargainBarter, menes der især funktionalitet, og ikke så meget det æstetiske udtryk.

Brugervenlighed er i høj grad blevet skabt igennem hjemmesidens design. I designet af hjemmesiden blev der tænkt over, at hjemmesiden skulle være let genkendelig og intuitiv at bruge. Konkret er det udført ved at bruge genkendelig elementer fra andre hjemmesider i designet. Disse elementer omfatter bl.a. navigationsbaren i toppen af siden. Mange elementer går igen på alle sider på hjemmesiden, så oplevelsen er ensartet. Navigationsbarens navne og links er blevet forfinet gennem projektet således, at de blev entydige og let forståelige. Et billede af den endelige navigationsbar kan ses på figur 9.19.

**Figur 9.19:** Navigationsbaren for BargainBarter

I projektet er der blevet lagt vægt på, at der skal kunne navigeres rundt på hele hjemmesiden på få klik. Anvendelsen af ASP.NET MVC har gjort det simpelt at lave et layout, der går på tværs af de forskellige sider. Dette gøres ved et fælles layout der er blevet defineret i Shared.cshtml viewet.

10. Test

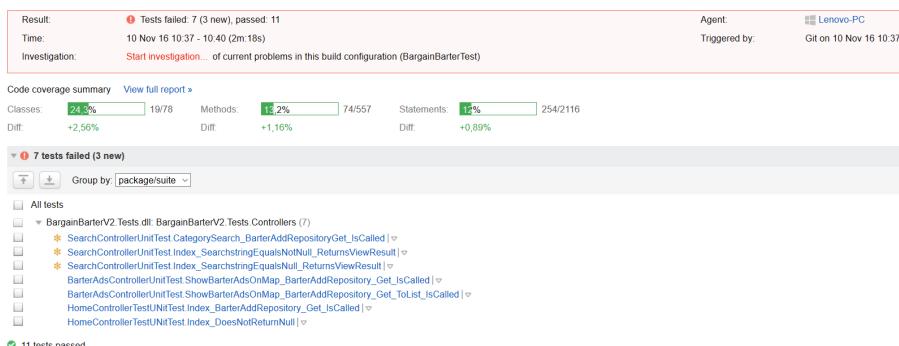
Projektet stiller krav om løbende udvikling og ikke mindst krav om kvalitet. Automatiserede tests har dermed været et naturligt valg. De automatiserede test har konstant og systematisk hjulpet os som udviklere til at sørge for, at kvaliteten i vores system opretholdes, når ny funktionalitet er blevet tilføjet. Der er specielt blevet lagt fokus på unit test i system. Detektion af fejl i gammel kode, når ny kode tillægges er en afgørende faktor for dette valg.

10.1 Continuous Integration

Projektet er sat op, så der blev benyttet continuous integration. Dette betød, at projektet bliver bygget på en lokal maskine, hver gang der bliver tilføjet en ændring til projektet via git. CI er blevet brugt for at undgå integrationsproblemer i projektet. Disse kan nemt opstå, når flere personer arbejder parallelt på projektet. CI har ved at bygge projektet hele tiden holdt styr på projektets status. Så hvis der opstod problemer på projektet kunne de nemt lokaliseres til et enkelt push. Dette har været med til at overskueliggøre processen for alle gruppemedlemmer. TeamCity blev sat op med to build som var afhængige af hinanden. Efter hvert push til repositoriet blev projektet bygget, og hvis det lykkedes, kørte testene.

TeamCity [3] blev anvendt som CI-server, hvor der blev opsat et projekt, der blev sat til at pege på gruppens github-repository. Projektet blev sat op til, at hver gang der blev "pushed" til repositoriet skulle TeamCity kører to "Build Steps". I første step blev projektet bygget ved brug af MSBuild Tools 2015. Hvis det første skridt gik godt blev anden step udført, der bestod i at køre Test-projektets NUnit-tests. Testene blev kørt ved brug af en NUnit.ConsoleRunner, der var blevet installeret som en NuGet-Package i projektet. CI-projeket blev også tilknyttet en dotCover-test, der skulle fungere som et pejlemærke for, hvor stor en del af projektet testene dækker. Da der ikke var nogen specielle krav til Coverage-analysen blev JetBrains standard dotCover fil benyttet.

Nedenunder på figur 10.1 kan et eksempel af et build på TeamCity serveren ses.



Figur 10.1: Testresultat på baggrund af kørt test på TeamCity

Som det fremgår af figur 10.1 gav TeamCity et let og overskueligt overblik og projektets status, der kunne blandt andet ses Coverage-analysen og hvilke test, der fejlede mm.

Det var planen, at CI skulle have kørt på en ekstern server, så alle for gruppen kunne have fuldt processen. Men gruppen havde desværre ingen ekstern server til rådighed, hvilket betød, at CI blev

lagt på en lokal server/PC. Det var et bevidst valg fra gruppens side at benytte TeamCity i stedet for Jenkins, der er opnået erfaring med i undervisningen i I4SWT, for at opnå erfaring med forskellige CI-servere og programmer.

10.2 Unit Tests af controllerne

Da systemet i starten af udviklingsfasen kun indeholdt meget minimal buisness-logic, ligger størstedelen af funktionaliteten i controllerne i MVC strukturen. Strukturen med at lægge funktionalitet i controller blev fastholdt igennem udviklingsfasen. Dette blev erfaret som en dårlig beslutning, da controllerne ofte uover buisness-logikken også benyttede sig af web-specifikke metoder¹, der gør unit test på disse metoder tæt på umulig. Designet skulle derfor have været refaktoreret, således at mere af buisness-logikken var blevet flyttet til et separat buisness-lag, så denne del af systemet kunne være blevet testet. Databasen i sig selv er svær at teste, og views i MVC'en kan nærmest ikke testes. Det der reelt kan testes er derfor, hvad controllerne giver videre til deres views, og ikke selve views'ne.

I Controllerne er der flere ting der er væsentligt at teste.

- Hvad gives med i viewbagen
- Buisness-logic funktioner som controllerne bruger
- At de korrekte fejl bliver kaldt når controllerne giver ugyldige værdier
- De enkelte redirects bliver kaldt korrekt
- At de enkelte controllers returnerer views

Disse er de udvalgte ting som er vurderet til at give mening at teste på.

Den fulde liste af test kan ses nedenfor.

¹HttpContext-metoder

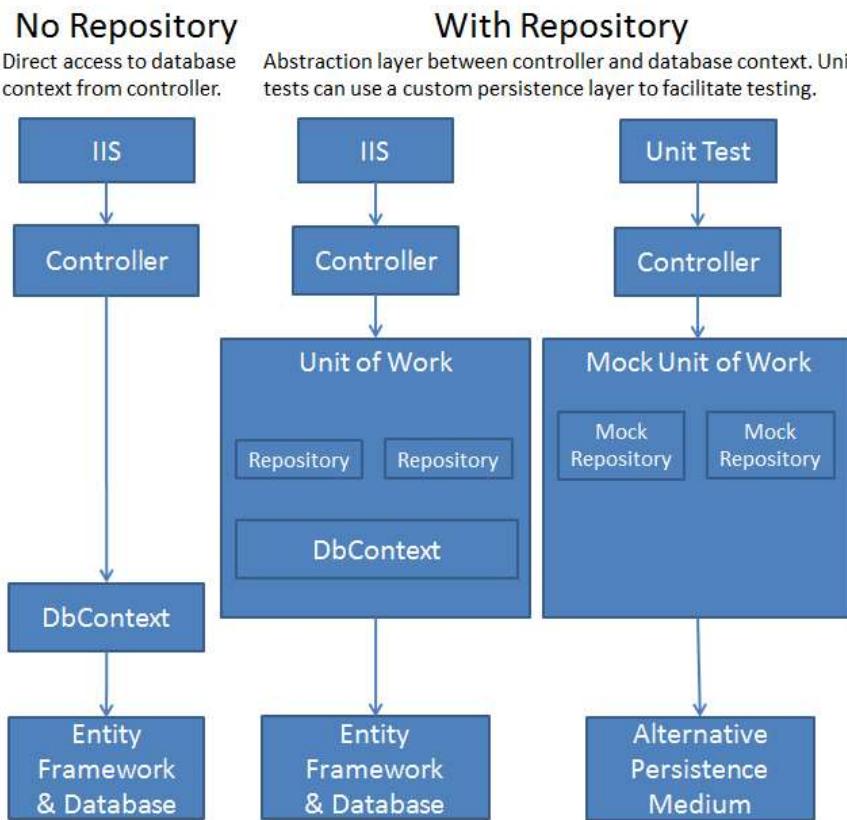
Controller	Test	Beskrivelse
BarterAds-Controller	Index_RedirectsToController-_HomeActionIndex ShowBarterAdsOnMap-_BarterAddRepository-_Get_IsCalled Details_NullId_Returns-_BadRequest Create_DoesNot_Return_Null Edit_NullId_Returns_Bad Request Delete_NullId_Returns_Bad Request	Tester om controlleren faktisk kalder den rigtige view til Home. Tester om controlleren kalder den rigtige metode for at vise alle barteradds på kortet Tester om hvis metoden Details får Null, som id, så returnerer controlleren en BadRequest, kode 400. Tester, at create controller retunerer et view Tester om hvis metoden Edit får Null, som parameter, så returnerer controlleren en BadRequest, kode 400. Tester om hvis metoden Delete får Null, som parameter, så returnerer controlleren en BadRequest, kode 400.
ManageController	ChangePassword_DoesNot ReturnNull	Tester, at ChangePassword metoden retuner et view, der er forskellig fra null
HomeController	Index_BarterAddRepository-_Get_IsCalled Index_DoesNotReturnNull AboutDoesNotReturnNull ContactDoesNotReturnNull	Tester om den metode der viser alle barteradds på Index siden, bliver vist. Tester, at index metoden retuner et view, der er forskellig fra null Tester, at About-metoden retuner et view, der er forskellig fra null Tester, at Contact-metoden retuner et view, der er forskellig fra null
SearchController	Search_Calls-_BarterAdsRepository CategorySearch_Calls-_BarterAdsRepository Index_DoesNotReturnNull Index_SearchstringIsNull_ ReturnsNotNull Categorysearch_SearchstringIsNonEmpty_ResultIsNotNull	Tester om den BarterAddRepository bliver kaldt når man søger efter en bestemt ting Tester om BarterAdsRepository anvendes at og bliver kaldt når man vælger kategori Tester, at Index-metoden retuner et view, der er forskellig fra null Tester om, at Index-metoden retunerer et view, der giver mening på trods af den modtager null som paramter Tester om, at Categorysearch-metoden retunerer et view, der giver mening, når det modtager en paramter

Controller	Test	Beskrivelse
HelperFunctions	CalculateAverage_Test	Tester om CalculateAverage udregner gennemsnittet rigtigt på baggrund af en nogen kontrolværdier.
	CalculateUserAvg_Test	Tester om CalculateuserAvgRating kan udregne udregne gennemsnittet rigtig for en bruger på baggrund af en liste af FinishedTrades
HelperFunctions Distance	GetDistance_Test	Tester om en adresse kan konverteres til koordinater og om distancen mellem koordinaterne er rigtig.

Det er væsentligt at pointere hvordan disse test bliver lavet isoleret, således at controllerne ikke er afhængige af DAL-laget.

10.2.1 Unit test igennem DAL

Som nævnt spiller design og test i høj grad sammen. Dette er grunden til, at der i projektet er blevet anvendt repository pattern, der uddybes i afsnit 9. På figur 10.2 kan det ses, hvordan repository-pattern og UnitOfWork i høj grad spiller sammen med test af systemet. Det er muligt at teste systemet direkte gennem dbcontexten, men ved ændring af database teknologi vil alle tests skulle omskrives. Dette er ikke ønsket, da selve unit testene blot skal teste om en klasse kaldet korrekt ned i DAL laget. En unit test skal også være genbrugelig, så en god unit test burde ikke skulle skrives om på baggrund af systemændringer. Unit testene skal ikke teste om kaldene ned i de andre klasser rent faktisk virker. Det er nemlig ikke controllerne i MVC-mønsteret, der har ansvar for, at database kaldene virker. Derfor bruges Unit of Work i testene som en mock, på baggrund af at der benyttes dependency injection i systemet. Mocken af Unit of Work er den, som controlleren kaldet ned i. På den måde opnås der ved brug af repository pattern, at controllerne kan testes uafhængigt af Data Access Laget og selve databasen.



Figur 10.2: Unit test med UnitOfWork-Mock [10]

10.3 Integrationstest

Efterfulgt af unit test af de enkelte controller, skal integrationen mellem de forskellige dele af systemet testes. Der er dog imidlertid ikke den store afhængighed mellem de forskellige controllers i systemet, hvilket MVC også lægger op til. Da controllerne også indeholder stort set alt business-logikken interagere de ikke rigtig med andre dele af systemet. Der er derfor ikke lavet nogen specielle integrationstest i systemet. Da der ikke er de store afhængigheder mellem de forskellige controllerlere og klasser i systemet. Et bedre design og opslitning af logik ville have gjort systemet nemmere at teste og specielt også at planlægge integrationstest på. Integrationstestene er derfor ikke blev udført som automatiske test, men i stedet som manuelle test, der vil blive beskrevet i det efterfølgende afsnit.

10.4 Manuelle tests

Grundet applikationens GUI og arkitektur har det ikke været muligt at automatisere alle tests. Derfor har de automatiserede unit tests været suppleret meget med manuelle tests. Dette er foregået ved, at en tester har klikket rundt på hjemmesiden og bekræftet om den nyimplementerede funktio-

nalitet virkede. I dette afsnit vil der følge et par eksempler på, hvordan de manuelle test har fundet sted. Ikke alle manuelle tests er beskrevet her, da det ville være for uoverskueligt. Men generelt gælder det, at stort set alt funktionalitet er blevet testet manuelt.

10.4.1 Test af byttehandel

Byttehandel funktionaliteten er blevet testet ved, at der for selve byttehandlen er blevet defineret en række af punkter, der skal gennemføres i forbindelse med en byttehandel. Disse punkter udgøre tilsammen et bytteflow. Bytteflowet kan ses på figur 10.3.



Figur 10.3: Flow over en byttehandel i BargainBarter

Ved test af bytteflowet, blev selve bytteflowet slavisk gennemgået, og afhængigt af dette resultat, blev funktionaliteten be- eller afkræftet.

10.4.2 Test af Chat

Chat funktionaliteten er skrevet i javascript, hvilket har gjort, at den ikke kunne testes automatisk ved brug af NUnit. Chat funktionaliteten er derfor blevet testet manuelt på baggrund af nedenstående flow, der ses på figur 10.4.



Figur 10.4: Flow over chatten på BargainBarter

Chatten blev testet på baggrund af flowet fra figur 10.4. Resultatet af chat-testen set fra en anden s web-klient ses på figur 10.5.

Du kan nu chatte med alle aktive brugere på siden.

Nyeste beskeder står nederst.

- **Steve Hansen:** har joinet chatten!
- **Steve Hansen:** er der nogen friske piger på linjen?

Figur 10.5: Test af chat

10.4.3 Opret annonce

Opret annonce funktionaliteten er blevet testet manuelt på baggrund af nedenstående annonceflow, der ses på figur 10.6.



Figur 10.6: Flow over oprettelse af annonce på BargainBarter

Overstående flow blev gennemført, hvor det til sidst blev bekræftet, at der var blevet oprettet en annonce for brugeren.

11. Accepttest

I starten af projektet blev der beskrevet en række krav til systemets funktionelle krav og de ikke funktionelle krav. Disse krav kan ses i afsnit 3.

11.1 Funktionelle krav

De funktionelle krav er beskrevet ved brug af Gherkin. Accepttesten er derfor lavet på baggrund af Gherkin-scenariet. I det følgende afsnit, beskrives udførslen og resultat af accepttesten til de funktionelle krav, som er defineret til projektet.

11.1.1 Oprette En Bytteannonce

EGENSKAB: Oprette en bytteannonce

Som bruger

Ønsker jeg at kunne oprette en bytteannonce

For at kunne bytte med andre brugere af systemet.

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Oprette en bytteannonce

Når bruger ønsker at oprette en bytteannonce i systemet

Så nавигerer han til menupunktet "Opret annonce"

Så udfylder han bytteannonce-skabelonen

Og trykker på "Opret annonce"-knappen

Det overnævnte scenario blev kørt igennem og en bytteannonce blev oprettet tilhørende den pågældende bruger.

Status - Godkendt

11.1.2 Opret En Brugerprofil

EGENSKAB: Opret en brugerprofil

Som bruger

Ønsker jeg at kunne oprette en brugerprofil på websiden

For at få adgang til websidens funktionalitet

BAGGRUND

Givet at bruger ønsker en brugerprofil

SCENARIE: Opret en brugerprofil

Når bruger ønsker at oprette sig i systemet

Så nавигerer han til Menupunktet "Ny bruger"

Så indtaster han brugernavn, e-mail, password mm.
Så verificerer brugeren sig på tilknyttede e-mail addresse
Og brugeren kan nu logge ind med sin nye brugerprofil

Det overnævnte scenario blev kørt igennem og der blev oprettet en brugerprofil til brugere, der nu kan benytte systemet.

Status - Godkendt

11.1.3 Se Geografisk Lokation For Bytteannonce

EGENSKAB: Se geografisk lokation for bytteannonce

Som bruger
Ønsker jeg at kunne se hvor den ønskede byttevare er
For at kunne se hvor en evt. byttehandel kunne finde sted

BAGGRUND

Givet at bruger Jakob er logget ind
Og han er inde på et annonceopslag lavet af personen Gitte

SCENARIE: Jakob vil se Gittes geografiske lokation

Når Jakob ønsker at se Gittes lokation
Så kigger han til højre for annoncebilledet
Og ser et kort, som angiver Gittes lokation

Det overnævnte scenario blev kørt igennem og bytteannoncens geografiske lokation blev præsenteret for brugeren.

Status - Godkendt

11.1.4 Glemt Password Eller Brugernavn

EGENSKAB: Glemt password eller brugernavn

Som bruger
Ønsker jeg at kunne få tilsendt mit brugernavn og et nyt password
For at kunne komme ind på min bruger på trods af at have glemt mit password eller brugernavn

BAGGRUND

Givet at bruger er oprettet i systemet
Og har glemt sit password eller brugernavn

SCENARIE: Glemt password eller brugernavn

Når bruger ønsker at få tilsendt sit brugernavn med et nyt password
Så navigerer han til websidepunktet "Glemt password eller brugernavn"
Så skriver brugeren sin e-mailadresse ind
Så finder brugeren den tilsendte e-mail
Og brugeren kan nu logge ind med det tilsendte password

Det overnævnte scenario blev kørt igennem, og brugeren fik et nyt password og kan igen benytte

systemet.

Status - Godkendt

11.1.5 Kommenter En Annonce

EGENSKAB: Kommentering af annoncer

Som bruger

Ønsker jeg at kunne kommentere annoncer

For at kunne afklaret spørgsmål etc. om annoncen

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Kommentere en annonce

Når bruger ønsker at kommentere en annonce

Så skal bruger navigere ned til en kommentarboks under annoncen

Så skal bruger kunne skrive en kommentar i kommentarboksen

Så skal bruger trykke "send" ved siden af kommentarboksen

Og kommentaren vises da under kommentarboksen

Det overnævnte scenario blev kørt igennem og der blev oprettet en kommentar til den pågældende bytteannonce.

Status - Godkendt

11.1.6 Søg Efter Bytteannoncer

EGENSKAB: Søg efter bytteannoncer

Som bruger

Ønsker jeg at kunne søge efter bytteannoncer

For at kunne finde en bestemt type vare

BAGGRUND

Givet at bruger er logget ind

SCENARIE: Søg efter bytteannoncer

Når bruger ønsker at søge efter bytteannoncer i systemet

Så nавигerer brugeren til menupunktet "Søg"

Så indtaster brugeren den varer som brugeren ønsker

Og trykker på "søg"-knappen

Det overnævnte scenario blev kørt igennem og der blev søgt efter 'stol'. Brugeren blev præsenteret for en af annoncen af en stol.

Status - Godkendt

11.1.7 Se Bytnehistorik

EGENSKAB: Se bytnehistorik

Som bruger

Ønsker jeg at kunne se min byttehistorik
For at kunne se, hvilke ting jeg har byttet mig til.

BAGGRUND

Givet at Bruger er logget ind

SCENARIE: Se Byttehistorik – Webapplikationen
Når bruger ønsker at se sin byttehistorik i systemet
Så nавигerer han til menupunktet "Brugerprofil"
Og trykker på "Byttehistorik-knappen"

Det overnævnte scenario blev kørt igennem og brugeren kunne se sin byttehistorik.

Status - Godkendt

11.1.8 Kontakt en bruger direkte

EGENSKAB: Kontakt en bruger direkte

Som bruger

Ønsker jeg at kunne kontakte en bruger direkte via en chat fra annoncen

BAGGRUND

Givet at en brugeren er logget ind

SCENARIE: Kontakt brugere direkte
Når Brugeren ønsker at kontakte en anden bruger direkte
Så nавигeres der til den anden brugers annonce
Så nавигeres der til knappen "Start chat"
Så trykkes der på "Start chat"
Og så kan der chattes med den anden bruger

Det overnævnte scenario kunne ikke gennemføres, da der ikke er implementeret et privat-chat mellem to brugere.

Status - Ikke godkendt

11.1.9 Anmeldelse af en byttehandel

EGENSKAB: Anmeldelse¹ af en byttehandel

Som bruger

Ønsker jeg at kunne vurdere en byttehandel
For at angivne, hvor trovædig den anden bruger er.

BAGGRUND

Givet at en brugeren er logget ind

Og har afsluttet en byttehandel med en anden bruger

SCENARIE: Anmeldelse af andre brugere

¹Der er her tale om en rating

Når Brugeren ønsker at vurdere byttehandlen en byttehandel med en anden bruger

Så navigatorer brugeren hen til "Afsluttede byttehandler"

Så trykkes "Vurder byttehandel" på den givne byttehandel

Så angives en score mellem 1 og 5

Så skrives der en kommentar (valgfrit)

Så trykkes "Send"

Og hans anmeldelse vises nu på den anmeldte brugers brugerprofil, og tælles med i den anmeldte brugers samlede brugervurdering

Det ovennævnte scenario blev gennemført og Gitte fik en brugeranmeldelse af Jakob. Gittes brugervurdering gik fra 3 til 3.5.

Status - Godkendt

11.2 Ikke funktionelle krav

I følgende afsnit beskrives udførslen og resultatet af accepttesten for de ikke funktionelle krav.

Krav	Beskrivelse	Forventet resultat	Resultat	✓ / X
Punkt 1	Der klikkes på "Annoncer"	Brugeren kommer ind på en oversigt over alle annoncer	Brugeren kommer ind på en oversigt over alle annoncer ved blot at være på forsiden	✓
Punkt 2	Hjemmesiden bliver tilgået af 10 brugere samtidig	Der sker ikke nogen ændring i funktionaliteten for de enkelte brugere på hjemmesiden	Ingen ændring for de enkelte brugere af hjemmesiden	✓
Punkt 3	Der skrives "Stol" i søgefeltet og der trykkes på søg, når der trykkes på søg startes et stop-ur. Uret stoppes når Søgeresultaterne bliver vist på skærmen	Stop-uret viser mindre end 15 sekunder	Stop-uret viser 2.3 sekunder	✓
Punkt 4	En computer tilkoblet et andet netværk end hjemmesidens server, går til hjemmesidens url-adresse (link til hjemmeside) via en Chrome Browseren	Browseren åbner forsiden til systemets hjemmeside	Browseren skriver: 'Der kan ikke oprettes forbindelse til dette website'	X
Punkt 5	Der søges efter annoncer inden for 10km	Den annonce, der ligger længst væk fra søgelokationen er mindre end 10100m	Den annonce, der er længst væk er 0.9km	✓
Punkt 6	En bruger skriver "Hej" til en anden bruger gennem det indbyggede chat-modul. Når der trykkes på send startes et stop-ur og uret stoppes, når den anden kan se beskeden	Stop-uret står på mindre end 10 sekunder	Testen er foretaget i den globale chat. Stop-uret viser 0.3 sekunder.	✓
Punkt 7	Skriv en brugervurdering på 500 tegn og prøver at skrive videre	Brugeren kan nu ikke tilføje flere tegn	Brugeren kan tilføje alle de tegn brugeren ønsker	X
Punkt 8	Skriv en brugervurdering på under 500 tegn og tryk "Send". Når der trykkes på "Send" startes et stop-ur og uret stoppes, når brugeranmeldelsen offentliggøres	Stop-uret viser mindre end 15 sekunder	Stopuret viser 1.5 sekunder	✓

Tabel 11.1: Accepttest af ikke-funktionelle krav del 1

Krav	Beskrivelse	Forventet resultat	Resultat	✓ / X
Punkt 9	Tilgå hjemmesiden fra en af de afgivne enheder i afsnit 4 via http://10.29.0.30/bargainbarter , log ind og opret en BarterAd	En BarterAd er blevet oprettet i systemet	En BarterAd er blevet oprettet i systemet	✓

Tabel 11.2: Accepttest af ikke-funktionelle krav del 2

12. Litteraturliste

- [1] CodeProject AdOps. *Three Layer Architecture in C# .NET*. <https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET>. 2016.
- [2] blackrockdigital. <https://blackrockdigital.github.io/startbootstrap-blog-post>. 2016.
- [3] JetBrains. *TeamCity*. <https://www.jetbrains.com/teamcity/>. 2016.
- [4] Microsoft. *ASP.NET*. <https://www.asp.net/>. 2016.
- [5] Microsoft. *Code First Migrations and Deployment with the Entity Framework in an ASP.NET MVC Application*. <https://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application>. 2016.
- [6] Microsoft. *Introduction to ASP.NET Identity*. <https://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity>. 2016.
- [7] Microsoft. *Introduction to LINQ Queries*. <https://msdn.microsoft.com/en-us/library/bb397906.aspx>. 2016.
- [8] Microsoft. *Introduction to SignalR*. <https://www.asp.net/signalr/overview/getting-started/introduction-to-signalr>. 2016.
- [9] Microsoft. *Model-View-Controller*. <https://msdn.microsoft.com/en-us/library/ff649643.aspx>. 2016.
- [10] Microsoft. *Repository Pattern*. <https://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>. 2016.
- [11] Microsoft. *The ADO.NET Entity Framework Overview*. [https://msdn.microsoft.com/en-us/library/aa697427\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/aa697427(v=vs.80).aspx). 2016.
- [12] Tutorials Point. *Design Pattern - Overview*. https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm. 2016.
- [13] Twitter. *Bootstrap*. <http://getbootstrap.com/>. 2016.
- [14] Kartik Visweswaran. *Bootstrap star rating*. <http://plugins.krajee.com/star-rating>. 2016.
- [15] w3schools. *CSS Basics*. <http://www.w3schools.com/css/>. 2016.
- [16] w3schools. *HTML Basics*. <http://www.w3schools.com/html/>. 2016.
- [17] w3schools. *JavaScript Basics*. <http://www.w3schools.com/js/>. 2016.