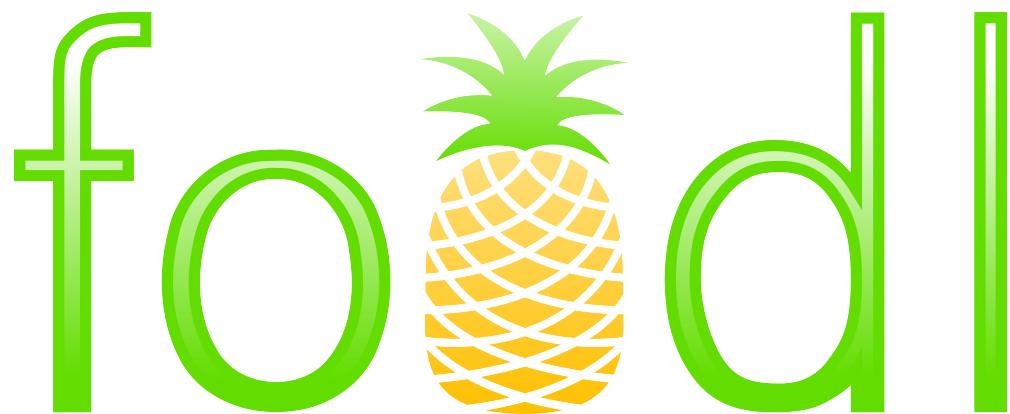


Aalborg Universitet

Datalogi



Anvendelse af Madrester

Forfattere:

Sebastian Wahl
Simon Buus Jensen
Elias Khazen Obeid
Niels Sonnich Poulsen
Kent Munthe Caspersen
Martin Bjeldbak Madsen

Vejleder:

Lise T. Heeager

September 2012 - December 2012

Titel:

fo~~o~~dl - Anvendelse af Madrester

Tema:

Udvikling af applikationer – fra brugere til data, algoritmer og test – og tilbage igen

Projektperiode:

P3, 3. september - 20. december 2012

Projektgruppe:

d303e12 (d303e12@cs.aau.dk)

Deltagere:

Sebastian Wahl
Simon Buus Jensen
Elias Khazen Obeid
Niels Sonnich Poulsen
Kent Munthe Caspersen
Martin Bjeldbak Madsen

Vejleder:

Lise T. Heeager

Afsluttet:

20. december 2012

Synopsis:

Denne rapport gør rede for udviklingsproces-
sen bag systemet, **foodl**, der er en webapplika-
tion. Undersøgelser viser, at personer, der bor i
parcelhuse, i gennemsnit smider 42 kg mad ud
om året. **foodl** har til formål at gøre det lette-
re for de madansvarlige, i de danske husstande,
at genbruge madrester, for dermed at mindske
spildet af mad.

Igennem en iterativ udviklingsmetode har vi
analyseret, designet, implementeret og testet
systemet. Vi benyttede en objektorienteret me-
tode, der er blevet præsenteret i bogen Objekt-
orienteret Analyse & Design [25]. Til udvikling
af systemet havde vi et tæt samarbejde med to
informanter, der hjalp os med at fortolke og
revidere problemstillingen. Informanterne var
en stor del af analysen, designet og kvalitetssik-
ringen af systemet.

I konklusionen har vi konkluderet, at **foodl** gør
det lettere for de madansvarlige at få inspira-
tion til at genbruge madrester. Vi har, i per-
spektivering, reflekteret over, hvad der kan
gøre systemet endnu bedre.

Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Forord

Denne rapport beskriver udviklingen af systemet **foodl** - en webapplikation, som er tilgængelig på <http://foodl.dk>. **foodl** er et system, der gør det lettere at få inspiration til, hvordan man kan anvende sine madrester, så man kan undgå at smide dem ud. **foodl** søger blandt opskrifter, fra Arla Foods hjemmeside <http://arla.dk>, og præsenterer de opskrifter, som indeholder flest af de madvarer, som brugeren har indtastet. Det er nødvendigt for brugeren at følge et link til Arla's hjemmeside, for at få vist hele opskriften og dens fremgangsmåde.

Udviklingsprocessen er baseret på teknikker fra Objektorienteret Analyse & Design [25], teknikker til Design og Evaluering af Brugergrænseflader [4] og inddragelse af to informanter, Merete og Keld, der har været med til at udforme systemet og afprøve dette. Informanterne er blevet inddraget ved forskellige aktiviteter, herunder interviews, test af prototyper og en usability-test af det endelige system.

Projektet er delt op i to overordnede dele, der er rapporten er præsenteret som: "Del 1 - Udviklingsrapport" og "Del 2 - Akademisk rapport".

Udviklingsrapporten beskriver processen og de tanker og overvejelser, vi har gjort os i forhold til udviklingen af systemet. Det er her, vi dokumenterer hele processen af systemudviklingen af **foodl**. I den akademiske rapport beskriver, reflekterer og perspektiverer vi de udviklingsmetoder og værktøjer, som vi har anvendt i projektet.

Logoet på forsiden repræsenterer programmets navn, Foodl.

Systemets kildekode kan findes via følgende link: <https://bitbucket.org/Acolarh/p3-project>

Tak til

- Informanterne, Merete og Keld
 - Tak for samarbejdet igennem hele projektet
- Arla
 - Tak for brug af opskrifter

Indhold

I Udviklingsrapport	1
1 Indledning	3
1.1 Problemformulering	3
2 Metode	5
2.1 Evolutionær metode	5
2.2 Samarbejde med brugere	6
3 Systemvalg	9
3.1 Situationen hos informanterne	9
3.2 Eksisterende systemer	10
3.3 Systemdefinition	16
4 Analyse af problemområdet	19
4.1 Klasser	19
4.2 Struktur	21
4.3 Adfærd	22
4.4 Sammendrag	25
5 Analyse af anvendelsesområdet	27
5.1 Brug	27
5.2 Funktioner	34
6 Design af arkitektur	37
6.1 Kriterier	37
6.2 Komponenter	39
7 Design af brugergrænsefladen	45
7.1 Prototyper	45
7.2 Brugergrænseflade	48
8 Implementering	57
8.1 Teknologier	57
8.2 Funktionalitet	58
8.3 Dataudtrækning	63
9 Kvalitetssikring	73
9.1 Unit test	73
9.2 Test af usability	74
10 Konklusion	77
10.1 Perspektivering	78

II Akademisk rapport	79
11 Indledning	81
12 Samarbejde med informanter	83
12.1 Møder med informanter	83
12.2 Prototyper og usabilitytest	84
13 Udviklingsmetoden	87
13.1 Objektorienteret Analyse & Design	87
13.2 Den evolutionære udviklingsmetode	87
14 Ruby on Rails	91
Litteratur	93
Bilag	99
A Fravagte klasser og hændelser	99
B Fravagte brugsmønstre	101
C Prototyper og møder med informanter	103
D Manuel mapping	113
E Valg af mapping metode	115
F Usabilitytest	117

Del I

Udviklingsrapport

1 Indledning

Madspild er et verdensomspændende problem[21]. Tal viser, at industrien (supermarkeder, restauranter o.l.) i mange vestlige lande smider over 30%, af alt det mad landet producerer, ud i affaldsspanden[21]. Ofte er der tale om spiseligt mad. Hvad er årsagen til, at så meget mad bliver smidt ud? En af grundene er eksempelvis, at hvis en fødevarer ser forkert ud eller ikke har den rigtige størrelse (eksempelvis en agurk, der er for skæv), så bliver fødevareren bare smidt ud, inden den når frem til f.eks. supermarkedet [35].

En anden grund til det enorme madspild, finder sted i de private husstande. I Danmark findes der omkring 2,6 millioner husstande [34], der dagligt skal få madlavningen til at gå op i en højere enhed. Dette kan ofte være en svær opgave i en travl og stresset hverdag, hvor der sjældent er tid til at kredse om maden, eller skænke madspild en tanke. Det sker derfor ofte, at danskerne, i deres iver på at få handlet hurtigt ind og få styr på dagens madlavning, køber for meget mad, og glemmer at få anvendt det mad, de i forvejen har i køleskabet. Resultatet af dette, kan ses i tal fra Politiken, som viser, at personer, der bor i parcelhuse, i gennemsnit smider 42 kilo mad ud om året [29]. Der er flere årsager til det store madspild blandt privatpersoner. Til et foredrag vedrørende madspild ved TED-konferencen kommenterer en bruger:

“I am a single person and I do waste a lot simply because of packaging. I don't know what % of people are single but I either have to freeze stuff if I can or eat the same dish for 2-3 days [33].” - Dee Simmons

Der ligger tydeligvis også et problem i størrelsen på de portioner, supermarkederne stiller til rådighed. Hakket oksekød findes typisk kun i pakker med 400-500 gram som det mindste. Til en enkelt person er dette ofte for meget, og man risikerer derfor at stå tilbage med halvdelen, som enten skal bruges dagen efter eller fryses ned, hvis ikke det skal gå til spilde. Idet cirka 40 % af alle husstande i Danmark, er beboet af en enkelt person [20], er der tale om et større problem.

Det er naturligvis ikke realistisk, at forestille sig en verden uden madspild. Dette ville kræve en enorm mængde tid og planlægning, som er urealistik for de fleste danskere. Størrelserne på supermarkedernes indpakning, styres af supermarkederne selv, og er modelleret efter, hvordan de kan få størst mulig indtjening. Det er ikke noget den enkelte kan styre. Til gengæld må det være muligt at mindske den enkeltes madspild, ved at gøre madlavning og anvendelse af de madrester, man allerede har i køleskabet, nemmere. Hvis man vil benytte madresterne og varerne i sit køleskab, kan det være svært og tidskrævende at finde en opskrift, hvori disse indgår. En metode, kunne være at bladre sine kogebøger igennem, men dette ville hurtigt blive uoversuelig og tidskrævende, da fremgangsmåden ved opslag i kogebøger er: 1) søge efter opskrift, som man har lyst og tid til at lave; 2) kigge opskriftens ingrediensliste igennem for at se om de madrester og varer, man har i køleskabet, indgår. Hvis det er tilfældet, at der mangler for mange ingredienser, så skal man søge efter en ny opskrift.

1.1 Problemformulering

De enorme mængder af spiselige fødevarer, der årligt bliver smidt ud, er en kæmpe økonomisk og miljømæssig flaskehals. Madspillet ude i industrien og i husstande er to vidt forskellige områder, der begge har det samme problem. I begge områder er madspildsproblemets enormt, men man bliver nødt til at fokusere på ét område, fordi vi mener, at disse områder skal løses på to vidt forskellige måder.

Vi er en gruppe bestående af datalogistuderende og temaet for dette projekt er “udvikling af applikationer - fra brugere til data, algoritmer og test - og tilbage igen”. Det betyder, at der er fokus på systemudvikling igennem projektforløbet. Vi har formuleret en problemformulering, der lyder således:

Hvordan kan man ved hjælp af et system gøre det muligt at anvende madrester i de danske husstandes madlavning?

2 Metode

Gennem projektforløbet har der været brug for planlægning og strukturering af arbejdsopgaver for at få projektet til at fungere og gå op i en højere enhed. Dette kapitel beskriver, hvilke arbejdsmetoder vi har brugt, og hvordan vi har samarbejdet med vores informanter. Informanterne skal ses som potentielle brugere af det færdige system, der er produktet af projektforløbet.

Ydermere har vi brugt udvalgte metoder og aktiviteter fra bogen, Objektorienteret Analyse & Design (OOAD) [25], hvis formål er at fastlægge systemspecifikationer. Det betyder, at OOAD-bogen er blevet brugt til at udvikle og forstå et system, dets kontekst og de betingelser, der forudsætter dets implementation. Bogens metoder gør det nemmere at udvikle et system, hvor det er veldokumenteret og overskueligt, hvordan en systemudviklingsprocess har foregået, og hvordan systemet løser et givent problem.

2.1 Evolutionær metode

Det er vanskeligt at planlægge et helt projekt for et større problem. Det kræver meget forarbejde, og man bliver nødt til at fortolke og revidere problemstillingen mange gange, indtil man har tilstrækkelig viden og forståelse for problemet. Dette gøres på bedste vis ved at have tæt kommunikation og interaktion med potentielle brugere af det fremtidige system. Vi har udviklet et system til brugerne, og det er til sidst dem, der er i direkte kontakt med systemet. Derfor er det vigtigt, at de kan finde ud af at bruge systemet, og på samme tid udvikler vi en bedre forståelse for problemet igennem interaktionen med brugerne.

Den evolutionære metode, bygger på en iterativ arbejdsproces, hvor der gennem hver iteration sker en udvikling i forståelsen af et problem. Metoden bygger på eksperimentering med bl.a. prototyper til at skabe sig en forståelse for problemet. Arbejdsmetodens iterationer kaldes også for faser. I hver fase reviderer og bearbejder man analysen, designet af systemet, implementering og kvalitetssikringen [14]. Når vi siger, at vi har benyttet en iterativ arbejdsproces, så betyder det ikke, at der ikke også har været konstruktive tilgange i planlægningen. Men det betyder, at hvis man ser på den samlede udviklingsproces, så er det den iterative tilgang, der dominerer.

Projektet strakte sig over fire måneder, og vi vurderede, at fire faser af tre ugers varighed var en fornuftig opdeling af tiden, og at det gav os tilstrækkelig tid til hver fase. Disse fire faser kan ses i tabel 2.1. Hver fase havde et formål, og et resultat. Tabellen har til formål at give læseren et hurtigt overblik over, hvad der er foregået i de forskellige faser. I tabellen står der eksempelvis hvilke dele af rapporten, der er blevet tilføjet eller revideret i den enkelte fase. I den første fase er der ikke noget at revidere, da vi ikke havde udført noget arbejde endnu. Derudover beskriver tabellen, hvordan vi har interageret med informanterne.

Informanterne, der bliver beskrevet i afsnit 2.2, hjalp os bl.a. med at forstå problemet og hvordan en eventuel løsning på problemet skulle udformes. Vi holdt et initierende møde med dem, for at få en bred forståelse af problemet. Vi præsenterede informanterne for prototyper, der havde til formål at visualisere, hvilke tanker vi havde gjort os i forhold til en eventuel løsning, og vi ønskede at undersøge, hvordan informanterne integrerer med disse. Vi havde to prototyper, hvis fokus lå på søgefunktionaliteten i systemet og en prototype, hvis fokus lå på andre funktioner, systemet kunne have. Prototyperne er beskrevet yderligere i afsnit 2.2.

2. Metode

Fase	Formål	Beskrivelser	
		Resultat	Informanternes inddragelse
1	At få indblik i informanternes problemstillinger, mht. madlavning og anvendelse af deres madrester, og at modellere disse. At definere et system, og forstå hvilke funktioner informanterne har brug for.	<p><i>Tilføjet:</i> Problmeområdet. Klasser. Hændelser. Hændelsestabell. Klassestruktur. Prototype med fokus på søgefunktionalitet.</p>	Møde med fokus på problemet. Møde med fokus på løsning. Prototype med fokus på søgefunktionalitet.
2	At sikre os, at de funktioner vi har i systemet passer med informanternes behov og at dokumentere og modellere disse.	<p><i>Tilføjet:</i> Problemområdet. Anvendelsesområdet. Brugsmønstre. Funktioner. Aktører. Kriterier. Forbilleder. <i>Revideret:</i> Problemområdet. Prototype med fokus på funktionalitet.</p>	Prototype med fokus på funktionalitet.
3	At modellere systemet, ved at implementere funktioner og sikre os, at de definerede kriterier bliver opfyldt.	<p><i>Tilføjet:</i> Implementering. Sammensætning af rapport. Komponenter. Udtrækning af data i opskrifter. <i>Revideret:</i> Problemområdet. Anvendelsesområdet. Design.</p>	
4	At implementere det designede system.	<p><i>Tilføjet:</i> Implementering. Kvalitetssikring. <i>Revideret:</i> Problemområdet. Anvendelsesområdet. Design.</p>	Bestemmelse af råvaretype fra ingrediens. Usability-test.

Tabel 2.1: Denne tabel giver et hurtigt og kortfattet overblik over projektets arbejdsfaser. Her ses de forskellige faser af den iterative arbejdsmetode med tilhørende formål og de resultater, vi har fået ud af de forskellige faser. Desuden kan man se, hvordan informanterne er blevet inddraget i processen.

2.2 Samarbejde med brugere

Gennem projektet har vi arbejdet sammen med to informanter, som vi vurderede kunne være potentielle brugere af et system, der kan løse problemet med madspild i danske husstande. Dette afsnit har til formål at præsentere disse to informanter for læseren og beskrive, hvordan vi har interageret med dem igennem projektforløbet. De to informanter vil blive refereret til via navnene Merete og Keld igennem hele rapporten.

Merete er en familiemor, der bor med sin mand. De har fire børn, som alle er flyttet hjemmefra. Merete står for madlavningen i husstanden. Både Merete og hendes mand har et arbejde, der skal passes, derfor føler de, at det ofte er svært at planlægge madlavningen i forvejen.

Keld er en familiefar, der bor med sin kone og deres to små døtre. Keld står for både madlavningen og indkøb af varer til familien. Familien føler ikke, at de har meget fritid i hverdagen, derfor vælger Keld at lave store portioner aftensmad, så han ikke behøver at lave mad hver aften. Både Keld og hans kone har et arbejde, der skal passes, og når de kommer hjem fra arbejde, så ønsker de at bruge så meget tid som muligt på at være sammen med deres døtre.

Begge informanter var med til at gøre det muligt for os at fortolke og forstå problemstillingen set fra deres perspektiv. De var en stor del af afprøvnings- og kvalitetssikringen af systemet. Denne samarbejdsproces er kort illustreret i tabel 2.1, hvor man kan se, hvorledes informanterne er blevet inddraget i arbejdet i de forskellige faser.

Indledningsvis mødtes vi med informanterne hver for sig, og diskuterede problemet omkring madspild, og hvordan de oplevede madlavningen i deres respektive husstande. Efter vi havde diskuteret og fortolket problemet, var vi nu i stand til at formulere to systemdefinitioner, beskrevet i afsnit 3.3, som vi kunne præsentere for informanterne. Der var nu tale om møder, hvor fokus lå på, hvordan vi kunne løse dette problem, vi havde diskuteret i de foregående møder.

Efter disse initierende møder, havde vi brug for at afprøve vores idéer og tanker vedrørende løsningsmetoder. Dette gjorde vi ved at udarbejde prototyper af de idéer, vi havde fået som resultat af diskussionerne med informanterne samt arbejdet med problemet. Tidligt i forløbet brugt vi pappersprototyper, som ikke krævede meget tid eller mange ressourcer at fremstille. Dette er en prototype, der er lavet af papir, som styres af et gruppemedlem, mens informant lader som om, at der bliver trykket på en knap. Alt efter, hvad informanten gør, ændrer prototypen sig i forhold til denne interaktion.

Lidt længere inde i projektforløbet benyttede vi diashow-prototyper, der havde til formål at undersøge systemets funktionalitet. Med sådan en prototype blev informanterne præsenteret for en prototype, der var dynamisk. Man kunne klikke på knapper og navigere rundt i diashowet. Denne interaktion foregik på en computer. Begge prototyper er forklaret yderligere i kapitel 7, hvor systemets brugergrænseflade bliver præsenteret. Systemets design er blevet til som følge af de afprøvninger, vi havde foretaget med prototyper.

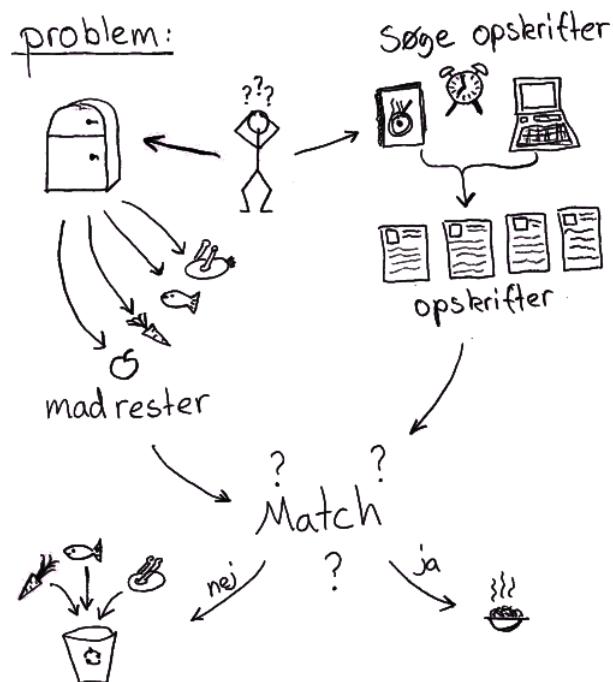
Afslutningsvis præsenterede vi det funktionsdygtige system for informanterne. Systemet var baseret på alle de foregående prototyper og designidéer, som var blevet præsenteret for og diskuteret med informanterne. Ved at præsentere informanterne for det funktionsdygtige system, blev vi i stand til at sikre os en vis kvalitet, inden vi afsluttede projektet og produktudviklingen. Til de sidste afprøvninger var der mulighed for at opdage kritiske fejl, i og med vi lod personer, der ikke havde været med til at udvikle systemet, bruge og teste det. Her kan brugeren foretage sig nogle valg i systemet, som vi måske ikke havde overvejet. Disse uforudsete valg skulle ikke være skyld i, at systemet blev ubrugelig eller gik ned, og denne sidste afprøvning var en god sikring mod dette. Disse afsluttende test bliver præsenteret i afsnit 9.2.

3 Systemvalg

Dette kapitel beskriver, hvordan vi er kommet frem til de overordnede egenskaber, som vores system skal have. Dette blev opnået vha. interviews med informanterne. På baggrund af interviewene, udformede vi systemdefinitioner, som definerede systemer, der havde til formål at mindske madspild. Til sidst vælges den endelige systemdefinition, der stemmer overens med informanternes behov og BATOFF-kriterierne, som er forklaret yderligere i afsnit 3.3.

3.1 Situationen hos informanterne

I de kommende små afsnit vil vi forklare, hvordan situationen er hos vores to informanter.



Figur 3.1: Rigt billede, der visualiserer problemet ved at skulle genbruge madrester.

Det rige billede i figur 3.1 viser en person, der ikke har tid eller ressourcer til at finde ud af, hvordan madresterne, som er i køleskabet, kan blive brugt i madlavningen. Dette rige billede er blevet udarbejdet ud fra de initierende samtaler, vi har haft med vores to informanter. Billedet har hjulpet os til at få en forståelse for situationen.

Madansvarlige i husstanden

Begge informanterne står for madlavningen i deres respektive husstande. De er begge opmærksomme på, at der er dele af aftensmaden, der bliver smidt ud. Denne madspild forekommer selvom, de prøver at genbruge madresterne ved bl.a. at fryse madresterne ned eller genbruge dem den kommende dag i f.eks. biksemad, supper osv. Det forekommer ofte, at familierne får gårdsdagens rester til dagens aftensmad.

Madvaner og madlavning

Merete har været vant til at lave aftensmad til to sønner og en datter ud over hende selv og hendes mand. Ægteparrets børn er flyttet hjemmefra, og de spiser derfor sjældent med hos forældrene mere. Det er svært at få mængden af aftensmad til at passe, når der ikke er ligeså mange, som der plejer. Merete er også meget opmærksom på holdbarhedsdatoerne på de forskellige madvarer. Hvis den dato bliver overskredet, så bliver maden smidt ud med det samme.

Derudover forklarer Keld, at han med vilje laver ekstra store portioner til aftensmaden, så familien kan få resterne fra dagens aftensmad den næste dag. Denne strategi benytter Keld sig af, fordi han mener, at der ikke altid er meget tid tilovers til madlavningen. Ægteparret har to små børn, der skal passes og bruges tid på. Udover at tage sig af børnene, så har de også hver deres job, som skal ses til. Derfor er tid ikke noget, som ægteparret har meget af, og de bruger lignende tricks til at bruge mindre tid på madlavningen og mere tid på at være sammen. Det er helt tydeligt, at tiden er en vigtig faktor for Kelds familie, og det er netop derfor, at familien ofte får de samme retter til aftensmad.

Indkøb

Når det kommer til indkøb af madvarer, så er det ikke altid, at der bliver brugt en indkøbsseddel til at planlægge indkøbet. Den person, der har tid, handler ind. Merete og hendes mand kan bedst lide at gå på opdagelse i supermarkedet, og se om de kan finde nogle gode tilbud, som de kan lave noget aftensmad ud af. Keld derimod står altid for indkøb, og han har ofte en plan i hovedet eller en liste i hånden over, hvad han skal have købt med hjem til aftensmaden.

Planlægning

Når ugens aftensmad skal planlægges, så benytter hverken Merete eller Keld sig af en madplan. De har ofte idéerne til aftensmaden i hovedet, og madlavningen er rutinepræget. Aftensmaden er meget ensformig, fordi fremgangsmåden er velkendt og derved nem og hurtig at lave. Derudover er det svært for familerne at planlægge tidspunktet for aftensmaden, fordi de alle har jobs, der skal ses til. Derfor ændrer deres planer sig pludseligt, og det vil være svært at styre en madplan, når arbejdstiderne kan variere.

3.2 Eksisterende systemer

På baggrund af samtalerne med informanterne, har vi udarbejdet systemdefinitioner og udvalgt en af disse, som vi har baseret vores videre arbejde på. For at være i stand til at udarbejde systemdefinitioner effektivt, valgte vi at undersøge nogle eksisterende systemer, der forsøger at løse en lignende problemstilling, som den vi arbejdede med.

Der eksisterer allerede en lang række systemer, som tilbyder en service, der ligner den, som vi ønsker at kunne tilbyde. Heriblandt valgte vi at undersøge eksisterende danske systemer, nemlig Forbrugerrådets For Resten [19], DK-kogebogen [3] og Opskrifter.dk [26]. Der eksisterer desuden også en række engelske hjemmesider, som vi ikke var interesseret i. Dette er grundet, at de danskssprogede anses som værende direkte konkurrenter til [foodl](#). For Resten er en mobilapplikation kun tilgængelig på iOS- og Android-smartphones. DK-kogebogen og Opskrifter.dk er webapplikationer tilgængelig på internettet. Efter de initierende møder med informanterne blev der diskuteret nogle kriterier, som de synes er vigtige for brugervenligheden og funktionaliteten, der gør opskriftshjemmesider så tiltrækende som muligt. Vi besluttede os for, at der var nogle egenskaber, der er mere relevante at undersøge end andre. De følgende egenskaber så vi nærmere på i de forskellige systemer:

- Antal opskrifter
- Kvalitet af opskrifter
- Fleksibilitet

- Opskriftssøgningsfunktion

Det er relevant at undersøge, hvor mange opskrifter systemerne har i deres databaser. Jo færre opskrifter de har, jo større er risikoen for, at man, som bruger, får nogle tomme resultater, når man søger efter opskrifter med en specifik ingrediens.

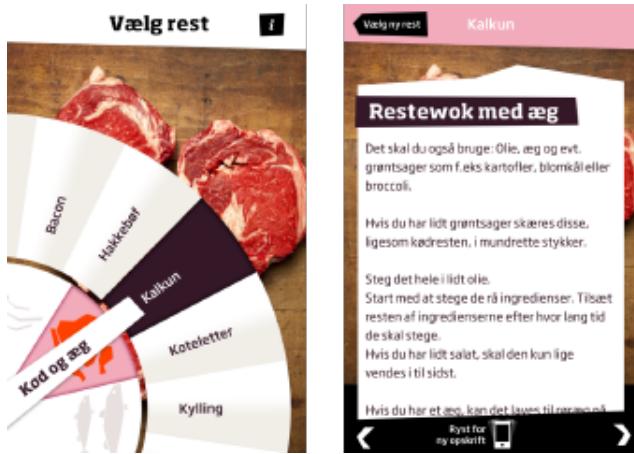
Kvaliteten af opskrifterne er også vigtig. Hvis systemet tillader alle og enhver at uploadere deres opskrifter til hjemmesiden, så er der en risiko for, at opskrifter vil være fejlfyldt og uden billede eller ligefrem ubrugelige, hvis der blot mangler et trin i fremgangsmåden. Oplever brugeren, gang på gang, fejl i opskrifter eller opskrifter med manglende indhold som søgningsresultater, så vil sidens troværdighed mindske.

Hjemmesidens fleksibilitet blev vurderet ud fra om brugeren har mulighed for f.eks. at op- eller nedskalere opskrifter således, at de er tilpasset flere eller færre personer; om det er muligt at sortere efter tilberedningstid eller andre ting, og om brugeren har mulighed for at sætte begrænsninger op for, hvilke opskrifter han/hun ønsker skal vises (f.eks. kun opskrifter uden svinekød, laktose, nødder osv.).

Den sidste funktion, som vi undersøgte, var opskriftssøgningsfunktionen. Her undersøgte vi, hvordan de tre systemer havde valgt at bygge deres tøm-køleskabs-funktion op, f.eks. hvordan man søger på ingredienser, eller hvor lettilgængelig funktionen er mm.

3.2.1 For Resten

For Resten er en gratis mobilapp, der er udgivet af Forbrugerrådet som en del af en kampagne mod madspild. App'en findes til Android og iOS og kan installeres fra henholdsvis Google Play og App Store. App'en fungerer ved, at man på to "hjul" vælger kategori ("Kornprodukter", "Mejeriprodukter", "Kød og æg" og fem andre kategorier) og rest (f.eks. "Mørbrad", "Kyrring", "Kødsovs" osv.), hvorefter brugeren præsenteres for en række opskrifter, som inkluderer den valgte rest. To skærmbilleder af brugergrænsefladen kan ses i figur 3.2.



Figur 3.2: Brugergrænseflade i For Resten. Valg af rest (t.v.) og visning af opskrift (t.h.).

For hver rest er der ca. 4 - 5 opskrifter, hvilket med 124 rester giver et samlet antal opskrifter på omkring 500 - 600. Der vises kun fremgangsmåde for opskrifterne, og ikke en ingrediensliste, hvilket vi mener, er en væsentlig del af en opskrift. Det er derfor heller ikke muligt at op- eller nedskalere opskriftsportionerne. Derudover er der ingen mulighed for at favorisere opskrifter eller på anden måde gemme resultatet af en søgning. I forhold til søgningen så kan man, som sagt, kun søge på én enkelt rest, og ikke sammensætte rester i form af flere råvarer, som ved andre løsninger. Samtidig har man kun mulighed for at vælge sin rest på hjulene, og har altså ikke muligheden for selv at skrive i et felt. Under afprøvningen var det derfor i nogle tilfælde svært at finde den ønskede ingrediens. Det er de samme problemstillinger, der kommer til udtryk i brugernes anmeldelser af app'en på Google Play. F.eks. skriver brugeren *TJA* [36]:

3. Systemvalg

“Fin ide, men den burde være blevet kålet lidt mere for, før den røg i play. Hvem laver restemad af én rest og så en masse ting der skal købes? Jeg har brug for at kunne søge på tre-fire rester for at se, hvordan de kan kombineres til noget spændende. Og hvis jeg finder en opskrift, jeg vil prøve, så kan jeg ikke gemme den i app’en, men skal søge den frem igen når jeg står i netto og skal købe de ting, der skal til - og søger den frem igen, når jeg skal lave maden. Alt for besværligt.”

Sammenlagt har app’en på Google Play bedømmelsen 2,4 stjerner ud af 5, baseret på 69 bedømmelser, hvoraf næsten halvdelen kun har givet app’en 1 stjerne. Et andet kritikpunkt, der kommer til udtryk i flere anmeldelser, er, at app’en bruger for mange systemressourcer og er langsom til at starte op.

3.2.2 DK-Kogebogen

DK-Kogebogen er en webapplikation, som tilbyder en lang række funktioner, hvoraf “tøm kåleskabet” er en af dem. Udover “tøm kåleskabet” tilbyder DK-kogebogen også en ugentlig madplan, et kæmpe opskrifterregister, en ekstern hjemmeside med fokus på viden omkring mad (energiindhold, vitaminindhold osv.) og en kalorieberegnere. Siden har et omfang på over 36.000 opskrifter, hvilket er det klart største antal blandt de undersøgte systemer. De mange opskrifter er indsendt og lavet af brugere af hjemmesiden. Dette er både positivt og negativt. Der kan forekomme en del fejl i opskrifterne, hvis de ikke først bliver gennemrettet. DK-Kogebogen skriver på deres hjemmeside, om de indsendte opskrifter:

“Opskriften kan ses med det samme, men der vil senere blive rettet lidt til i teksterne.” [15]

Det er altså muligt for alle at indsende opskrifter, som er fulde af fejl, og disse vil alligevel være synlige på siden. Til gengæld skaber det mulighed for, at den enkelte bruger føler sig mere knyttet til siden, da han/hun har et personligt engagement i den, fordi de indsender deres egne opskrifter; og dette vil resultere i en overordnet højere brugeraktivitet. Det skal noteres, at de 36.000 opskrifter ikke er unikke. Det vil sige, at der godt kan eksistere flere forskellige varianter af den samme opskrift; f.eks. er der 17 forskellige varianter af Boller i Karry, hvilket kan virke uoverskueligt.

“Tøm kåleskabs”-funktionen er placeret i toppen af DK-Kogebogens forside og er vist i figur 3.3. Her kan brugeren indtaste så mange ingredienser der er brug for, inden for en grænse på 55 karakter. For at systemet kan identificere ingredienser, skal der mellem hver ingrediens, som brugeren indtaster, være et mellemrumstegn. Begrænsningen på 55 tegn betyder, at der maksimalt kan søges på ca. 8-10 ingredienser af gangen (forudsat at ingrediensnavne er 5-7 bogstaver lange).



Figur 3.3: DK-Kogebogens “Tøm kåleskabet”-funktion. Funktionen er tilgængelig i toppen af forsiden og enhver anden underside.

Efter brugeren har søgt på opskrifter med en eller flere specifikke ingredienser, viser DK-Kogebogen en liste med alle de opskrifter, som indeholder ingredienserne. På figur 3.4 ses en del af den liste, som er resultatet ved søgning på “Tomat Paprika Kartoffel”. Ud fra opskrifterne kan der findes et kamera-symbol og/eller et dokument-symbol, som respektivt fortæller brugeren, om der er et billede af den pågældende opskrift, og/eller at opskriftens næringsindhold er beregnet og vist. Brugeren vælger derefter en af opskrifterne han/hun finder mest interessant.

Karakter - Stemmer	
9 - 1	Bacondolmere med italiensk sovs og ris
8 - 2	Bagt kartoffelmos med kædsauce
7.3 - 69	Barbecue-laks med lun kartoffelsalat
9.8 - 8	Bjørns oksehaleragout
8.6 - 121	Bif Stroganoff med kartoffelmos (klassisk)
8.3 - 4	Gulasch med Kartoffelmos
6.6 - 5	Heilstegt oksetyksteg med svømmende kartoffelsalat, grillede ferskner, majskolber og caesar salat
10 - 1	Heilstegt oksetyksteg med svømmende kartoffelsalat, grillede ferskner, majskolber og caesar salat
9 - 1	Isbjørne stroganoff

Figur 3.4: Liste af opskrifter, som indeholder ingredienserne Tomat, Paprika og Kartoffel.

Inde på opskriftsvisningsiden er det i nogle tilfælde muligt at op- eller nedskalere portionsstørrelsen, mens der i andre tilfælde skaleres på antal personer. Derudover er der også nogle opskrifter, hvor det slet ikke er muligt at skalere. Her er brugeren i stedet for tvunget til selv at finde ud af, hvor stor en portion opskriften ca. passer til. Der er altså ikke overensstemmelse med, hvad portionerne skal angives i. Denne mangel på konsistens er endnu en af problematikkerne ved, at det er brugerne selv, som indsender opskrifterne. En endelig vurdering af DK-Kogebogen, findes i afsnit 3.2.4.

3.2.3 Opszkrifter.dk

Opszkrifter.dk er endnu en webapplikation, der tilbyder en “Tøm køleskabet”-funktion, som kan bruges til at finde opskrifter i deres samling af opskrifter. Ligesom i For Resten kan ingredienser kun vælges fra kategorier, dog er der i dette system hele 624 ingredienser, fordelt over 27 kategorier. Modsat For Resten kan man her vælge mere end én ingrediens. Dette gøres ved først at vælge en kategori og derefter finde og vælge en ingrediens og klikke på knappen “Tilføj >”. Man kan ligeledes fjerne allerede valgte ingredienser ved at markere dem og klikke på knappen “< Fjern”, eller fjerne alle valgte ingredienser ved at klikke på knappen “< Fjern alle”. Når man har valgt de ingredienser, man ønsker at inkludere, kan man foretage sin søgning ved at klikke på knappen “Søg”.

The screenshot shows the user interface of the Opskrifter.dk website. At the top, there's a navigation bar with links for 'Forside', 'Opskrifter', 'Drinks', 'Sundhed', 'Mit område', 'Om Opskrifter.dk', 'Sitemap', 'Svenska', and 'Dansk'. Below this, a breadcrumb trail indicates the user is at 'Du er her: Opskrifter > Tøm køleskabet'. On the left, a sidebar menu lists various sections like 'Opskrifter', 'Grill Tema', 'En nemmere hverdag', 'Spørg Kokken', 'Brugersopskrifter', 'Film fra køkkenet', 'Menuer', 'Ingredienser', 'Kokkens Hjørne', 'Kokken fra Samsoe', 'Tom køleskabet', 'Kaffe Hjørnet', 'Artikler', and 'Den Daglige'. Under 'Tom køleskabet', there are links for 'Opskrift søgning' and 'Log ind'. The main content area has a title 'Tøm køleskabet'. It features three dropdown menus labeled '1. Kategori:', '2. Ingredienser:', and '3. Valgte ingredienser:'. The first dropdown shows categories like 'ALLE KATEGORIER', 'Brod', 'Diverse', 'Drikkevarer, Alkoholiske', etc. The second dropdown shows ingredients like 'A38', 'Abrikos', 'Abrikosmarmelade', 'Agurk', 'Ahornsirup', etc. The third dropdown shows selected ingredients like 'Haj', 'Hamburgerryg', 'Krabbe', 'Peberfrugt', 'Skumfiduser', and 'æg'. Arrows point from the first dropdown to the second, and from the second to the third. Buttons for 'Tilføj >', '< Fjern', and 'Søg' are located at the bottom of each list.

Figur 3.5: Brugergrænsefladen for Opszkrifter.dk's “Tøm køleskabet”-funktion.

Søgningen foretages blandt de ca. 2700 opskrifter, som er tilgængelige på Opszkrifter.dk. Resultaterne vælges ud fra, om de inkluderer minimum én af de valgte ingredienser. Under hver opskriftnavn vises antallet af valgte ingredienser,

3. Systemvalg

som opskriften inkluderer, men det er umiddelbart ikke muligt at sortere resultaterne efter dette tal. Klikker man på et resultat, åbnes den valgte opskrift til højre for resultaterne, og altså ikke på en ny side eller i et nyt vindue/faneblad. Opskrifterne viser informationer såsom tilberedningstid samt alle ingredienserne og deres mængder. Det er på alle opskrifter muligt at skalere opskriften til et bestemt antal personer. Figur 3.6 viser et eksempel af et søgningsresultat.

Kvaliteten af opskrifterne er forholdsvis høj og ca. 40 % af alle opskrifter er med billede. Dette skyldes sandsynligvis, at det ikke er muligt for almindelige brugere direkte at indsende opskrifter. Almindelige brugere kan derimod indsende opskriftforslag, som først skal gennemlæses og tilføjes af en administrator.

The screenshot shows the search results for "Tøm køleskabet". The main search interface includes a search bar, a button for "Ny søgning", and sorting options. Below this, a list of seven recipes is displayed:

- Filopakker med krabbefyld (3 af dine ingredienser)
- Baskisk skinke og æg (2 af dine ingredienser)
- Calamarisalat (1 af dine ingredienser)
- Den lækkre Citrontærte (1 af dine ingredienser)
- Fastelavnsboller (1 af dine ingredienser)
- Fransk citrontærte

On the right, a detailed view of the first recipe, "Filopakker med krabbefyld", is shown. It includes the title, a small image, a star rating of 3, and a brief description. Below this, there are buttons for "Send opskrift til ven", "Anmeld opskrift", and "Læg i min kogebog". The detailed view also shows the recipe's type (Hovedret), preparation time (1 time), cooking time (30 min), and ingredient list for 2 persons:

Ingredienser:	2 pers.
4/5 spsk olivenolie	(Læs om)
1/3 stk kartoffel	(Læs om)
1/3 stk rød peberfrugt	(Læs om)
1/3 stk løg	(Læs om)
1/3 stk fennikelknold	(Læs om)

Figur 3.6: Resultatsiden vist ved søgning med Opskrifter.dk's "Tøm køleskabet"-funktion. Her er markeret en opskrift, der ikke indeholder et billede af retten.

Opskrifter.dk har også et brugersystem, der tillader brugere at registrere sig og logge ind. Dette giver mulighed for, at man bl.a. kan gemme de opskrifter, man har fundet (ved at klikke på "Læg i min kogebog"). Derudover husker "Tøm køleskabet"-funktionen de ingredienser, man har indtastet, til næste gang man besøger siden. Under "Tøm køleskabet"-funktionen er det også muligt for brugeren at skrive kommentarer til funktionen. Disse kommentarer har givet et indblik i, hvad Opskrifter.dk's brugere synes om funktionaliteten. Nogle kommentarer går på manglende ingredienser, mens en stor del kommentarer går på, at funktionen finder opskrifter, som man ikke kan lave uden at skulle købe en masse ind. F.eks. skriver brugeren Jytte Hasselriis:

"Hvordan skulle jeg kunne lave fasan i flødesovs, når jeg ikke har en fasan i køleskabet?????" [22]

Dette synes at udtrykke en vis utilfredshed med den måde, hvorpå Opskrifter.dk's "Tøm køleskabet"-funktion vælger resultater på.

3.2.4 Sammendrag

De tre systemer; For Resten, DK-Kogebogen og Opskrifter.dk er i foregående afsnit blevet undersøgt og analyseret. Der blev lagt vægt på fire hovedpunkter: antallet af opskrifter i systemet, kvaliteten af opskrifterne, systemets

fleksibilitet og opskriftssøgningsfunktionen (også kaldet “Tøm køleskabet”-funktionen). De fire hovedpunkter er specifiseret i detaljer i afsnit 3.2. For at skabe et samlet overblik, har vi valgt at samle de vigtigste og mest karakteristiske dele fra hver af systemerne i tabel 3.1.

Funktioner	System		
	For Resten	DK-kogebogen	Opskrifter.dk
Kvalitet af opskrifter	ringe	svingende	god
Antal opskrifter	550	36.500	2.700
Fleksibilitet	meget ringe	middel	god
Opskriftssøgningsfunktion	meget ringe	middel	middel

Tabel 3.1: *Oversigt over opfyldelse af kriterier af de eksisterende systemer.*

Kvalitet af opskrifter

Kvaliteten af opskrifterne på DK-Kogebogen er meget varierende. Brugeren risikerer at støde på opskrifter, som er ubrugelige.

Kvaliteten af opskrifterne på For Resten kunne være meget bedre. Opsiakterne er udelukkende lavet eller tilføjet af folkene bag app'en, og derfor er opskrifternes opbygning og design konsistent, hvilket naturligvis havde været en god egenskab, hvis det ikke var for det faktum, at opbygningen er uoverskuelig, og at der ingen billeder eksisterer af opskriften. Der er ingen ingrediensliste på opskriften og beskrivelsen af fremgangsmåden er kortfattet.

Kvaliteten af opskrifterne på Opskrifter.dk er høj. Dette skyldes, at opskrifterne bliver gennemgået af en administrator, inden de bliver tilgængelige på Opskrifter.dk's side, hvilket er modsat af DK-Kogebogen, hvor opskrifterne bliver tilgængelige med det samme. Desuden er Opskrifter.dk's opskriftsopbygning konsekvent i alle opskrifter, hvilket også er i modsætning til DK-Kogebogens opskrifter. Der mangler dog billeder på nogle opskrifter.

Antal opskrifter

Som det ses i tabel 3.1, har DK-Kogebogen det langt største antal opskrifter, mens Opskrifter.dk har ca. fem gange flere opskrifter end For Resten. Dermed har DK-Kogebogens “Tøm køleskabet”-funktion også langt bedre chance for at give brugeren et resultat, når der søges på opskrifter med specifikke ingredienser.

Fleksibilitet

Der er stor forskel på fleksibiliteten fra system til system. I For Restens app, er det slet ikke muligt at skalere portionsstørrelse. På DK-Kogebogens side er det kun muligt med nogle opskrifter, mens det på Opskrifter.dk er muligt at skalere portionsstørrelse i alle opskrifter. Denne relativ simpel funktion ses som meget brugbar, da brugeren undgår selv at skulle gange igennem mængder op.

Kun Opskrifter.dk tilbyder brugeren muligheden for at sortere i resultaterne af en opskriftssøgning. Her kan der sorteres efter alfabetisk orden, opskrifter med billeder, opskrifter med kød samt flere. Som en bruger på Opskrifter.dk dog pointerer, mangler den sorteringsmulighed, som sorterer efter de opskrifter, som indeholder flest af de ingredienser, som brugeren har indtastet. Denne sorteringsmulighed anses for os, som værende den mest relevante, da man som bruger er interesseret i at få anvendt så mange af ens madrester som muligt.

Opskriftssøgningsfunktion

De tre systemer er vidt forskellige i deres måde at håndtere søgning på. Der er mellem DK-kogebogen og Opskrifter.dk en markant forskel på, hvordan resultater findes. I DK-Kogebogen findes kun opskrifter, som inkluderer alle de indtastede ingredienser, mens Opskrifter.dk finder alle opskrifter, som indeholder bare én af de valgte ingredienser. Dvs., at man med DK-Kogebogen får færre resultater, jo flere ingredienser man skriver, mens det med Opskrifter.dk er direkte modsat, idet antallet af resultater stiger voldsomt med antallet af ingredienser, man skriver. Opskrifter.dk's måde at gøre det på, kombineret med deres manglende sortering, giver en uoverskuelig mængde af resultater, hvor en stor del af disse måske kun indeholder en af de valgte ingredienser.

3.3 Systemdefinition

En systemdefinition er en kortfattet beskrivelse af et system udtrykt i naturligt sprog. Vi har udarbejdet to systemdefinitioner, som bliver beskrevet i afsnit 3.3.1. En af disse to systemdefinitioner blev udvalgt i samarbejde med informanterne (afsnit 3.3.2). Systemdefinitionerne er udtrykt på en måde, så informanterne, der er inddraget i projektet, kan forstå det og godkende det. For at sikre os at vores systemdefinition blev så præcis som mulig, udarbejdede vi vores systemdefinition efter BATOFF-modellen, der indeholder seks kriterier[25, s. 37]:

- **Betingelser** - *betingelser for systemets udvikling og brug*
- **Anvendelsesområde** - *de organisationsdele, der administrerer/ overvåger/ styrer et problemområde*
- **Teknologier** - *den teknologi, som systemet udvikles til og ved hjælp af*
- **Objekter** - *de væsentligste objekter i et problemområde*
- **Funktioner** - *de systemdefinitioner, som understøtter arbejdsopgaver i anvendelses området*
- **Filosofi** - *den filosofi, der ligger bag IT-systemets anvendelse*

Disse kriterier har til formål at støtte udviklingen af systemdefinitioner ved at vurdere de forhold, der er gældende for et givet systems funktion i forhold til en organisation eller forbruger og omverden. Derudover benyttede vi BATOFF-kriterierne, fordi de fastsætter nogle rammer i forhold til opsætningen og indholdet af systemdefinitioner samt opretter en form for standard, der gør det muligt at sammenligne flere forskellige systemdefinitioner på en logisk måde.

3.3.1 Alternative systemdefinitioner

Vi startede med at definere selve systemdefinitionerne og derefter sikre os, at systemdefinitionerne stemmer overens med BATOFF-kriterierne. Når definitionerne var blevet udarbejdet og kontrolleret, så blev de præsenteret for informanterne.

Efter møder med vores informanter fik vi forståelsen af, at madspild er et reelt problem for de to familier. Det er blevet forklaret, hvad der ofte er grunden til madspillet, og på baggrund af møderne og det rige billede, som blev præsenteret i afsnit 3.1 i figur 3.1, er to systemdefinitioner blevet udarbejdet.

Systemdefinition 1 (S1)

Systemet skal fungere som et online opskrifsregister, der giver brugeren idéer til opskrifter, som kan laves ud fra de madvarer brugeren har. Systemet fokuserer på at mindske madspild, da forbrugere smider mad ud på grund af et manglende formål med anvendelsen af resterne. Brugerne af systemet er en del af en husholdning og vil have meget varierende erfaringer inden for brug af internettet. Udviklerne af systemet er ulønnede studerende. Deadline for det færdige system kan ikke ændres. Systemet skal køre på en server, der kan tilgås via en webapplikation fra en internetbrowser på enhver type computer. På baggrund af en mængde fødevarer som input, findes forskellige opskrifter, der bedst muligt matcher disse fødevarer. Opszifterne skal kunne sorteres på flere måder, og ingredienser skal kunne tilføjes til en indkøbsliste. Ligeledes skal man kunne gemme favoritopskrifter til senere brug.

Systemdefinition 2 (S2)

Systemet skal fungere som et planlægningsværktøj, som har til formål at planlægge madlavningen over en given tidsperiode (f.eks. dage, uger, måneder) ud fra de opskrifter, der er blevet lavet over de sidste par dage. Formålet med planlægningsværktøjet er at sikre, at brugeren får en sund og varieret kost igennem udvalgte opskrifter, hvor der også tages højde for ingrediensers vitaminindhold. Brugerne af programmet vil være husstande, der har varierende erfaringer inden for brug af internettet. Udviklerne af systemet er ulønnede studerende. Systemet skal køre på en server, der kan tilgås via en webapplikation fra en internetbrowser på enhver type computer.

Efter udarbejdelsen af systemdefinitionerne undersøgte vi, om de overholdte BATOFF-kriterierne. Vi havde indbyrdes i gruppen diskuteret, hvorvidt systemdefinitionerne stemmer overens med BATOFF-kriterierne. Vi har indsat de forskellige kriterier ind i tabel 3.2, hvor vi har indtastet de relevante forklaringer for de seks kriterier. Tabellen giver derudover et bedre overblik over systemdefinitionerne.

Kriterier	Systemdefinitioner	
	Online opskrifter (S1)	Madplanlægger (S2)
Betingelser	Ulønnede udviklere.	Ulønnede udviklere.
Avendelsesområde	Dele af husholdninger, f.eks. forældre eller lignende.	Dele af husholdninger, f.eks. forældre eller lignende.
Teknologier	Internetforbindelse. PC. Tablet. Mobiltelefon.	Internetforbindelse. PC. Tablet. Mobiltelefon.
Objekter	Opskrifter. Ingredienser.	Opskrifter. Ingredienser. Vitaminer.
Funktioner	Søgningsværktøj. Find opskrifter indeholdende valgte ingredienser	Planlægningsværktøj. Planlægge madlavning ud fra tidligere dages retter.
Filosofi	Folk smider mad ud, fordi de mangler et sted at bruge deres rester.	Folk spiser ikke varieret nok, hvilket er en trussel mod folkesundheden.

Tabel 3.2: Tabel med forklaringer over BATOFF-kriterier for systemdefinitionerne S1 og S2.

3.3.2 Valg af systemdefinition

Vi holdte os meget åbne for nye løsningsforslag, da vi gerne ville gøre det muligt for informanterne at komme med nye idéer, også selvom de var markant anderledes end vores systemdefinitioner. Vi ønskede at høre om informanterne ville være i stand til bruge lignende systemer. Derfor foregik møderne som semistrukturerede interviews, hvilket betyder, at vi havde udformet et interview med spørgsmål, som vi ønskede at stille, men der var luft i interviewet til at informanterne kunne introducere nye emner til diskussionen. Disse nye emner ville ikke ødelægge interviewet. Det var vigtigt for os at få informanternes idéer til hvilke funktioner et sådan system skulle have, og hvilke krav de stiller.

Der blev taget højde for informanternes respons (Al dokumentation og alle referater fra de afholdte møder med informanterne kan findes i bilag C.) på de udarbejdede systemdefinitioner, og der blev truffet et valg, som alle parter kan se som en mulig løsning på problemerne mht. madlavning og madspild i danske husstande. Efter møderne var det helt klart, hvilken systemdefinition, der var den, informanterne ønskede. Med hensyn til systemdefinition 2 (se afsnit 3.3.1) mente informanterne, at de simpelthen ikke ville komme til at bruge et planlægningsværktøj, da det ville være både for tidskrævende og give dem mindre frihed. Systemdefinition 1 (se afsnit 3.3.1) var informanterne derimod meget entusiastiske overfor. De så systemdefinition 1 som et meget brugbart system, der ville gøre det muligt for dem at bruge deres gamle madrester på en spændende måde. På baggrund af dette valgte vi at udvikle systemet som beskrevet i **systemdefinition 1**. Denne beslutning satte nogle fastere rammer for projektets videre forløb. En visualisering af systemet kan ses i figur 3.7.

Systemdefinition 1 er blevet valgt. Det er nu på tide at overveje, hvordan systemet skal fungere, hvordan det skal designes, og hvilke funktioner, der skal inkluderes i systemet.



Figur 3.7: Rigt billede, der visualiserer anvendelsen af systemet udtrykt i systemdefinition 1.

4 Analyse af problemområdet

Analysen tager udgangspunkt i en metode fra bogen Objektorienteret Analyse & Design (OOAD)[25, s. 43]. Problemområdet er den del af omgivelserne, som skal administreres, styres og overvåges af vores system. I vores tilfælde er problemområdet: madresterne og madlavningen i de danske hustande. Formålet med analysen er at skabe et overblik over, hvilke klasser og hændelser systemet skal have, for at kunne udføre de funktioner, vi ønsker, og for at systemet bliver så brugbart som muligt for dets fremtidige brugere.

Vi har valgt at dele analysen op i følgende tre aktiviteter: identificering og udvælgelse af klasser og identificering og udvælgelse af hændelser, som beskrives i afsnit 4.1. Strukturering af klasser og hændelser (struktureres vha. et klassediagram), som beskrives i afsnit 4.2, og analyse og beskrivelse af adfærdsmønstre for klasserne, som beskrives i afsnit 4.3, vha. tilstandsdiagrammer. Resultatet af de tre aktiviteter er en hændelsestabell, som ses i tabel 4.1, der er præsenteret i slutningen af afsnit 4.1 for at give et overblik over, hvad der kommer i de efterfølgende afsnit i dette kapitel.

4.1 Klasser

En klasse er en beskrivelse af en samling af objekter med samme struktur, adfærdsmønster og attributter [25, s. 51]. To klasser kan have en association mellem dem, men det er ikke altid helt klart, hvad denne association skal betyde. Vi har løst dette problem ved at navngive nogle af associationerne mellem klasser. Ser vi på figur 4.1 i afsnit 4.2, så ser vi eksempler på navngivne associationer. Her er en ”Vare” associeret med en ”Person”, og associationen er navngivet ”Indkøbsliste”.

Overblikket over hvilke klasser systemet skal have, kan skabes ved først at finde så mange klasserkandidater som muligt, og dernæst at afgrænse disse, så kun de mest relevante klasser er tilbage. I vores tilfælde anvendte vi en kombination af rige billeder, som kan ses i figur 3.1 og figur 3.7, og systemdefinitionen som hjælpemidler til at finde frem til de klasser, som vi vil modellere i problemområdet.

Grundet den evolutionære arbejdsmetode, som vi har arbejdet ud fra, er klasser undervejs i forløbet blevet tilføjet og fjernet. De klasser som er blevet fravalgt, kan ses i bilag A.1. Herunder ses de valgte klasser, samt beskrivelser og begründelser for, hvorfor de er med i vores problemområde.

Person

En person er en, der er ansvarlig eller hjælper til med madlavningen i husstanden. En person kan være i besiddelse af en indkøbsliste, der bruges til at håndtere fremtidige indkøb af varer. Derudover kan en person have bogmærket nogle opskrifter, som vedkommende synes godt om. Ydermere kan en person opdage fejl vedr. en opskrift, som personen kan vælge at indmelde.

Opskrift

En opskrift er den centrale klasse i problemområdet. En opskrift kan findes i en opskritsamling, i en kogebog eller på nettet. Optrifter indeholder en ingrediensliste. En opskrift kan være ikke-bogmærket eller bogmærket, hvis en person ønsker, at opskriften skal være let tilgængelig eller ej.

Råvaretype

En råvaretype findes i køleskabene, på madhylderne i husstanden og i supermarkederne. Råvaretype adskiller sig fra klassen ingrediens ved, at en råvaretype ikke har nogen enhed eller mængde. Et eksempel på råvaretyper kunne være ”gulerødder”, ”mælk”, ”hakket oksekød” osv.

Ingrediens

En ingrediens tilhører en råvaretype, og består af en mængde og en enhed. En ingrediens kan befinde sig på en ingrediensliste på en opskrift. Klassen ingrediens skiller sig ud fra råvaretype, netop fordi den uddover at være en råvaretype, også har en mængde og en enhed. Eksempler på ingredienser er: "3 styk gulerødder", "4 liter mælk", "500 gram hakket oksekød" osv.

Vare

En vare er en vare, som man kender det fra supermarkeder. En vare, i modsætning til ingredienser, tilhører ikke en råvaretype. En vare kan f.eks. være toiletpapir, vaskepulver eller blyanter. En vare kan befinde sig på en persons indkøbsliste.

Fejl

Fejl opstår som f.eks. en mængdefejl i en opskrift eller en manglende instruktion i fremgangsmåden. Fejl er uventede, og det er svært at sige, hvor de opstår og i hvilket omfang.

Ud fra de valgte klasser skal vi have formuleret nogle hændelser, som beskriver klassernes adfærd. For klassen "opskrift", kan en hændelse f.eks. være "opskrift fundet". En hændelse er en bestemt adfærd for en klasse, som beskrives med udsagnsord. En hændelse kan involvere en til flere klasser. Som i eksemplet før, så involverer hændelsen "opskrift fundet", både "opskrift" men også "ingrediens", da vi vurderer, at man skal finde en opskrift, før man kan arbejde med ingredienser, fordi en opskrift består af ingredienser. Ligesom med klasser så har vi gennem projektforløbet tilføjet og fjernet hændelser. De hændelser, som er blevet fravalgt, kan ses i bilag A.

I tabel 4.1 ses de valgte hændelser, og hvilke klasser disse hændelser har en indflydelse på. Hver hændelse forårsager et tilstandsskift, og disse tilstande kan ses i tilstandsdiagrammerne i afsnit 4.3. Hændelsestabellen er resultatet af analysen af problemområdet, men vi præsenterer den allerede nu, da den giver et godt overblik over de hændelser og klasser, vi er endt op med.

I hændelsestabellen benytter vi \circlearrowleft -symbolet til at illustrere, at den tilhørende hændelse kan forekomme flere gange i den pågældende klasse. Det betyder, at denne hændelse kan ses som en iteration i tilstandsdiagrammerne i afsnit 4.3. Derudover benytter vi $+$ -symbolet til at illustrere, at den tilhørende hændelse blot kan forekomme en gang i klassen. Det betyder, at hændelsen kan ses som en selektion eller en sekvens i tilstandsdiagrammerne.

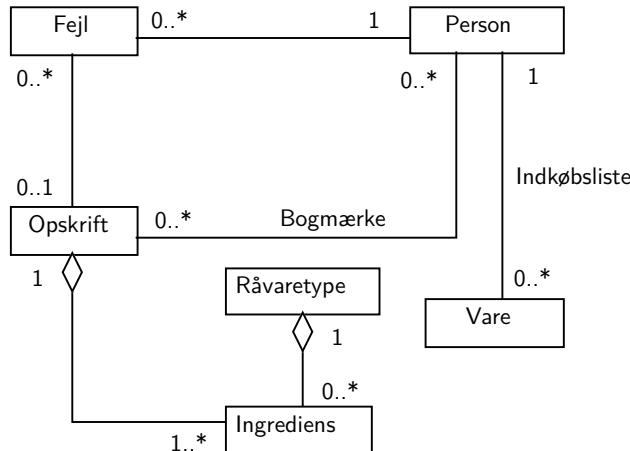
Hændelser	Klasser					
	Opskrift	Person	Vare	Ingrediens	Råvaretype	Fejl
Opskrift smidt ud	+				+	
Opskrift fundet	+				+	
Bogmærke sat ind	\circlearrowleft	\circlearrowleft				
Bogmærke fjernet	\circlearrowleft	\circlearrowleft				
Skrevet på indkøbsliste	\circlearrowleft	\circlearrowleft	+		\circlearrowleft	
Fjernet fra indkøbsliste		\circlearrowleft	+		\circlearrowleft	
Råvare opbrugt					\circlearrowleft	
Råvare købt					\circlearrowleft	
Fejl rapporteret	\circlearrowleft	\circlearrowleft				+
Tilbagemelding modtaget			+			+
Fejl set bort fra						+

Tabel 4.1: Hændelsestabell for klasserne opskrift, person, vare, ingrediens, råvaretype og fejl

Problemområdet består af madrester og madlavning i husstanden, derfor er det vigtigt, at problemområdet bliver repræsenteret ordentligt, i form af klasser. Madlavning skal ikke forståes som, at vi ønsker at overvåge, om der bliver lavet mad i husstanden, da det vi ønsker blot er at udvikle et system, der giver mulighed for at genbruge madrester, og give inspiration til madlavningen.

4.2 Struktur

Strukturen på vores klasser beskrives ved hjælp af et klassediagram. Med et klassediagram skabes der et overblik over relationen mellem de forskellige klasser og objekter. **foodl** består af 6 klasser, som det er vist på figur 4.1. Relationen mellem kasserne kan ses i klassediagrammet i figur 4.1. En relation mellem to klasser markeret med en rumbe-pil, er en aggregerings-struktur, hvilket er en relation, som beskrives ved udsagnet “har-en” og “indgår-i”. Eksempelvis så er ingrediens en aggregering af råvaretype, hvilket betyder, at en råvaretype har ingredienser. En relation mellem to klasser markeret med en streg, er en associeringsstruktur, som betyder, at de to klasser har en sammenhæng. I vores struktur associerer kasserne fejl og opskrifter med hinanden. En associering er ikke, ligesom en aggregering, nem at knytte et udsagn til. Men i vores tilfælde kan en associering mellem fejl og opskrift beskrives som et “hører-til”-forhold. Fejl “hører-til” en opskrift, og en opskrift kan muligvis “høre-til” fejl. Hver relation har desuden multiplicitet tilknyttet. Dette beskriver forholdet mellem kasserne. Det kan eksempelvis være et et-til-et-forhold, eller et 0-til-mange-forhold. På figur 4.1 ses det, at forholdet mellem klassen opskrift og klassen ingrediens er et 1-til-mange-forhold (*-tegnet betyder mange), mens forholdet mellem ingrediens og opskrift er et 1-forhold. Det betyder, at der på hver opskrift er minimum én ingrediens, og at hver ingrediens hører til præcis én opskrift. Hver enkelt klasses relationer, forhold og multiplicitet forklares herunder nærmere:



Figur 4.1: Klassediagram for problemområdet.

Associering mellem person og vare (indkøbsliste)

En person associerer 1-til-mange varer, modelleret på baggrund af indkøbslisten. En indkøbsliste kan benyttes af en person til at holde styr på hvilke varer, der skal handles ind. Det er muligt for en person at være relateret til mange varer. Hvert vareobjekt er unikt for hvert personobjekt, dvs. at det ikke er muligt for personer at dele varer mellem sig. En indkøbsliste kunne have været en klasse for sig selv, men fordi hver person kun har netop en indkøbsliste, er dette overflødig.

Associering mellem person og fejl

Fejl associerer med person-klassen, da det er personen, der opdager fejl. En person kan opdage flere fejl i den samme opskrift, eller i forskellige opskrifter i løbet af personobjektets levetid.

Associering mellem opskrift og fejl

En opskrift kan have 0-til-mange fejl tilknyttet. En fejl behøver dog ikke være tilknyttet en specifik opskrift og kan maksimum være tilknyttet en opskrift.

Associering mellem opskrift og person (bogmærke)

Personer kan associeres med flere opskrifter i et forhold udtrykt som bogmærker. Støder personen på en opskrift, som personen synes er god, og ønsker at gøre den let tilgængelig til en anden gang, så skal det være muligt at bogmærke denne. Et bogmærke er dog ikke udtrykt i modellen som en klasse, da hvert bogmærke ikke har nogen identitet, udover associeringen med en opskrift og en person. Hver opskrift kan derudover også associeres med flere personer, dvs. at flere forskellige personer godt kan have et bogmærke på den samme opskrift.

Aggregering mellem opskrift og ingrediens

En opskrift består af 1-til-mange ingredienser. Hvis opskriften ikke bestod af nogle ingredienser, ville det ikke være en opskrift og skulle derfor ikke modelleres. Hver ingrediens tilhører netop en opskrift.

Aggregering mellem ingrediens og råvaretype

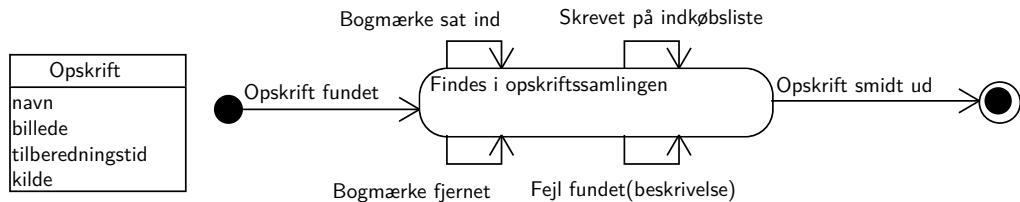
Hver ingrediens tilhører netop én råvaretype, mens en råvaretype kan indeholde mange forskellige ingredienser. Ingrediensen "300 g hvedemel" tilhører f.eks. råvaretypen "hvedemel".

4.3 Adfærd

Den sidste aktivitet i analyse af problemområdet består i at analysere og beskrive adfærdsmønstre for klassers objekter. Formålet med dette er at få en bred forståelse af, hvordan objekter kan opføre sig, hvilket vi vil bruge senere som inspiration, når vi skal beskrive de funktioner **food** skal have. Dermed opnåes der også en mere glidende overgang over mod analyse af anvendelsesområdet i kapitel 5. Desuden har vi anvendt adfærdsmønstrene til at få overblik over, hvilke attributter klasser skal have, om de hændelser vi har valgt, er fyldestgørende, og ydermere, til at få inspiration til nye hændelser. Adfærdsmønstre har vi valgt at beskrive ved hjælp af tilstandsdiagrammer. For hver klasse følger der et tilstandsdiagram, som illustrerer adfærdens for objekter af den pågældende klasse. I tilstandsdiagrammerne skal de afrundede rektangulære bokse med tekst anses som tilstande, som den pågældende klasse kan have. Pilene der fører til en tilstand, skal anses som hændelser, som kan være skyld i et tilstandsskift for objektet. Som eksempel (se figur 4.4) kan et objekt af klassen råvaretype skifte mellem tilstandene "eksisterer" og "brugbar" ved hjælp af hændelserne "Råvare købt" og "Råvare opbrugt". Går en pil med en hændelse til og fra den samme tilstand, er der tale om en løkke. En løkke er en hændelse, som kan ske igen og igen, uden at objektet ændrer tilstand. Eksempelvis kan hændelsen "vare tilføjet" ske igen og igen for et indkøbsliste-objekt. Som oftest vil en klasse have en starthændelse, som "sætter gang" i klassen, og en sluthændelse, som "slutter" klassen. En starthændelse er markeret med en fyldt sort cirkel, og en sluthændelse er markeret med en fyldt sort cirkel, som ligger inde i en anden sort cirkel. I figuren med tilstandsdiagrammerne, er der også en illustration af den klasse, som tilstandsdiagrammet hører til. Inden i klassen ses de attributter, vi har fundet frem til. Attributterne vil desuden beskrives nærmere i afsnit 6.2.4.

4.3.1 Opskrift

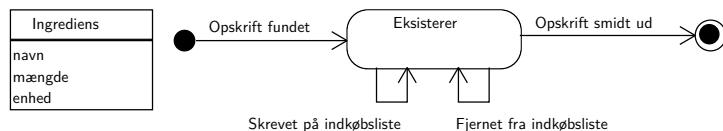
En opskrift kan befinde sig i en kogebog, på et stykke papir eller på en hjemmesiden. I figur 4.2 ses adfærdens for klassen "Opskrift". En opskrift eksisterer ved hændelsen *opskrift fundet*. Her befinder den sig i tilstanden *findes i opskriftssamlingen*. Hvis man er rigtig glad for en opskrift, kan man sætte et bogmærke på opskriften, så man hurtigt kan finde den igen. Sådan en hændelse kaldes for *bogmærke tilføjet*. Hvis man bestemmer sig for at fjerne bogmærket, indtræffer hændelsen *bogmærke fjernet*. Derudover kan man være nødsaget til at handle ind til en specifik opskrift, fordi man mangler nogle ingredienser. Det betyder, at man skal tilføje nogle opskrifter til en indkøbsliste. Denne hændelse kaldes for *skrevet på indkøbsliste*, hvor der også findes den modsatte hændelse *fjernet fra indkøbsliste*. Der kan også eksistere en fejl i opskriften. En fejl kunne være et manglende billede til opskriften, eller at der står 20 minutter i ovnen i stedet for 40. Dette er beskrevet ved hjælp af hændelsen "fejl fundet". Disse fire hændelser medfører ikke et tilstandsskift. Opskriften ryger kun ud af tilstanden *findes i opskriftssamlingen*, når opskriften fjernes vha. hændelsen *opskrift smidt ud*.



Figur 4.2: Tilstandsdiagram for klassen opskrift. Klassen har én tilstand (*findes i opskriftssamlingen*) og en afsluttende hændelse “*opskrift fjernet*”, der fører klassen ud i en sluttilstand.

4.3.2 Ingrediens

En ingrediens kommer til verden sammen med en opskrift. Dette er fordi, at en ingrediens ikke er en del af problemområdet, før den optræder i en opskrift. Det er nemlig først der, at man i en husstand behøver at bekymre sig om ingredienser. Hvis de ikke fandtes i nogle opskrifter, var der ikke nogen grund til at indkøbe en råvaretype svarende til ingrediensen. Denne hændelse kaldes *Opskrift fundet*. Mens ingrediensen eksisterer, er den i tilstanden ”eksisterer”. Når man kender til en ingrediens, kan man skrive den på sin indkøbsliste. Dette gør man, hvis man vil være sikker på at huske at købe en råvare svarende til ingrediensen, når man er ude at handle ind. Man kan selvfølgelig også fjerne ingrediensen fra sin indkøbsliste, hvis man har ombestemt sig og ikke ønsker at huskes på at købe den tilsvarende råvare. Disse to hændelser kaldes *skrevet på indkøbsliste* og *fjernet fra indkøbsliste*. Når opskriften, der indeholder en given ingrediens, smides ud, er hændelsen *Opskrift smidt ud* netop indtruffet, og ingrediensen bringes i sin sluttilstand. Det bør bemærkes, at 2 forskellige opskrifter, der begge indeholder pasta, anses for at indeholde hver sin ingrediens. Dette er en vigtig adskillelse, da ingredienser kan bestå af en mængde og enhed, f.eks. 200 g pasta, hvilket en råvaretype ikke kan.

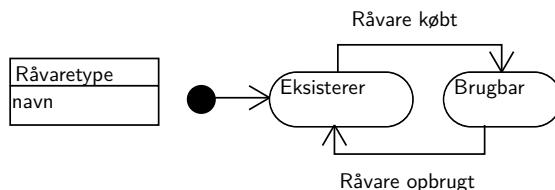


Figur 4.3: Tilstandsdiagram for klassen ingrediens. I dette tilfælde har klassen en tilstand ”eksisterer” og en starthændelse ”*Opskrift fundet*” og en afsluttende hændelse ”*Opskrift smidt ud*”, der fører klassen ud i en sluttilstand.

4.3.3 Råvaretype

Når nogen køber en råvaretype, indtræffer hændelsen *råvare købt*. Denne hændelse fører råvaretype-klassen i en ny tilstand, der hedder *brugbar*. Denne råvaretype er klar til at blive benyttet til madlavning. Inden der bliver handlet ind, og inden der sker et tilstandsskift, så er råvaretypen i en tidligere tilstand, der hedder *eksisterer*. Der eksisterer ingen starthændelse på klassen, da råvaretyper kan ses som altid eksisterende objekter i problemområdet. Dette tilstandsdiagram har tilsvarende ingen afsluttende hændelse, fordi vi vurderer, at en råvaretype altid vil eksistere, selvom man ikke er i besiddelse af den. Men den er først brugbar, når man har købt den.

Indehaverne af en råvare har nu to muligheder. Månen kan enten bruge hele råvaren, eller smide den ud. Disse to hændelser samler vi under en fælles hændelse, som vi kalder *råvare opbrugt*. Denne hændelse fører klassen i den forrige tilstand, der hedder *eksisterer*, da den ikke længere er brugbar i husstanden.

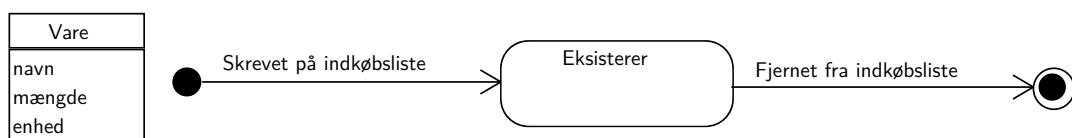


Figur 4.4: Tilstandsdiagram for klassen råvaretype. Klassen har to tilstande “eksisterer” og “brugbar”. Råvaretype har ikke en igangsættende eller afsluttende hændelse, da der altid vil eksistere en råvaretype, uanset om man har den i sin husstand eller ej.

4.3.4 Vare

En vare påbegynder sin eksistens, når den skrives på en persons indkøbsliste, dvs. ved hændelsen “Skrevet på indkøbsliste”. En vares eksistens ophører, når den fjernes fra indkøbslisten igen, ved hændelsen “Fjernet fra indkøbsliste”.

Klassen er kommet til verden ud fra gruppeditisioner under komponentarkitekturdesignet. Grunden til, at vi har valgt at tilføje vare-klassen, er, at vi mener det skal være muligt for personer også at tilføje og fjerne ikke-ingredienser til indkøbslisten.



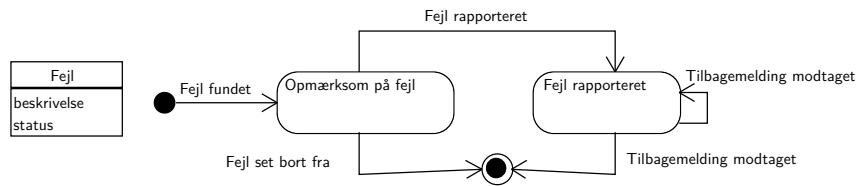
Figur 4.5: Tilstandsdiagram for klassen vare. Klassen har én tilstand, nemlig “eksisterer”.

4.3.5 Fejl

Fejl opstår ved, at opskriftsskriventer glemmer at tilføje nogle vigtige ingredienser i en opskrift, eller f.eks. mangler et trin i fremgangsmåden. Det skal derfor være muligt for personer at indrapportere deres input, så de kan hjælpe forelaget med at korrigere de fejl, der måtte eksistere i opskriften. Dette kan være med til at forbedre kvaliteten af udgivelsen. Tilstandsdiagrammet for klassen “fejl” ses i figur 4.6.

Starthændelsen *fejl fundet* opstår, når læseren opdager en fejl i en opskrift. Derefter eksisterer der en selektion, hvor brugeren har valget mellem at rapportere fejlen til opskriftsskriventeren (hændelsen *fejl rapporteret*) eller blot springe den over og gå videre til en anden opskrift (hændelsen *fejl set bort fra*, hvilket fører til henholdsvis en ny tilstand “fejl rapporteret” eller sluttilstanden. Hvordan fejlen bliver rapporteret er ikke relevant i forhold til denne modellering af problemområdet, vi er bare interesseret i, at det bliver gjort. Når fejlen er rapporteret, er der mulighed for både en iterativ og en tilstandsændrende (til sluttilstanden) form for tilbagemelding (hændelsen *tilbagemelding modtaget*). Dette afhænger af konteksten mellem personen og opskriftsskriventeren, om kontakten kun sker envejs, eller om der er løbende kontakt mellem de to parter.

Klassen er, ligesom vare-klassen, kommet til verden ud fra gruppeditisioner under komponentarkitekturdesignet. Dette er grundet, at vi mener forskellige opskriftshjemmesider har varierende kvalitet af opskrifter. Derfor skal man, som forbrugere, kunne gøre andre opmærksomme på diverse fejl, der skulle befinde sig på en opskrift. Så har modtageren mulighed for at kigge på fejlene og eventuelt rette dem, så brugere får en endnu bedre oplevelse.

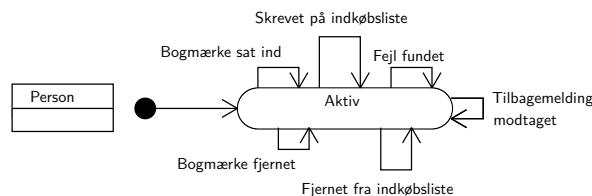


Figur 4.6: Tilstandsdiagram for klassen "fejl". Klassen har to tilstande, som begge kan føre til sluttilstanden i forhold til de valg, som der bliver foretaget

4.3.6 Person

En person er en, der er ansvarlig for eller hjælper til ved madlavningen i husstanden. Figur 4.7 viser tilstandsdiagrammet for person-klassen. En person bliver aktiv, når personen bliver inddraget i madlavningen.

Herefter vil personen være med til at indsætte og fjerne bogmærker fra bl.a. kogebøger. Man kan også være med til at tilføje eller fjerne varer til en indkøbsliste, som man senere skal bruge til at handle ind. Derudover kan personen finde fejl i opskrifter og evt. få rettet fejlen ved at indsætte en mail eller lignende til forlaget. Til slut kan man modtage en tilbagemelding på fejlen, hvis forelaget ønsker at rette fejlen.



Figur 4.7: Tilstandsdiagram for klassen person.

Person har ingen attributter, fordi vi f.eks. ikke er interesserede i at overvåge personens navn eller lignende. Det har ingen indflydelse på madlavningen.

4.4 Sammendrag

De forskellige klasser og hændelser i problemområdet er nu blevet analyseret og beskrevet. Grundet den evolutionære arbejds metode, er der gennem projektforløbet blevet tilføjet og fjernet klasser og hændelser igen og igen. Til at finde klasser anvendte vi en kombination af rigtige billeder, interviews med informanter, samt brainstorming. Det samme gjorde sig gældende for hændelser, men her brugte vi også tilstandsdiagrammer som hjælp. Resultatet af analysen af problemområdet er en hændelsestabel, som ses i tabel 4.1. Denne illustrerer forbindelsen mellem klasser og hændelser. Adfærdens for hver enkelt klasse er også blevet beskrevet, og vi er klar til at begynde på analyse af anvendelsesområde.

5 Analyse af anvendelsesområdet

I følgende kapitel analyseres anvendelsesområdet. Formålet med analysen er at fastlægge krav til brugen af **foodl**. Analysen tager udgangspunkt i en metode, som er introduceret i bogen Objektorienteret Analyse & Design[25, s. 113].

Analysen indeholder to hovedaktiviteter, *brug* og *funktioner*. Samlet set giver de to aktiviteter os indblik i, hvordan **foodl** skal interagere med personer, der bruger systemet, samt andre systemer. [25, s. 115].

5.1 Brug

Denne analyse af anvendelsesområdets brug har til formål at gøre det klart, hvordan **foodl** interagerer med brugere og andre systemer. Vi analyserer, hvilke aktører, der benytter **foodl** og beskriver disse. Måden, hvorpå aktørerne interagerer med **foodl**, beskriver vi med brugsmønstre. Hver enkelt brugsmønster vedrører en eller flere aktører. Relationerne vises i tabel 5.1. Et flueben betyder, at brugsmønstret i samme række vedrører aktøren i samme kolonne.

Aktører	Brugsmønstre		
	Bruger	Administrator	Dataudtrækker
Søgning	✓	✓	
Favorisering	✓	✓	
Indkøbslistehåndtering	✓	✓	
Login	✓	✓	
Fejlhåndtering		✓	
Rapportering	✓	✓	
Dataudtrækning			✓

Tabel 5.1: *Aktørtabel for foodl*.

5.1.1 Aktører

For at finde aktører har vi spurgt os selv, hvem der skal bruge systemet. Vi har identificeret 3 aktører: Bruger, Administrator og Dataudtrækker, der vil kunne interagere med **foodl**. Disse aktører er vist i tabel 5.1. Aktørerne Bruger og Administrator er personer, der benytter systemet, mens Dataudtrækker er et andet system. Bruger repræsenterer den samme person, som afspejles af person-klassen i problemområdet. Aktøren har et andet navn en brugeren, for klart at skelne mellem problem- og anvendelsesområdet. Der er nemlig ikke "brugere" i problemområdet, da der ikke er noget system at bruge. Aktørerne specificeres med en beskrivelse af aktøren samt en, eller om nødvendigt, flere eksempler på hver enkelt aktør. Disse beskrivelser har til formål at give læseren en bedre forståelse for, hvilke egenskaber og karakteristika en sådan aktør besidder.

Bruger

Formål: En person, der ønsker at bruge systemet **foodl** til at finde opskrifter, der er mulige at lave med de råvaretyper, som personen er i besiddelse af.

Karakteristik: Systemets brugere inkluderer mange personer i forskellige aldersgrupper med vidt forskellig erfaring inden for computerbrug.

5. Analyse af anvendelsesområdet

Eksempler: Bruger A er en 23-årig universitetsstuderende, der føler sig sikker med at navigere rundt på internettet og gør det flere gange dagligt. A kan godt lide at afprøve de forskellige funktioner som hjemmesider stiller til rådighed for at undersøge, hvad de gør. A er meget lerenem, når det kommer til at benytte funktioner på hjemmesider. A bor alene, og har pga. supermarkedernes familietilpassede portioner, ofte madrester til overs, som A ønsker at bruge.

Bruger B er en 45-årig familiemor eller -far, der hovedsagligt bruger computeren til arbejdsrelaterede opgaver og til at holde sig opdateret ved at læse nyheder på diverse nyhedshjemmesider. B benytter systemet til blandt andet at få brugt madrester fra den foregående dags aftensmad eller til at få inspiration til den kommende aftensmad. B vil være gerne være i stand til at dele indkøbslisten med ægtefællen, på tværs af enheder.

Administrator

Formål: En person, der har til formål at administrere og håndtere eventuelle fejl i systemet, der rapporteres af systemets brugere.

Karakteristik: Systemets administrator har et højt erfaringsniveau med hensyn til systemet. De har også kontakt til systemets udviklere, der kan rette eventuelle seriøse og systemkritiske fejl.

Eksempler: Administrator A er en ubetalt studerende, som håndterer fejl på **foodl** i sin fritid. A gennemgår fejlrapporter og vurderer om en fejl er så kritisk at han bør kontakte en systemudvikler, eller om han selv kan udbedre fejlen, f.eks. ved at slette et dødt link.

Dataudtrækker

Formål: Dataudtrækkeren er et system, der tilføjer madopskrifter til **foodl** i form af data, repræsenteret på en måde, som **foodl** kan forstå. **foodl** forstår opskrifter som noget, der har et navn, billede, tilberedningstid, portionsstørrelse, en liste af råvaretyper og en fremgangsmåde. En Dataudtrækker kan f.eks. være et system, der besøger en hjemmeside med madopskrifter, udtrækker den nødvendige data omkring forskellige opskrifter, oversætter data og sender det til **foodl**. En opskrift består af ingredienser, som ikke nødvendigvis har en tilsvarende råvaretype i **foodl** med nøjagtig samme navn. Derfor skal disse ingredienser også oversættes til en af de råvaretyper, som **foodl** kender. Ingrediensens mængde og enhed skal også identificeres.

Karakteristik: Dataudtrækkeren er utrolig systematisk i måden, hvorpå den læser og oversætter opskrifter. Den er ikke fleksibel i dens arbejde, da den kun kan oversætte opskrifter med et layout, der er magen til det layout, den har lært at læse. En dataudtrækker er heller ikke kritisk over for det materiale, den læser. Hvis den pludselig støder på en opskrift, hvis billede er af noget mærkeligt, der ikke har noget med opskriften at gøre, ville den ikke bemærke dette, og stadigvæk forsøge at overføre dette billede til **foodl**. Dataudtrækker A fungerer på en hjemmeside med opskrifter. Den bevæger sig imellem alle de forskellige opskrifter på hjemmesiden og udtrækker informationen omkring opskrifterne fra sidernes kildekode. Opskrifternes billede er gemt i et, af **foodl**, ukendt format, hvorfor Dataudtrækker A konverterer billedet til et almindeligt billedformat.

5.1.2 Brugsmønstre

For at beskrive hændelserne, der vedører en given aktør, modellerer vi en række brugsmønstre. Brugsmønstrene præsenteres både i form af tilstandsdiagrammer og brugsmønsterspecifikationer. Tilstandsdiagrammet giver et visuelt overblik over brugsmønstret, hvor brugsmønsterspecifikationen giver en mere detaljeret beskrivelse af brugsmønstret.

I forbindelse med modelleringen af brugsmønstre, har vi fokuseret på at identificere så mange brugsmønsterkandidater som muligt. Vi har benyttet denne teknik i et forsøg på at sikre, at vi ikke har overset et vigtigt brugsmønster. Kandidaterne er valgt i forbindelse med en brainstorming, og er hver især blevet analyseret for relevans efter brainstormen. Dette har resulteret i, at nogle af kandidaterne er blevet fravalgt. Der har været flere grunde til, at vi har fravalgt kandidater:

- Brugsmønstret har ikke været en del af anvendelsesområdet
- Brugsmønstret har været for simpelt
- Brugsmønstret har været været en del af eller magen til et andet brugsmønstre

De fravalgte kandidater, samt begründelsen for fravælgelsen, fremgår af bilag B.

Aktøren “bruger” vil være den vigtigste aktør i forhold til brugen af systemet. Det er meningen, at der skal være rigtig mange brugere, der kan få gavn af systemet. Disse brugeres kompetencer mht. computerbrug vil variere utroligt meget, og det skal vi tage højde for.

Søgning

Brugsmønster: En søgning igangsættes af *brugeren*. *Brugeren* bliver præsenteret for et tom søgefelt, der illustrerer, at man kan indtaste nogle søgekriterier i form af råvaretyper i det tomme felt. En søgning fuldføres, når man taster på knappen “søg”, hvilket kun er muligt, når der mindst er én råvaretype som søgekriterie. Det er muligt at indtaste og fjerne så mange råvaretyper fra søgefeltet, som *brugeren* ønsker. Se figur 5.1.

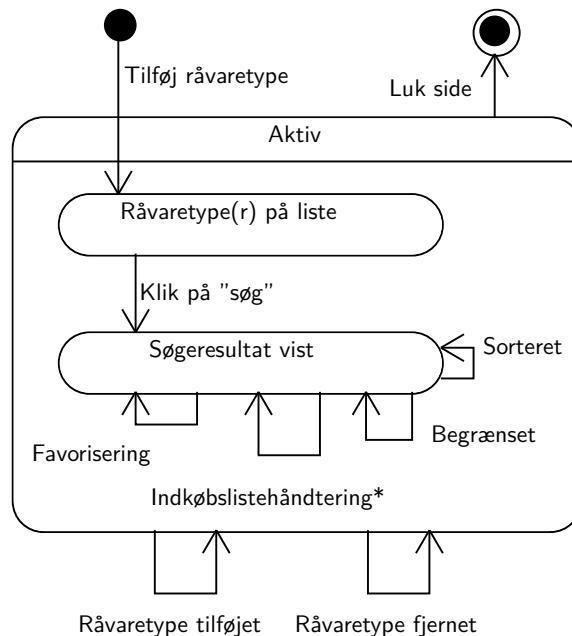
Når der er blevet foretaget en søgning, så bliver der vist et søgeresultat, der er en liste af opskrifter, der indeholder en eller flere af de indtastede råvaretyper. *Brugeren* har nu endnu en gang mulighed for at fjerne eller indtaste nye råvaretyper fra denne resultatside. *Brugeren* kan søge på de nye søgekriterier direkte fra denne side uden at genlæse visningen i sin internetbrowser.

Opskrifterne sorteres i første omgang efter, hvor godt deres ingredienser matcher de valgte råvaretyper. *Brugeren* kan vælge to andre sorteringsmuligheder. Udover den primære sortering, så kan opskrifterne sorteres efter tilberedningstid og alfabetisk orden. Det er også muligt at begrænse søgeresultatet efter tilberedningstid. Her har *brugeren* mulighed for at bestemme om, der skal bruges lidt eller meget tid på madlavningen. Når *brugeren* har valgt, så bliver listen dynamisk opdateret.

Søgningen kan under alle tilstænde afsluttes ved at lukke siden. Brugsmønstret skiller sig ud fra de andre brugsmønstre, ved at det har et brugsmønster integreret i sig, nemlig indkøbslistehåndtering. Det er markeret med en *-tegn, for at symbolisere, at det er et brugsmønster.

Objekter: Opskrift, Råvaretype

Funktioner: Søg efter opskrifter, Skaler søgeresultat, Sorter søgeresultat, Begræns søgeresultat



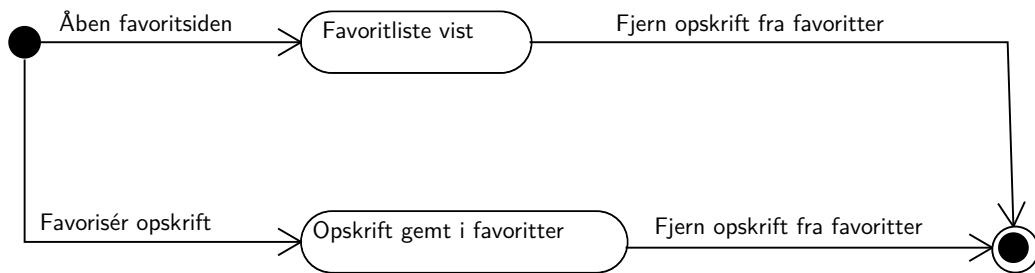
Figur 5.1: Brugsmønster for søgning.

Favorisering

Brugsmønster: Favorisering igangsættes af *brugeren*. Når *brugeren* har lavet en søgning, og flere forskellige opskrifter vises som søgeresultater, kan *brugeren* klikke på favoriser-knappen tilhørende den enkelte opskrift, for at favorisere denne. Se figur 5.2. Når en opskrift er blevet favoriseret, tilføjes denne til en liste af favoritopskrifter. Det er muligt at fjerne opskriften fra favoritlisten på to måder. Enten fra samme sted som favoriseringen blev tilføjet eller direkte i favoritlisten.

Objekter: Opiskrift, Person

Funktioner: Opret favorit, Slet favorit, Vis favoritter



Figur 5.2: Brugmonsteret favorisering

Rapportering

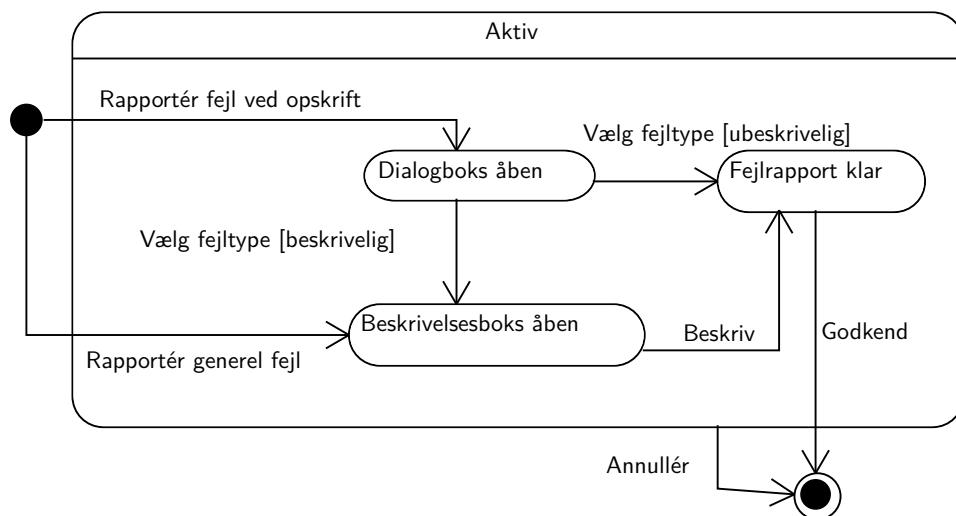
Brugsmønster: Rapportering igangsættes af *brugeren*, når *brugeren* opdager en fejl på hjemmesiden. Hvis *brugeren* opdager en fejl, der har med et søgningsresultat (opskrift) at gøre, så klikkes der på en rapporteringsknap, der er

ved det enkelte søgningsresultatet. Se figur 5.3. Når der skal rapporteres en fejl vedrørende opskrifter, så åbnes en dialogboks på siden, hvor *brugeren* herefter skal vælge en fejltyp. Der skelnes mellem beskrivelige og ubeskrivelige fejltyper. Et eksempel på en ubeskrivelig fejltyp er bl.a., hvis et link ikke fungerer, som hører under fejtypen "dødt link". De ubeskrivelige fejltyper behøver ingen beskrivelse, da fejtypen er beskrivelse nok i sig selv. Vælger *brugeren* derimod en beskrivelig fejltyp, så præsenteres en beskrivelsesboks for *brugeren*, hvor fejlen beskrives med tekst. Derefter er rapporten klar, og *brugeren* skal nu godkende fejlrapparten, inden den bliver sendt til *administratoren*.

Derudover er der en generel rapporteringsknap, der vedrører andre generelle fejl på siden. Når denne knap benyttes, så dirigeres *brugeren* direkte hen til en beskrivelsesboks, hvor fejlen beskrives med tekst. Til slut skal rapporten godkendes af *brugeren*, inden den bliver sendt til *administratoren*. Det er altid muligt at annullere rapporteringen under alle tilstade i brugsmønstret.

Objekter: Fejl, Opskrift, Person

Funktioner: Opret fejl



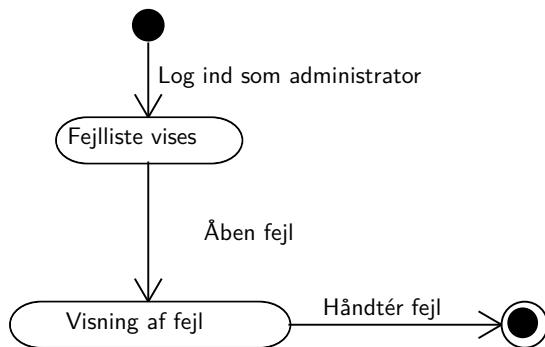
Figur 5.3: Brugmonsteret rapportering

Fejlhåndtering

Brugsmønster: Fejlhåndtering igangsættes af *administratoren*. *Administratoren* logger ind på hjemmesiden, og bevæger sig ind på fejlhåndteringssiden. Se figur 5.4. Her præsenteres en liste af fejl, der, via *brugeren*, er blevet rapporteret og dokumenteret i systemet. *Administratoren* kan derefter klikke på en rapporteret fejl i listen for at se en detaljeret beskrivelse af den givne fejl, som derefter kan håndteres.

Objekter: Fejl, Person

Funktioner: Åben fejl, Luk fejl



Figur 5.4: Brugmonsteret fejlhåndtering

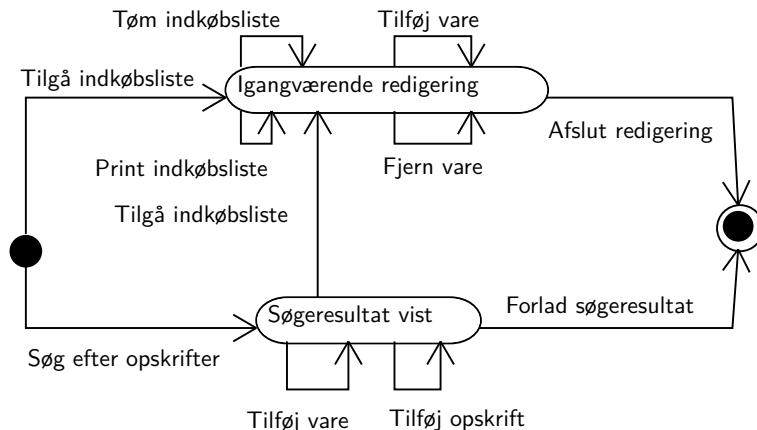
Indkøbslistehåndtering

Brugsmønster: Håndteringen af indkøbslisten følger materialemønsteret[25, p. 128]. I materialemønstret er det muligt for aktøreren, at lave flere handlinger i en vilkårlig rækkefølge, da der i brugmønstret typisk ikke vil være særlig mange tilstande i forhold til handlinger. Håndteringen igangsættes af *brugeren*, se figur 5.5. *Brugeren* kan tilgå indkøbslisten fra en vilkårlig underside på **food!**. Når *brugeren* har tilgået indkøbslisten, så er redigeringen igangsat, og det er muligt at tilføje/fjerne varer fra indkøbslisten. Ingredienser bliver ”omdannet” til varer, når de tilføjes på indkøbslisten. Alle elementer på indkøbslisten er af vare-klassen, hvilket gør det muligt for *brugeren* at indtaste vilkårlige varer, ikke blot ingredienser, der er relateret til f.eks. en opskrift. *Brugeren* har også mulighed for at tilføje varer eller alle opskriftens ingredienser direkte fra søgeresultatet, der er en liste af opskrifter. Når *brugeren* forlader søgeresultatet, så er indkøbslisten gemt, og den kan stadig tilgås fra en vilkårlig underside på **food!**.

Der kan altid kun være én indkøbsliste ad gangen pr. bruger. Fra indkøbslistevisningen er det muligt at printe indkøbslisten samt tømme indkøbslisten for alt indhold. Håndteringen af indkøbslisten afsluttes når *brugeren* lukker ned for redigering - altså forlader indkøbsliste-siden. Redigering startes igen, når man tilgår indkøbslisten, og alle de tilføjede varer, der ikke er blevet slettet, vil stadig være tilgængelige.

Objekter: Vare, Opskrift, Person

Funktioner: Håndter indkøbsliste



Figur 5.5: Brugmonster for håndteringen af indkøbslisten, hvilket er relationen mellem ”person”- og ”vare”-klasserne i problemområdet.

Indlogging

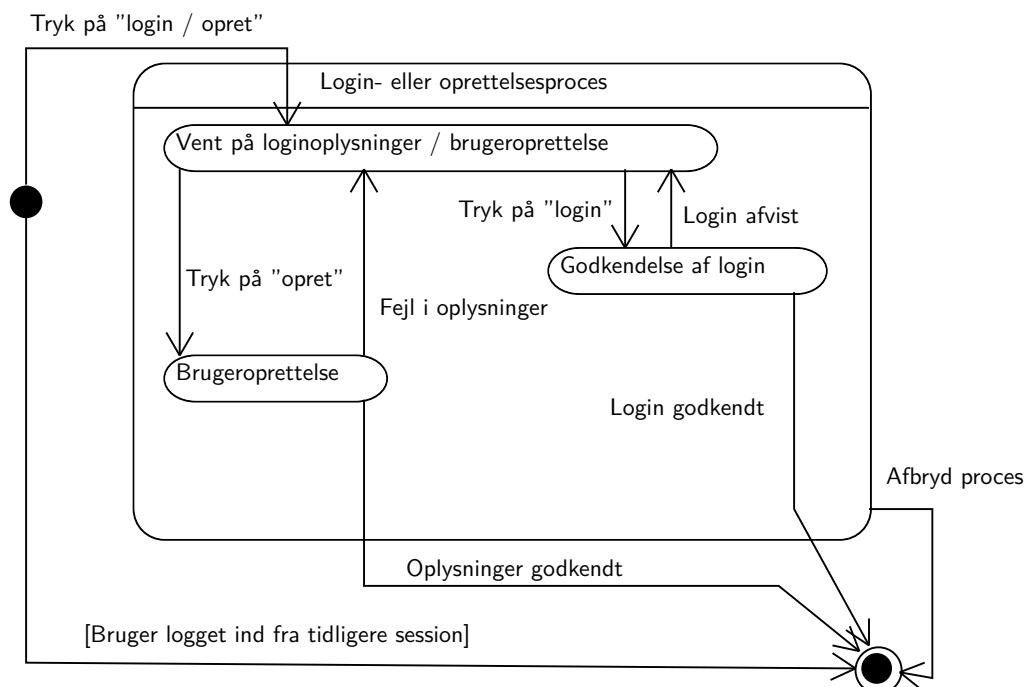
Brugsmønster: Indlogging igangsættes af *brugeren*. Der er to startmuligheder, og de ender begge i samme sluttilstand. Se figur 5.6. Den første mulighed er, hvis *brugeren* var logget ind fra en tidligere session, så hentes oplysningerne fra denne session automatisk og *brugeren* vil blive logget ind. Den anden mulighed er, at *brugeren* trykker på "Login / Opret", som kan tilgås fra en vilkårlig **foodl**-underside. Systemet venter nu på *brugers* loginoplysninger. *Brugeren* indtaster oplysningerne, og systemet påbegynder godkendelsesprocessen. Hvis oplysningerne bliver afvist, så bliver *brugeren* bedt om at genindtaste oplysningerne. Hvis de bliver godkendt, så bliver *brugeren* logget ind på siden. En anden mulighed er, hvis *brugeren* ønsker at lave en ny konto i systemet. Dette sker på samme del af systemet, hvor *brugeren* kan logge ind. *Brugeren* skal nu indtaste e-mail og adgangskode i systemet. Hvis der opstår en fejl i oplysningerne, så skal *brugeren* genindtaste oplysningerne. Når oplysningerne bliver godkendt, så bliver *brugeren* logget ind med den e-mail og adgangskode, som er blevet indtastet. På hvilket som helst tidspunkt, har *brugeren* mulighed for at annullere loginprocessen, hvorefter man naturligvis ikke bliver logget ind eller oprettet i systemet.

Hvis der er favoriseringer eller hvis indkøbslisten indeholder varer fra, hvor *brugeren* ikke var logget ind, så overføres disse til den nyligt oprettede bruger til yderligere redigering og tilføjelse.

Hvis *brugeren* er logget ind på en konto, så bliver *brugers* indkøbsliste og favoritter indlæst. Det muliggør også fejlrapportering på de dele af systemet, der understøtter det. Disse kan tilgås fra en vilkårlig **foodl**-underside. *Brugeren* kan nu oprette eller fjerne favoritter og tilgå og håndtere indkøbslisten.

Objekter: Person

Funktioner: Registrer bruger, Log ind, Log ud, Ret brugerdata, Slet bruger



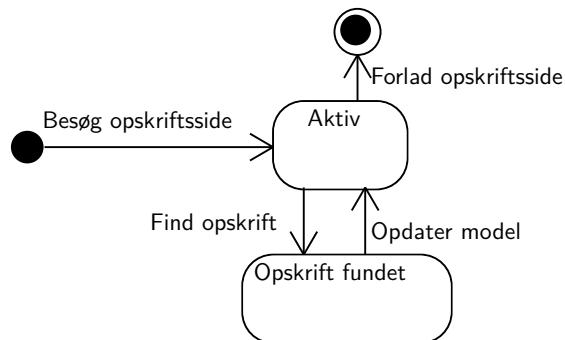
Figur 5.6: Brugsmønster for indlogging.

Dataudtrækning

Brugsmønster: Dataudtrækning igangsættes af *dataudtrakkeren*, se figur 5.7. Under dataudtrækningen besøges opskrifter på en opskriftsside (én af gangen). *Dataudtrakkeren* analyserer opskrifterne for at kunne skelne hvilke dele af den, der er ingredienser, fremgangsmåde, serveringsstørrelse, billede af retten, m.m. Alle opskrifter, der bliver fundet, bliver omsat, så de passer til systemets model af en opskrift, der tilføjes til systemets indeks.

Objekter: Opskrift, Ingrediens

Funktioner: Udfør dataudtrækning



Figur 5.7: Brugsmønsteret for dataudtrækning

5.2 Funktioner

For at skabe overblik over systemets funktionalitet er en funktionsliste blevet udarbejdet, hvilket beskriver, hvad systemet skal kunne. Dette skal som minimum dække over, hvordan aktørerne vil interagere med systemet.

Funktionslisten kan ses i tabel 5.2 og indeholder, ud over listen af funktioner, to parametre for hver funktion, funktionstype og kompleksitet. Funktionstype klassificerer generelle typer af funktioner på, hvordan de forbinder systemet med omgivelserne. Aflæsnings- og opdaterings-funktioner aflæser og ændrer i modellen, hvor beregnings-funktioner aflæser modellen og udarbejder noget nyt data, som vises i anvendelsesområdet. Kompleksitet er vores bedømmelse af, hvor svært det er at implementere funktionen, og kun de komplekse funktioner vil blive nærmere uddybet her.

Navn	Funktionstype	Egenskaber	Kompleksitet
Udfør dataudtrækning	Beregning		Kompleks
Opret favorit	Opdatering		Simpel
Slet favorit	Opdatering		Simpel
Vis favoritter	Aflæsning		Simpel
Slet bruger	Opdatering		Simpel
Håndter indkøbsliste	Opdatering		Medium
Søg efter opskrifter	Beregning		Medium
Skaler søgeresultat	Beregning		Simpel
Begræns søgeresultat	Beregning		Simpel
Ret brugerdata	Opdatering		Medium
Log ind	Opdatering		Simpel
Log ud	Opdatering		Simpel
Registrer bruger	Opdatering		Simpel
Opret fejl	Opdatering		Medium
Åbn fejl	Opdatering		Simpel
Luk fejl	Opdatering		Simpel

Tabel 5.2: *Funktionsliste over systemet.*

Funktionerne i ovenstående tabel bliver udført af aktørene på systemet. De første fire funktioner i tabel 5.2 er funktioner, der skal blive udført af dataudtrækkeren. Denne har til formål at trække relevant data ud af specifikke hjemmesider, og evt. oprette, slette eller opdatere opskrifter alt efter hvilken funktion, der er relevant. Hvis der f.eks. er sket en ændring i en af opskrifterne, så skal denne opdateres som følge heraf. De øvrige funktioner bliver udført af brugeren eller administratoren og kan ses i brugsmønstrene.

5.2.1 Specifikation af komplekse funktioner

Den komplekse funktion “udfør dataudtrækning” er beskrevet ved at dekomponere funktionen i mindre komplekse funktioner. Funktionslisten for “udfør dataudtrækning” kan ses i tabel 5.3. Under funktionen “find opskrift” udfører dataudtrækkeren dataudtrækning af en opskriftshjemmeside, og alle opskrifter identificeres. Funktionen “find ingredienser” finder alle ingredienser for opskrifterne, og funktionen “gem opskrift” tilføjer det hele til modellen.

Navn	Funktionstype	Egenskaber	Kompleksitet
Find opskrift	Beregning		Medium
Find ingredienser	Beregning		Medium
Gem opskrift	Opdatering		Simpel

Tabel 5.3: *Funktionsliste over den komplekse funktion “udfør dataudtrækning”.*

6 Design af arkitektur

Dette kapitel vil beskrive designet af systemet som følge af den foregående analyse af problemområdet i kapitel 4 og analyse af anvendelsesområde i kapitel 5. Vi vil først beskrive de kriterier, som skal gælde for vores system, samt vigtigheden af disse. Derefter beskriver vi designet af de komponenter systemet består af.

6.1 Kriterier

I dette afsnit vurderer vi hvor væsentlige forskellige kriterier er for det færdige system. Vi opstiller prioriteter for hver af disse kriterier for at skabe overblik over, hvor vi skal koncentrere vores arbejdskraft. Vincent J. har defineret 12 kriterier[38], der er som følger: *brugbart, sikkert, effektivt, korrekt, pålideligt, vedligeholdbart, testbart, fleksibelt, forståeligt, genbrugeligt, flytbart* og *integrerbart*. Hver af disse kriterier består desuden af 3-5 faktorer, der gør det nemmere at vurdere kriteriet. Feks. har kriteriet brugbart 3 faktorer: *træning, meddelsomhed* og *funktionalitet*.

Hvert kriterie har en vigtighedsgrad fra *irrelevant* til *meget vigtig*, eller vigtighedsgraden *trivial*, hvilket vil sige at kriteriet bliver opfyldt som følge af en af de andre kriterier.

Nogle af kriterierne strider imod, eller har større indflydelse på andre, kriterier. Feks. hænger *genbrugbart* og *effektivitet* ikke særlig godt sammen, da det ene kriterie modstrider det andet, og det kan derfor gøre en stor forskel i vurderingen (der refereres til [38, s. 18], hvis mere information om “trade-offs” imellem kriterierne ønskes). Hvis to modstridende kriterier er vigtige for et system, vil det tage ekstra lang tid at designe og implementere systemet.

Eftersom mængden af tid og arbejdskraft er begrænset er det vigtigt at der balanceres mellem mængden af vigtige kriterier og mindre vigtige kriterier. En tommefingerregl er derfor at man ikke bør have mere end 2 “meget vigtige” kriterier.

6.1.1 Begrundelser for hvert kriterie

Kriterium	Vigtighed				
	Meget vigtig	Vigtig	Mindre vigtig	Irrelevant	Trivialt
Brugbart	✓				
Sikkert				✓	
Effektivt		✓			
Korrekt			✓		
Pålideligt			✓		
Vedligeholdbart		✓			
Testbart			✓		
Fleksibelt		✓			
Forståeligt		✓			
Genbrugbart				✓	
Flytbart			✓		
Integrerbart			✓		

Tabel 6.1: Vurdering af designkriterier for foodl.

Gruppen er i sammenråd med vores informanter, blevet enige om, hvad der rent designmæssigt er væsentligt for at foodl bliver mest attraktivt. Her under en liste over kriterierne og en vurdering af kriteriernes betydning for foodl. Vurderingerne er ligeledes opsummeret i tabel 6.1.

Brugbart (meget vigtig) Systemet skal være håndgribeligt for brugerne. Det skal være nemt at bruge og nemt at gå til uden nogen form for oplæring, fordi vi har vurderet, ud fra interviews med informanterne, at de fleste personer, ikke ønsker at skulle bruge meget tid på at skulle sætte sig ind i vores system.

Sikkert (irrelevant) Systemet behandler ikke personfølsomme oplysninger, som f.eks. kreditkortnumre. Folks sikkerhed afhænger på ingen måde af systemet er sikkert, som f.eks. systemer der styrer lyskryds gør. Sikkerhedskriteriet vurderes som irrelevant for systemets endelige design.

Effektivt (vigtig) Generelt er det vigtigt, at søgninger og navigation i en webapplikation foregår hurtigt og responsivt. Hvis man bruger en webapplikation, hvor søgefunktioner og navigation ikke reagerer i løbet af relativt kort tid, bliver man hurtigt træt af det pågældende system og kan finde på at benytte et andet system. Marissa Mayer, tidligere leder hos Google, nu præsident og direktør for Yahoo, udtalte følgende: [18]

“When the Google Maps home page was put on a diet, shrunk from 100K to about 70K to 80K, traffic was up 10 percent the first week and in the following three weeks, 25 percent more.”

Det skal være hurtigt at bruge **foodl** til at finde en anvendelse af sin madrest i en opskrift. Det er utrolig hurtigt blot at smide madresten i skraldespanden, så hvis folk skal bruge **foodl**, så er det vigtigt at de ikke mister tålmeldigheden fordi de hele tiden skal vente på at en proces på **foodl** bliver færdig.

Korrekt (mindre vigtig) Korrektheden kategoriseres som mindre vigtigt, fordi gruppens fokus ligger på brugbarhed og effektivitet. Et par forkerte søgeresultater eller en bogmærket opskrift, der alligevel ikke blev bogmærket, kan måske skrämmme nogle brugere væk, men det er ikke et livsnødvendigt kriterie for systemet. Brugere vil stadig kunne få inspiration, til hvordan de kan bruge deres madrest, selvom 1 ud af 10 opskrifter måske ikke passer særlig godt på den madrest de vil bruge.

Pålideligt (mindre vigtigt) Pålideligheden vurderes som mindre vigtig, fordi det indebærer konsistens og nøjagtighed, hvilket vi anser som indre vigtigt. Hvis mængden af en ingrediens er en smule forkert er systemet stadig brugbart. Brugere kan endda klikke ind på den originale opskrift og se de helt korrekte mængder. Hvis søgemaskinen ikke er konsistent og ikke altid finder de samme opskrifter med den samme søgning, så gør det heller ikke det store. Det vil faktisk give brugeren endnu bedre inspiration til hvordan man kan anvende sine madrester, end ved blot at vise samme rækkefølge af søgeresultaterne hver gang samme søgning foretages.

Det ville dog være et problem, hvis et system går ned, fordi brugeren har udført en handling, der ikke er taget højde for. Systemet skal være i stand til at melde tilbage til brugeren, at der er sket en fejl uden at systemet går ned.

Vedligholdbart og fleksibelt (vigtig) Det er vigtigt for implementeringen af systemet, at det er nemt at udbygge, viderefudvikle og vedligholde. I og med at gruppen benytter sig af en iterativ arbejdsproces, så er det vigtigt, at gruppen kan vende tilbage til koden efter f.eks. en måned og, uden problemer eller forsinkelser, være i stand til at læse og modifcere i systemets funktioner. For at kunne udvide systemet til andre sprog eller tilføje opskrifter fra mange forskellige sider, er der god grund til at vedligholdelse og fleksibilitet er vigtige kriterier.

Testbart (mindre vigtig) Systemet kan gøres testbart ved at dele koden op i mange små funktioner, så hver af disse kan testes individuelt. Men opdelingen i for mange små stumper kode kan medføre en lavere effektivitet. Det er vigtigt at **foodl** er effektivt, så vi vurderer testbarheden mindre vigtigt, men har det i baghovedet at vi til en vis grad selvfolgelig skal kunne teste om systemet lever op til de funktioner, der kræves.

Forståeligt (viktig) **foodl** er ikke et system, der vil have et fast antal brugere. Derfor er det svært at forudsige behovet for antallet af udviklere tilknyttet systemet. Hvis **foodl** bliver meget populært kan der være behov for at få hjælp fra flere udviklere, og derfor er det vigtigt at systemet er forståeligt, så nye udviklere forholdsvis hurtigt kan komme i gang.

Genbrugbart (irrelevant) Da systemdefinitionen ikke stiller krav om at **foodl** skal kunne bruges af andre systemer vurderes dette som irrelevant.

Flytbart (mindre viktig) Systemet skal kunne flyttes frit mellem forskellige tekniske platforme, såsom Windows og Unix, hvis det bliver nødvendigt. F.eks. skal det være muligt at køre systemet på andre webhostingsservices eller skalere systemet op, hvis behovet overstiger det tekniske systems ydelse. Derfor skal vi tænke over hvilke eksterne systemer, der bliver benyttet i det færdige produkt. Nogle eksterne systemer har platformsafhængigheder, der begrænser valget af platforme, som produktet kan køre på. Derfor skal gruppen være i stand til konsekvent at benytte cross-platform-biblioteker og implementere selve systemet, sådan at det er let at flytte over til andre platforme. Det er ikke sikkert at **foodl** bliver så populært at webhosten ikke ydelsesmæssigt kan følge med og det er også uvist om en anden teknisk platform vil klare den belastning bedre. Derfor vurderes kriteriet som mindre vigtigt.

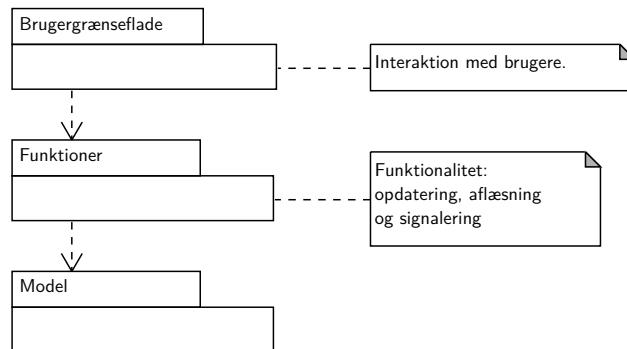
Integrerbart (mindre viktig) Da modularitet og standardisering af datarepræsentationer er en væsentlig del af implementationen af projektet, og kriterierne fleksibilitet samt vedligeholdbarhed er vurderet som vigtige, vil vi foretage nogle overvejelser i forhold til integrerbarhed. Det vurderes som mindre vigtig fordi systemet ikke skal benyttes i andre systemer (se “genbrugbart”). I forhold til at koble på andre systemer refereres til “flytbart”-kriteriet ovenover.

6.2 Komponenter

Når der hentydes til komponenter i sammenhæng med systemudvikling og programmeringsparadigmer, så er en komponent en samling af programdele, der udgør en helhed og har veldefinerede ansvarsområder. En programdel kan også være et helt delsystem, men i og med at platformen for vores system er objektorienteret, så vil de fleste programdele være de klasser, der findes i systemet[25, s. 191]. Resultatet ved at udarbejde en komponentarkitektur er et revideret klassediagram, der indeholder operationer, attributter og eventuelle nye klasser.

6.2.1 Generel komponentarkitektur

En 3-lag arkitektur, der ses i figur 6.1 er en meget generel komponentarkitektur[16]. Komponenten “brugergrænseflade” har ansvaret for at aflæse brugerens interaktion med systemet og ud fra denne opdatere, hvad brugeren får vist på skærmen. Funktionskomponenten er ansvarlig for systemets funktionalitet. Funktionaliteten bliver leveret til brugergrænsefladen som et set operationer på de offentlige klasser i funktionskomponenten. Modelkomponentens primære ansvarsområde er at holde styr på objekterne, der repræsenterer problemområdet. Sker der noget relevant i problemområdet, så skal modelkomponentens objekter skifte tilstand i overensstemmelse hermed.



Figur 6.1: Den generiske lagdelt komponentarkitektur.

Ud fra den simple lagdelte struktur, som er vist i figur 6.1, er der blevet udviklet en overordnet komponentarkitektur, som kan ses i afsnit 6.2.3.

6.2.2 Model-View-Controller

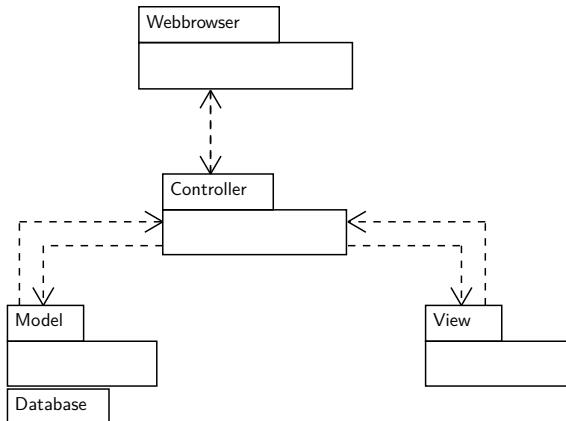
3-lag arkitekturen minder meget om Model-View-Controller mønstret, som vi benytter og kort har beskrevet i afsnit 6.2.2. Dog skal det nævnes, at de ikke er helt ens. I en 3-lags komponentarkitektur er det ikke muligt for præsentationslaget at tilgå modellen. MVC-mønstret er mere trekantet idet, at views kan tilgå både funktionskomponenten og modelkomponenten og dermed blive opdateret direkte fra modellen[17]. MVC-arkitekturen kan ses i figur 6.2. Der findes mange forskellige frameworks, der bygger på MVC-arkitekturen. Fordelen ved at benytte et eksisterende og anerkendt mønster, er dets mange funktioner og brugervenlighed, der gør det muligt at hurtigt kunne implementere de nødvendige dele af systemet.

Arkitekturen består af tre hovedkomponenter: model, view og controller. Modelkomponenten bærer de data, som typisk er repræsenteret som tabeller, i en database, der skal kunne manipuleres. Controllerkomponenten fungerer som et slags bindeledd mellem modellen og views ved at sende forespørgsler til modellen eller views, alt afhængig af brugerinputtet fra de tilsluttede views. Derudover sender controlleren også forespørgsler i form af tilstandsændringer til modellen, hvis brugeren f.eks. opdaterer noget i et view. Et view præsenterer brugeren for data i modellen og muliggør ændringer i tilstanden af modellen.

6.2.3 Den overordnede komponentarkitektur for foodl

En komponentarkitektur har til formål at skabe en fleksibel og forståelig strukturering af systemet. Med en god komponentarkitektur gør vi systemet let at forstå og organiserer designarbejdet. Igennem modelarkitekturen bliver det samlede design reduceret til en række mindre komplekse opgaver[25, s. 185]. Der findes mange forskellige frameworks i forskellige programmeringssprog, der gør det lettere at opbygge en god komponentarkitektur. Vi valgte at benytte programmeringssproget Ruby med frameworket Ruby on Rails, der bygger på MVC-arkitekturen. Ruby on rails beskrives i afsnit 8.1. Så snart en ny Railsapplikation bliver genereret, bliver der dannet mapperne “models”, “views” og “controllers”, så på den måde er man tvunget til at implementere MVC-arkitekturen i sit system. Det er et eksempel på principippet “konvention over konfiguration”, som Ruby on Rails tager udgangspunkt i. Et diagram over den konkrete måde, hvorpå Ruby on Rails implementerer MVC, kan ses i figur 6.2.

foodl benytter klient-server arkitekturmønstret (se figur 6.2). Det er oplagt af flere grund. Et vigtigt kriterie er, at systemet skal være vedligeholdbart. Det opnås bedst ved at lade systemet fungere som en webapplikation på en server. Ved at lade alle brugere af webapplikationen være klienter, kan vi hurtigt ændre eller tilføje noget til systemet, uden at brugeren skal have opdateret hele systemet, de skal blot genindlæse siden, fordi systemet ikke befinner sig lokalt hos den enkelte bruger. Ved blot at lade dette ansvar forblive på serveren, får brugerne et mere vedligeholdbart system. Vi lader derfor webbrowserkomponenten fungere som klient, der tilkobler sig serveren for at sende forespørgsler. Det er controller-komponenten, der sørger for at sende data frem og tilbage mellem klienterne og serveren. Der er altså kun én server, som integrerer med mange forskellige klienter igennem et netværk og deler fælles ressourcer med alle de klienter, der er tilkoblet. Serverkomponenten stiller diverse funktioner tilgængelig for klienterne, f.eks. at muliggøre søgning eller lagring af oplysninger i modelkomponenten. Serveren, i dette tilfælde, og i modsætning til klient-servermønstret beskrevet i OOA&D[25], kan godt kende noget til klienteren. F.eks. er det muligt, at vise andre views, alt afhængigt af hvilken webbrowser, der bliver tilkoblet serverkomponenten.



Figur 6.2: Et tilpasset generiske Model-View-Controller mønster indkapslet i et klient-server arkitekturmønster[24].

Forskellen på viewkomponenten, set i figur 6.2, og brugergrænsefladekomponenten, set i figur 6.1, er at et view er en slags skabelon for det, der skal sendes til brugergrænsefladen. Et view bliver kaldt af controllerkomponenten og bliver udfyldt med indhold fra modellen, igen gennem controllerkomponenten før den sendes til brugergrænsefladen. Brugergrænsefladens opgave er så at muliggøre interaktionen mellem bruger og system ved at vise indholdet i et view. Typisk vil brugergrænsefladen være en webbrowser på klientens side, der tilslutter sig webapplikationen ved at pege på webapplikationens webadresse. Et eksempel kunne være, at der skal en specifik visning af noget data i modellen til forskellige typer brugere, hvor førstegangsbrugere burde blive budt velkommen på siden, burde en administrator måske blive fortalt om brugere har indrapporteret fejl. Et view gør dette muligt, da det har et dynamiske indhold og henter ting fra modellen, alt afhængigt af brugerens adgangsniveau og sender det relevante layout op til brugergrænsefladen.

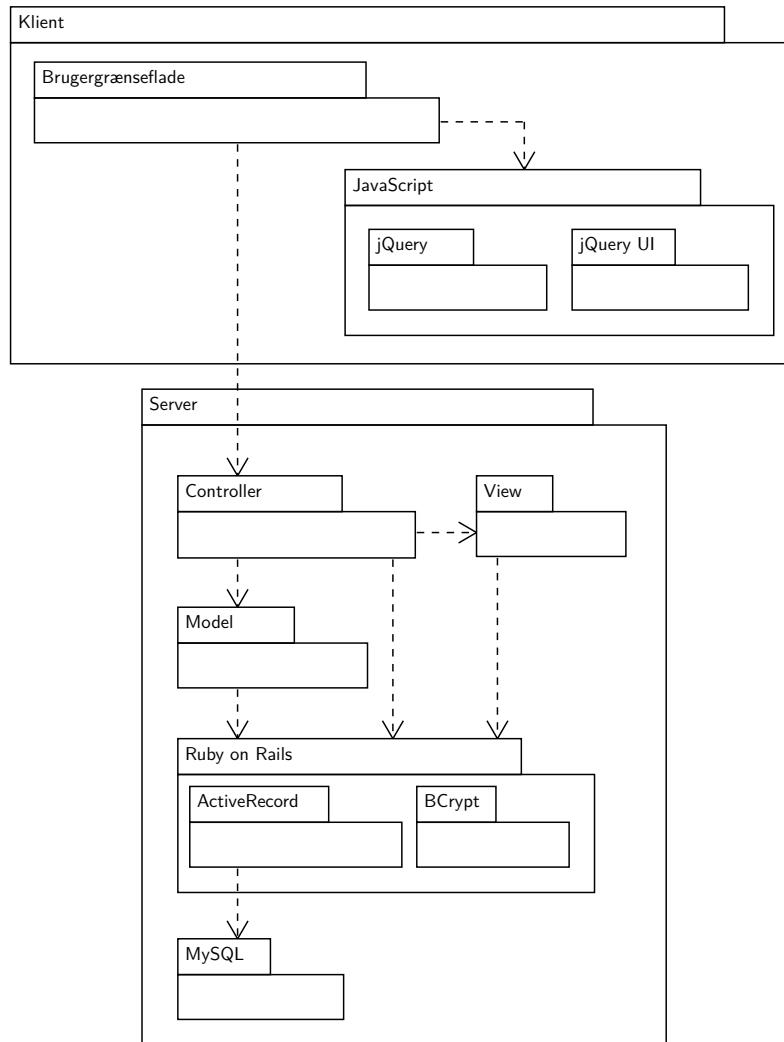
Som det blev nævnt i afsnit 6.2, så startede vi med en meget simpel lagdelt struktur for at få et godt udgangspunkt for det videre arbejde med komponentarkitekturen. Herefter har vi arbejdet på at identificere de forskellige komponenter, der er i vores system og få dem placeret i de korrekte forhold til hinanden. Resultatet af bearbejdningen kan ses i den nye komponentarkitektur vist i figur 6.3.

I forhold til den generiske lagdelte komponentarkitektur set i figur 6.1, er der kommet to lag mere på i bunden af figur 6.3, der specificerer systemets tekniske platform. Det første lag, lige under *model*, er et Ruby on Rails-lag. Ruby on Rails laget tilgår og ændrer data i en database. I vores system benytter vi en MySQL database, der kræver at man bruger MySQL-forespørgsler til at tilgå og ændre i databasen. Dette illustreres som det nederste lag i komponentarkitekturen. Viewkomponenten er også ny og er en del af MVC-mønstret. Den tillader en nem måde, at fremstille skabeloner for de forskellige sider i webapplikationen. Controllerkomponenten har overtaget funktionskomponentens plads men fungerer ligeledes som laget, hvorfra handlinger udføres, ligesom det gjorde i funktionskomponenten.

Komponentarkitekturen i figur 6.3 viser, hvordan de forskellige komponenter har forskellige ansvarsområder. Brugergrænsefladen kommunikerer bl.a. med controllerkomponenten ved at videresende brugerens interaktion til controlleren. Controlleren kan ud fra den interaktion enten sende et forespørgsel for at få vist et specifikt View, eller kommunikere med modelkomponenten, hvis brugerens interaktion kræver en ændring eller visning af objekter fra modellen af problemområdet. For at gøre brugergrænsefladen dynamisk, benytter vi JavaScript, som denne har direkte kontakt til. JavaScript er et eksisterende system, som vi benytter til klient-side-scripting. Sammen med JavaScript benytter vi jQuery og jQueryUI, som er frameworks til JavaScript, der giver os mulighed for lettere at tilgå forskellige elementer i brugergrænsefladen og bruge nogle indbyggede grafiske elementer og symboler.

Både Controller, View og Model har direkte kontakt med den tekniske platform, hvorfra der er kontakt til databasen. Active Record-komponenten indbygget i Ruby on Rails giver muligheden for at kommunikere med databasen, uanset hvilket system den underliggende database består af. I softwareudvikling er Active Record et udviklingsmønster,

der findes i diverse framweorks, som repræsenterer data fra relationelle databaser i et objektorienteret miljø. Ruby on Rails har en implementering af Active Record, som vi benytter med MySQL-databasen. Controlleren, hvori et objekt er i overenstemmelse med dette mønster, vil omfatte funktioner som "Insert", "Update" og "Delete", samt egenskaber, der ca. svarer til kolonnerne i den underliggende databasetabel[39]. Derudover består den tekniske platform af BCrypt, som er et eksternt system, der bliver brugt til hashing af kodeord, så disse ikke blive gemt som ren tekst i databasen.



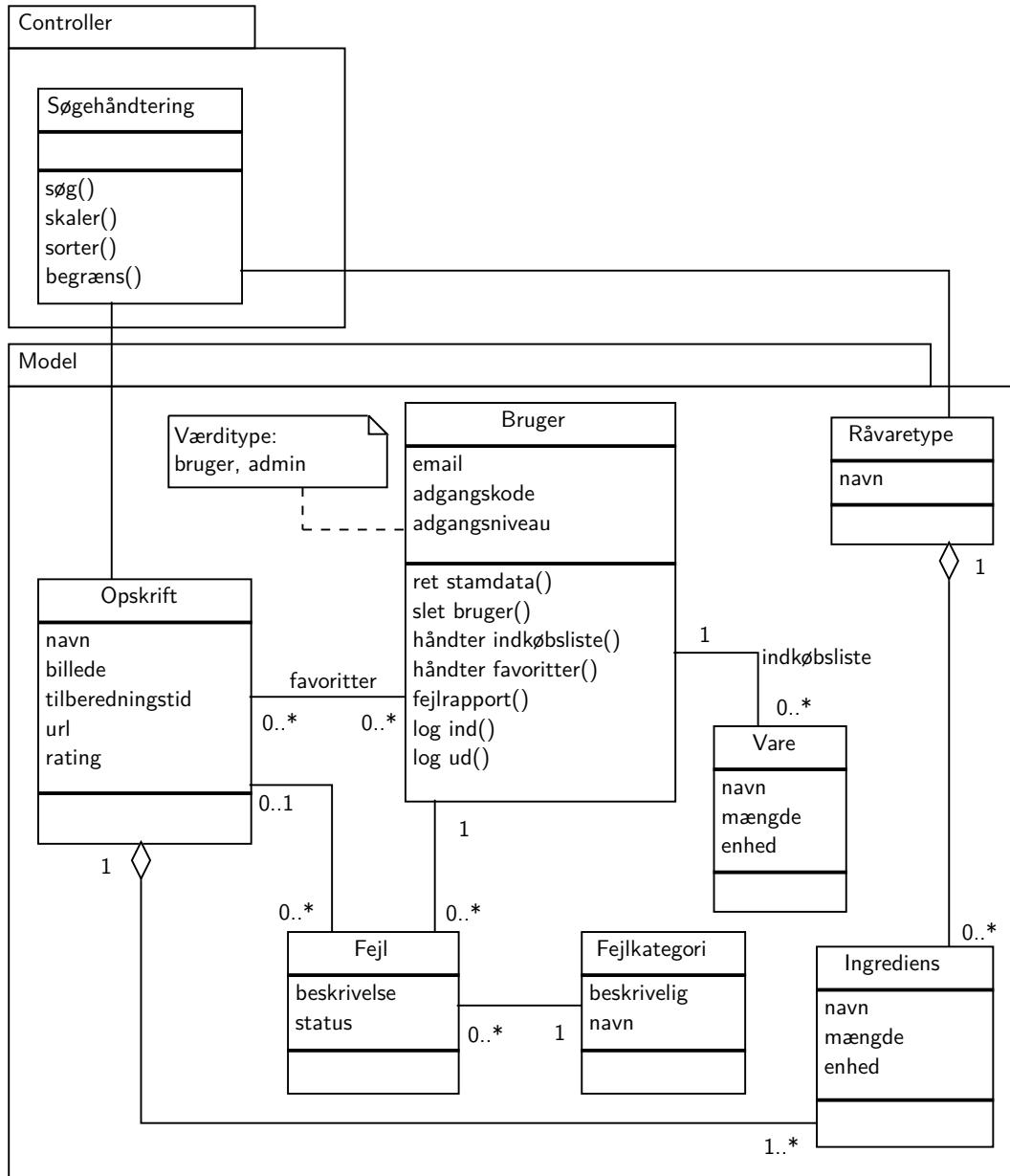
Figur 6.3: Den overordnede komponenetarkitektur for *foodl*.

Klienten vil bestå af Brugergrænsefladen og JavaScript, som er grænsefladekomponenten. Controllerkomponenten og modelkomponenten vil køre på selve serveren.

6.2.4 Model- og controllerkomponent

Model- og controllerkomponenterne, i figur 6.4, repræsenterer objekterne og deres funktionalitet i problem- og anvendelsesområderne. Vi har opdateret klassediagrammet fra problemområdet i kapitel 4 ved at benytte klassernes adfærd, der er beskrevet i afsnit 4.3, til at danne attributter og nye klasser. Operationerne på klasserne er blevet

dannet ud fra funktionerne, som er beskrevet i anvendelsesområdet i kapitel 5. Klasserne er opdateret eller ændret på anden vis ud fra analysen af problemområdet i kapitel 4.



Figur 6.4: Det endelige klassediagram.

Klassen Bruger afspejler de personer, der kan bruge systemet. Der findes to slags brugere af systemet. En bruger, der har et adgangsniveau som administrator, og en bruger, der ikke har adgangsniveau som administrator. Dette håndteres af værditype-attributten *adgangsniveau*, som ses i figur 6.4. En bruger, der er administrator kan læse, der er indberettet af en almindelig bruger. Derudover kan en administrator slette andre brugere af systemet. Vi kunne godt have brugt rollemønsteret, hvor klassen Bruger kunne have en rolle som administrator eller en rolle som almindelig bruger. Vi mente dog at forskellen på de to roller (administrator og bruger) var for lille til, at vi ville benytte rollemønsteret, og benyttede i stedet en attribut til at beskrive deres rolle. Alle brugere har adgang til funktionerne “ret stamdata()”, “håndter indkøbsliste()”, “håndter favoritter()” og “fejlrapporrt()”, som går ud på at indberette en fej

6. Design af arkitektur

vedr. siden eller en opskrift. For at en bruger, der har benyttet systemet tidligere, kan få adgang til sin indkøbsliste og favoritter, tilføjes attributterne *email* og *adgangskode* til Bruger-klassen. På den måde kan systemet hurtigt finde den rigtige Bruger-klasse og forhindrer samtidig andre at få adgang til det samme Bruger-objekt, medmindre de selvfolgelig kender den pågældende email og adgangskode.

Det er ikke obligatorisk at være logget ind på systemet for at bruge det. Klassen Bruger har fortsat de samme associationer til Fejlrapport, Opskrift og Vare, som klassen havde i problemområdet. Klassen Fejlrapport er dog blevet delt op i to klasser; Fejl og Fejlkategori.

I problemområdet er hændelsen “fejl rapporteret” en iteration, hvilket er grundlag for at klassen Fejl oprettes. Man kan forestille sig, at der kan forekomme mange forskellige typer fejl. Det kan være fejl i en opskrift, på indkøbslisten eller på søgesiden. For at lette overblikket over håndteringen af disse fejl for en administrator, tilføjer vi klassen Fejlkategori. Når brugere rapporterer en fejl, kan de tilknytte fejlen til en kategori. Fejlkategori-klassen kunne være erstattet af en værditype-attribut på klassen Fejl, men i så fald ville det være svært at tilføje nye fejlkategorier. Derfor har vi modelleret en håndteringen af fejlkategorier som en ny klasse (Fejlkategori). Nogle af de fejl, som brugerne rapporterer, kan have behov for at få tilknyttet en lille beskrivelse af fejlen, mens andre fejl ikke behøver dette. En fejl som f.eks. et dødt link til en opskrift behøver ikke at have en beskrivelse tilknyttet, da det blot medfører at klassen Fejl bliver associeret med en opskrift.

Klassen Vare er associeret til klassen Bruger i form af en indkøbsliste, hvorpå brugerens varer er skrevet på. Vi har ikke modelleret indkøbslisten som en klasse, fordi hver bruger blot har én enkelt indkøbsliste af gangen. En bruger kan altså være tilkoblet mange varer (tilføjet manuelt eller gennem en opskrift) og denne liste af varer bliver dermed gemt på indkøbslisten. Det samme gælder for associationen mellem Bruger og Opskrift, som er en association i form af en favoritliste. Vi har ikke modelleret favoritter som en klasse, fordi vi har vurderet, at det er blot et forhold mellem brugeren og opskriften. Men derfor skal Opskrift være en klasse for sig.

Klassen Opskrift har i problemområdet attributten *kilde*. Da vi i vores aktørspecifikationer kun har givet et eksempel på at Dataudtrækkeren benytter hjemmesider som kilde til opskrifter, og ikke kan forestille os at *foodl* vil benytte andre kilder, vælger vi i modelkomponenten at omdøbe attributen *kilde* til *url*. Informanterne fortalte under møde 2, at det ville være smart, hvis det var muligt, at kunne se hvad andre brugere synes om en opskrift. Derfor tilføjer vi attributten *rating* til opskriften, der vil afspejle hvor mange andre brugere, der har valgt at favorisere denne opskrift i forhold til den opskrift, der har opnået flest favoriseringer. Hvis den opskrift, der har fået flest favoriseringer, har fået 200, og *Gulerodssuppe* har fået 50, så får *Gulerodssuppe* ratingen 25 %, og alt efter hvordan dette designmæssigt bedst vises på en brugergrænseflade, kan de 25 % mappes til f.eks. x ud af y stjerner.

I controllerlaget findes klassen Søgehåndtering, som indeholder funktionerne “søg()”, “skaler()”, “sorter()” og “begrens()”. Søgehåndtering er dannet ud fra den viden, vi har opnået ved at analysere problemområdet i kapitel 4, hvor man er nødt til at udføre en form for søgning efter opskrifter for at finde opskrifter, man kan lave. Søgehåndtering er associeret med klasserne Opskrift og Råvaretype, fordi man udfører en søgning baseret på råvaretyper, og søgeresultatet er en liste af opskrifter. Det er klassen Søgehåndtering, der gør det muligt bl.a. at søge på opskrifter. Søgningen er baseret på brugerindtastede råvaretyper. En bruger kan derefter bruge funktionerne, der findes i Søgehåndtering til at skalere, sortere eller på anden vis begrænse søgeresultatet.

7 Design af brugergrænsefladen

Dette kapitel har til formål at præsentere systemets brugergrænseflade som følge af interaktionen med de to informanter. Vi benyttede prototyper til at inddrage informanter i designbeslutninger, hvor vi præsenterede vores idéer, og vi iagttog deres interaktion med prototyperne. I afsnit 7.2 præsenteres og beskrives den endelige brugergrænseflade i det færdigudviklet system.

7.1 Prototyper

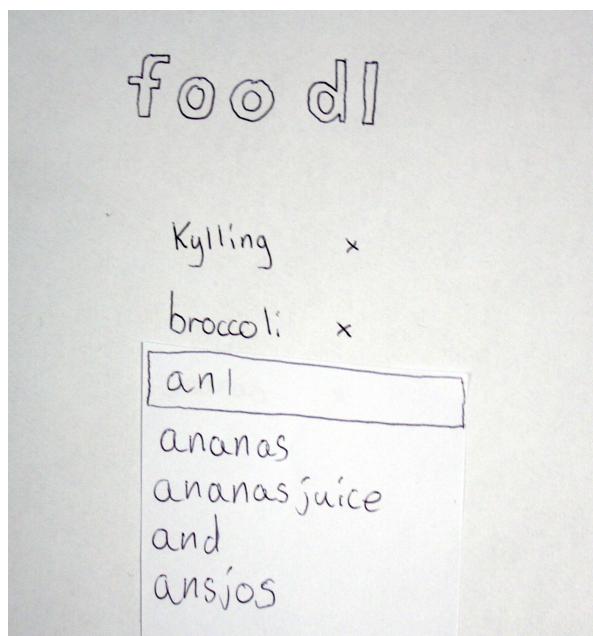
Vi udarbejdede to prototyper, der fokuserede på søgefunktionaliteten, som bliver præsenteret i afsnit 7.1.1, og en større prototype, der fokuserede på systemets andre funktioner, som bliver præsenteret i afsnit 7.1.2.

Vi udførte prototypeafprøvningerne hjemme hos begge informanter. Informanterne blev bedt om at udføre en case, som havde til formål at få dem igennem hver hoveddel af prototyperne, så vi havde mulighed for at overvåge dem, og efterfølgende diskutere prototyperne med dem. Afprøvningerne er dokumenteret ved hjælp af videooptagelser, som refereres til i bilag C.3 for prototyperne, der havde fokus på søgefunktionalitet, og i bilag C.4 for prototypen, der havde fokus på systemets andre funktioner.

Prototyperne med fokus på søgefunktionalitet blev afprøvet i første fase af projektforløbet, og prototypen med fokus på systemets andre funktioner blev afprøvet i anden fase, hvilket også er illustreret i tabel 2.1 i afsnit 2.1.

7.1.1 Prototyper med fokus på søgefunktionalitet

Systemets vigtigste funktionalitet er, at brugerne skal være i stand til at udføre en søgning på nogle råvaretyper, som skal give dem et brugbart resultat i form af en liste af opskrifter. Uden en søgefunktion er systemet ikke meget værd. Derfor har vi udarbejdet og afprøvet to prototyper, der fokuserer på to forskellige måder, hvorpå man kan udføre en søgning. Den første prototype, som kan ses på figur 7.1, anvendes ved at stave til en råvaretype. Systemet vil ud fra de indtastede bogstaver, give forslag til, hvilke råvaretyper, der er søgbare i systemet. Ud fra den liste, som bliver præsenteret, skal informanten vælge den ønskede råvaretype.



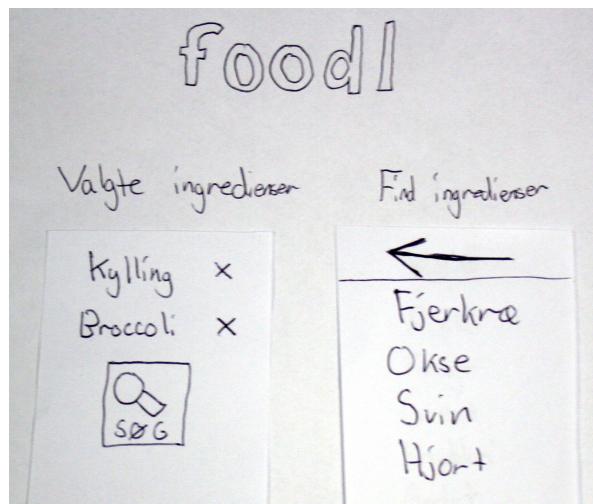
Figur 7.1: Visualisering af prototype 1A, der har fokus på søgefunktionalitet.

7. Design af brugergrænsefladen

På figur 7.1 skal det forestille sig, at informanten i forvejen har valgt to råvaretyper (kylling og broccoli). Disse valgte råvaretyper kan fjernes fra søgningen igen ved at klikke på det lille kryds ud for den råvaretype, man vil fjerne. Derudover er informanten i færd med at indtaste et ord, og der er blevet indtastet bogstaverne "a" og "n". Detmedfører, at systemet giver nogle forslag til søgbare råvaretyper. Man kan vælge at skrive hele ordet færdig eller navigere ned og klikke på den ønskede råvaretype.

Søgefunktionen på figur 7.1 minder meget om, hvordan en søgning på f.eks. Google bliver foretaget. Det er meget minimalistisk, da der et logo og et søgefelt i centrum og dermed i fokus. Google bliver brugt af riktig mange mennesker[2]. Der er blevet foretaget undersøgelser vedrørende menneskers hukommelse i forhold til genkendelse af objekter og det at skulle komme i tanke om et objekt. Mennesker er meget bedre til at genkende ting, og det er grundet til, at vi prøver at udvikle en søgefunktion, der minder meget om Googles. Der en større chance for, at brugere af systemet kan genkende søgefunktionaliteten og derved være i stand til at udføre en søgning uden problemer, hvis de kan genkende designet [4, p. 340].

En anden måde, hvorpå en søgefunktion kan designes er præsenteret i figur 7.2. Her skal man manuelt vælge råvaretyper fra kategoriserede lister. Feks. kan man finde råvaretypen "kylling" under kategorien "Fjerkræ".



Figur 7.2: Visualisering af prototype 1B, der har fokus på søgefunktionalitet.

Søgefunktionen på figur 7.2 præsenterer brugeren for to vinduer, hvor man i det venstre vindue kan se, hvilke råvaretyper, man allerede har valgt, og man vælger nye råvaretyper i det højre vindue. Det er i det højre vindue, hvor man skal navigeres sig igennem kategorierne og finde den ønskede råvaretype. Når man har tilføjet alle de ønskede råvaretyper, så kan man udføre en søgning ved at klikke på knappen "Søg", som er placeret i det venstre vindue lige under de allerede valgte råvaretyper.

Begge informanter var enige om, at prototypen 1A virkede mere intuitiv og mindre rodet end prototype 1B. Informanterne mente begge, at prototype 1B var langsommere at bruge, og de syntes ikke om den, fordi den faktisk var svær at navigere rundt i. Som et eksempel på, at prototype 1B ikke var intuitiv, er at Merete var i tvivl om, hvilken kategori hun skulle vælge kylling fra (kød eller fjerkræ?).

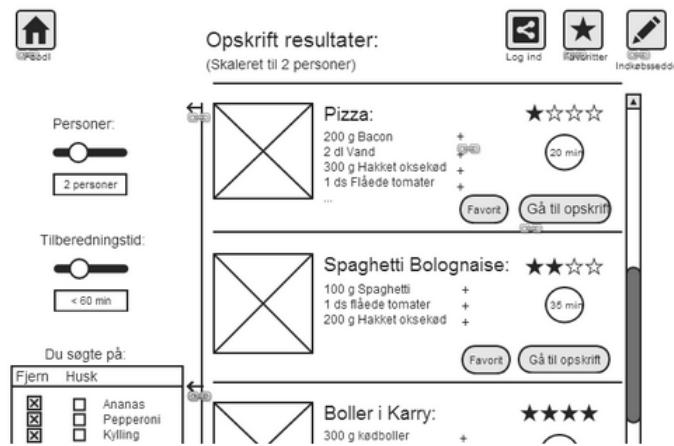
Kategorierne kan laves på mange forskellige måder, og uanset hvordan de vælges, vil der med garanti være nogle brugere, der vil være i tvivl om, hvor de skal lede efter en bestemt råvaretype. Begge informanter var enige om, at prototype 1A var hurtig, nem og effektiv at bruge.

På baggrund af informanternes feedback, valgte vi at arbejde videre med prototype 1A. I afsnit 7.2 introducerer vi brugergrænsefladen for systemet, hvilket bærer meget præg af vores test med informanterne og disse prototyper.

7.1.2 Prototype med fokus på systemets funktionalitet

Ud fra afprøvningerne af prototyperne, hvor der var fokus på søgerfunktionen, har vi udarbejdet en prototype, der fokuserer på resten af systemets funktioner. Til at test disse andre funktioner, brugte vi en diashow-prototype, som er dynamisk, hvilket giver en fornemmelse af et fungerende system. Denne prototype kalder vi for prototype 2, og den er vist på figur 7.3. Afprøvningerne af prototyperne i afsnit 7.1.1 havde til formål at give os mulighed for at træffe et kvalificeret valg i forhold til, hvordan søgerfunktionen skulle designs.

Figur 7.3 viser opskrifter, der er resultatet af en søgning, som er blevet udført af informanten. Opskrifterne er blot eksempler, som vi selv har lavet for at det skal ligne så realistisk, som muligt. Der var fokus på, hvorvidt informanterne var i stand til at navigere rundt på siden, og at undersøge om funktionernes placering, synlighed og brug var intuitiv for informanten.



Figur 7.3: Visualisering af prototype 2, der har fokus på systemets funktionalitet. Udviklet vha. <http://gomockingbird.com>.

I toppen af prototype 2 ses fire knapper. Ser vi på knappen i venstre side, så skal denne forestille en "hjem"-knap, og derefter kommer en "log ud/ind"-knap, og så en "favorit"-knap og en til højre en "indkøbsliste"-knap. Vi har benyttet os af både figurer, der minder om den hændelse, der vil ske, når man klikker på knappen, og kort beskrivende tekst. Her benytter vi os af menneskers gode evne til at genkende objekter og deres betydning [4, s. 340]. Begge informanter kunne forstå, hvad knapperne i toppen af siden skulle bruges til. Merete var dog en smule i tvivl om "favorit"-knappen havde til formål at tilføje opskrifter til en favoritliste eller om den viste de favoritter, man havde gemt.

I venstre side af prototype 2 ses en sidebar, hvor der er nogle funktioner til at begrænse eller på anden vis ændre søgeresultatet. Fra sidebaren er det bl.a. muligt at skalere opskrifter mht. antal personer, begrænse søgeresultatet i forhold til tilberedningstid eller i forhold til råvaretyper (hvilket er nyttigt, hvis man f.eks. er allergiker eller ikke spiser noget specifikt). Derudover kan man se sin nuværende søgning i sidebaren og fjerne eller huske de indtastede råvaretyper. Denne sidebar er normalt gemt, og man skal klikke på en tast i venstre side af søgeresultatet for at åbne sidebaren. Begge informanter havde problemer med at opdage, at der var en sidebar, fordi den var gemt væk til at starte med. De var enige om, at den sagens kunne forblive åben, når man havde udført en søgning. Informanterne var også enige om, at det var meget nyttigt med de forskellige funktioner til at begrænse opskrifterne med, men f.eks. ved skaleringsfunktionen behøvede man ikke at være i stand til at skalere til f.eks. 30 personer, da der netop var tale om brug af madrester.

Som det ses på prototype 2 så vises opskrifterne i en liste, og hver opskrift har nogle funktioner, der er tilknyttet til den ene opskrift. Herfra er det muligt at tilføje ingredienserne til en indkøbsliste vha. de små +'er ud for hver

7. Design af brugergrænsefladen

ingrediens. Derudover kan man favorisere en opskrift, så den er gemt i favoritlisten og man kan se, hvor populær en opskrift er ved de stjerner, der er ved hver opskrift, samt tilberedningstiden for den enkelte opskrift lige under. Når man har valgt en opskrift, så kan man gå til opskriftens hjemmeside, hvor fremgangsmåden (og alt andet indhold) kan findes. Merete havde ikke overvejet at +'et skulle bruges til at tilføje en ingrediens til indkøbslisten. Derudover var begge informanter enige om, at de nuværende funktioner var nytte, og at det var helt fint, at man kun kunne se, hvilke ingredienser man skulle bruge og så navigere til den originale side for at se hele fremgangsmåden.

7.1.3 Sammendrag

Merete var en smule forvirret omkring brugen af knapperne i toppen af siden. Et forslag var, at vi kunne tilføje mere sigende tekst som "Vis favoritter" i stedet for, at der blot står "Favoritter" eller lignende. Informanterne synes også, at små funktioner som +'et, der bruges til at tilføje en ingrediens til indkøbslisten, skal være mere intuitiv.

Informanterne var enige om, at en så stor del af funktionalitet, som sidebaren er, skal være synlig igennem hele processen, fra man er på søgeresultatsiden. De mente ikke, at den skulle gemmes væk.

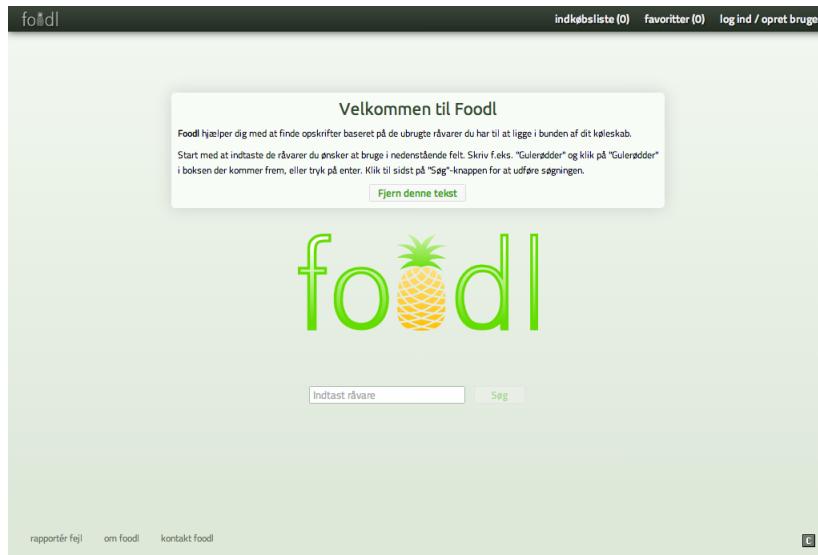
Derudover kunne informanterne godt finde ud af at bruge de andre funktioner i systemet, f.eks. indkøbslisten.

7.2 Brugergrænseflade

Som resultat af prototypeafprøvningerne med informanter, som er beskrevet i afsnittene 7.1.1 og 7.1.2, følger her en præsentation og beskrivelse af den endelige brugergrænseflade. Vi har udviklet en webapplikation, som vi har valgt at kalde **foodl**.

7.2.1 Forside

Når man taster sig ind på hjemmesiden <http://foodl.dk>, bliver man mødt af en velkomsthilsen, der meget kort beskriver hjemmesidens formål og brug, som kan ses på figur 7.4. Denne hilsen kan brugeren vælge at lukke ned. En cookie bliver gemt i browseren, så velkomsthilsnen ikke bliver vist igen, medmindre browserhistorikken bliver ryddet.

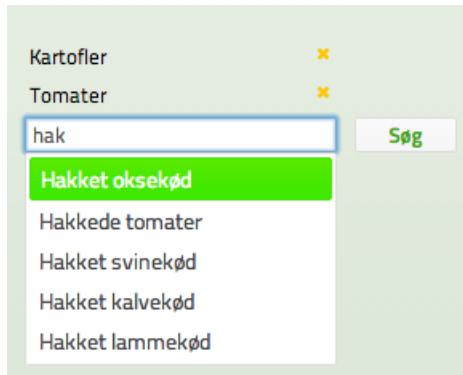


Figur 7.4: Denne figur har til formål at give et overblik over systemets forside.

Navnet **foodl** er også en del af webapplikationens logo. For at gøre det klarere for en ny bruger, hvad siden handler om, erstattede vi et “O” i navnet med en stor ananas, fordi det er noget, der kan spises, og sidens formål er at give folk mulighed for at genbruge deres madrester.

På toppen af alle undersider af **foodl** er det muligt at tilgå sidehovedet. Her er der mulighed for at navigere tilbage til forsiden ved at klikke på den mindre version af det logoet. Derudover kan man tilgå både en indkøbsliste, der er nærmere beskrevet i afsnit 7.2.3, og en liste af favorit-opskrifter, som brugeren selv vælger fra hjemmesiden. Favoritlisten bliver beskrevet nærmere i afsnit 7.2.4. Både indkøbslisten og favoritter har et tal i parentes, der fortæller brugeren, hvor mange varer, der er i den nuværende indkøbsliste, eller hvor mange opskrifter, der er gemt under favoritter. Dette kan ses på toppen af figur 7.4. På sidehovedet kan man også logge ind i systemet eller oprette en bruger, hvilket er forklaret nærmere i afsnit 7.2.5.

Efter brugeren føler sig tryg ved hjemmesiden og evt. har lukket velkomsthilsnen ned, så er det tid til at indtaste alle de råvaretyper, der ønskes brugt til madlavningen. Figur 7.5 viser, hvordan en sådan søgning foregår. Der bliver løbende indtastet bogstaver, og systemet undersøger for dele af tekstrække, der matcher det, som bliver indtastet. Ud fra disse match gives der forslag til hvilke råvaretyper, man kan vælge. Man kan ikke indtaste, hvad som helst som et søgekriterie i systemet. Der er en lang række råvaretyper at vælge imellem. Hvis der f.eks. bliver indtastet kød i søgefeltet, så kommer der en liste af matchende råvaretyper som forslag, se figur 7.5. Der er ingen begrænsning for, hvor mange råvaretyper, der kan indtastes som søgekriterier.



Figur 7.5: Denne figur viser systemets søgefelt.

For at fuldføre en søgning skal man blot trykke på “Søg”, der er til højre for søgerfeltet. Når brugeren trykker “Søg”, så arbejder systemet på at finde alle de opskrifter, der minimum har én ingrediens, der matcher en af de indtastede råvaretyper. Det er efter sådan en søgning, at brugeren finder ud af, hvad der er muligt at lave ud fra de råvaretyper, der er til rådighed (resultatet er afgrænset til den database, man har over opskrifter).

7.2.2 Resultatside

Når en søgning bliver udført, så bliver brugeren nавигeret videre til søgeresultatsiden, hvor man får præsenteret en liste af opskrifter, der matcher de søgekriterier, der er blevet søgt på. Figur 7.6 giver et overblik over, hvordan sådan en liste kan se ud.

I første omgang er opskrifterne sorteret efter relevans, dvs. hvor mange ingredienser, der matcher de forskellige indtastede råvaretyper. Figur 7.6 viser et eksempel af, hvordan sådan en liste ser ud. De ingredienser, der matcher søgekriterierne bliver markeret med fed skrift.

På hjemmesiden vises der kun hvilke ingredienser, der skal til for at lave opskriften, men selve fremgangsmåden er ikke vist nogen steder. Man er nødt til at tilgå den oprindelige hjemmeside, hvorfra opskriften stammer fra. Dette

7. Design af brugergrænsefladen

The screenshot shows a search interface with a sidebar on the left containing a search bar and filters for preparation time (less than 30 min, 30-60 min, more than 60 min) and number of people (2). The main content area displays two recipe cards:

- Fylde pitabrød - af en rest boller i karry**: A card for "Fylde pitabrød - af en rest boller i karry". It includes a photo of the dish, a list of ingredients (karrysauce, yoghurt, ananas, etc.), a preparation time of 30 min, and a rating of 5 stars.
- Marcipankage med hvid chokoladecreme**: A card for "Marcipankage med hvid chokoladecreme". It includes a photo of the dessert, a list of ingredients (Arla Karolines Køkken®, kormene fra 1 stang vanilje, etc.), and a rating of 5 stars.

At the bottom left, there are links for reporting errors, contacting foodl, and reporting a bug. At the top right, there are links for shopping list, favorites, logging in, and creating a new user.

Figur 7.6: Denne figur har til formål at give et overblik over systemets resultatside.

gøres ved at trykke på enten opskriftens titel eller billedet. Begge elementer består af et link til opskriftens originale hjemmeside.

Opskrifternes fremgangsmåde kan ikke ses på **foodl**, men alle andre vigtige elementer af opskriften er synlige. En opskrift består af følgende elementer:

- Titel
- Billede
- Vurdering
- Tilberedningstid
- Relevans (antal matchende ingredienser)
- Ingredienser
- Knapper
 - Tilføj alle ingredienser til indkøbsliste
 - Tilføj / fjern fra favoritter
 - Tilføj enkelte ingredienser til indkøbsliste
 - Indmeld en fejl med opskriften

Alle opskrifter består af en beskrivende titel og et relevant billede, der skal vise brugeren, hvordan opskriften kan se. Billedet er med til at vække en interesse hos brugeren og var i øvrigt ønsket fra informanterne. Tilberedningstiden er også en vigtig ting at være klar over, og denne kommer under billedet. De matchende ingredienser er markeret med fed skrift, så har brugeren nemmere ved at gennemskue, hvad der f.eks. er relevant at handle ind ud fra alle ingredienserne. Derudover er der et sæt knapper, som brugeren kan bruge. Se figur 7.7. I øverste højre hjørne er der

en knap, der har et notesblok-lignende ikon. Denne knap tilføjer alle ingredienserne til indkøbslisten. Der er også mulighed for at tilføje de enkelte ingredienser ved at trykke på de små +’er ud for ingredienserne. Indkøbslisten bliver beskrevet yderligere i afsnit 7.2.3. Ydermere er der mulighed for at tilføje og fjerne en opskrift til ens favoritliste. Dette gøres ved at trykke på den hjerte-formede knap. I nederste højre hjørne er der en advarselsknap, der bruges til at rapportere om eventuelle fejl ved den specifikke opskrift.



Figur 7.7: Her ses et eksempel på en opskrift, der kan være et resultat på en søgning.

I toppen af resultatsiden, som er vist på figur 7.6, er der en toolbar, hvorfra brugeren har forskellige muligheder for at manipulere søgningsresultatet. Der er blevet implementeret en toolbar i toppen af siden, der følger brugerenens bevægelser mht. at scrolle op og ned. På denne måde behøver brugeren ikke at scrollle helt til toppen for at udføre en handling på søgningsresultatet.

Figur 7.8 præsenterer toolbaren. I venstre side er der en samling af tre knapper, der bruges til at begrænse søgningsresultatet mht. tilberedningstiden. Her er der mulighed for at markere flere af gangen, og default kriteriet er, hvis ingen er markeret, så er alle markeret. Dette betyder, at systemet starter med at vise alle resultater. I midten af toolbaren er der et skaleringssværktøj, der kan bruges til at skalere opskrifternes portioner mht. antal personer. Man kan skalere dem ned til en person og op til 10 personer. Vi valgte 10 som maximum, fordi det er relativt let at skalere yderligere, hvis dette er ønsket og det er de færreste gange, man serverer rester for over 10 mennesker. I højre side af toolbaren er der endnu en samling af tre knapper, men disse benyttes til at sortere opskrifterne. Default er “relevans”. De to knappesamlinger er forskellige på to måder; hvad de bruges til, og at der kun er mulighed for at markere en knap af gangen ved sorteringsknapperne (højre side) og mulighed for markering af flere af gangen ved afgrænsningsknapperne (venstre side).



Figur 7.8: Systemets toolbar, der er direkte under sidehovedet.

Ud over toolbaren, så viser figur 7.9 en sidebar, hvilket gør det muligt for brugeren at følge med i, hvad der blev søgt på, og den giver brugeren mulighed for at lave endnu en søgning. Man kan herfra slette og/eller tilføje nye råvaretyper til en ny søgning. Denne sidebar følger også med brugeres skærmrulning, da det skal være nemt og hurtigt at lave en ny søgning, hvis dette bliver aktuelt, uanset hvor mange opskrifter, der bliver vist som resultat.



Figur 7.9: Systemets sidebar, der vises ved resultatsiden.

Hvis en bruger vælger at benytte sig af indkøbslisten, så kan denne tilgåes via sidehovedet, ved at trykke på “indkøbsliste”. I sidehovedet kan man også se, hvor mange varer, der allerede er blevet tilføjet til listen.

7. Design af brugergrænsefladen

Hvis brugere opdager en fejl med en af opskrifterne, så er det muligt at rapportere fejl i systemet. På alle opskrifterne er der en trekantet advarsels-knap i nederste højre hjørne, som kan bruges til at rapportere en fejl vedr. en opskrift. Der popper en lille boks op, og baggrunden af siden bliver mørk for at fokusere koncentrationen på oprettelse af fejrapporten. Her kan man nu specificere, hvad fejlen handler om og evt. give en beskrivelse, inden man vælger at indsende fejlen. Beskrivelsesfeltet vises dynamisk i forhold til den valgte fejlkategori.

7.2.3 Indkøbsliste

Ud over muligheden for at tilføje opskrifternes ingredienser til indkøbslisten, så kan man også tilføje almindelig tekst til, så det er muligt at lave en indkøbsliste, der indeholder andet end ingredienser til madlavningen. Så man kan skrive andre varer på, som man kan købe med fra f.eks. supermarketdet. Systemets indkøbsliste kan ses i figur 7.10.

The screenshot shows the foodi website interface. At the top, there's a navigation bar with 'foodi' on the left, 'indkøbsliste (7)', 'favoritter (0)', and 'log ind / opret bruger' on the right. Below the navigation, a green banner says 'Reklamer!!!'. To the right, there's a black box with an upward arrow icon and the text: 'Husk at oprette en bruger for at gemme din indkøbsliste og favoritter.' In the center, there's a yellow shopping list card titled 'Indkøbsliste' with a 'Udskriv' and 'Slet alt' button. The list contains the following items:

Indkøbsliste
40 g perlebyg, perlespelt eller perlerug
1/4 tsk stødt allehånde
1/2 lille knust fed hvidløg
1/2 øeg
10 sorte oliven med sten
1/2 groftrevet, afdryppet agurk
87,5 g rygeost 10% (40+)

At the bottom of the list card, there's a text input field labeled 'Tilføj til indkøbsliste:' and a 'Tilføj' button. Below the list card, there are links for 'rapporter fejl', 'om foodi', 'kontakt foodi', and a small logo.

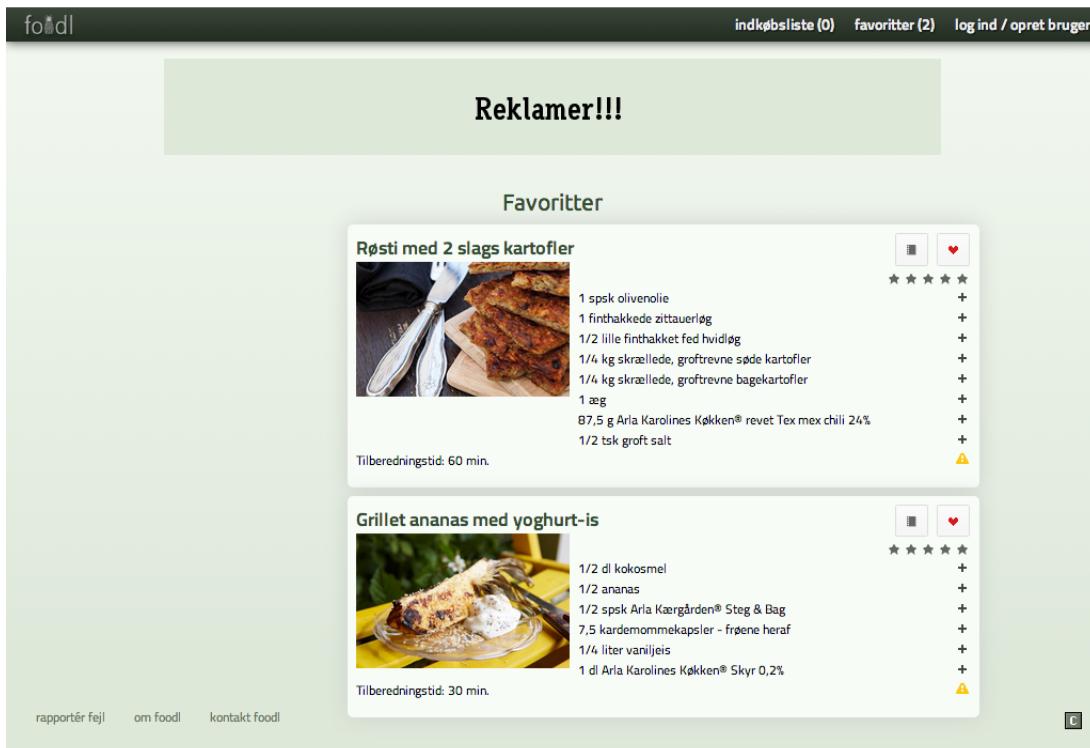
Figur 7.10: Denne figur har til formål at give et overblik over systemets indkøbsliste.

Brugeren har mulighed for at tilføje varer i feltet "tilføj til indkøbsliste" og trykke på "tilføj" i bunden af siden. Der er mulighed for at slette alle varer fra indkøbslisten, ved at trykke på knappen "slet alt" i øverste højre hjørne af indkøbslisten, og ligeledes at slette enkelte varer, ved at trykke på de små gule kryds'er ud for alle varerne. Derudover er der implementeret en knap, til at udskrive indkøbslisten, som vi naturligvis kalder for "udskriv".

Hvis brugeren ikke er logget ind, vil de se i øverste højre hjørne af figur 7.10 (under sidehovedet) en boks, som informerer brugeren om, at man skal være logget ind for at systemet skal være i stand til at gemme indkøbslisten og favoritter. Oprettelse af bruger og indlogging bliver beskrevet nærmere i afsnit 7.2.5.

7.2.4 Favoritliste

Når der bliver tilføjet en opskrift til en brugers favoritliste, så kan denne opskrift findes under "favoritter", som kan findes via sidehovedet i toppen af siden. Et eksempel af en kort opskriftsliste under favoritter kan ses på figur 7.11.



Figur 7.11: Denne figur har til formål at give et overblik over systemets favoritside.

En opskrift bliver tilføjet til favoritlisten ved at trykke på den hjerte-formede knap i øverste højre hjørne af en opskrift. Hvis en opskrift ikke er favoriseret, så er det hjerteformede område i knappen gråt. Når den bliver favoriseret, så bliver hjertet rødt, og dette kan ses i figur 7.11.

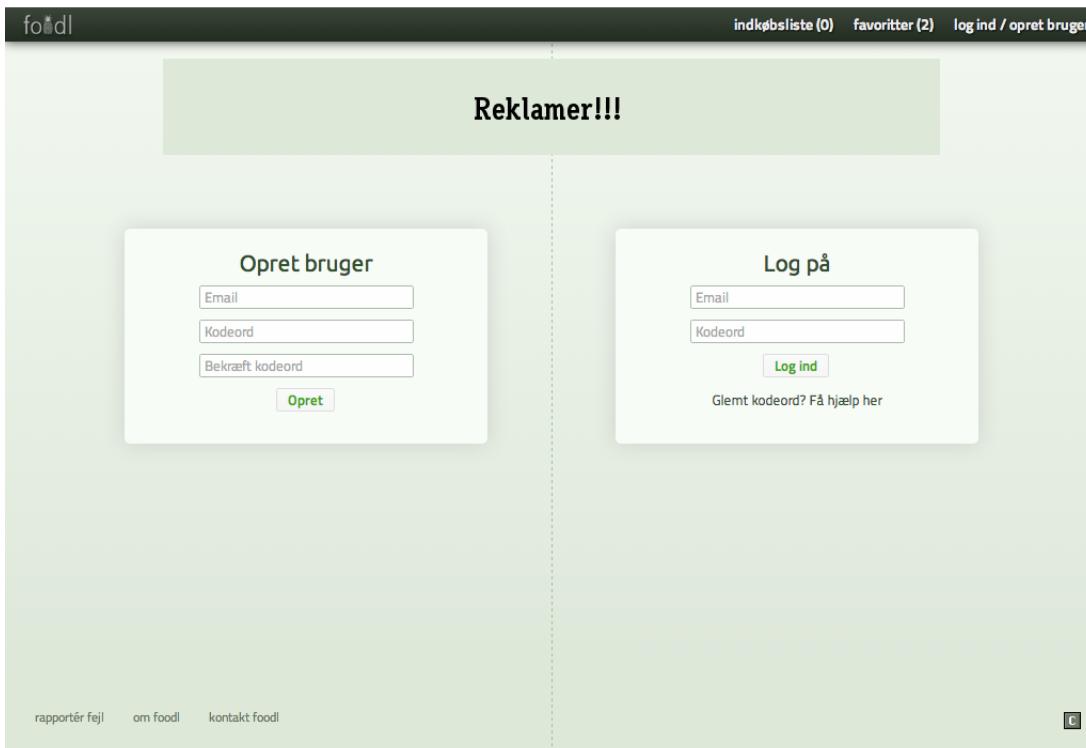
Idéen med favoritlisten er at give brugerne mulighed for at gemme opskrifter, de finder interessante og at de gerne vil bogmærke den til næste gang. De er meget nemmere at finde frem, når man kan finde dem under favoritlisten i stedet for at skulle udføre en ny søgning og prøve at finde den samme opskrift igen.

7.2.5 Brugeroprettelse

Alle har mulighed for at oprette en bruger på [foodl](#). Hvis man har en bruger og logger ind på systemet, så kan man kan gemme sin indkøbsliste og listen over favoritter. Det er dog ikke obligatorisk at have en bruger for at kunne bruge systemet, da vi ikke ønsker at binde brugerne til at oprette noget som helst. Man kan bruge hele systemet, om man har en bruger eller ej.

Man opretter en bruger på samme side, hvor man logger ind på systemet. Dette er præsenteret i figur 7.12. Man navigerer til denne side ved at klikke på "log ind / opret bruger" i højre side af sidehovedet, som også er synlig på samme figur.

7. Design af brugergrænsefladen



Figur 7.12: Denne figur har til formål at give et overblik over systemets brugeroprettelsesside.

Man skal bruge sin email og en adgangskode for at lave en bruger. Hvis man allerede har en bruger, så skal man blot logge ind med de rigtige oplysninger.

Når man er logget ind, så ændrer sidehovedet sig en smule. Figur 7.13 viser, at der nu er mulighed for at gå ind i en menu, der hedder "indstillinger" og at logge ud igen.



Figur 7.13: Det andrede sidehovede, når brugeren er logget ind.

Hvis man ønsker at skifte sin adgangskode, så sker det ved at trykke på knappen "indstillinger" i sidehovedet.

7.2.6 Generelt

Som de fleste andre hjemmesider, så har vi også en "om" og en "kontakt" side, som kan tilgås fra nederste venstre hjørne af enhver underside af foodl. Derudover kan man rapportere en generel fejl ved siden, hvis man støder på sådan en. Figur 7.14 viser disse tre knapper.



Figur 7.14: I nederste venstre hjørne af systemet kan man rapportere en generel fejl, læse mere om systemet og kontakte udviklerne.

Under "om foodl" kan brugeren kort læse om dette projekt og om udviklerne bag systemet, altså denne gruppe.

7.2.7 Designovervejelser

Under designet af foodl's brugergrænseflade har vi gjort os nogle designmæssige overvejelser for at øge kvaliteten af programmet. Vi har blandt andet taget højde for følgende principper for god usability[4, s. 90]:

Visibility

Vi har valgt, at de vigtigste funktioner altid skal være synlige for brugeren. Det omfatter links til at navigere til siderne *indkøbsliste*, *favoritter*, *log ind* / *opret bruger* og *forside*. De nævnte links er altid synlige allerøverst på siden i sidehovedet.

Consistency

Vi er konsistente med måden hvorpå knapper og lignende, hvorfra brugeren kan interagere med systemet, ser ud. Bl.a. bliver knapper grønne, når musen føres over dem. Knapper har den samme firkantede form med svagt afrundede hjørner, grålig baggrund og mørkere grå kanter. Har en knap tekst på sig, er teksten som udgangspunkt grøn. Hvis knappen er blevet trykket i bund og altså er aktiveret, er skriften orange. Hvis en knap er inaktiv, og man ikke kan trykke på den, så er kontrasten mellem knappens baggrund, tekst og kant meget lav. Dette kaldes normalt *grayed out*, og er måden man typisk viser, at en knap ikke kan trykkes på.

Familiarity

Vi benytter symboler, som folk i forvejen kender, som f.eks. krydset man trykker på for at slette en råvaretype, man har indtastet, eller når man vil slette en ingrediens fra indkøbslisten. Når man skal tilføje en ingrediens til indkøbslisten trykker man på et plus-symbol, som ofte forbindes med at tilføje noget.

Når en søgning er sat i gang, så vises en cirkulær animation, som ofte bruges, når en proces, der kan tage et stykke tid, er i gang.

Logoet linker til forsiden, hvilket er ret normalt for de fleste hjemmesider. I og med at det er normalt og velkendt af mange personer, så er det menneskers evne til at genkende ting, vi prøver at ramme.

En opskrifts vurdering kan læses som 5 stjerner, hvoraf nogle af stjernebladene bliver farvet fra venstre mod højre, ligesom folk kender det fra f.eks. avisers anmeldelser af film.

Affordance

Vi har designet foodl's logo, samt forsiden hvor logoet præsenteres, på en sådan måde at folk med det samme får et indtryk af, hvad systemet kan. Logoets tekst, *Foodl*, får tankerne over på mad, hvilket yderligere forstærkes af, at det ene "O" er udskiftet med en ananas. Navnet Foodl minder en smule om Google, og har til formål at lede folks tanker i retning af en søgermaskine, hvilket yderligere forstærkes af sidehovedet og forsiden's design, der minder en hel del om Googles søgermaskine.

Når man skal favorisere en opskrift, altså tilkendegive at man synes om denne, så skal man trykke på en knap med et hjertesymbol. Hjertet skal gerne give folk en ide om, at man altså skal trykke på knappen, hvis man synes om opskriften.

Indkøbslisten ligner et rigtig linieret A4-papir. Varer på listen vises med en skrifftype, der ligner håndskrift, dog ikke i så høj grad, at det bliver svært at læse teksten. Vi benytter altså en metafor her, ved at tage et kendt objekt og mappe det til noget virtuelt, der ser ud og fungerer på samme måde.

En stjerne bliver associeret med noget godt. Derfor viser vi et antal stjerner ved en opskrift, for at folk skal forbinde det med, hvor god en opskrift er.

Navigation

Vi har forsøgt at gøre navigation imellem foodl's sider mere enkel ved at have et sidehoved, der sørger for, at de overordnede navigeringsmuligheder imellem undersider altid er synlige.

Hvis man ikke er logget ind, kommer der en sort firkantet boks frem med teksten "Husk at oprette en bruger for at gemme din indkøbsliste og favoritter". I boksen er der en pil, der peger lige netop på det link i sidehovedet, man skal trykke på for at oprette en bruger.

7. Design af brugergrænsefladen

Når man har udført en søgning, får man allerførst vist alle resultater. Derefter kan man sortere og begrænse søgningen. Dette opfylder designprincippet *Overblik først, derefter detaljer*.

Control

Når musen er over en knap, der kan interageres med, bliver knappens baggrund grøn. Så ved man, at man kan trykke på knappen, og på samme måde finder man hurtigt ud af, at en slider kan trækkes i.

Når der trækkes i slideren, der skalerer opskrifterne i et søgeresultat, så ændres antal personer og ingredienserne mængder sig med det samme, og ikke først når slideren er sluppet. På den måde får brugeren bedre kontrol og kan hurtigt ramme den rigtige skalering.

Feedback

Systemet giver brugeren feedback, så man er klar over, at systemet har reageret på interaktionen. Brugeren bliver informeret, når noget er gået galt, eller hvis han har lavet en fejl.

En animation af en cirkulær bevægelse viser, at en søgning er i gang. På den måde tror brugeren ikke, at han har klikket ved siden af en knap, hvis søgning blot tager lidt længere tid end forventet.

En besked popper op på siden, når brugeren skal have feedback. Det kan f.eks. være, når brugeren har oprettet en bruger, eller har indtastet en forkert kode i et forsøg på at logge ind.

Revocery

Hvis en bruger har glemt sin kode, kan man få den tilsendt igen.

Style

Vi har også gjort os nogle overvejelser, vedrørende hvilke farver vi benytter i systemet. Igennem hele systemet har vi benyttet nogle få farver, hvilket stemmer overens med regler for godt design af grænseflader [4, s. 344]. Vi har forsøgt at følge vestlige farvekonventioner.

Den lidt kølige og grønne baggrundsfarve, som kan ses igennem hele systemet, fungerer godt som en rolig baggrundsfarve, som også er indbydende for brugeren. I følge farvekonventionen så er den kølige farve god til baggrundsinformation. Den grønne farve er beskrevet som en tryg og klar farve [4, s. 344]. Den bliver ikke signaleret noget hastende. Dvs., at brugeren skal tage sig tid til at kigge lidt på siden, og se hvilke funktioner der er på forsiden inden der bliver foretaget nogle søgninger.

Vi har brugt nogle mere advarende farver til f.eks. fejlmeddelelser eller knapper, der skal rapportere en fejl i systemet. Fejlmeddelelser bliver præsenteret i en rød boks, som, ifølge de vestlige farvekonventioner, signalerer farer, med hvid tekst, som signalerer neutralitet. Vi vurderede, at disse to farver var en god kombination, fordi den hvide farve neutraliserer den farlige røde en smule. Derudover har vi en rapporteringsknap, som er farvet gul, som signalerer forsigtighed. Man skal ikke bare klikke på denne knap hele tiden. Den skal bruges til nogle alvorlige rapporteringer.

Conviviality

Hvis en bruger laver en fejl, så forholder vi os relativt roligt og viser brugeren en lille tekstmeldelse med feedback. Vi afspiller ikke en lyd, der får folk til at spilde kaffen.

Når noget er gået galt, giver vi ikke brugeren skylden for dette direkte. Hvis en person forsøger at logge ind uden at have indtastet gyldig email-adresse, så skriver vi blot *Angiv en email-adresse*, i stedet for *Du har indtastet en forkert email-adresse*. På samme måde med alle fejlmeddelelser, bebrejder vi ikke brugeren direkte.

8 Implementering

Dette kapitel har til formål at give læseren et indblik i hvilke metoder og teknologier, der er blevet brugt til udviklingen af **foodl**. Teknologierne, som indbefatter bl.a. Ruby on Rails, JavaScript og CSS mm., vil vi ikke gå i dybden med, men blot introducere kort. Derudover vil essentielle algoritmer, samt tankerne bag disse, blive forklaret, og webapplikationen vil blive præsenteret, for at give et overblik over, hvordan **foodl** fungerer.

8.1 Teknologier

Systemet er en webapplikation, hvor der er lagt vægt på enkelt og forståeligt design, effektivitet, fleksibilitet og brugbarhed. For at kunne opfylde alle disse krav og kriterier, som også er beskrevet i afsnit 6.1, har vi gjort brug af nogle forskellige webudviklings- og programmeringsteknologier.

Vi har brugt følgende teknologier:

- HTML og CSS
- MySQL
- JavaScript, jQuery, jQuery UI, AJAX og JSON
- Ruby on Rails

Vi har brugt HTML[42] til at designe brugergrænsefladen af hjemmesiden med. HTML fungerer godt til opmærkning og strukturering af hjemmesider, men ren HTML er ikke særlig flot at se på. For at give en pådere repræsentation af indholdet af hjemmesiden har vi brugt CSS[41]. CSS er et sprog, der bruges til at beskrive, hvordan man ønsker indholdet af bl.a. et HTML-dokument skal præsenteres i f.eks. en webbrowser.

Hvad angår databaser, så har vi brugt MySQL[48], der er en flertrådet SQL-databaseserver, som understøtter flere samtidige brugere.

For at gøre hjemmesiden dynamisk og brugervenlig, bruger vi JavaScript[44], som er et objektorienteret scriptsprog, som de fleste moderne webbrowsere forstår. Derudover har vi også brugt jQuery[30] sammen med JavaScript. jQuery er et programbibliotek, der bygger oven på JavaScript. Det forøger kraftigt funktionalitet af JavaScript og samtidig formindsker antallet af linjer kode, da jQuery-objekter indeholder mange indbyggede metoder. AJAX[40] og JSON[46] bliver yderligere brugt til at gøre webapplikationen asyntkont. Vi udnytter forskellige widgets og animationer inkluderet i jQuery UI[45] med et eksisterende tema. jQuery UI bygger oven på jQuery og tilbyder yderligere funktionalitet i forbindelse med designet af webapplikationer. Figur 8.1 illustrerer tre forskellige jQuery UI “widgets”, som brugeren kommer i kontakt med, når der bliver udført en søgning på **foodl**-hjemmesiden. På denne figur er der forskel på de to knappesamlinger (venstre og højre side). Knapperne til venstre fungerer som afkrydsningsbokse, hvilket vil sige, at man kan markere flere af gangen, hvorimod knapperne til højre er radioknapper, hvilket vil sige, at man kun kan markere én af gangen.

AJAX gør det muligt at udveksle data mellem hjemmeside og server uden, at siden skal gennemgå en fuld sideopdatering hver gang, der sker en dataoverførsel, for at vise det nye indhold. Hovedsageligt sker alle dataoverførsler i baggrunden og brugeren præsenteres for disse ændringer med det samme. F.eks. bliver søgeresultatsiden opdateret dynamisk, når man ændrer i sorteringen. AJAX står for “Asynchronous JavaScript and XML”, og navnet antyder, at XML bruges til udvekslingen af oplysninger fra serveren til klienten (webbrowseren). Nu om dage anvendes JSON dog som regel som erstatning for XML i forbindelse med AJAX. JSON er et simpelt dataudvekslingsformat, der

8. Implementering

gør det nemt at udveksle forskellige datastrukturer. Det bygger på JavaScript-syntax, og understøtter de simple JavaScript typer, såsom tal, tekststrenge, arrays og objekter. Det gør det ideelt til f.eks. at sende et objekt eller et array af objekter fra serveren til klienten.



Figur 8.1: Figuren viser tre forskellige eksempler på widgets, der er lavet vha. jQuery UI. Til venstre og højre ses en samling af tre knapper. Midt for ses en justerbar slider.

Derudover har vi anvendt Ruby on Rails [49], der er et webudviklingsframework bygget oven på programmerings-sproget Ruby, til at lave webapplikationer. Intentionerne bag Rails var at skabe et framework, der gjorde det lettere og hurtigere for webudviklere at udvikle webapplikationer, hvilket også er en af de væsentlige årsager til, at vi vælger at anvende Rails.

Målet med Rails blev bl.a. opnået vha. af følgende filosofi:

“konventioner over konfigurationer”

Filosofien har både sine fordele og ulemper. Det er en ulempe, at webudvikleren skal have arbejdet meget med Rails i forvejen, for at kunne huske de mange konventionerne og kommandoer, eller skal bruge en masse tid på at slå dem op, når de skal bruges. Det er på den anden siden en fordel, at webudviklere, vha. Rails’ konventioner, kan lave applikationer, som kan udrette meget, ud fra få linjers kode. Det gør det også lettere at ligge på andre, eksisterende Rails-applikationer uden, at man bliver nødt til at sætte sig ind i et hav af konfigurationsfiler. Rails viser allerede fra første møde, at der skal meget lidt arbejde til, for at få et stort afkast.

```
darx@darkstar /home/darx/Workspace % rails new food
      create README.rdoc
      create Rakefile
      create config.ru
      create .gitignore
      create Gemfile
      create app
      create app/assets/images/rails.png
      create app/assets/javascripts/application.js
      create app/assets/stylesheets/application.css
      create app/controllers/application_controller.rb
      create app/helpers/application_helper.rb
      create app/mailers
      create app/models
```

Figur 8.2: Rails-kommando, der indtastes i kommandoprompten, hvorefter Rails genererer en mappe med en fuldt fungerende webapplikation, kaldet ‘Food’

Enhver Railsapplikation tager udgangspunkt i arkitekturen Model-View-Controller (MVC). Mappen med webapplikationen består af en lang række undermapper, hvori “models”, “views” og “controllers” bl.a. befinner sig. Kort sagt genererer Rails, vha. **new**-kommandoen, hele skelettet for webapplikationen, og derefter er det “bare” at fylde det indhold, man ønsker i sin webapplikation, i de rigtige mapper. Model-View-Controllerarkitektur i detaljer i afsnit 6.2.2.

8.2 Funktionalitet

Da **food** er en Rails-applikation består den, som andre MVC-applikationer, hovedsageligt af modeller, controllers og views. Modeller er klasser, der repræsenterer data, dvs. f.eks. tabeller i databasen. Controllers er klasser bestående af actions, eller handlinger, som er metoder, der har til formål at reagere på en aktørs interaktion med serveren. Den sidste grundlæggende del af MVC-designet er views, som er template-filer, som bliver udfyldt med data gennem controlleren og sendt tilbage til aktøren. I **food** er der flest HTML-templates, som har til formål at præsentere data i brugerens webbrowser.

Derudover består en Rails-applikation også af helpers og assets. Helpers er hjælpefunktioner, som skal være tilgængelige flere steder i applikationen, dvs. i forskellige controllers og views. Assets er de JavaScripts, stylesheets og billeder, som applikationen bruger på klientsiden.

8.2.1 Controllers

Det centrale i implementeringen af controllerkomponenten i vores webapplikation er dets controllers. Controller-komponenten styrer, hvorpå man i en MVC-applikation fordeler ansvaret for de forskellige modelklasser mellem flere controller-klasser. Som hovedregel laver man en controller for hver modelklasse, med det formål at styre klients interaktion med den pågældende modelklasse. Webapplikationen indeholder følgende controllers:

ApplicationController

Den overordnede applikationscontroller, som alle andre controllers i **foodl** nedarver fra. Den kan f.eks. indeholde forskellige helper-metoder, som skal kunne tilgås overalt i applikationen.

FavoritesController

Denne controller håndterer brugerens favoritter, hvad enten denne er logget ind eller ej. Controlleren har tre actions, en til at liste favoritter, en til at tilføje favoritter og en til at fjerne favoritter.

HomeController

Denne controller håndterer applikationens to statiske sider, ”Om foodl” og ”Kontakt foodl”.

IssuesController

Fejlhåndteringen foregår med denne controller. Dvs. oprettelsen af de fejl, der rapporteres og listen af fejl.

RecipesController

Kun én action er indeholdt i denne controller, en action til at servere det billede, der hører til en opskrift.

SearchController

Søgning håndteres i denne controller. Søgningen er beskrevet i detaljer i afsnit 8.2.4.

SessionsController

Denne controller håndterer brugersessions, dvs. når en bruger vil logge ind eller ud.

ShoppingListController

Indkøbslisten håndteres med denne controller. Tilføjelsen af elementer, hvad enten det er ingredienser, råtekststrenge eller hele opskrifter. Sletning af elementer muliggøres også af denne controller.

UsersController

Denne controller håndterer alt i forbindelse med brugere, dvs. f.eks. opretning af bruger, opdatering af kodeord og funktionen ”Glemt kodeord”.

8.2.2 Modeller

Enhver af de nedenstående modeller er repræsenteret i klassediagrammet, beskrevet i afsnit 6.2.4. Disse modeller har en direkte kobling til klassediagrammet og skal ses som den endelige implementering af designfasen. Herunder er hver model beskrevet med, hvad de indeholder, og hvad deres funktion består af.

User er en af de eneste modelrepræsentationer, der benytter sig af Active Record Callbacks for at sørge for, at der f.eks. ikke kan oprettes to brugere med samme emailadresse, og at emailen er ”gyldig” (f.eks. at den skal indeholde et ”@”-tegn). Dette gør vi, da vi ikke mener, at andre modeller er i direkte kontakt med brugeren på samme måde. De andre klasser bliver hovedsagligt håndteret af systemet, og derfor behøver de f.eks. ikke gyldighedskontrolleres ligesom **User**. Hvis systemet skulle bruges i produktion og håndteres af flere mennesker, ville det være åbenlyst at tilføje mere kontrol på modellerne.

FoodType

Denne model repræsenterer “råvaretype”-klassen i klassediagrammet. **FoodType** er en liste af råvarer, som hver ingrediens i alle af opskrifterne svarer til. Meningen er, at hver ingrediens i alle opskrifter, benytter sig af én **FoodType**. Grunden til dette er, at nogle opskriftshjemmesider benytter varemærker i ingrediensnavne eller bruger forskellige ord for den samme type råvare, f.eks. havsalt og salt eller schalotteløg og skalotteløg. Dette kan skabe forvirring og gør søgning meget besværligt, hvis der ikke blev søgt på “bagekartofler” når man indtaster “kartofler” i søgningen. De første råvarer er taget fra en liste af ingredienser[27], hvorefter de manuelt er blevet indtastet, hvis en ingrediens fra en opskriftshjemmeside ikke eksisterer i forvejen. De forskellige instanser af **FoodType** bliver f.eks. vist som søgeforslag, når man foretager en søgning på forsiden. Alle råvarer er skrevet i flertal for at få konsistens; og vi vurderer, at det er yderst sjældent, at man har lyst til f.eks. at bruge én enkelt gulerod som ingrediens i sin rester.

Ingredient

Ingrediensmodellen indgår udelukkende i opskrifter og indeholder navn, mængde og enhed. Hver **Ingredient** har en tilsvarende **FoodType**. F.eks. svarer “ostindisk karry” til “karry”. Navnfeltet er taget fra opskriftshjemmesiderne, da de typisk er mere beskrivende end blot en **FoodType**. Det gør, at navnfeltet ikke har nogen betydning for søgningen, da der ikke bliver søgt på ingrediensnavnet. Enheden for hver **Ingredient** er beskrevet vha. SI-systemet og er også taget fra ingredienslisten på opskriftshjemmesiden (Arla). Mængdefeltet beskriver mængden af ingrediensen skaleret ned til en enkelt person. Det muliggør en nemmere skaleringsfunktion på **foodl**, så man ikke behøver, at navigere videre til selve opskritsiden, hvis man allerede er klar over, hvor mange personer, der skal laves mad til.

Recipe

Opskriftsmodellen indeholder al information om opskrifter: navn, billede, vurdering, url, tilberedningstid, samt flere instanser af klassen **Ingredient**. Billedet er taget direkte fra opskriftshjemmesiden og er krævet, da vi i forhold til informanternes ønsker, ikke interesserer os for opskriftshjemmesider uden et billede af opskriften. I forhold til favoritter, har opskrifter også mulighed for at tilhøre brugere, men dette er repræsenteret af en anden tabel i databasen.

IssueCategory

Fejltypemodellen beskriver de forskellige typer fejl, som brugeren kan rapportere til administratorerne. Modellen består af et navnfelt og en sandhedsværdi. Navnet på en fejkategori kunne f.eks. være “dødt link” eller “stavefejl”. Fejtyperne kan derudover være beskrivelige eller ubeskrivelige, da der nødvendigvis ikke behøves en forklaring af brugeren på f.eks. et dødt link, hvorimod andre mere generelle fejtyper kræver en form for beskrivelse, før de kan rapporteres.

Issue

En fejl tilslutter sig en **IssueCategory** for, at administratorerne på siden har mulighed for at sortere og filtrere listen af fejl i forbindelse med deres kategori. Derudover indeholder **Issue** en bruger og/eller en beskrivelse, alt afhængigt af hvilken fejkategori den indgår i. Dvs., at hvis brugeren er logget ind under oprettelse af en fejlrappport, så kan administratorerne tilgå brugerens mailadresse og eventuelle oplysninger, hvis de ønsker at kontakte brugeren for yderligere oplysninger eller lignende.

ListItem

Når ingredienser fra en opskrift eller en tekststreng bliver tilføjet til indkøbslisten, så bliver de repræsenteret som et **ListItem**. Dette gøres for at undgå, at indholdet i indkøbslisten ikke ændres, hvis der bliver fjernet eller ændret i en ingrediens og vice versa. Et **ListItem** har alle egenskaber som en **Ingredient** har, hvor den derudover kræver en **User** for at koble brugeren med de varer, der er tilføjet til indkøbslisten.

User

Brugermodenlen repræsenterer de oprettede brugere på **foodl**. Brugere skal oprette sig med en emailadresse samt kodeord. Email er den unikke måde, hvorpå brugeren bliver identificeret. Der kan altså på intet tidspunkt være oprettet to brugere med samme emailadresse. Kodeord bliver gemt i databasen vha. et Ruby-bibliotek,

der hedder BCrypt, der hasher kodeordet i stedet for at gemme det i plain-text. Brugere har i øvrigt mange **ListItems**, **Issues** og har også en tilslutning til favorit-tabellen, som er beskrevet i **Recipe**-modellen.

8.2.3 Tabeller

Hver model svarer til en tabel i databasen. Vores tabelstruktur kan ses i figur 8.3. Hver kasse repræsenterer en tabel i databasen, og hver pil repræsenterer en relation mellem to tabeller. Pilen vender således, at den peger væk fra den tabel, som har en fremmed nøgle. Det ses f.eks. mellem **ingredients** og **food_types**. Her indeholder tabellen **ingredients** en kolonne **food_type_id**, der er en fremmed nøgle, som bruges til at pege på en række i **food_types**-tabellen, derfor pilen fra **ingredients** til **food_types**.

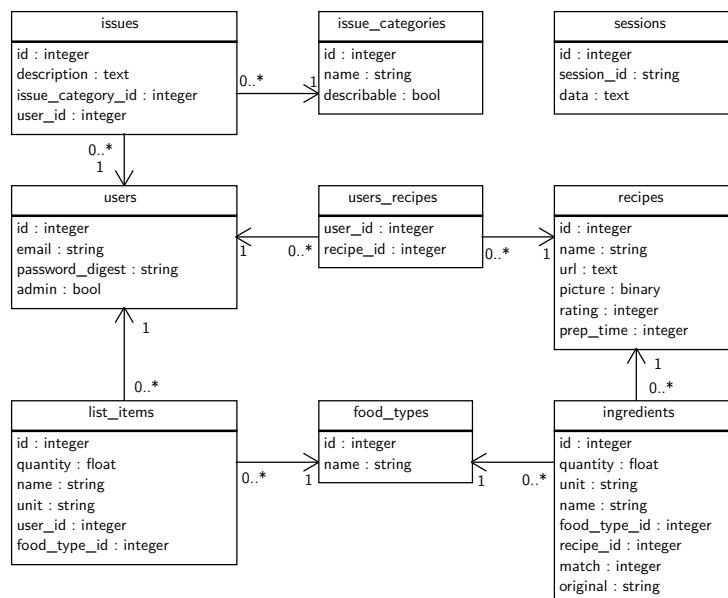
sessions-tabellen bruges af en intern Session-model i Rails, som er nødvendig for at gemme sessionsdata på serveren, dvs. f.eks. den besøgandes favoritter og indkøbsliste, når denne ikke er logget ind.

Ydermere er der tabellen **users_recipes**, som er nødvendig for at udtrykke mange-til-mange-forholdet mellem brugere og opskrifter. Dvs., at hver relation mellem en bruger og en opskrift (en favorisering) svarer til en række i denne tabel, som indeholder både et id for en opskrift og et id for en bruger. Når tabellen først er oprettet, sørger Rails selv for at abstrahere denne relation, så længe en **has_and_belongs_to_many**-relation, der peger på **User**-modellen, er sat op i **Opskrift**-modellen og vice versa. Dette medvirker til, at man f.eks. nemt kan tilføje en opskrift til en brugers favoritter vha. en enkelt linje Ruby-kode, som er illustreret i eksempel 8.1.

Eksempel 8.1: Hvis man har et **User**-objekt i **user** (som f.eks. returneret med `User.find_by_id(42)`) og et **Recipe**-objekt i **recipe**, kan opskriften associeres med brugeren med denne linje Ruby-kode.

```
1 user.favorites << recipe
```

Som illustreret i figur 8.3, så er der udeover mange-til-mange-forholdene også mere simple relationer. Det ses blandt andet mellem opskrifter og ingredienser, og mellem ingredienser og råvaretyper. Dette betyder, at ser man på **Ingredient**-modellen, så er der sat to **belongs_to**-relationer op, som peger på henholdsvis **Recipe** og **FoodType**. Ligeledes er der i både **Recipe**- og **FoodType**-modellerne sat en **has_many**-relation op, som peger på **Ingredient**.



Figur 8.3: Databasetabelstrukturen og associeringerne mellem tabellerne.

8.2.4 Søgning

Søgningen er den vigtigste del af **foodl**. Som beskrevet i afsnit 3.3, så er målet med projektet netop at kunne søge blandt mange opskrifter og returnere de opskrifter, som indeholder de råvarer, som brugeren har valgt.

Alt dette styres af klassen **SearchController**, som består af tre actions. Ligeledes er aktiviteten spredt ud over to views; **index**, som er selve forsiden af **foodl** med mulighed for at indtaste råvarer, og **result**, som er resultatsiden, hvor de relevante opskrifter bliver præsenteret.

Den tredje action i controlleren er **autocomplete_food_types**, hvortil der ikke er knyttet et view. Denne action har til formål at returnere en liste over råvarer baseret på brugerens indtastning. Denne funktionalitet er beskrevet i detaljer i nedenstående afsnit.

Søgeforslag

En central del af forsiden og brugerens mulighed for at vælge råvarer er, at systemet kommer med en liste over forslag, som er baseret på brugerens indtastning.

Det eneste denne action gør er at generere en SQL-forespørgsel, som sendes til MySQL-databasen og returnerer et array bestående af fem råvareforslag. Dette array konverteres til JSON-format og returneres. Dermed kan man via JavaScript tilgå denne action med en AJAX-forespørgsel, mens brugeren indtaster, og næsten med det samme fremvises resultatet.

Et eksempel på en genereret SQL-forespørgsel kan ses i eksempel 8.2. SQL-forespørgslen er en **SELECT**-forespørgsel, hvilket vil sige, at vi ønsker at læse fra en tabel i databasen. I første linje vælges kolonnen **name**, således at kun indholdet af denne kolonne returneres. I linje 2 vælges hvilken tabel forespørgslen skal udføres på, i dette tilfælde **food_types**-tabellen, altså tabellen med alle råvarer. I linje 3 sættes en betingelse for hvilke rækker fra tabellen, der skal returneres, nemlig alle dem, hvor råvarenavnet indeholder ordet "pølse". "%" -tegnet er wildcard-karakter i SQL, og matcher nul eller flere karakterer. Dvs. "%pølse%" matcher både "pølser", "pølsebrød" og "wienerpølse". I fjerde linje påbegyndes sorteringsudtrykket. Der sorteres på baggrund af tre parametre. Først og fremmest sorteres resultatet efter en **CASE**, hvor rækvens navnefelt testes for lighed med det indtastede. Er dette tilfældet er udtrykket lig 1, ellers er det 0. Da der sorteres i faldende rækkefølge (**DESC**), vil den række (hvis den eksisterer), hvor navnet er lig med det indtastede, altså komme først. Den næste sorteringsparameter, har en lignende **CASE**, dog med en wildcard-karakter i enden. Dermed vil alle rækker, hvor navnet begynder med det indtastede, komme før de rækker, hvor det indtastede står inde midt i råvarenavnet. Den sidste sorteringsparameter, sorterer efter længden af råvarenavnet i stigende rækkefølge. Dette gøres for, at de mest relevante foreslag kommer øverst. Den sidste linje i forespørgslen (**LIMIT**), begrænser mængden af resultater til fem.

Eksempel 8.2: Hvis en bruger indtaster "pølse" udføres denne SQL-forespørgsel.

```
1 SELECT name
2 FROM food_types
3 WHERE name LIKE "%pølse%"
4 ORDER BY CASE
5     WHEN name LIKE "pølse" THEN 1
6     ELSE 0
7 END DESC,
8 CASE
9     WHEN name LIKE "pølse%" THEN 1
10    ELSE 0
11 END DESC,
12 LENGTH(name) ASC
13 LIMIT 5
```

Resultatet af SQL-forespørgslen i eksempel 8.2 er derfor en liste med 5 forslag til råvarer. Det returnerede JSON-array for “pølse” kan ses i eksempel 8.3.

Eksempel 8.3: Et returneret JSON-array for “pølse”.

```
1 [ "Pølser", "Pølsebrød", "Spegepølse", "Wienerpølse", "Røde pølser" ]
```

De valgte råvarer tilføjes med JavaScript til en liste, som sendes videre til **result**-siden, når brugeren klikker “Søg”.

Søgeresultat

Resultatsiden håndteres af en action ved navn **result**, som foretager søgningen på opskrifterne i databasen.

Den primære parameter er en liste af råvaretyper, som er en tekststreg separeret med “|”-tegn. Det første, der bliver gjort med denne streng, er, at den bliver splittet op i et array bestående af råvarenavne. Derefter genereres en SQL-forespørgsel, som henter id’erne for de valgte råvaretyper fra **food_types**-tabellen.

For søgestrenge “**Pølser|Wasabi|Peber|**” returneres derfor array’et [1116, 1113, 2]. Disse id’er bruges til at generere endnu en SQL-forespørgsel, med det formål at finde opskrifter, som indeholder de valgte råvaretyper. Et eksempel på dette kan ses i eksempel 8.4.

Eksempel 8.4: Ved en søgning på “Pølser”, “Wasabi” og “Peber” udføres denne SQL-forespørgsel.

```
1 SELECT recipes.*, COUNT(*) AS relevance
2 FROM ingredients
3 RIGHT JOIN recipes ON recipe_id = recipes.id
4 WHERE food_type_id IN ( 1116, 1113, 2 )
5 GROUP BY recipes.id
6 ORDER BY relevance DESC
7 LIMIT 0, 50
```

Resultatet grupperes efter opskrifternes id, således at man kan tælle, hvor mange af de valgte råvarer opskriften indeholder. Denne værdi gemmes i variablen **relevance**, og bliver (som standard) brugt til at sortere resultatet med. Den bliver ligeledes sendt videre til view’et sammen med hver opskrift, således at det fremgår af resultaterne, hvor mange af de valgte ingredienser, hver opskrift indeholder.

Opskrifterne returneres på to måder alt efter, hvordan siden forespørges. Ved en almindelig browser-forespørgsel returneres hele resultatsiden som forventet. Men foretages forespørgslen via AJAX, så returneres kun listen med opskrifter. Dette gør, at resultatsiden dynamisk kan hente en ny opskriftsliste fra serveren, hvis f.eks. søgeparametrene ændres.

8.3 Dataudtrækning

Dette afsnit beskriver, hvordan vi har udtrukket data til brug i det endelige system. Systemet består af opskrifter, som vi har fået lov til at benytte af Arla, hvilket bliver beskrevet i afsnit 8.3.1. Systemet lader en bruger søge på råvaretyper, hvilket bliver forklaret i afsnit 8.3.2. Derudover beskrives hvilke dele af opskrifterne, der bliver udtrukket af Dataudtrækkeren, som gennemgår sidens opskrifter. Til slut beskriver vi nogle forskellige metoder, hvorpå vi har oversat ingredienser til råvaretyper, som en bruger kan søge på.

8.3.1 Samarbejde med Arla

Når en bruger har indtastet en mængde råvaretyper og trykker søg, så skal søgeresultatet vise opskrifter, der indeholder de indtastede råvarer. For at gøre dette muligt, er det nødvendigt, at **foodl** indeholder data om en masse forskellige opskrifter. Da der findes mange opskriftshjemmesider på internettet, ville det være oplagt at benytte disse. At benytte data fra en specifik opskriftshjemmeside kræver, at vi har ejerens tilladelse til det. I dette projekt har vi af Arla fået lov til at benytte deres opskrifter, der kan ses på Arla.dk, forudsat, at vi overholder disse krav:

- Al den information, som vi ønsker at vise brugerne af **foodl**, skal gemmes lokalt. Det vil sige, at vi ikke må loade billeder, ingredienser og lignende fra deres server, hver gang en bruger benytter **foodl**.
- Vi må ikke vise **foodl**'s brugeres opskrifternes fremgangsmåde. **foodl** skal linke videre til Arla.dk, når folk vil se, hvordan en opskrift laves.

Under udviklingen af **foodl** har vi derfor taget højde for disse krav.

8.3.2 Råvaretypetabellen

Efter afprøvning af prototyperne 2A og 2B på informanterne blev vi klar over, at de råvaretyper, som en bruger ønsker at søge på, skal autocompletes. Det vil sige, at når en bruger skriver teksten *ba* i søgerfeltet, så skal **foodl** foreslå råvaretyper som f.eks. *banan* og *balsamico* osv. En autocomplete-funktion skal have noget data at stille forslag fra, og i dette tilfælde har vi brug for en tabel med alle de forskellige råvaretyper, man kan forestille sig, at en bruger vil indtaste. Denne tabel over råvaretyper vil fremover blive kaldt råvaretypetabellen.

Da opskrifterne, der søges på, kommer fra Arla, ville det have været nemt hvis Arla havde offentliggjort en liste over de forskellige ingredienser de benytter, således vi kunne bruge denne data som vores råvaretypetabel. En sådan liste var ikke tilgængelig, så derfor overvejede vi muligheden i at lave råvaretypetabellen, ved at udtrække ingredienserne direkte fra Arlas opskrifter. Et eksempel på ingredienserne navne i Arlas opskrifter:

- 375 g rød peberfrugt i strimler
- 3 røde peberfrugter (ca. 600 g)
- 4 røde peberfrugter i store tern (ca. 500 g)

En bruger, der indtaster teksten *pe* i forsøget på at indtaste *rød peberfrugt*, skal kun præsenteres for forskellige råvaretyper, og altså ikke samme råvaretype i forskellige mængder og former (skiver, strimler, hakket, m.m.). Det vil sige, at de mange forskellige ingredienser, der alle består af rød peberfrugt, kun skal medføre, at *rød peberfrugt* findes én gang i råvaretypetabellen. Vi vurderede, at Arlas navngivning af ingredienser har været for forskellig til at blive brugt som kilde til vores råvaretypetabel. Råvaretypetabellen er i stedet blevet lavet ud fra en liste af råvarer offentliggjort af hjemmesiden madopskrifter.nu[27]. Et udpluk af listen ser således ud:

- Rød peberfrugt
- Persille
- Citron

Listen fra madopskrifter.nu blev brugt som vores råvaretypetabel af følgende grunde:

- Listen indeholdt kun 4 dubletter (som vi har fjernet)
- Selve råvaretypen indeholder kun den rå råvaretype, ingen mængder eller andre betegnelser (vaskede, skrællede, m.m.)
- Listen omfattede 933 råvarer, hvilket vi anså som tilstrækkeligt

Med en liste over råvaretyper, bliver det muligt for en bruger at indtaste en række råvaretyper. Udover dette har vi også brug for data, der fortæller noget omkring de opskrifter brugeren kan søge på. Denne data udtrækker vi fra Arlas hjemmeside, selvfølgelig med deres accept.

8.3.3 Udtrækning af links til opskrifter

For at kunne gemme informationer om en opskrifts forskellige bestanddele, såsom navn, tilberedningstid, m.m., var det nødvendigt at udtrække denne data fra Arlas hjemmeside. Dette begreb kalder vi for dataudtrækning. Formålet var at indsætte data i databasen, så brugere har mulighed for at få præsenteret foreslag til opskrifter. Da dataudtrækningen kun havde til formål at indsætte indhold i databasen, var der ikke brug for views og controller, hvorfor vi valgte at benytte Ruby **uden** Rails frameworket til dette.

Ved at kigge på Arlas forside fandt vi et link, der hed *vis alle opskrifter*. Ved at følge linket kom vi ind på en ny side, hvor vi fik vist 16 opskrifter af gangen og kunne bladre fra side 1 til og med side 57. I alt havde vi altså adgang til $57 \times 16 = 912$ opskrifter. Når vi bladrede fra side 1 til side 2, kunne vi se browserens url ændre en forespørgsel på følgende måde:

```
?...&paging=1&... til ?...&paging=2&...
```

En variabel ved navn *paging* ændrer i queryen værdi fra 1 til 2. Siderne fra 1 til 57, der hver viser links til 16 opskrifter, er derfor lette at finde ved blot at lade variablen *paging* i queryen gå fra 1 og øges indtil ingen opskrifter fremkommer.

I stedet for at gøre dette manuelt for 57 opskrifter, benyttede vi Ruby-modulet **open-uri**. Modulet gør det muligt at hente indholdet i et open-uri-objekt af en hjemmeside ved at benytte funktionen **open(string url)**. I Ruby kunne vi så lade *paging*-variablen i url'en gå fra 1 til og med 57, og på den måde åbne alle 57 opskrifts-indekseringssider. De 16 links, en indekseringsside indeholdt, så i HTML således ud:

```
<h2><a href=''/opskrifter/Suppe'>Suppe</a></h2>
```

Efter at have åbnet en indekseringsside med metoden **open(url)**, var det nu nødvendigt med en metode til at finde de 16 opskriftlinks på hver side. Problemet var bare, at siden ikke kun indeholdt de 16 opskriftlinks, men også links til nyheder, kontaktinformation og lignende. For at kunne filtrere i siden, så vi stod tilbage med de 16 ønskede opskriftlinks, benyttede vi Ruby-modulet **Nokogiri**[1], der er en parser til HTML. Med **Nokogiri** var det let at filtrere en side efter bestemte tags og klasser. I vores tilfælde, havde vi brug for alle *<a>* tags inde i *<h2>* tags, som det kan ses af linkeksemplet ovenfor. Dette blev gjort ved at benytte en af **Nokogiri**'s mange metoder:

```
xpath('//h2//a').map |link| "http://arla.dk"+link['href']
```

Kaldet til **xpath** fandt alle *<a>* tags inde i *<h2>* tags og returnerede et array af hashtabeller. En hashtable er et array af nøgler med tilhørende værdier. Hver hashtable havde i dette tilfælde kun én nøgle, *href*, og værdien af nøglen var url'en til den pågældende opskrift. For at udtrække url'erne brugte vi Rubys enumerator **map**. Enumeratoren **map**, itererer over alle elementer i et array og returnerer et nyt array, hvor en eller flere operationer er blevet udført på alle elementer[32]. Et eksempel er **[1, 2, 3].map{|x| x+10}**, der returnerer arrayet **[11, 12, 13]**. Da vi benyttede **.map{|link| "http://arla.dk"+link['href']}**, itererede vi over alle elementer i arrayet, og for hvert element (hashtable), udtræk vi værdien af *href*-nøglen, der altså svarede til url'en fra HTML-tagget *Suppe*. På samme tid blev præfixet **http://arla.dk** også tilføjet foran. Resultatet var altså et array af links til de 16 opskrifter på indekseringssiden.

En forudsætning, der skulle være tilstede, for at kunne benytte denne metode, var at samme syntaks, i dette tilfælde *<h2><a>*, ikke blev benyttet til andet end de links, vi var interesserede i. Dette var heldigvis tilfældet.

8.3.4 Udtræk af opskrifter

Med en mulighed for at finde alle opskrifters unikke url begyndte vi at undersøge opbygningen af siden, der viser en enkelt opskrift. På samme måde kunne Nokogiri også bruges her til at finde informationer omkring tilberedningstid, opskriftens navn, billede af opskriften og portionsstørrelse. Alle disse ting kunne indsættes som en ny række i databasens tabel ‘recipes’. Opszriftens navn var særlig nem at dataudtrække, da det var sidens titel, og det kunne derfor tages fra HTML-tagget `<title>`. Der var dog tilføjet lidt mere information, adskilt af karakteren “|”. Et eksempel: *Opszrifts navn | Kløver® | Produkter |*. Ved at bruge Ruby-funktionen `split`[32], var det muligt at splitte navnet ved alle “|”-karakterer, således at de forskellige dele, der var adskilt af “|”, hver især blev til ét element i et array. Opszriftens navn kunne så udtrækkes som det første element i arrayet.

Ingredienserne i en opskrift blev udtrukket på en lignende måde med Nokogiri, men hvordan de udtrukne ingredienser skulle håndteres var omfattende, og vil blive forklaret i afsnit 8.3.5.

8.3.5 Mapping af ingredienser

En råvaretypetabel i vores database indeholder en liste over navnene på næsten 1000 råvaretyper. Denne liste giver brugeren mulighed for at indtaste en mængde råvaretyper i søgerfeltet på [foodl's](#) forside. Hvis han indtaster råvaretypen *gulerødder*, skal brugeren have muligheden for at udføre en søgning, der finder alle opskrifter, der indeholder gulerødder. Det er derfor nødvendigt for vores system at kunne oprette relationer imellem vores råvaretyper, og de opskrifter, der indeholder ingredienser magen til en råvaretype. Ingredienserne navne i Arlas opskrifter er meget inkonsistente, og indeholder ofte også mængder, enheder og forslag til Arlas egne produkter. Derfor vil navnene på en ingrediens kun i få tilfælde svare til navnet på en råvaretype. Det er derfor nødvendigt med en metode til at finde den råvaretype, som en ingrediens minder mest om. Denne proces vil vi fremover kalde for mapping. For at kunne finde ud af hvilken råvaretype en given ingrediens minder mest om, er det nødvendigt med et godt grundlag at sammenligne på. En ingrediens bør derfor analyseres på en måde, så mængde, enhed og andre ting, der er ligeegyldige for sammenligningen, udelades. Dette er blevet håndteret af en klasse, vi har kaldt **IngredientAnalyzer**. Dernæst er det også nødvendigt med en metode, der kan sammenligne to strenge og afgøre, hvor meget de minder om hinanden. Denne metode har vi kaldt **CompareStrings**.

Optimering

Hvis **CompareStrings** køres hver gang der udføres en søgning, vil det stille høje krav til hastigheden af denne funktion, for at brugeren skal kunne udføre en hurtig søgning. For ikke at sætte krav til hastigheden af funktionen, benyttede vi en relationstabell. For hver opskrift fundet på Arla sammenlignede vi hver enkelt ingrediens i opskriften med alle råvaretyperne i råvaretypetabellen ved at bruge funktionen **CompareStrings**. For hver ingrediens blev én relation indsat mellem ingrediensen og den råvaretype i råvaretypetabellen, som ingrediensen bedst matcher ifølge vores **CompareStrings**. Når en bruger udfører en søgning, vil **CompareStrings** slet ikke blive benyttet. Det vil kun være nødvendigt at undersøge de indtastede råvaretypes relationer til ingredienser, og blot præsentere de opskrifter, der er relateret til de fundne ingredienser, som et søgeresultat for brugeren.

IngredientAnalyzer

Inden en ingrediens sammenlignes med en råvaretype, så er det nødvendigt at foretage en analyse af ingrediensen. Ingredienserne på Arlas opskrifter består nemlig af både navn, mængde og enhed, mens indholdet i vores råvaretypetabel kun består af en råvaretypes navn. Første skridt i behandlingen af de udtrukne ingredienser fra Arlas opskrifter, var at fjerne mængder og enheder. Dette fik klassen **IngredientAnalyzer** ansvaret for. Arla var meget konsistente i deres syntaks, hvad angik mængde og enhed. Syntaksen var altid: **“mængde enhed navn”**. På den måde kunne vi nemt udtrække en ingrediens’ mængde, enhed og navn, ved at lade **IngredientAnalyzer** starte ved første karakter. Hvis denne er en numerisk karakter, fortsættes der indtil et mellemrum, og den fundne streng blev bestemt som ingrediensens mængde. Dernæst tjekkes om den resterende streng starter med en enhed, som f.eks. *dl*,

kg, bundt, glas efterfulgt af et mellemrum. Vi havde gemt en liste over alle de mængder Arla benyttede. Hvis en enhed kunne identificeres, blev denne bestemt til at være ingrediensens enhed. Den resterende streng blev bestemt til at være ingrediensens navn. Nogle specielle ingredienser var uden mængde og enhed, f.eks. *vand*, men fordi en sådan ingrediens ikke starter med en numerisk karakter, springes mængden over. Enheden springes også over, da *vand* ikke findes på listen over enheder, og for den sags skyld heller ikke ender med et mellemrum. Derfor vil ingrediensens navn stadig blive bestemt til at være *vand*, uden en mængde og enhed.

CompareStrings

Der findes mange forskellige algoritmer til at sammenligne to strenge for lighed. Det er vigtigt for os at finde en god og brugbar metode, der så korrekt som muligt, og kan mappe alle ingredienserne i Arlas opskrifter over til en passende råvaretype i vores råvaretypetabel. Dette fik funktionen **CompareStrings(string a, string b)** ansvaret for. Det har ikke været muligt for os at opnå en 100 % korrekt mapping ved brug af en algoritme til dette, og vi har derfor gennemgået alle mappings manuelt. Vi har erfaret, at der ved brug af en algoritme nemt kan opstå problemer omkring ingredienser der minder meget om hinanden, som f.eks. ved mapping af ingrediensen *hakket løg*. En algoritme vil have svart ved at vide, at *hakket løg* skal mappes til råvaretypen *løg* og ikke *hakket oksekød*. En korrekt mapping vil i dette tilfælde kræve kendskab til at ordet *hakket* er et udsagnsord, og derfor bør fjernes før der forsøges at mappe. Men hvis vi vil mappe ingrediensen *hakket oksekød*, så er det nødvendigt, at ordet *hakket* stadig indgår under mappingen (på trods af at det er et udsagnsord), fordi der findes mange former for oksekød, og vi ønsker at skelne imellem de forskellige former.

Efter at dataudtrækkeren havde fundet en ingrediens' navn, blev det undersøgt hvilken råvaretype fra vores råvaretypetabel denne mindede mest om. Både ingredienser og råvaretyper er strenge, så vi brugte en metode til at sammenligne, hvor ens to strenge er, og derved finde ud af hvilken streng i råvaretypetabellen, der mindede mest om ingrediensens navn. Vi har brugt tilfældigt udvalgte ingredienser fra Arlas opskrifter til at teste 5 forskellige metoder til tekstsammenligning. I mange tilfælde kunne alle 5 metoder mappe en ingrediens til en korrekt råvaretype. I enkelte tilfælde skete der dog en fejlmapping, og disse fejlmappings brugte vi til at sammenligne de forskellige tekstsammenligningsmetoder, hvilket ses i tabel 8.1.

Forklaring af de forskellige **CompareStrings** funktioner

1. Egen algoritme (lineær)
2. Egen algoritme (polynomial)
3. Levenshtein (1 point for slet, tilføj og udskift)[37]
4. Levenshtein (1 point for slet, tilføj og udskift. Score divideres med længste streng)[37]
5. Levenshtein (1 point for slet og tilføj. 2 point for udskift)[37]

Ingrediens	Metode				
	1	2	3	4	5
dildkvist	dild	dild	dild	sildefilet	dild
groft salt	salt	citron saft	frugtsaft	frugtsaft	salt
grofthakkede krydderurter, fx koriander, persille og dild	krydderurtemix	tikka <i>(indisk krydderi)</i>	hakkede tomater	hakkede tomater	hakkede tomater
basmatiris eller luftige urteris	basmati ris	herbamare urtebouillon	basmati ris	basmati ris	basmati ris
ostindisk karry	karrypasta	kinaradise	sød sherry	sød sherry	karry
frisk salvie	salvie	salvie	fiskesauce	fiskesauce	salvie
koncentreret tomatpure	tomatpure	tomatpure	soltørrede tomater	soltørrede tomater	tomatpure
hakket svinelever	hakket svinekød	svinelever	hakket svinekød	hakket svinekød	svinelever
store kapers med stilk	kapers	syltede ar- tiskokhjerter	trefarvet is	trefarvet is	kapers
hvidvin fx rieslin friskpresset limesaft kartoffel	hvidvin limesaft kartofler	hvidvin limesaft kartofler	hvidvinseddike appelsinsaft kartofler	hvidvinseddike appelsinsaft kartofler	hvidvin limesaft kartoffelmel
Total:	11	7	3	2	10

Tabel 8.1: Test af flere forskellige **CompareStrings** funktioner, der bruges til at mappe en ingrediens til en råvaretype. Fed skrift betyder at begge vores informanter har godkendt mappingen. Se bilag E

Som det ses i tabel 8.1, var metode 1 den, der gav den bedste mapping. Metoden fik 11 rigtige ud af 12 mulige, men det bør bemærkes, at sammenligningen kun er foretaget på de ingredienser, som mindst én metode mappede forkert. Vi kom forbi 56 ingredienser, udover de ingredienser vist i tabellen, før vi tilsammen havde 12 ingredienser, som én eller flere metoder mappede forkert.

Udover metode 1 og 2, som er algoritmer, vi selv har forsøgt at skräddersy til formålet, sammenlignede vi også med en kendt tekstsammenligningsalgoritme, ved navn Levenshtein. Levenshtein beregner, hvor mange operationer det kræver at ændre den ene af de to sammenlignede strenge til den anden[47]. Det er ret almindeligt at benytte 1 straf point hver gang en karakter tilføjes, fjernes eller ændres, hvilket er brugt i metode 3. Metode 4 tilføjer at resultatet af metode 3 divideres med den længste af de to strenge. Metode 5 giver 2-straf point for ændringen af en karakter.

Metode 1 er en algoritme, vi selv har udviklet, der er beskrevet med pseudokode i algoritme 8.1. Vi har benyttet pseudokode i stedet for Ruby-kode, da pseudokoden gør algoritmen kortere, mere overskuelig og ikke stiller krav til at læseren har kendskab til Ruby-kode. Algoritmen tager to strenge som input, og returnerer en værdi mellem 0 og 100. Højere returværdi betyder større lighed mellem de to inputtede strenge. 100 point opnås kun ved to identiske strenge. Idéen bag algoritmen er, at to inputtede strenge, a og b , hele tiden sammenlignes for at finde den længste streng, de har til fælles. Denne streng fjernes fra a og b , og en score med længden af fællesstrengen i anden potens bliver lagt til variablen $score$. Sådan fortsættes indtil a og b ikke længere har nogen streng til fælles. Dette er også tilfældet, hvis én eller begge er tomme strenge. Ligheden mellem de to strenge beregnes ved at finde ud af, hvor stor en procentdel den endelige score udgør af den maksimalt opnælige score, der fås kun hvis $a = b$. I tabel 8.2 ses et eksempel på, hvordan algoritmen sammenligner *hummersuppe* med *suppe med hummerhale* og kommer frem til resultatet 13.8.

Trin	Tekststreg 1	Tekststreg 2	Sammenligninger Scoreudregning
1	hummersuppe	suppe med hummerhaler	$max_length = 21$
2	hummersuppe	suppe med hummerhaler	$score = 6^2 = 36$
3		med haler	no common substrings found
4			$max_score = 21^2 = 441$
5			$return \frac{61 \times 100}{441} = 13.8$

Tabel 8.2: Her vises et eksempel på, hvordan metode 1 sammenligner hummersuppe med suppe med hummerhaler.

Metode 2 var magen til metode 1 bortset fra, at scoren kun blev forøget med længden af fællesstrengen, og ikke i anden potens som i metode 1 (se algoritme 8.1, linje 8). På samme måde blev variablen *max_score* i linje 11 også beregnet lineært.

Algoritme 8.1 Algoritmen *CompareStrings* udregner hvor ens to streme er.

```

1: function CompareStrings(str1, str2)
2:    $max\_size \leftarrow \max(str1.length, str2.length)$ 
3:   if  $max\_size == 0$  then
4:     return 100
5:   end if
6:    $score \leftarrow 0$ 
7:   while a longest common substring lcs exists in both str1 and str2 do
8:      $score \leftarrow score + lcs.length^2$ 
9:     remove lcs from str1 and str2
10:    end while
11:    $max\_score \leftarrow max\_size^2$ 
12:   return  $\frac{score}{max\_score*100}$ 
13: end function

```

Algoritme 8.2 Algoritmen *LongestCommonSubstring*, der benyttes af *CompareStrings* i linje 7, finder den længste fælles streng, som er indeholdt i begge de inputtede streme. Se figur 8.4 for en visualisering af algoritmen.

```

1: function LongestCommonSubstring( $a = \{a_0, a_1, \dots, a_n\}, b = \{b_0, b_1, \dots, b_m\}$ )
2:    $max\_length \leftarrow 0$ 
3:   for  $x = 0 \rightarrow n$  do
4:     for  $y = 0 \rightarrow m$  do
5:        $count[x, y] \leftarrow 1$ 
6:       if  $x > 0 \wedge y > 0$  then
7:          $count[x, y] \leftarrow count[x, y] + count[x - 1, y - 1]$ 
8:       end if
9:       if  $count[x, y] > max\_length$  then
10:         $max\_length \leftarrow count[x, y]$ 
11:         $end\_pos \leftarrow x$ 
12:       end if
13:        $start = end\_pos - max\_length - 1$ 
14:       return  $\{a_{start}, a_{start+1}, \dots, a_{end\_pos}\}$ 
15:     end for
16:   end for
17: end function

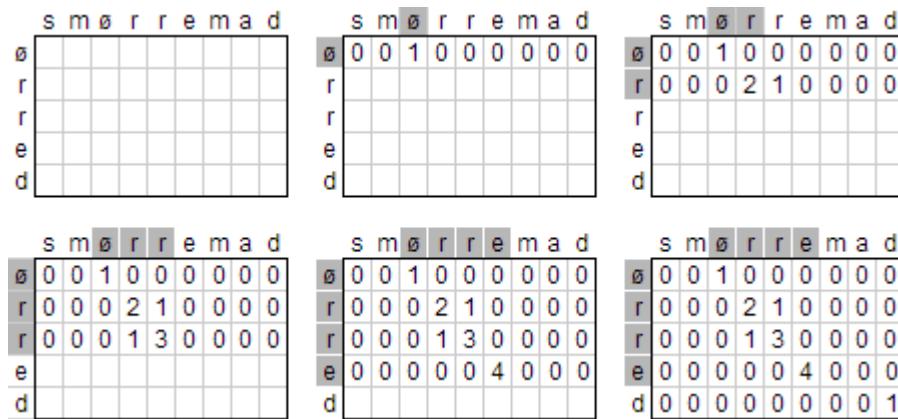
```

Algoritme 8.2 er en pseudokode, vi selv har skrevet, for at beskrive en Ruby-implementering[43] af funktionen **LongestCommonSubstring**. **LongestCommonSubstring** er en algoritme, der har til formål, blandt to strenge, at finde den længste streng, som disse to har til fælles.

Eksempel: **LongestCommonSubstring**(*ostemad*, *olivensten*) = *sle*

Et eksempel på hvordan algoritmen beregner den længste fælles streng blandt *smørremad* og *ørre* til at være *ørre*, kan ses i figur 8.4. Algoritmen laver et 2-dimensionelt array af størrelsen $n \times m$, hvor n og m er længden af de inputtede strenge a og b . Hver karakter i a og b sammenlignes i læseretningen af arrayet, se figur 8.4. Hvis karakteren a_x ikke er lig med b_y , så indsættes værdien 0 i arrayet. Er de ens, så indsættes værdien $1 + [a_{x-1}, b_{y-1}]$. Variablen max_length indeholder altid den største værdi i arrayet, mens end_pos holder den værdi x havde, da max_length sidst blev assigget til. Når algoritmen er færdig, er end_pos i dette tilfælde lig med 5, hvor begge strenge har karakteren *e*. max_length er 4, så ved at gå 3 karakterer tilbage fra end_pos fås de 4 karakterer *ørre*, som returneres i linje 14. Kompleksiteten af denne funktion beregnes ved at betragte løkken på linje 3, der eksekveres n gange, hvori løkken på linje 4 er indeholdt, der eksekveres m gange. Alle operationer inden i løkken på linje 4's scope kan udføres i konstant hastighed, hvilket betyder, at algoritmens kompleksitet er $O(n \times m)$.

Funktionen **CompareStrings**, der kalder funktionen **LongestCommonSubstring**, kan nu beregnes ved at betragte løkken på linje 7 i **CompareStrings**. For hver iteration af løkken, fjernes minimum én karakter fra begge de strenge, der sammenlignes. Hvis én af de to strenge er tomme, trædes der ud af løkken. Derfor vil løkken maskimalt blive itereret n gange, hvor n er længden af den korteste af de to inputtede strenge. Da hver iteration af løkken kalder **LongestCommonSubstring** én gang, hvis kompleksitet er $O(n \times m)$, bliver **CompareStrings** kompleksitet $O(n^2 \times m)$. Hvis inputtet er strenge af samme længde, er kompleksiteten $O(n^3)$.



Figur 8.4: Et eksempel på hvordan funktionen *longest_common_substring* virker.

Mapping af ingredienser til råvaretyper

Den automatiske mapping omfattede 10,234 ingredienser, blandt 921 opskrifter. Tidsforbruget var ca. 4 timer, og langt de fleste ingredienser blev mappet korrekt. Ind i mellem gik mappingen galt, fx. blev ingrediensen *grøtkvarnet peber* hver gang mappet til råvaretypen *bvid peber* i stedet for råvaretypen *peber*. For at få en bedst muligt mapping besluttede vi os for at gå alle mappings igennem manuelt. Under mappingen havde vi gemt returværdien af **CompareStrings(string, string)**, der fortæller hvor ens de to sammenlignede tekstrstreng var. Ved at have gemt returværdien behøver vi ikke at kontrollere mappings, der har returneret 100, da det vil sige, at ingrediensen er blevet mappet til en identisk råvaretype. På denne måde kunne vi se bort fra 1,236 ingredienser, der ikke behøvede manuel kontrol. Til at mappe ingredienserne hurtigst muligt lavede vi et meget simpelt WinForms program skrevet i C# (se bilag D). Programmet gjorde det muligt at få vist 20 labels med ingredienser. Ud for hvert label blev vist et tekstfelt med den råvaretype, som ingrediensen var blevet mappet til. Ved at ændre i tekstfeltet kunne man hurtigt få lavet en korrekt mapping. Vi tilføjede flere funktionaliteter for at øge hastigheden vi kunne mappe med.

- Ingredienserne blev sorteret alfabetisk. Omkring 120 ingredienser i træk var *grøfthakket peber*.
- Autocomplete gjorde det nemt at se, hvilke råvarer man kunne vælge imellem og også hurtigere at indtaste råvaretypen.
- Råvaretyper kunne tilføjes, hvis ingen fandtes, der matchede en given ingrediens.
- Ingredienser som f.eks. *grillspyd* og *lagkageflag* kunne fjernes i programmet.
- En hel side kunne godkendes med ét klik, hvis alt var mappet korrekt på forhånd.

Tidsforbruget på remappingen var ca. 6 mandetimer, hvilket kan omregnes til $\frac{10234 - 1236}{6} = 1500$ ingredienser pr. mandetime.

9 Kvalitetssikring

I følgende kapitel beskriver vi de unit tests vi har foretaget på systemet. Vi vil derudover beskrive den usability-test vi foretog på vores to informanter. Usability-testen bestod af en tænke højt session, sammen med en Instant Data Analysis for at finde frem til nogle af de kritiske, seriøse og kosmetiske fejl der findes i systemet.

9.1 Unit test

For at verificere at programmet opfører sig som vi ønsker, har vi benyttet RSpec, hvilket er et Ruby unit test framework, der først og fremmest er minded mod “behavior-driven development”, men også kan bruges til at skrive unit tests med. Vi har unit testet bruger-modellen, fordi den er central for systemet, og den er i direkte kontakt med brugeren og kan have kritiske fejl. Derudover har vi også unit testet enkelte forespørgelser (requests i RSpec), der simulerer en brugeres interaktion med systemet. Feks. testes der login-systemet samt det HTML, der vises, efter man har logget ind. Vi har ikke foretaget unit tests af hele systemet, hvilket er grundet nød af ressourcer. Idet pålidelighedskriteriet, specificeret i kapitel 6, er blevet vurderet som “mindre vigtigt”, har vi valgt at nedprioritere dem. Eksempler på nogle af de unit test vi har foretaget på bruger-modellen, ved hjælp af RSpec, ses i eksempel 9.1 og eksempel 9.2. For resten af tests refereres der til kildekoden.

Det skal nævnes, at “`@user`” er en instansvariabel defineret tidligere i unit test filen og er et objekt af klassen “`user`” (se afsnit 6.2.4), der indeholder de attributter og metoder, der hører med til alle instanser af klassen. Eksempel 9.1 viser en enkel unit test for brugeroprettelsesprocessen. Her skal kodeordfeltet og kodeordbekræftelsesfeltet være præcis ens, ellers kan objektet ikke være valid. Metoden `valid?` på bruger-objektet er en Active Record metode (bruger-modellen nedarver fra ActiveRecord-klassen) og returnerer blot sand eller falsk, alt afhængigt om objektet kan gemmes i databasen eller ej. Denne metode bliver kaldt på linje 3 i eksemplet.

Eksempel 9.1: Unit test af brugerens har indtastede password, når han/hun opretter sig som bruger

```
1 describe "when password is not present" do
2   before { @user.password = @user.password_confirmation = " " }
3   it { should_not be_valid }
4 end
```

Eksempel 9.2 viser unit test af emailadressefeltet, også fundet i brugermodellen. Her testes der på validiteten af emailadressen. Emailadresser i bruger-modellen bliver tjekket af et regular expression, inden objektet gemmes i databasen. Derfor testes der på adresser, der strider imod det regular expression og burde derfor ikke godkendes ved brug af `@user.valid?`.

Eksempel 9.2: Unit test af en brugers email er valid, når han/hun opretter sig som bruger

```
1 describe "when email format is invalid" do
2   it "should be invalid" do
3     addresses = %w[user@foo.com user_at_foo.org example.user@foo.
4                      foo@bar_baz.com foo@bar+baz.com]
5     addresses.each do |invalid_address|
6       @user.email = invalid_address
7       @user.should_not be_valid
8     end
9   end
10 end
```

Der er foretaget flere lignende unit test på bruger-modellen samt login-systemet. Der er ikke nogen konkret kode-dækningsprocent, da det er svært at vurdere, hvorvidt de forskellige actions er blevet testet.

9.1.1 Metode til at finde lighed mellem tekststrenge

Vi benyttede en ret simpel fremgangsmåde til at unit teste metoden **CompareStrings**, der tester ligheden mellem to tekststrenge. Vi testede denne funktion i Ruby, ved at initialisere et array som vist i eksempel 9.3. Der er ikke brugt RSpec til unit tests af dette program.

Eksempel 9.3: Et eksempel på en række testcases til brug ved unit test.

```
1 test_cases = [
2   ["aa12345", "aa67890", "aa"],
3   ["aa12345", "678aa90", "aa"],
4   ["aa12345", "67890aa", "aa"]
5 ]
```

Arrayet **test_cases** fyldes med testdata og forventet værdier. Selve arrayet består af et sub-array, hvor hvert element i nævnte rækkefølge er: [input1, input2, forventet returnværdi]. For hvert element i arrayet, blev det testet om **CompareStrings(input1, input2)** returnerede den forventede værdi. Eksemplet er kun et udpluk af 26 test-cases, der fremgår af kildekoden.

9.2 Test af usability

For at teste usability i forbindelse med vores system, har vi benyttet os af Instant Data Analysis-metoden[23]. Vi har dog kun haft 2 tænke-højt sessions, hvor det anbefalede for instant data analysis-metoden er 4-6. Først lavede vi en liste over alle de ting vi gerne ville have testpersonerne til at udforske. Det kunne f.eks. være at skifte sin adgangskode eller at favorisere en opskrift. På baggrund af denne liste konstruerede vi en case bestående af opgaver, som testpersonerne skulle udføre i en bestemt rækkefølge, for netop at komme omkring alle de udvalgte ting vi ville have udforsket. Casen kan ses i bilag F. Vi udførte de to tænke-højt-sessions på informanterne Keld og Merete i nævnte rækkefølge. Vi havde ikke mulighed for at benytte os af Usability Lab på Cassiopeia, da mange andre grupper havde booket en tid, og vi kunne ikke få tiderne til at passe sammen med vores informanters ønsker. I stedet for at bruge Cassiopeias Usability Lab til at overvåge processen, installerede vi programmet Teamviewer på den bærbare computer informanten udførte casen på. De 3 personer, der ikke var ude ved informanten kunne fra deres egen bærbar logge på Teamviewer og se hvad der foregik på skærmen, samtidig med at det indbyggede webcam og mikrofon gjorde det muligt at se informantens reaktion og høre ham tænke højt. De andre 3 i gruppen tog ud til informanten, hvor en havde rollen som “test monitor”, den anden var “data logger” og den sidste var teknologiansvarlig, ved at stille computer samt software til rådighed.

Videoer af de 2 tænke-højt-sessions er blevet uploadet på YouTube [12] [13].

Efter de 2 tænke-højt sessions, samlede vi os uden informantene og lavede en brainstorm over de usability-problemer informantene var stødt på i forbindelse med testen. Én logfører havde under alle tænke-højt sessions noteret hver gang han opdagede et problem. Problemerne vi kom frem til blev klassificeret enten som kritisk, seriøst eller kosmetisk.

9.2.1 Usability-problemer

Vi blev opmærksomme på følgende usability-problemer efter de to tænke-højt sessions. En opgørelse over antallet af kritiske, seriøse og kosmetiske fejl kan ses i tabel 9.1. Fejlene kategoriseres som kritiske (alvorlige), seriøse eller kosmetiske, der defineres på følgende måde:

Alvorligt problem: Brugeren anvender uforholdsmæssig lang tid på at løse opgaven, men klarer sig igennem uden hjælp. Problemets bør rettes i næste udgave [31].

Kritisk problem: Brugeren kan ikke løse en stillet opgave på egen hånd, eller (katastrofe) brugeren synes, at netstedet beder ham om at gøre noget, som er helt urimeligt. Problemet bør rettes omgående [31].

Kosmetisk problem: Et mindre problem, som forsinket brugeren kortvarigt, eller som fremkalder en irriteret bemærkning fra brugeren. Problemet rettes ved lejlighed [31].

- Indkøbsliste
 - Det var svært at finde linket til indkøbslisten, der er placeret i toppen. (**kritisk**)
 - Knappen til at udskrive indkøbslisten tog lang tid at finde. (**kosmetisk**)
 - Opdatering af indholdet på indkøbslisten i sidehovedet skete ikke automatisk. (**kosmetisk**)
 - Han ønsker en form for bekræftelse, når en opskrift bliver tilføjet til indkøbslisten. (**kosmetisk**)
 - Nogle ingredienser kunne have mængden 0, f.eks. “0 tsk salt”, “0 citronskal i strimler”. (**kosmetisk**)
 - Indkøbslisten var så lang, at det var svært at finde feltet til at tilføje endnu en vare, da feltet var i bunden. (**kosmetisk**)
 - En informant prøvede at klikke under sidste vare på listen, for at tilføje en ny.
- Søgning
 - En stavfejl, i en ingrediens indtastet af brugeren, førte til at der ikke kom forslag til råvarer i søgerfeltet. (**seriøs**)
 - 2 råvaretyper blev indtastet i søgerfeltet på samme tid, separeret af et komma. Råvaretyper skal indtastes og tilføjes én af gangen. (**kritisk**)
 - En informant foreslog af ændre placeholder-teksten i søgerfeltet til “indtast én råvare” i stedet for “indtast råvare”.
 - En informant ville søge efter to råvaretyper. Den anden ingrediens blev skrevet i feltet, men ikke tilføjet. Da brugeren trykkede på søg, blev den ignoreret og der blev dermed kun søgt på den første råvaretype. (**seriøs**)
- Søgeresultat
 - En informant formåede ikke på egen hånd at tilføje en ingrediens fra en opskrift til indkøbslisten. (**seriøs**)
- Sidehoved og toolbar
 - Både sidehovedet og toolbaren virkede usynlige for brugeren (**seriøs**)
 - En informant prøvede at sortere opskrifter ved at højreklikke på siden
 - En informant prøvede at sortere opskrifter ved at klikke på “Indstillinger” i sidehovedet.
 - foodl-logoet blev overset, som en funktion til at gå tilbage til forsiden. (**seriøs**)
 - En informant foreslog at vi kunne bruge konventionen hus-ikon
 - Funktionen af knapperne i toolbaren var ikke selvbeskrivende. (**seriøs**)
 - Sliderens håndtag, til at skalere opskrifter med, træder ikke klart nok frem. (**seriøs**)
 - En informant prøvede at klikke på tallet, der justeres af slideren, og ændre det med tastaturet.
 - En informant forventede ikke at knappen “Navn” ville sortere opskrifterne i alfabetisk orden. (**seriøs**)
- Favorisering
 - Favoriserede opskrifter forsvinder ikke med det samme, når de bliver fjernet fra favoritter. (**kosmetisk**)
- Kontakt os
 - En informant kunne ikke finde ud af at ordet mail i teksten “kontakts os per mail.” var et link, da formateringen af hele sætningen var forkert. (**seriøs**)

9. Kvalitetssikring

Fejltypen	Kritiske	Type	
		Seriøse	Kosmetiske
Antal	2	9	6

Tabel 9.1: *Antallet af usabilityproblemer kategoriseret efter type*

Udover usability-testen, benyttede vi også mødet med informanterne, til at høre hvad de syntes om foodl som helhed. Efter udførelse af test-casen havde vi en ustuctureret snak om hvad de syntes om systemet. Begge informanter var enige om, at systemet var brugbart og ville gøre det nemmere at finde en anvendelse af deres madrester. De var mildt sagt ret imponeret.

10 Konklusion

I vores projektforløb har vi arbejdet med at løse et problem, som i problemformuleringen blev beskrevet som følger:

“Hvordan kan man ved hjælp af et system gøre det muligt at anvende madrester i de danske husstandes madlavning?”

Til udvikling af systemet anvendte vi den objektorienterede metode fra bogen “Objektorienteret Analyse & Design” [25]. Metoden er meget anvendelig i forhold til at analysere og designe objektorienterede systemer, f.eks. webapplikationer ved brug af en iterativ arbejds metode på problemer med høj usikkerhed og lav kompleksitet. Igennem projektforløbet har vi haft et tæt samarbejde med to informanter, Merete og Keld. Sammen med informanterne formulerede vi en systemdefinition, som kan ses i afsnit 3.3.1. Systemdefinitionen opfylder BATOFF-modellens punkter, som beskriver, hvad systemet skal kunne, hvor det skal kunne tilgåes fra, under hvilke forhold systemet bliver lavet, mm.

Alt det analyserede er blevet designet. Af det designede, er der dog ting vi ikke har valgt at implementere af hensyn til deadline for projektet:

- Dataudtrækkeren gennemgår kun opskriftssiderne på Arla én gang. Der er ikke implementeret muligheden for at vende tilbage til disse sider og tjekke om der er sket ændringer, for i givet fald at opdatere modellen.
- Fejlrapportering af opskrifter er kun delvist implementeret. Man kan rapportere fejl i en opskrift fra søgesiden ud fra den opskrift, der er fejl i. Den generelle fejlrapporteringsknap, der findes som en footer på alle sider har dog ingen funktion.

Samarbejdet med informanterne omfatter møder, som har hjulpet os med at blive mere bevidste om, hvilke kriterier systemet skulle opfylde, for at blive så attraktiv som muligt. Samarbejdet har desuden omfattet test af systemet ved hjælp af prototyper og en usability test af det endelige system. Ved brug af prototyper, blev vi mere bevidste om, hvordan **foodl**'s brugergrænseflade skulle opstilles, for at være brugbar, samt hvilke funktioner systemet skulle indeholde.

Til at løse problemet udviklede vi en webapplikation, kaldet **foodl** (<http://foodl.dk>), som ved hjælp af indtastning af nogle madvarer, kommer med forslag til relevante opskrifter. Med relevante, menes opskrifter, som indeholder så mange af de indtastede ingredienser og madvarer som muligt, så disse kan blive anvendt.

Vi kan konkludere, at **foodl** gør det lettere for danskerne, at få inspiration til, hvordan de kan få anvendt deres madrester. **foodl** gør det muligt at anvende madrester, ved at præsentere opskrifter, hvor disse madrester indgår. Systemet er blevet usability-testet af to informanter, der begge stod for madlavningen til deres respektive husstand. Begge informanter havde begrænset erfaring med brug af IT-systemer, men de var alligevel i stand til at benytte **foodl**.

10.1 Perspektivering

I dette afsnit vil vi reflektere og perseptivere over de valg, vi har taget mht. det endelige system (**foodl**).

Formålet med **foodl**; at mindske madspild i danske husstande, er ikke et nyt koncept. **foodl** er relativt nyt, så der vil højest sandsynligt være mange punkter, hvor der kan ske ændringer, som vil gøre systemet bedre for befolkningen. Her prøver vi at reflektere over, hvordan dette kunne ske.

foodl implementerer ikke nogle muligheder for f.eks. allergikere at begrænse søgeresultater til f.eks. at udelukke opskrifter med nødder eller lignende. Dette kunne være en fornuftig feature at have, hvis man nemt og hurtigt ville have et overblik over de opskrifter, man selv kan have glæde af at lave.

Til webapplikationen **foodl** har vi kun kontaktet Arla og fået lov til at benytte os af deres, omkring 1000, opskrifter. Ved kun at hente opskrifter fra denne ene side støder vi på nogle ulemper, på grund af de særtræk, der er ved Arlas opskrifter. Enkelte ingredienser er Arlas egne produkter, hvilket giver en anledning til at tro at en lignende ingrediens måske kunne være lige så god eller bedre. Alle opskrifter virker til at være ret krævende med hensyn til antallet af ingredienser der bruges, f.eks. Arlas opskrift på øllebrød, der uddover det basale: ol, brød og sukker, også indeholder kanel, appelsinskal, appelsinsaft og flødeskum. Ved at indhente opskrifter fra flere forskellige opskriftshjemmesider, vil det blive muligt at give brugerne adgang til mange flere opskrifter. Det vil hjælpe på problemet med at ikke alle råvarer er forbundet med en opskrift, og det vil også afhjælpe ulempene ved de særtræk Arlas opskrifter har.

Ved kun at benytte Arla som kilde, vil vi få et problem hvis Arlas sider er nede i et par dage. Ved at benytte 100 forskellige kilder, ville vi blot kunne deaktivere søgningen på Arlas opskrifter mens deres side er nede, hvilket ville få minimal betydning blandt 99 andre kilder, sammenlignet med nu, hvor **foodl** er afhængig af tilgængeligheden af Arlas hjemmeside.

Madspild forekommer ikke kun i Danmark. Derfor kan man også udvikle systemet til at understøtte andre sprog, og muligvis arbejde sammen med opskriftshjemmesider fra forskellige lande for at kunne levere webapplikationens ydelse til andre lande, der også kunne have gavn af mindre madspild.

Ser vi på optimering af algoritmer, så skal der være særlig fokus på, hvordan en søgning foretages. På nuværende tidspunkt, så vises søgeresultater, der indeholder ingredienser, der er indtastet af brugeren. Altså den opskrift, der indeholder de fleste matchende ingredienser bliver vist først. Man kunne f.eks. sortere ud fra hvor stor en procentdel af alle opskriftens ingredienser, der matcher. På denne måde mindsker vi det eventuelle indkøb, der skal til for at lave en opskrift. Et eksempel kunne være en søgning på råvarerne kylling og pastinak. Hvis to opskrifter fremkommer, den ene med 10 ingredienser og den anden med 25, så vil det, at lave en af opskrifterne, kræve at man skaffer yderligere 8 eller 23 nye råvarer (forudsat man ikke i forvejen har disse). Det ville være smartest at få de opskrifter præsenteret først, der kræver mindst muligt indkøb. Ved at udvikle en mere intelligent søgning, så kan vi også mindske det fremtidige madspild, fordi vi begrænser det eventuelle indkøb af manglende råvarer.

Del II

Akademisk rapport

11 Indledning

Til et systems udviklingsproces kan der være anvendt en række metoder og værktøjer. Med værktøjer menes eksempelvis softwareværktøjer, informanter eller andre remedier. Nogle metoder og værktøjer har forskellige fordele og ulemper. Det vigtigt for os at reflektere over hvorvidt de anvendte metoder og værktøjer har været anvendt, har været optimal. Hvis det ikke er tilfældet, hvad er årsagen så til dette? Hvilke andre metoder eller værktøjer eksisterer, som i så fald kunne være blevet anvendt i stedet?

I projektforløbet har vi været seks datalogistuderende, som har haft en tidsperiode på cirka fire måneder, til at udvikle et system, som skulle løse et realistik og eksisterende problem. I vores tilfælde valgte vi at arbejde med problemet madspild i et forsøg på at gøre det nemmere for danskerne at mindske dette. I udviklingsprocessen anvendte vi både en evolutionær og en konstruktiv udviklingsmetode [14]. Vi havde en objektorienteret tilgang til problemet, hvor vi fulgte en metode fra bogen Objektorienteret Analyse & Design [25], og i den forbindelse indragede vi to informanter. Vi anvendte et framework kaldet Ruby on Rails.

I denne akademiske rapport har vi valgt at beskrive og reflektere over følgende tre punkter:

- Samarbejdet med informanter (se kapitel 12)
- Den evolutionære- kontra den konstruktive udviklingsmetoden, samt den objektorienterede tilgang (se kapitel 13)
- Vores tekniske platform, som var Ruby on Rails (se kapitel 14)

12 Samarbejde med informanter

Vi har haft et tæt samarbejde med to informanter fra Aalborg. De har hjulpet os med at fortolke problemet ved at deltage i interviews og diskussioner vedr. problemet. Dette har gjort det muligt for os at skabe en brugbar analyse til design af vores system. Informanter har været inddraget i forbindelse med 2 interviews hver, afprøvning af 3 prototyper og endelig en færdig test af programmet. Informanterne er Merete og Keld. Merete er en familiemor, der bor med sin mand. Merete står for madlavningen i husstanden. Ligelides er Keld en familiefar, der står for madlavningen og bor med sin kone og deres to små døtre. For en længere beskrivelse, se afsnit 2.2.

Vi valgte at arbejde med netop disse informanter, fordi de var potentielle brugere af det system, som vi ønskede at udvikle. Begge informanter har været yderst interesserede i at hjælpe os igennem projektet, og de har været gode til at udtrykke deres idéer, når vi havde diskussioner.

Vi er klare over, at vores to informancers madvaner ikke kan generaliseres til hele Danmark. Vi vurderer dog, at samtalerne med informanterne giver et godt billede af, hvordan situationen, mht. madlavningen og madvaner, ser ud i nogle danske husstande.

De to informanter har ca. samme vilkår mht. familien, job og IT-kompetencer, hvilket var meget tydeligt, da vi ofte fik meget lignende respons fra begge informanter. Hvis vi havde ressourcer til at arbejde sammen med flere informanter, der havde flere forskelle, der kunne påvirke hvordan de bruger systemet, ville vi helt sikkert gøre det. Det kan være at folk, der er eneboende sammenlignet med familieboende oplever problemet med madspild og brugen af madrester på en anden måde. Det kunne også være der var en forskel på rige kontra fattige. Vi mener, at dette ville være en fordel, og hvis der er mulighed for at have flere informanter i et fremtidigt projekt, så vil vi helt klart forsøge at få informanter med forskellige baggrunde med i projektet. Vi synes informanternes inddragelse var helt tilpas. Vi havde hele tiden viden nok at arbejde med til at kunne videreudvikle systemet. Med færre afholdte møder ville vi nok have været i tvivl om hvordan dele af systemet skulle se ud med risiko for at skulle lave det om. Med flere afholdte møder ville vi trække for meget på vores egen men også informanternes tid, selvom det måske kunne have afklaret nogle mindre ting vi var i tvivl om. Vi valgte i stedet at ringe vores informanter op når vi havde nogle små tvivlsspørgsmål. Det skete f.eks. da vi var i tvivl om hvorvidt informanterne ønskede mængdeangivelser på deres indkøbsliste.

12.1 Møder med informanter

Projektets problemstilling var meget bred, og vi ønskede at skabe os et lidt bedre overblik og få mulighed for at fortolke problemet, som vi ønskede at arbejde med. De indledende møder med informanterne var derfor fokuseret på selve problemet, og hvordan de oplevede madspildet i deres respektive husstande. Hver gang vi skulle holde et møde eller præsentere noget for informanter, så tog vi hjem til dem og holdte møderne. Dette valgte vi at gøre, fordi vi vurderede, at informanterne måske ville føle sig mindre pressede af os og mere trygge, hvis de var i deres naturlige omgivelser, end hvis de skulle med os op til Universitet hver gang. Vi mente, at hvis informanterne var trygge, så ville de også være mere villige til at diskutere og tale med os.

På baggrund af de indledende møder havde vi viden nok til at diskutere og fortolket problemet, og deraf formulere to systemdefinitioner, som vi kunne præsentere for informanterne. Systemdefinitioner er en del af den objektorienterede tilgang til et projekt, hvilket vi mener fungerede rigtig godt, fordi systemdefinitionerne gav os mulighed for at præsentere vores idéer for informanterne, og informanterne havde mulighed for at give os feedback på, hvad de kunne se som en fornuftig løsning på problemet. Hvis det ikke var for systemdefinitionerne, så havde vi højst sandsynligt endt ud med et produkt, som indeholdt rigtig mange funktioner, som brugerne ikke ønskede at tage i

brug. Det feedback, vi har fået af informanterne bl.a. vedr. systemdefinitionerne, har været rigtig godt, og det har ledt os i den rigtig retning.

Vi ønskede ikke at have truffet nogle faste beslutninger omkring systemet, inden vi havde afholdt møder med informanterne. Derfor arbejdede vi med at udvikle semi-strukturerede interviews. Dette betyder, at vi havde en struktur og nogle spørgsmål, som vi ønskede at stille informanterne, men hvis de introducerede et emne, som vi ikke havde taget højde for, så ville det ikke ødelægge interviewets flow. Der var altså luft til eventuelle spørgsmål og nye emner midt i interviewet.

12.2 Prototyper og usabilitytest

Ud over interviews og diverse diskussioner med informanterne, præsenterede vi vores idéer for dem ved at udvikle prototyper. Vi benyttede os af papirsprototyper til de initierende afprøvninger. Denne form for prototype er ikke tidskrævende og er en billig form for præsentation. Da det var initierende afprøvninger, var det godt, ikke at bruge for mange kræfter på at udvikle prototyperne, da vi det sted i processen havde stor usikkerhed omkring hvordan systemet skulle fungere. Samtidig kunne det også have en ulempe at bruge meget tid på prototyperne, nemlig at vi ville have svært ved at give slip på idéerne, hvis informanterne ønskede noget helt andet end det prototyperne illustrerede.

I de senere forløb præsenterede vi informanterne for en hifi-prototype, der var brugt længere tid på at lave, og som til forveksling kunne ligne et rigtigt system. Vi lavede en diashow-prototype, der havde til formål at undersøge hvilke funktioner systemet skulle bestå af. Prototypen var dynamiske, så man kunne klikke på knapper og navigere rundt i diashowet. Systemets brugbarhed var en vigtig faktor for os, og vi ønskede at gøre det lettere for den madansvarlige i husstanden at bruge sine madrester i madlavningen. Derfor skulle systemet være intuitivt og nemt at gå til. Med diashow-prototypen fik vi mulighed for at sikre os, at vores designidéer blev forstået af informanterne. Hvis informanterne f.eks. havde svært ved at finde nogle funktioner, så kunne det være, at de skulle gøres mere synlige. Lignende spørgsmål blev besvaret relativt tidligt i systemets udviklingsfase, hvilket var en fornuftig ting. Det gav os mere tid til at rette fejl og komme på nye designidéer, hvilket blev nødvendigt.

Det fungerede rigtig godt med papirsprototyper til de indledende afprøvninger. Vi kunne overraskende hurtigt præsentere vores idéer for informanterne og afprøve om idéerne helt basalt var brugbare. Informanterne kunne nemt forestille sig prototyperne afspejle et rigtigt system og finde fordele og ulemper ved dem, selvom de blot var lavet af papir. Det var ikke nødvendigt for os at lave helt nye prototyper, fordi informanterne gav os rigtig god feedback, som vi arbejdede videre med, men hvis det skulle blive nødvendigt, så ville det have været nemt at udvikle nye prototyper i de indledende faser af projektet, fordi de var hurtige og nemme at lave. Et eksempel kunne være, hvis det viste sig, i vores to første prototyper, at informanterne ikke kunne finde ud af at tilføje råvaretyper at søge på. Diashow-prototyperne blev også præsenteret på et fornuftigt tidspunkt. Informanterne fik en smagsprøve af vores idéer til hvordan systemet skulle se ud, og hvilke funktioner man kunne bruge. Det var rigtig godt for os som udviklere at få lov til at afprøve vores idéer for at se, hvordan de ville reagere på systemet. Vi fik rigtig meget ud af det, og vi reviderede systemet ud fra de test, vi fik lavet med informanterne.

Brugen af prototyperne var generelt en god idé, for det gjorde det meget nemmere for informanterne at følge med i vores tankegange, fordi vi visualiserede idéerne for dem. Det er helt sikkert noget, vi vil tage med os videre i fremtidige projekter.

Som en sidste afprøvning af systemet præsenterede vi det funktionsdygtige system for informanterne. Til de sidste afprøvninger var der mulighed for at opdage kritiske fejl, fordi systemet nu var færdigt. Med et færdigt system har brugerne større mulighed for at være kritisk. I en papirsprototype vil en bruger måske tænke, at en fejl blot er med vilje og vil blive rettet, for det er jo bare en skitse han har fået. En anden forskel er, at vi nu sætter krav til systemet. Et eksempel er vores papirsprototyper, hvor det var facilitatoren, der kom med forslag når en bruger indtastede starten af en råvaretype i søgerfeltet. I det færdige system lå ansvaret på systemet, og systemet gør ikke brugerens opmærksom

på forslagene til det han indtaster i søgerfeltet på samme måde, som når facilitatoren ved papirsprototyperne siger “Vent lidt, jeg skal lige skifte layout, du får lige denne liste sat ind under dit søgerfelt”.

Til den afsluttende test af systemet afholdt vi en test-session, hvor begge informanter skulle udføre en case. Efter casen benyttede vi Instant Data Analysis[23] til at analysere hvad der skete under casen og finde flest mulige usability-problemer. Casen var lavet for at få informanterne omkring flest mulige dele af systemet. Et punkt i casen kunne f.eks. være ”Opret en bruger på Foodl”. Fra tidligere afprøvninger havde vi bemærket, at informanterne virkede en smule anspændte under afprøvninger, så i stedet for at hele gruppen var til stede under vores afprøvning, installerede vi TeamViewer på den bærbare computer, som informanterne udførte casen på. TeamViewer sorgede for at dele bærbarens webcam, mikrofon og en live stream af skærbilledet med en anden computer. På den måde kunne vi nøjes med kun at have 2 personer til stede ved casens udførelse. En facilitator, der guider informanten igennem casen og en logfører, der noterer hvad der sker. Resten af gruppen, der ikke var til stede under afprøvningen, fulgte meget præcist med i casen fra grupperummet på Cassiopeia og føgte også en log imens.

Da de to cases var blevet udført, forlod vi efterfølgende informanterne og begyndte at brainstorme hvilke usability-problemer de to cases havde afsløret. Vi brugte hukommelsen og logbøgerne, og da vi efterhånden følte os meget sikre på, at vi havde fået alle de væsentlige problemer med, men kunne samtidig mærke en vis usikkerhed på, hvornår vi skulle stoppe, for der kunne jo lige pludselig dukke et problem mere op, som logføreren ikke havde noteret og som lå gemt dybt i hukommelsen. På dette punkt ville en videodata-analyse have en fordel, da vi blot ville kunne analysere et lille stykke film af gangen og være sikker på at finde alle usability-problemer i hver stump film. På den måde ville der ikke være samme usikkerhed omkring, hvornår vi burde stoppe, og hvor mange fejl vi ikke fandt. Når alt film var analyseret ville vi være færdige. Vi mener dog, at vi identificerede langt de fleste problemer, og at eventuelle mangler primært ville være kosmetiske problemer. Hvilken analysemetode vi vil vælge for et fremtidigt projekt vil selvfølgelig afhænge af hvilket system, der testes. Til et projekt lignende dette, ville vi til enhver tid benytte Instant Data Analysis, hvis lave tidsforbrug var altafgørende. Informanterne virkede trygge ved brugen af Teamviewer, og de gruppemedlemmer, der ikke var til stede, mener alle at have fået lige så stort udbytte, hvis ikke større, som hvis de havde været til stede under testen. Det var nemlig nemmere at se informantens handlinger og tilhørende ansigtsudtryk på samme tid igennem Teamviewer. På den måde kunne man lægge mærke til, når der skete noget uventet for informanten.

I flere tilfælde i dette projekt har vi følt at informanterne var håbløst uvidende om hvordan en meget simpel ting skulle gøres i **foodl**. Det kunne f.eks. være at gå ind på sin indkøbsliste. Årsagen kan være at vi, datalogistuderende, har stor erfaring med brug af systemer, hvilket betyder at vi har brugt almindelige måder at gøre ting på **foodl**, som informanterne ikke har kendt til. Et eksempel kan være at man ved at trykke på sidens logo kommer tilbage til forsiden. Vi har erfaret, at ved at arbejde på et projekt i næsten et halvt år, så får vi et indgående kendskab til mange små dele af systemet. Det indgående kendskab får os til at føle at en avanceret funktion er meget simplere end den vil fremstå for en ny bruger af siden. Derfor har vi nogle steder ikke givet nødvendig vejledning til hvordan en funktion bruges, som f.eks. søgerfeltet på forsiden af **foodl**. I det store hele har vi lært hvor vigtigt det er at inddrage brugere når man designet et system.

12.2.1 Løsning af usability-problemer

Der var ikke tid til at løse alle usability-problemerne og vi opsummerer her de ændringer vi ville have lavet hvis vi havde haft tid.

Indtastning af råvaretyper

Begge informanter havde problemer med at indtaste råvaretyper første gang de brugte siden. Dette mener vi er fordi det ikke er tydeligt at man skal vælge en råvaretype på en liste, i stedet for at man kan skrive fritekst som i de fleste andre søgemaskiner.

Vi vil derfor prøve at give bedre tilbagemeldning så brugeren bedre kan forstå hvad man skal gøre. Hvis der skrives en råvaretype som ikke eksistere i systemet dukker der ikke nogen forslag op og det ser ud som om at alt er som det skal være. Men da dette ikke er tilfældet vil vi vise en besked hvor forslagene normalt dukker op som gør brugeren opmærksom på problemet. Man kunne supplere dette med et billede på forsiden første gang **food**besøges, hvor søgefeltet vises med en halvt indtastet tekst og nogle forslag i drop-down boksen.

En skrivefejl førte også til at systemet ikke kunne give nogle forslag, hvilket førte til samme problem som før. Men da dette let kan ske burde systemet se bort fra disse og stadig give relevante forslag.

Et andet problem var at man hvis man ikke havde tilføjet den sidste ingrediens, men bare havde skrevet den i feltet, så søgte den uden den sidste ingrediens. For at undgå at dette sker skal knappen kun virke hvis feltet er tomt, så brugeren kan se at man skal tage stilling til det.

Navigation

Indkøbslisten var svært at finde for informanterne fordi de ikke lagde mærke til toolbaren i toppen af skærmen. For at gøre brugeren opmærksom på indkøbslisten, skal den oplyses når man tilføjer en opskrift eller ingrediens til indkøbslisten.

Konventionen at sidens logo fører til forsiden var heller ikke tydelig, og for at forbedre dette vil vi tilføje et ikon af et hus ved siden af logoet.

At skifte sorteringsorden på resultatsiden var også svært at finde, for at gøre det lettere at forstå vil vi gøre teksten mere selvførlærende. Man kunne eventuelt tilføje teksten "Sorter efter." i forbindelse med knapperne.

13 Udviklingsmetoden

I følgende kapitel vil vi beskrive og reflektere over den objektorienterede analyse- og designmetode, som vi er blevet introduceret for i bogen Objektorienteret Analyse & Design [25], samt over den evolutionære udviklingsmetode som vores projektforløb har taget udgangspunkt i.

13.1 Objektorienteret Analyse & Design

Objektorienteret Analyse & Design er titlen på den bog, som vores tilgang og projektforløb har taget udgangspunkt i. Bogen beskriver en objektorienterede metode til udvikling af systemer. Metoden består af fire hovedaktiviteter: Analyse af problemområde, Analyse af anvendelsesområde, Design af arkitektur og Design af komponenter [25, p 15]. Hvilke af aktiviteterne man udfører først, er ikke fast, men er helt op til det enkelte projektforløb. Vi modtog, sideløbende med vores projekt, undervisning i kurset (Systemudvikling), hvor vi først blev introduceret for Analyse af problemområde, derefter Analyse af anvendelsesområde, Design af arkitektur og til sidst Design af komponenter. Derfor var det naturligt for os at følge denne rækkefølge, hvilket vi gjorde meget slavisk. Bogen præsenterer desuden en masse værktøjer, såsom rige billeder, introduktion til prototyper mm., som er meget anvendelig i en udviklingsproces, hvor der er informanter/brugere involveret.

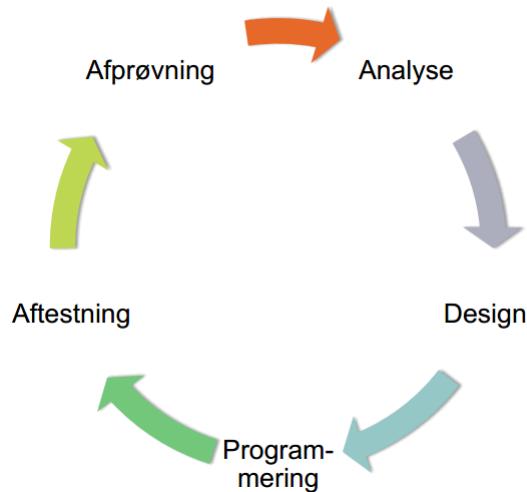
En af fordelene ved at anvende bogens objektorienterede analyse- og designmetode er, at programmeringen og implementeringen af systemet, bliver nemmere, da der efter analyse- og designdelen, ikke hersker nogen tvivl om, hvad systemet skal kunne, hvordan det er modelleret, og hvordan det skal designes. Ulempen er, at der skal bruges meget tid på at analysere og designe. I vores projektforløb brugte vi endda ekstra lang tid, fordi vi først skulle sættes ind i, og forstå kurssets begreber, før vi kunne anvende dem. Desuden var vi meget grundige med vores analyse- og designdel, og alle gruppens medlemmer deltog aktivt i eksempelvis, hvilke objekter, klasser og hændelser, der var i problemområdet, hvordan klassernes tilstandsdiagrammer skulle se ud, hvilke aktører vi skulle have osv. I det hele taget, brugte vi meget tid på at bearbejde og revidere diverse diagrammer. Tid, som vi kunne have brugt mere effektivt andet steds. Vi kunne eksempelvis have delt opgaverne ud, så alle ikke skulle side og kigge på det samme komponentdiagram, tilstandsdiagram osv.

Til fremtidige projekter vil vi anvende dele af den objektorienteret analyse og design metode, som kurset har introduceret. De mange gruppeditiskussioner vi i fællesskab har haft, har ledt os til en bedre forståelse for systemudvikling og de ting, feltet indebærer, og den objektorienteret metode vil formentlig kunne fungere endnu bedre, i et fremtidigt projekt, fordi vi nu har fået erfaring i at bruge den. Vi vurderer tilgengæld, at vi sandsynligvis ville kunne have lavet et ligeså godt produkt, med samme kvalitet, med mindre brug af den objektorienterede analyse- og designmetode. Der blev brugt alt for meget tid på mindre detaljer, som hele gruppen brugte mange resurser på, som vi efterfølgende føler, ikke havde den store betydning, eller var trivielle og ville blive tilføjet eller redigeret naturligt undervejs, pga. den iterative udviklingsmetode, som vi beskriver i følgende afsnit 13.2.

13.2 Den evolutionære udviklingsmetode

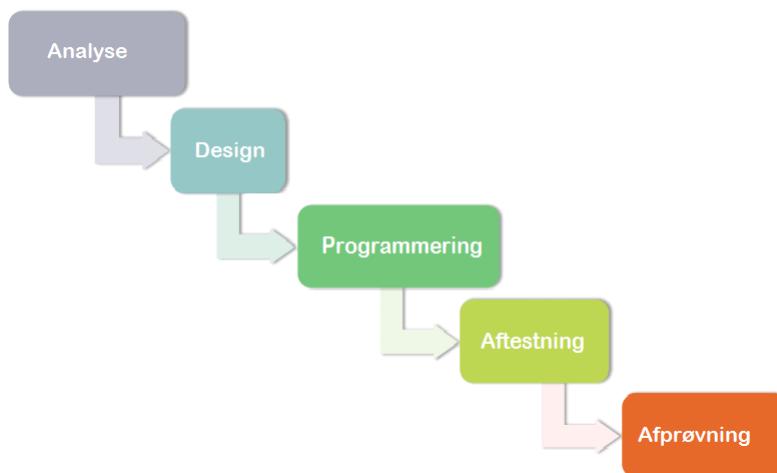
Den evolutionære udviklingsmetode er velegnet i udviklingsprocesser af systemer og tager udgangspunkt i iterationer ved, at et forløb deles op i en række faser, som hver gennemgår samtlige af projektes dele. Et systemudviklingsprojekt kunne eksempelvis være delt op i følgende punkter: analyse, design, programmering, aftestning og afprøvning, som præsenteret i figur 13.1. Figuren er, ligesom figur 13.2, taget fra slides, fra Objektorienteret Analyse & Design [28], og tilpasset til vores eget projektforløb. Deres visuelle stil er genbrugt i figur 13.3.

For hver fase vil der blive tilføjet, fjernet eller på anden vis redigeret i indholdet fra delene, så de hele tiden er opdateret i forhold til det problem, der skal løses. Metoden er specielt anvendelig i forhold til udviklingsprocesser, hvor det givne problem ikke er veldefineret, eller ændres undervejs i forløbet. I forhold til vores egen udviklingsprocess, havde vi i løbet af projektet specificeret et klassediagram, men det endelige klassediagram blev fastlagt i slutningen af projektforløbet, da vi først her var færdige med at fortolke og revidere vores forståelse af problemet. Her kan udviklingsproesen, ved hjælp af den evolutionære metode, nemt ændres og dirigeres henimod det rette problem igen.



Figur 13.1: Figuren illustrerer en fase i den evolutionære udviklingsmetode. En enkelt fase i udviklingsforløbet gennemgår samtlige dele af projektet.

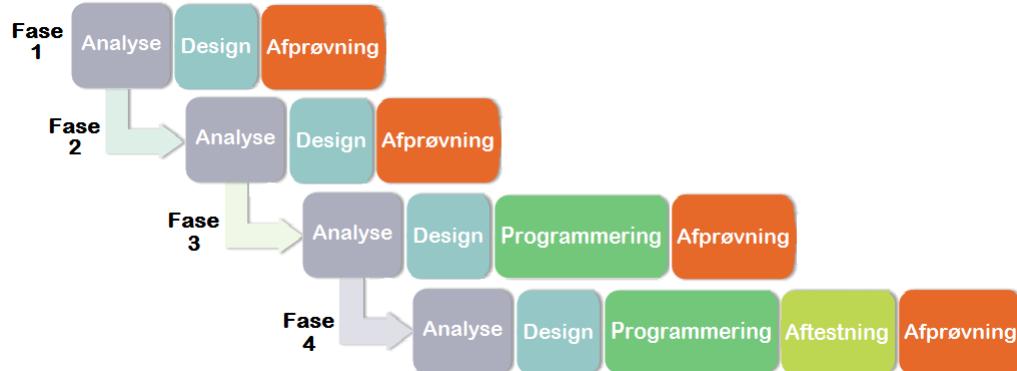
Modsat findes den konstruktive udviklingsmetode (også kaldet vandfaltsmetoden), som har en linier tilgang til udviklingsprocessen. Den konstruktive udviklingsmetode er også delt op i faser, men hver fase er, modsat i den evolutionære metode, fokuseret henimod en specifik projektdel (se figur 13.2). Typisk vil et systemudviklingsforløb med udgangspunkt i den konstruktive udviklingsmetode starte ud med en analyse. Når analysen er færdig, påbegyndes design, og derefter programmering osv. Metoden bruges ofte i systemudviklingsprocesser, hvor det problem, der arbejdes ud fra, er veldefineret og klart. Det er grundet, at metoden ikke, på samme måde som den evolutionære udviklingsmetode, er dynamisk i forhold til, hvis problemet ændrer sig undervejs i forløbet.



Figur 13.2: Figuren illustrerer den konstruktive udviklingsmetode.

En forskel på de to metoder er bl.a. deres måde at interagere med brugere/informanter på. Mens der foregår et

tæt samarbejde med informanter i den evolutionære udviklingsmetode, hvor der anvendes prototyper; har informanterne en mere passiv rolle i den konstruktive metode, hvor de blot godkender beslutninger, og fungerer som ressourcer til information. En anden forskel mellem de to metoder er deres tilgang til det endelig produkt/system. Den konstruktive udviklingsmetode har en meget stringent og langsigtet tidsplan, og man er ikke i tvivl om, hvornår produktet/systemet er færdigt. I en systemudviklingsproces med den evolutionære udviklingsmetode kan det modsat være svært at vurdere, hvornår systemet er færdigt. Det vil altid være muligt at gå igennem endnu en iteration, og lave nye tilføjelser og redigeringer.



Figur 13.3: Figuren illustrerer gruppens udviklingsmetode, som har været en blanding mellem den evolutionære- og den konstruktive udviklingsmetode.

Der er altså store forskelle på de to udviklingsmetoders tilgang til en systemudviklingsproces, og det er derfor op til situationen, hvilken der vil fungere bedst. Det er dog væsentligt at nævne, at det næsten aldrig vil være muligt udelukkende at arbejde ud fra den ene udviklingsmetode. Som regel vil elementer fra begge udviklingsmetoder blive implementeret. I vores projektforløb, anvendte vi også elementer fra begge metoder, men udgangspunktet var den evolutionære udviklingsmetode. Dette var grundet, at problemet som vi fokuserede på, ikke var fuldkommen klart i den initierende del af forløbet. Vi havde kurser, sideløbende med projektet, som gjorde, at vi ikke kunne komme igennem hver eneste projektdel, da vi først skulle introduceres for nogle begreber, i vores kurser omhandlende Systemudvikling [25] og Designing Interactive Systems [4]. Vores udviklingsmetode er illustreret i figur 13.3. Vi delte processen op i fire faser, hvor vi i de to første faser, udelukkende arbejdede med analyse, design og afprøvning ved hjælp af prototyper. Først i tredje fase begyndte vi at programmere og teste systemet, og det samme gjorde vi i fase fire. Den måde vores forløb var iterativt på, var ved at vi i hver fase gik tilbage, og tilføjede, redigerede eller slettede noget i de dele, som vi havde gennemgået i den foregående fase.

Til fremtidige projekter, ville den evolutionære udviklingsmetode bestemt være en metode gruppen ville overveje at bruge igen i lignende forløb. Den tætte interaktion med brugere/informanter, mener vi er essentiel i systemudvikling, fordi det i sidste ende er brugerne, som skal bruge det færdige system, og derfor er det vigtigt at systemet opfylder brugernes behov. Derudover er der det faktum, at metoden er så fleksibel som den er, hvilket gør den mere anvendelig til projektforløb end den konstruktive udviklingsmetode. Dette er grundet, at der sideløbende med projektet, ofte er kurser, som introducerer begreber, der skal implementeres til projektet, hvilket er nemmere med den evolutionære metode. Dette mener vi tilgengæld også er den evolutionære metodes største svaghed, da de konstante tilføjelser og redigeringer, også kræver en revidering af dokumentationen i form af rapporten. Det vil sige, at selv i et projektforløbs afsluttende fase, kan der stadig blive revideret i rapportens analysedel, designdel osv., og det giver en stor udfordring i forhold til at bibeholde en naturlig sammenhæng i rapporten, hvilket også er meget tidskrævende.

14 Ruby on Rails

Ruby on Rails er et framework for programmeringssproget Ruby til udvikling af webapplikationer, som vi har benyttet i dette projekt til at implementere [foodl](#).

Vi traf valget om at arbejde med Ruby on Rails på baggrund af overvejelser omkring hvilket sprog, der ville være nemmeste at implementere systemet med, og hvad vi ville lære mest af. Vi ønskede at udvikle vores kompetencer mht. programmering og webudvikling ved at vælge et sprog, som vi ikke var fuldt bekendte med.

Vi overvejede følgende programmeringssprog og frameworks:

- PHP med CakePHP
- C# med ASP.NET MVC
- Ruby med Ruby on Rails

PHP er anerkendt som et programmeringssprog for webudvikling, men Ruby er mere brugervenligt, og Ruby er nemmere at sætte sig ind i forhold til PHP. Dette er vigtigt, da hver mand i gruppen har forskellige kompetencer. I forrige semester blev vi oplært og udarbejdede et system i C#, men vi valgte at gå videre med Ruby i stedet, fordi vi ønsker at blive bekendte i et nyt programmeringssprog.

Det tager selvfølgelig ekstra tid at skulle sætte sig ind i nye sprog, når man samtidig har nogle overordnede mål, for et projekt, man skal opnå inden en deadline. Udenfor programmeringssproget Ruby, så var der mange essentielle sprog til webudvikling (HTML, CSS, JavaScript og MySQL), hvor mange af gruppemedlemmernes kompetenceniveau ikke var særlig højt. For at mindske oplæringstiden og undgå begynderfejl benyttede vi parprogrammering i starten, hvor vi kombinerede gruppemedlemmer med et godt kendskab til sprogene sammen med gruppemedlemmer med et dårligt kendskab til sprogene. På denne måde kunne vi samarbejde med hinanden og få hjælp, hvis der var tvivl om noget. Dette var en rigtig god metode, der gjorde det muligt for os at arbejde mere effektivt, da vi hurtigt kunne spørge hinanden om hjælp.

Ruby on Rails benytter ‘konvention over konfiguration’ til lynchurtigt at oprette et fungerende system. Det fik Ruby on Rails til at virke “magisk”, hvilket var både positivt og negativt. På den ene side var det hurtigt at få et fungerende system, på den anden side var det til tider svært præcist at vide, hvad der foregik. Det tog derfor længere tid at lære Rails at kende end forventet, fordi vi var nødt til at sætte os ind i de forskellige konventioner. Konventionerne kan se ud til at virke på “magisk” vis, hvis man ikke vidste, hvad der foregik i koden. Rails’ fokus på MVC passede til gengæld godt med vores design, og efter vi havde lært de grundlæggende principper, så hjalp det til at gøre det lettere at implementere vores system.

Tager vi i betragtning, at vi arbejdede med et fremmede programmeringssprog, så kan vi konstatere, at det ikke havde nogen negativ effekt på vores arbejde med systemet. Dette kan måske skyldes, at vi arbejdede sammen i par, hvor den mindre erfarne programmør fik lov til at programmere, mens den mere erfarne kunne hjælpe og dirigere den mindre erfarne, hvis man var ved at skrive noget forkert.

Under udviklingen af systemet oplevede vi stabilitets- og ydelsesproblemer med Rails. Vores server havde kun 1GB RAM, og efter et par dage stoppede Rails med at fungere, fordi der ikke var mere hukommelse tilbage. Vi blev nødt til periodisk at genstarte Rails for at undgå, at systemet gik ned, mens vi ikke holdt øje med det.

Rails køres enten i ‘developement mode’ eller ‘production mode’, hvor den første tillader, at man let kan ændre i kodelfilerne og se ændringerne med det samme, hvor ‘production mode’ er beregnet til et færdigudviklet system,

hvor man ikke har brug for ofte at ændre kodelinjerne. En af forskellene er, at alle JavaScript- og CSS-filer bliver kombineret til en enkelt JavaScript-fil og en CSS-fil i ‘production mode’, hvilket ikke sker i ‘development mode’. For at øge organiseringen af vores kode og for at mindske konflikter i forbindelse med vores system til revisionskontrol, så var vores kode delt op i mange filer. Da alle filer skal overføres individuelt, hver gang man tilgår en side, blev systemet langsomt i ‘development mode’.

Et andet ydelsesproblem var i forbindelse med databasen. Mens vi udviklede på systemet på vores egne computere, valgte vi at tilgå databasen på serveren i stedet for at bruge en lokal kopi på vores egne computere. Vi valgte at gøre det på denne måde for at være sikre på, at alle arbejdede på de samme data. Vi oplevede, at systemet kørte meget langsommere på vores egne computere, da systemet i så fald skulle forbinde til en server, der ikke var lokal. Problemet var specielt omfattende på søgeresultatsiden, der udførte mange SQL-forespørgsler. Når hver forespørgsel skulle sendes over internettet, tog det cirka 150 millisekunder at udføre hver forespørgsel.

På søgeresultatsiden blev der udført så mange forespørgsler, at det tog næsten 1,5 minut for siden at blive genereret og yderligere et halvt minut for at indlæse alle JavaScript- og CSS-filerne. Dette betød, at udviklingen af netop den side tog længere tid, da man skulle vente 2 minutter for at se effekten af hver eneste lille ændring man lavede. Dette var specielt problematisk i forhold til det visuelle design, da man ofte justerer størrelser, farver og lignende, og hele tiden skal evaluere forskellen. Selvom det ikke var nærliggende at få løst, så det blev mere effektivt.

I fremtiden vil vi have en lokal kopi af databasen på vores computere og tilgå denne i stedet. På trods af at vi fandt og rettede ydelsesproblemet, så påvirkede det vores produktivitet meget. Yderligere giver det også den fordel, at vi kan arbejde på systemet uden at være forbundet til internettet, så man f.eks. kan arbejde på systemet, mens man sidder i toget.

Selvom situationen blev bedre efter, at serveren blev sat til at køre i ’production mode’, har det efterladt os kritiske over, hvorvidt Rails kan benyttes i et system, hvor effektivitet er meget vigtigt.

Ser vi på det store billede, så var udbyttet af at arbejde med Ruby on Rails i forhold til de problemer, som vi oplevede ved brugen af Rails, meget godt. Der var nogle problemer, men vi fik så meget ud af Rails, at vi stadig er glade for Rails. Dette betyder, at vi gerne vil benytte Rails i fremtiden for webudvikling. Men ønsket om at lære alternative programmeringssprog vejer tungt og kan resultere i valget af et andet system.

Litteratur

- [1] Patterson Aaron, Dalessio Mike, Harada Yoko, and Elliott Tim. Ruby nokogiri gem. <http://rubygems.org/gems/nokogiri>, December 2012. (set 25.11.2012).
- [2] Alexa. Siteinfo google.com. <http://www.alexa.com/siteinfo/google.com>, November 2012. (set 17.12.2012).
- [3] Alletiders Kogebog. Dk-kogebogen. <http://www.dk-kogebogen.dk/>, Oktober 2012. (set 30.10.2012).
- [4] David Benyon. *Designing Interactive Systems - a comprehensive guide to HCI and interaction design*. Pearson Education Limited, second edition edition, 2010. ISBN: 978-0-321-43533-0.
- [5] D303E12. Møde 1 merete.
https://www.dropbox.com/s/zz34nlbid9wq48h/interview_informant_merete.amr.
- [6] D303E12. Møde 2 keld.
https://www.dropbox.com/s/8phem942nkgvdxa/interview1_informant_keld.mp3.
- [7] D303E12. Møde 2 merete.
https://www.dropbox.com/s/qtvlx1ty33ah5me/interview_informant_merete2.amr.
- [8] D303E12. Prototypbe 1b merete. <http://www.youtube.com/watch?v=rUJexwTpu48>.
- [9] D303E12. Prototype 1a merete. <http://www.youtube.com/watch?v=E-8WA6QrZo4.com>.
- [10] D303E12. Prototype 2 keld. <http://www.youtube.com/watch?v=kjry9p4Cu7M>.
- [11] D303E12. Prototype 2 merete. <http://www.youtube.com/watch?v=Nb8dT PgzzXc>.
- [12] D303E12. Usabilitytest keld. <http://www.youtube.com/watch?v=m3tcy96dY5g>.
- [13] D303E12. Usabilitytest merete. <http://www.youtube.com/watch?v=lwoK04ae5oc>.
- [14] Lars Dahlbom, Bo og Mathiassen. *Computers in context : the philosophy and practice of systems design*. Wiley-Blackwell, august 1993.
- [15] DK-Kogebogen. Indtast opskrift.
<http://www.dk-kogebogen.dk/opskrifts-service/indsend-opskrift.php>, november 2012. (set 08.11.2012).
- [16] Wayne W. Eckerson. Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Information Systems*, 10(1), 1995.
- [17] Erich Gamma et al. *Design Patterns: elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley, 1995. <http://www.worldcat.org/title/design-patterns-elements-of-reusable-object-oriented-software/oclc/31171684>.
- [18] Dan Farber. Google's marissa mayer: Speed wins.
<http://www.zdnet.com/blog/btl/googles-marissa-mayer-speed-wins/3925>, November 2006. (set 23.10.2012).
- [19] Forbrugerrådet. For Resten. <http://forresten.taenk.dk/>, August 2012. (set 30.10.2012).

- [20] Landbrug & Fødevarer. Madspild koster danskere 16 milliarder. *Ingen*, September 2010.
<http://www.lf.dk/Aktuelt/Nyheder/2010/September/Madspild.aspx#.UFHTDqQ737E>.
- [21] Jenny Gustavsson, Christel Cederborg, Ulf Sonesson, Robert van Otterdijk, and Alexandre Meybeck. Global food losses and food waste. *Interpack2011*, page 30, 2011.
- [22] Jytte Hasselriis. Kommentar. <http://opskrifter.dk/Toem-koeleskabet.149.0.html>, marts 2012. (set 08.11.2012).
- [23] M. B. Skov og J. Stage J. Kjeldskov. Instant data analysis: Evaluating usability in a day. Technical report, Aalborg Universitet, 2004.
- [24] Kalid. Intermediate rails: Understanding models, views and controllers. <http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/>, december 2012. (set 06.12.2012).
- [25] Peter Axel Nielsen og Jan Stage Lars Mathiassen, Andreas Munk-Madsen. *Objektorienteret Analyse og Design*. Forlaget Marko ApS, Aalborg, Danmark, 3. edition, 2001. ISBN 87-7751-153-0.
- [26] Legro a/s og Wagawaga. Opskrifter.dk. <http://opskrifter.dk/>, Oktober 2012. (set 30.10.2012).
- [27] Madopskrifter.nu. Mest anvendte ingredienser.
<http://www.madopskrifter.nu/Ingredienser.aspx>, november 2012. (set 06.12.2012).
- [28] Andreas Munk-Madsen Peter Axel Nielsen og Jan Stage Lars Mathiassen. Konstruktion, evolution og prototyping. Slides. (set 13.12.2012).
- [29] Politiken. Danskere smider 42 kilo god mad ud om året. <http://politiken.dk/tjek/tjekmad/produktion/ECE1636012/danskere-smider-42-kilo-god-mad-ud-om-aaret/>, Maj 2012. (set 14.09.2012).
- [30] Jon Resig. jquery: write less, do more. <http://jquery.com/>, december 2012. (set 05.12.2012).
- [31] Christian Gram Rolf Molich. Et øjebliksbillede af brugbarheden af ni store danske net-steder ("web-sites"). 1998.
- [32] Ruby. Ruby-dokumentation. <http://www.ruby-doc.org/core-1.9.3>, December 2012. (set 11.12.2012).
- [33] Dee Simmons. Comment on Tristam Stuarts talk. <http://www.ted.com/profiles/1551637>, November 2012. (set 03.12.2012).
- [34] Danmarks Statistik. Husstande.
<http://www.dst.dk/da/Statistik/emner/husstande-familier-boern/husstande.aspx>, Oktober 2012. (set 18.10.2012).
- [35] Tristam Stuart. The global food waste scandal.
http://www.ted.com/talks/tristram_stuart_the_global_food_waste_scandal.html, May 2012. (set 08.11.2012).
- [36] TJA. Under middel. <https://play.google.com/store/apps/details?id=com.nodes.forresten&reviewId=17644663083364936688>, Oktober 2012. (set 30.10.2012).
- [37] Erik Veenstra. Ruby levenshtein gem. <http://rubygems.org/gems/levenshtein>, December 2012. (set 25.11.2012).
- [38] Sinclair J. Vincent J., Waters A. *Software quality assurance : volume I practice and implementation*, volume 1. Prentice-Hall, Englewood Cliffs, New Jersey, 1st edition, 1990. 0-13-821860-9.

-
- [39] Wikipedia. Active record pattern. http://en.wikipedia.org/wiki/Active_record_pattern, November 2012. (set 05.12.2012).
 - [40] Wikipedia. Ajax (programming). [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)), November 2012. (set 25.11.2012).
 - [41] Wikipedia. Cascading style sheets. http://en.wikipedia.org/wiki/Cascading_Style_Sheets, November 2012. (set 25.11.2012).
 - [42] Wikipedia. Html. <http://en.wikipedia.org/wiki/HTML>, November 2012. (set 25.11.2012).
 - [43] Wikipedia. Implementation of LongestCommonSubstring. http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring, December 2012. (set 04.12.2012).
 - [44] Wikipedia. Javascript. <http://en.wikipedia.org/wiki/JavaScript>, November 2012. (set 25.11.2012).
 - [45] Wikipedia. jquery ui. http://en.wikipedia.org/wiki/JQuery_UI, November 2012. (set 25.11.2012).
 - [46] Wikipedia. Json. <http://en.wikipedia.org/wiki/JSON>, November 2012. (set 25.11.2012).
 - [47] Wikipedia. Levenshtein. <http://www.levenshtein.net/>, December 2012. (set 10.12.2012).
 - [48] Wikipedia. Mysql. <http://en.wikipedia.org/wiki/Mysql>, November 2012. (set 25.11.2012).
 - [49] Wikipedia. Ruby on rails. <http://en.wikipedia.org/wiki/Rubyonrails>, November 2012. (set 25.11.2012).

Bilag

A Fravalgte klasser og hændelser

A.1 Fravalgte klasser

Herunder ses de fravalgte klasser, som gruppen ikke fandt relevante i forhold til problemområdet. De listes her, da der har været meget diskussion, om hvorvidt klasserne skulle med i systemet eller ej.

Husholdning

En husholdning repræsenterer et hjem, som indeholder én til flere personer. Det er ikke en del af problemområdet at holde styr på eller at kommunikere med andre husstande.

Køkken

Køkken og husholdning dækker over samme del af problemområdet, og vi fjerner derfor også køkken, af samme grund som vi fjernede husholdning.

Køleskab/skab/opbevaringsskab

Om råvarene befinner sig i et køleskab eller i en skuffe er, for os, uinteressant, derfor er vi ikke interesseret i at modellere disse skabe som en klasse.

Køkkenredskab/komfur

Ligesom ved køleskab/anden opbevaring er vi ikke interesseret i at modellere hvilke redskaber husholdningen har adgang til.

Service

Vi har valgt at fjerne klassen service (bestik), da service er noget, der bliver brugt når man er i færd med at spise maden, og ikke under selve madlavningen.

Butik

Vi har valgt at fjerne klassen Butik, da vores system fokuserer på madspild, og ikke madindkøb. En modellering af butikker ville være relevant, hvis vores fokus lå på at begrænse udgifter på mad, men da dette ikke er situationen i problemområdet, bliver den udeladt.

Typisk/atypisk ingrediens

Det er ikke en del af problemområdet at folk ikke er klar over hvilke ingredienser der er normale/unormale at have.

Enhed/Mængde

Enhed og mængde er ikke klasser, men attributter til en ingrediens.

Madplan

Informanterne syntes ikke at en madplan var særlig nødvendig. Den indgik i systemdefinition S2, som informanterne fravalgte. Madplanen anses derfor som overflødig, hvorfor denne fjernes som klasse.

A.2 Fravalgte hændelser

De hændelser som vi har fravalgt ses herunder. De fravalgte hændelser har en kort forklaring, der beskriver hvorfor hændelsen er blevet fravalgt.

A. Fravalgte klasser og hændelser

- Køkkenredskab benyttet (systemet skal ikke holde styr på køkkenredskaber)
- Råvare benyttet (systemet skal ikke holde styr på mængden af råvarer hos bruger)
- Mæthed opnået (fra fravalgte klasser: bruger, person)
- Madrest opstået (systemet skal ikke behandle råvarer forskelligt om det er rester eller ej)
- Bord opdækket (fra fravalgt klasse: husholdning)
- Opvask taget (fra fravalgt klasse: køkken)
- Service benyttet (fra fravalgt klasse: service)
- Køleskab åbnet (fra fravalgt klasse: opbevaringsskab)
- Køleskab lukket (fra fravalgt klasse: opbevaringsskab)
- Opskrift vurderet (opskrift valgt indebærer, at man har vurderet opskriften)
- Opskrift anmeldt (ikke en del af problemområdet at anmeldte opskrifter)
- Sult opstået (fra fravalgte klasser: bruger, person)
- Madlavning afsluttet (fra fravalgte klasser: bruger, person)
- Madlavning påbegyndt (fra fravalgte klasser: bruger, person)
- Råvare identificeret (overvåges i form af “råvare købt”-hændelsen med samme resultat)
- Ingrediens identificeret (indgår i hændelsen opskrift valgt)
- Madplan lagt (fra fravalgt klasse: Madplan)
- Madplan startet (fra fravalgt klasse: Madplan)
- Madplan afsluttet (fra fravalgt klasse: Madplan)
- Opskrift fravalgt (fra fravalgt klasse: Madplan)

B Fravalgte brugsmønstre

Skalering

Skalering af ingredienser i opskrifter i forhold til antallet af personer. Fjernet, da skalering indgår i søgningen, netop fordi der ikke er nogen grund til at skalere opskriften i hver visning og kan dermed indgå som en slags ”filter” i søgningen.

Råvarehåndtering

Holde styr på hvilke råvarer man har i sit køleskab. Fjernet, da råvarehåndtering indgår i søgningen.

Overvågning

Vi mener ikke, at vi har behov for at overvåge noget og har derfor fjernet dette brugsmønster.

Begrænsning

At tilføje en begrænsning, så man kun får vist opskrifter uden gluten, uden svinekød, m.m. Fjernet, da dette hører under søgning.

Sortere

At sortere opskrifter efter anmeldelse (antal stjerner), navn eller lignende. Fjernet, da dette hører under søgning.

Madplanlægning

Fjernet, da klassen madplan ikke er en del af problemområdet.

Synkronisering

I stedet for et loginsystem kunne vi benytte cookies og tilbyde muligheden for at synkronisere flere enheder, så de er tilknyttet hinanden. Således at en ændring på indkøbslisten på én enhed betyder at samtlige enheder kan se ændringen. På baggrund af det fjerde møde med informanterne, hvor vi afprøvede prototyper, er det gjort klart, at vi skal benytte et login-system i stedet, og fjerner derfor dette brugsmønster.

C Prototyper og møder med informanter

C.1 Møde 1

Møde 1 med Merete er blevet optaget[5].

Formål Igennem et semistruktureret interview med vores to informanter, ønsker vi opnå viden omkring, hvilke problemer informanterne har, i forbindelse med madlavning i det private. På baggrund af denne viden vil vi lave en eller flere systemdefinitioner, som beskriver et eller flere systemer, som vi forventer vil kunne løse disse problemer.

Formål Igennem et semistruktureret interview med vores to informanter, fik vi opnået viden omkring hvilke problemer informanterne havde i forbindelse med madlavning i det private. På baggrund af denne viden har vi lavet to systemdefinition (beskrevet i afsnit 3.3.1), der beskriver de systemer vi forventer vil kunne løse disse problemer.

Spørgsmål Informanterne blev stillet følgende spørgsmål.

- Hvad gør du for at undgå madspild?
- Hvordan planlægger du dine indkøb? Hvordan foregår de?
- Hvordan finder du ud af, hvad du skal spise til aftensmad?
- Gør du noget for at spise varieret? (Hvordan/hvorfor ikke?)
- Beskriv hvordan I i jeres husstand håndterer madlavningen?
 - Hvem laver mad?
 - Hvem bestemmer hvad I skal have?
 - Hvem køber ind?
 - Hvor tit handler I ind?
- Føler du, at du smider meget mad ud?
- Laver du en madplan? (hvordan/hvorfor ikke?)
 - Hvad skal der til for, at du vil anvende en madplan?

Møde med informant Merete Merete bor med sin mand og det er hende, der bestemmer, hvad for noget mad de skal have til aften. Det er altid hende der laver den, men nogle gange kommer manden dog hjem med takeaway.

Merete føler at hun smider meget mad ud. Maden bliver smidt ud når den bliver for gammel, da hun er meget opmærksom på holdbarhedsdatoer. En anden grund til madspild, er at hun før i tiden, har været vant til at lave mad til en hel familie, dengang hendes to sønner og datter boede hjemme, men nu er der kun hende og hendes mand tilbage i husstanden, hvilket har været svært at vænne sig til. Derfor bliver der lavet for store portioner. Hvis mad bliver tilovers, bruges det ofte i en sammenkogt ret næste dag (eksempelvis supper, biksemad osv.).

Merete bruger ikke madplaner af flere grunde. Med en madplan føler de ikke de får særlig meget mad for pengene, da der ikke er nogle madplaner, som tager højde for gode tilbud. Samtidig har Merete og hendes mand et job, som gør at deres planer ofte ændrer sig, og når de er på farten, er det let selv at lave mad, eller at tage højde for en madplan. Hvis Merete skulle bruge en madplan skulle den være baseret på hvad hun har i køleskabet, i en kombination med supermarkedernes gode tilbud. "Mad leveret til døren"-tilbud fungerer ikke for hende, da hende og manden som før nævnt ofte er på farten, og deres planer ofte ændrer sig

impulsivt. Både Merete og hendes mand kan stå for at handle ind. De planlægger sjældent indkøbet, men kan bedre lide at gå på opdagelse efter gode tilbud i forretningen. Merete gør ingenting for at spise varieret, da hun synes det tager for lang tid at tage højde for at få kosten til at blive varieret.

Møde med informant Keld Keld bor med sin kone og to små døtre i en hyggelig lille lejlighed. Han bestemmer, hvad familien skal have at spise hver aften, og han køber ind og laver al maden til familien. Når der bliver lavet mad, så bliver portionen som regel lavet så står, at der er nok til to dage. Familien ønsker nemlig ikke at lave mad hver aften, da der ikke er meget tid i hverdagen. De får oftest spist hele portionen i løbet af to dage, men hvis der bliver noget ekstra tilovers, fryser de den ned, så der bliver smidt så lidt mad ud som muligt. Keld planlægger oftest aftensmad to til tre dage i fremtiden. Når der skal handles ind, så bruges der en indkøbsliste. Dvs., at de har en liste klar, når der skal handles ind. Der kommer oftest også andre sager med i indkøbskurven, fordi de er nemme ofte for impulskøb. Keld handler ca. tre til fire gange om ugen.

Der bliver sjældent spist varieret mad. Det er oftest de samme "almindelige" og hurtige danske retter, fordi han har erfaring med disse og mener, at det er nemt at lave dem. Dvs., at tid er en vigtig faktor, når det kommer til familiens aftensmad. Dog kan det hænde, at der eksperimenteres med nye retter, men dette sker kun i weekenden, når der er lidt ekstra tid.

Madplan er ikke noget, som de bruger, fordi Keld normalvis har en plan i hovedet, som han går efter. Resten af familien har ikke den store indflydelse på madlavningen. Han kommenterer dog, hvis han skulle bruge en madplan, så skulle denne have opskrifter, der ikke tog meget tid, og hvor ingredienserne var håndgribelige. Med dette menes der, at ingredienserne ikke skulle være alt for forskellige, da man pga. af dette ville have sværere ved at mindske sine madrester fra aftensmaden før. Derudover skulle der være nogle billeder til hver opskrift, så man kunne få et hurtigt indblik i hvordan maden skal se ud, og på den måde kan man se, om en opskrift er noget for en.

C.1.1 Sammendrag

Vi har nu hørt om to informanders erfaringer inden for privat madlavning. De to informanter har det til fælles, at de begge oplever madspild og ikke benytter en madplan. Netop fordi ingen af informanterne benytter en madplan, kan det være at det blot er en sådan der skal til for at mindske deres madspild. Det kan også være at ingen af dem benytter en madplan fordi de simpelthen ikke kan overtales til dette. For nærmere at undersøge hvordan vi skal løse problemet med madspild, konstruerer vi to forskellige systemdefinitioner ud fra mødet med informanterne. Begge systemer forsøger at mindske madspild. Systemdefinition S1 benytter en løsning der ikke involverer en madplan, mens systemdefinition S2 netop benytter en madplan.

C.2 Møde 2

Møde 2 med Merete er blevet optaget [7].

Møde 2 med Keld er blevet optaget [6].

Formål For at kunne begynde at modellere anvendelsesområdet, har vi behov for mere viden om informanternes tanker omkring systemet. En sådan viden vil vi gerne opnå igennem dette møde. Informanterne præsenteres for systemdefinitionerne S1 og S2, hvorefter vi gerne vil høre hvilket system de ønsker at vi skal udvikle. Derefter har vi et par ret lukkede spørgsmål omkring måden de vil bruge systemet på og hvilke funktioner der er nødvendige for at opfylde deres behov.

Huskeliste Listen herunder er en huskeliste for gruppen, når vi skal tale med informanterne.

- Præsenter systemdefinition S1 og S2 for informanten

- Hvor ville du bruge et sådan system? Bærbar, iPhone, iPad, stationær?
- Skal programmet kunne huske ingredienser til næste gang, og hvilke ingredienser? Vil du fjerne de ingredienser du bruger under madlavningen?
- Hvilke opskrifter skal der findes? Forretter, hovedretter, desserter, m.m?
- Hvordan skal opskrifter sorteres?
- Retter med eller uden billeder?
- Andre forslag til programmet?

Noter fra møde med Merete Merete mener hun helt sikkert ville bruge et program i stil med det nævnt i vores systemdefinition 1. Systemdefinition 2 siger hende ikke rigtig noget. Hun er ret sikker på hun ikke vil bruge en madplan. Hun kan godt lide at have frihed til at lave det hun lige har lyst til. Systemdefinition 1 mener hun vil være nytigt til at få brugt alle resterne i køleskabet på en smart måde, så de ikke skal smides ud. Hun vil primært bruge programmet på sin bærbar. Hun ville også bruge programmet selv om hun ingen rester havde, for at få gode idéer til retter hun kan lave. Køleskabet skal ikke holde styr på ens varelager, hun vil kun bruge programmet inden madlavningen, og ikke efter. Det vil blive for bøylet hvis man hver gang efter madlavning skal huske at fjerne ingredienser fra programmet. Man har nok at gøre med at rydde op efter maden. Første gang man bruger programmet skal man kunne indtaste alt hvad man har, også krydderier og mælk. Med en ”husk mig”knap, kan man gemme de ingredienser, man ikke vil skrive på hver gang. Programmet skal kun finde almindelige hverdagsretter, ikke desserter, morgenmad og så videre. Man skal kunne vælge tilberedningstid. Hvis hun har travlt vil hun ikke have foreslægt retter der tager halvanden time at lave. Merete foreslår 3 knapper: 0 - 30 min, 30 - 60 min, > 60 min En knap at sætte flueben i ”Vis mig kun retter uden kød”, ville være rar. Programmet skal sortere opskrifter efter ”dem man kan lave”, så ”dem man mangler 1 ting til”, ”2 ting til”, osv. Inden for hver af de netop nævnte lister ville det være rart at kunne sortere efter kalorier, popularitet og tilberedningstid. Vis kun retter med billeder.

Noter fra møde med Keld Jeg forklarede hvordan systemet vil virke efter systemdefinition S1 og S2 og stillede følgende spørgsmål:

- Hvad systemdefinition kan du bedst lide? Systemdfinition S1.
- Hvor ville du bruge sådan et system? Bærbar, mobil, tablet, stationær
- Skal systemet kunne huske ingredienser til næste gang? Ja
- Hvilke opskrifter skal der findes? forretter, hovedretter, dessert? Jeg laver normalt ikke forretter og dessert, når vi bare er derhjemme - det er kun, når vi får besøgende
- Hvordan skal opskrifter sorteres? Årstiden, mængde af passende ingredienser
- Andre forslag til programmer?
 - Kommentarer på opskrifter
 - Ingrediensmængdeberegning i forhold til antal personer
 - Printfunktion, så man kan få det på papir
- Retter med eller uden billeder? med billeder!

Han lavede tit ensformig mad, når han har tid, kan noget nyt godt laves vha. inspiration fra kogebøger samt internetsider. Info om energi/kulhydrater/vitaminer/mineraler osv. betyder ikke så meget for ham, så længe billedet og opskriften ser lækker og inspirerende ud. Krydderier er ikke vigtige, når man søger på opskrifter.

C.2.1 Sammendrag

Enighed blandt begge informanter:

- Systemet bruges på en bærbar
- Programmet skal fokusere på hovedretter

- Der skal vises billeder af opskrifterne
- Opskrifterne skal sorteres efter mængde af passende ingredienser
- Opskrifter skal kunne skaleres i forhold til personer
- Systemet skal kunne huske ingredienser til næste søgning

Foreslæt af enkelt informant:

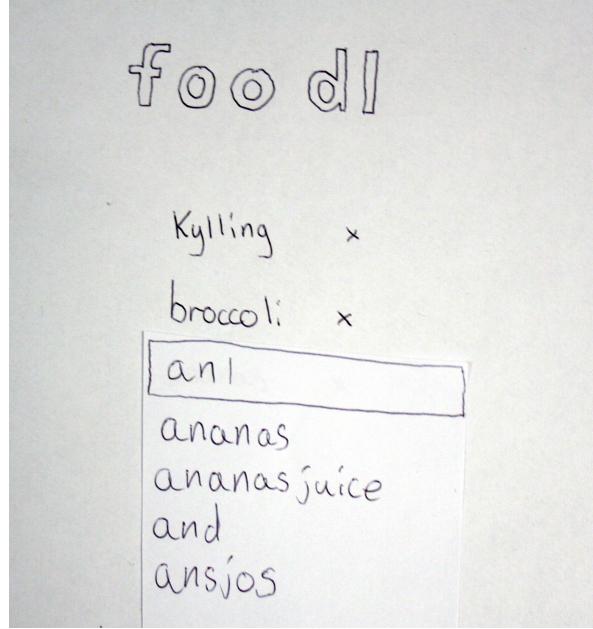
- Kommentarer på opskrifter
- Udprinte opskrifter
- Sorter opskrifter efter årstid, kalorier, tilberedningstid
- Krydderier er ikke vigtige når man søger på opskrifter

C.3 Prototype 1

Afprøvning af Prototype 1A på Merete er blevet filmet[9].

Afprøvning af Prototype 1B på Merete er blevet filmet[8].

Formål Vores system kan ikke benyttes uden at brugeren indtaster en mængde råvaretyper, som de vil udføre en søgning på. For at tilbyde en brugervenlig metode til indtastning af disse råvaretyper vil vi gerne teste 2 forskellige metoder på informanterne. Disse 2 metoder testes med hver deres prototype i papirsform, prototype 1A og 1B.

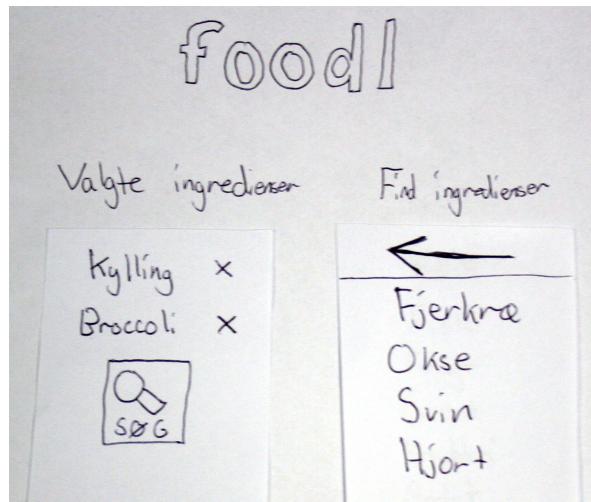


Figur C.1: Visualisering af prototype 1A.

Prototype 1A Præsenterer en søgeboks for brugeren, der minder meget om Google's søgefelt. Når man indtaster et bogstav, fx "k", kommer der en række forslag frem, såsom kylling og kartoffel, også på samme måde som ved Google, blot med den forskel at der kun foreslås råvaretyper. Man kan nu klikke på forslaget eller trykke enter. Man kan også skrive råvaretypens navn færdig manuelt.

Tanken bag denne metode er at man hurtigt kan indtaste en råvaretype hvis man blot ved hvordan de første få bogstaver staves. Brugerne har med stor sandsynlighed kendskab til denne metode, da den bruges af Google, og samtidig har vi også konstrueret logoet og sidens design så det også minder om Google, netop for at gøre det intuitivt for brugerne.

Ulempen er at man har brug for et tastatur og skal tænke over hvordan man staver til råvaretypen. Det er også muligt at overse forslagene og tro man er nødsaget til at stave et meget langt ord, som for eksempel



Figur C.2: Visualisering af prototype 1B.

Prototype 1B Fokuserer på et valg af råvaretyper blandt kategorier. Man vælger først en bred kategori, som for eksempel fjerkræ, kød, brød, frugt og grønt. Dernæst vælger man et antal gange en underkategori, indtil man til sidst kan vælge en råvaretype fra en liste.

Fordelen ved denne metode er at brugerne ikke har behov for et tastatur. Det er også nemt at benytte på tablets og smartphones, da der kan skal klikkes. Ulempen er muligheden for mange kategorier og forvirring omkring hvilken kategori en råvaretype findes i. Måske vil den sidste kategori der vælges stadig indeholde rigtig mange ignredienser, sådan at man skal bladre i denne liste for at finde den ønskede råvaretype.

Sammendrag Prototype 1A var hurtig, nem og effektiv. Informanten kunne bedst lide denne metode, og hun havde ikke brug for vejledning for at kunne finde de 3 råvaretyper: kylling, ananas og broccoli.

Prototype 1B var langsommere at bruge og informanten syntes ikke om den. Hun var i tvivl om hvilken kategori hun skulle vælge kylling under. Kategorierne kan laves på mange forskellige måder, og uanset hvordan de vælges, vil der med garanti være nogle brugere der er i tvivl om hvor de skal lede efter en bestemt råvaretype. Et eksempel kan være kartoffelstivelse. Nogle vil lede efter kartoffelstivelse i kategorien grøntsager (fordi kartofler findes der), mens andre måske vil lede efter en kategori med navnet brød og gryn.

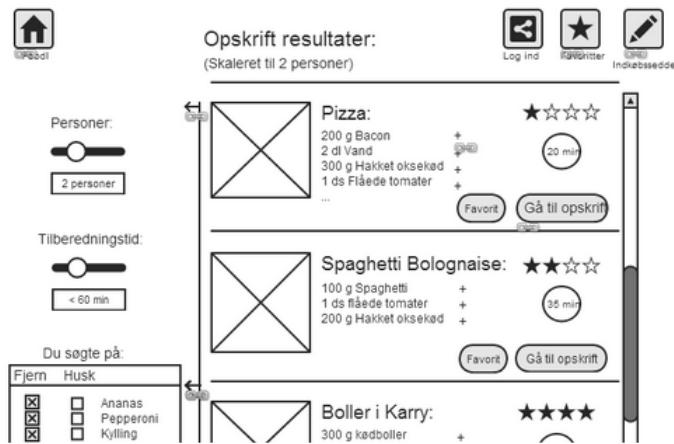
På baggrund af informantens valg, vælger vi at benytte metoden fra prototype 1A til at vælge råvaretyper.

C.4 Prototype 2

Afprøvning af Prototype 2 på Merete er blevet filmet[11].

Afprøvning af Prototype 2 på Keld er blevet filmet[10].

Formål På baggrund af møde 2, hvor informanterne kom med krav til systemets funktioner, har vi nu lavet en diashow-prototype på gomockingbird.com, hvor systemets funktion er vist. Når informanterne præsenteres for funktionerne i noget der, med lidt god vilje, ligner en rigtig hjemmeside, så kan det være at informanten bliver klar over at en funktion enten mangler, eller at en tidligere foreslægt funktion er overflødig. Formålet med mødet er derfor primært at ud af, om der er de funktioner, som informanten har brug for. Derudover vil vi også gerne finde ud af om brugeren kan finde de nødvendige funktioner, altså om programmet og dets funktioner som helhed er intuitive at bruge for informanterne.



Figur C.3: Visualisering af prototype 2.

Først udføres en case for at få ledt brugeren rundt blandt alle funktionerne.

Case Følgende case blev udført af informanterne.

1. Udfør en søgning på ingredienserne (pepperoni, ananas og kylling). Ingredienser tilføjes ved at klikke i søgefeltet
2. Skjul alle opskrifter med nødder
3. Gå til den første opskrift, der fremkommer
4. Gå tilbage og tilføj 200 g Bacon (fra pizzaens ingredienser) til din indkøbsliste
5. Print indkøbslisten ud
6. Gør klar til en helt ny søgning
7. Foretag en søgning på pepperoni
8. Du fandt ingen opskrifter, så du vil gerne tilføj ananas, inden du søger igen (du søger altså på pepperoni og ananas)
9. Føj den første opskrift, du finder, til dine favoritter

Efter casen tages en snak om hver af disse funktioner og muligheder med systemet.

Informanten bliver præsenteret for mange funktioner, hvor vi her beskriver hvad deres mening var omkring disse funktioner efter at have udført casen.

- Begrænse søgeresultat efter tilberedningstid
 - Merete synes idéen er god, men foreslår nu kun 2 valgmuligheder "kort" eller "lang" tilberedningstid
 - Keld synes også dette er en god ide. En opdeling på en 30 min. burde være fint, måske 15 min. Hvis det er tilberedningstid på over en time, må skalaen godt springe mere end 15-30 min.
- Sidebar
 - Merete opdagede ikke at denne sidebar kunne trækkes ud

- Den ligger ikke logisk for en. Selvom den er stor. Den bliver skjult i designet
- Keld synes ikke det ser for rodet ud, selvom sidebaren er ude hele tiden
- Skalere en opskrift til x personer
 - Merete ser det som en nyttig funktion, hun vil skalere til mellem 2-4 personer
 - Det er en god funktion, som Keld er sikker på mange vil få brug for. Opskalering til 5 burde være nok. Måske 1-20, så er gæstebehov også dækket ind. Men det er trods alt til rester, så til 5 personer burde være nok.
- Fjerne ingredienser inden en søgning udføres (på forsiden)
 - Merete synes det er meget brugbart
 - Keld synes det er fint at det er på forsiden
- Fjerne ingredienser efter en søgning er udført (på søgeresultatsiden)
 - Merete synes idéen med at kunne fjerne ingredienser mens der vises søgeresultater er god
 - Keld synes det er fint, at det også er muligt i sidebaren
- Huske ingredienser til næste søgning
 - Merete synes måden det fungerer på i prototypen er god. Hun vil ikke have at de huskede ingredienser vises på forsiden
 - Keld synes det er rart at det er muligt at huske nogle ingredienser. Det ville måske også være rart, hvis det også var muligt at gøre fra forsiden. Keld er dog i tvivl om, hvor på forsiden det skulle være. Det er måske alligevel bedst hvis forsiden er simpel. Han synes funktionen er brugbar
- Skjule opskrifter indeholdende bestemte ting
 - Merete synes det virker godt
 - Keld synes det er en god ide. Han fik hurtigt fundet funktionen, så snart toolboxen blev åbnet
- Browsers tilbageknap går til forsiden (beholder ingredienser)
 - Merete opdagede ikke denne funktion
 - Keld opdagede funktionen, og anvendte den også. Måske ville en tilbage- og fremknop på selve siden være brugbar, foreslår Keld
- Home knap går tilbage (fjerne ingredienser)
 - Merete synes det virkede naturligt
 - Keld synes det er godt at have en knap som går helt tilbage. Men han foreslår endnu engang at have en frem- og tilbageknap som supplerer home-knappen
- Visning af opskrifter (ekspander ved mouse over)
 - Merete synes opskrifterne blev vist fint. Hun kunne godt lide idéen med at ekspandere opskriften ved mouseover
 - Keld synes bare man skal have vist de væsentligste ingredienser. Han synes det ville være smart, hvis det var muligt at se hele ingredienslisten ved hjælp af et mouse-over
- Gå til opskrifter (evt link ved klik på navn)
 - Merete synes ikke knappen "Gå til opskrift", er overflødig. Hun ville blive forvirret hvis den ikke var der, og man bare skulle trykke et sted på opskriften (navnet, eller baggrunden, hvor baggrundsfarve ændrer sig eller lignende)
 - Keld synes det er godt med en "Gå til opskrift"-knap. Det er brugervenligt
- Tilføj opskrift til favoritter
 - Merete synes ikke man skal gå fra et søgeresultat og over til favoritsiden hver gang man tilføjter en opskrift til favoritter. Opszriften skal blot tilføjes. Måske den skal sige en lyd og fjerne knappen man trykkede på. Favoritter-ikonet kunne lyse op
 - Skal lave en "Fjern fra favorit-knap", så man hurtigt kan ombestemme sig
 - Keld synes det er smart nok at man sendes ind på favoritlisten, så man er sikker på at opszriften er kommet derind. Havde man samtidig en frem- og tilbageknap ville det være endnu bedre
- Favoritter (knappen, der viser ens favoritter)

- Merete var lidt forvirret med hensyn til om den tilføjede en opskrift til favoritter, eller hvad den gjorde
- Keld synes at det skal være muligt at skalere opskrifterne på favoritsiden og at det er muligt nemt at fjerne opskrift fra favoritsiden igen
- Visning af favoritter (layout)
 - Merete syntes layoutet var godt, og kunne godt lide at det mindede om layoutet ved visning af søgeresultat
 - Keld synes godt om layoutet
- Visning af indkøbsliste
 - Merete synes det er en god idé at man kan tilføje tekst
 - Keld foreslår at man har ingredienslisten fra den opskrift man har været inde på, ved siden af indkøbslisten, så det er muligt hurtigt at tilføje flere ingredienser derfra
 - Keld synes indkøbslisten er meget brugbar. Det er for ofte man glemmer nogle ting, uden indkøbslisten
- Tilføje opskrifts ingredienser til indkøbsliste
 - Merete vidste ikke hvordan man gjorde. Hun troede ikke man kunne trykke på +'et
 - Keld kunne godt tilføje en ingrediens til indkøbslisten
- Home-knappen sender en til en helt tom forside
 - Merete kunne godt lide dette. Det virkede helt naturligt for hende
 - Keld synes det giver mening
- Logge ind (få afklaret med brugerens, hvordan det skal foregå)
 - Log ind mest forståeligt
 - Kender meget til login, intet til synkronisering
 - Log ind er klart mest forståeligt for Keld. Keld har ikke lyst til at indtaste mere end mail-adresse, navn eller brugernavn og adgangskode. Helst ikke mere end det. Det er fint at der kommer en bekræftelsesmail til ens indbakke, men helst ikke aktiveringsmail
 - Sikkerheden har ikke høj-prioritet for Keld, da der alligevel ikke er nogle følsomme informationer på siden
- Informants forslag til flere funktion
 - Merete havde ingen forslag, udover at der skal være billede af opskrifterne, hvilket ikke var vist i prototypen
 - Keld foreslår at sidebaren også er på favoritsiden
 - Keld foreslår en frem- og tilbageknap
 - Filtrering af forskellige landes køkkener, så det eksempelvis var muligt at se italienske retter, kinesiske retter osv. (kun de mest kendte køkkener: nordiskekökken, kinesiske, italienske, græske f.eks.)

C.4.1 Sammendrag

I forhold til funktionalitet i prototype 2, er der nogle få ting, der skal tilføjes, fjernes eller ændres:

1. Når man på søgeresiden tilføjer en opskrift til favoritter, skal man forblive på søgeresiden. Knappen man trykkede på skal erstattes med en "Fjern fra favoritter"-knap
2. Sidebaren på søgeresultatssiden skal være nemmere at få øje på. En løsning er at gøre den synlig fra starten og give brugerens muligheden for at skjule den
3. Knappen i toppen, der viser de favoritter man har gemt, skal være mere sigende. "Vis favoritter" kunne der stå under den

4. På søgeresultatsiden, hvor en opskrifts vises, skal +'et ud for ingredienserne, der tilføjer en ingrediensen til indkøbslisten, være mere intuitiv
5. Brugeren skal muligvis have at vide, at man kan benytte browserens tilbageknap for at gå tilbage til forsiden, uden at ingredienser fjernes. Det kan være at informanten overså denne funktion fordi prototypen blev vist i form af et diashow på en hjemmeside, og informanten derfor var bange for at gå væk fra hele diashowets side
6. Skalering og andre funktioner til visning af favoritter
7. Frem og tilbage knap, der giver brugeren tryghed når han navigerer rundt, så man ikke skal være bange for at browseren forsvinder fra siden

D Manuel mapping

Ingredients left: 0

basmatiris eller luftige urteris	<input type="text" value="Basmati ris"/>
blå birkes	<input type="text" value="Birkes frø"/>
citron i skiver	<input type="text" value="Citroner"/>
dildkviste	<input type="text" value="Dild"/>
friske dildkviste	<input type="text" value="Dild"/>
friskpresset citronsaft	<input type="text" value="Citronsaft"/>
friskpresset citronsaft	<input type="text" value="Citronsaft"/>
groft salt	<input type="text" value="Salt"/>
groft salt	<input type="text" value="Salt"/>
grofthakkede krydderurer	<input type="text" value="Krydderuremix"/>
grofthakkede valnøddekerner	<input type="text" value="Valnøddemeler"/>
knuste dildfrø	<input type="text" value="Dild"/>
olivenolie	<input type="text" value="Olivenolie"/>
ostindisk karry	<input type="text" value="Karry"/>
skrællede gulerødder	<input type="text" value="Gulerødder"/>
små knuste fed hvidløg	<input type="text" value="Hvidløg"/>
små regnbueørred	<input type="text" value="Regnbueørred"/>
smør	<input type="text" value="Smør"/>
tamarindpasta	<input type="text" value="Tamarind"/>
yoghurt naturel 2%	<input type="text" value="Yoghurt"/>

Figur D.1: Program skrevet i C# til kontrol og remapping af ingredienser

E Valg af mapping metode

Ingrediens	Metode 1	Metode 2	Metode 3	Metode 4	Metode 5
dildkvist	(dild)	(dild)	(dild)	sildefilet	(dild)
groft salt	(salt)	citron saft	frugtsaft	frugtsaft	(salt)
grofthakkede krydderurter, fx koriander, persille og dild	(krydderurtemix)	tikka (indisk krydderi)	hakkede tomater	hakkede tomater	hakkede tomater
basmatiris eller luftige urteris	(basmati ris)	herbamare urtebouillon	(basmati ris)	(basmati ris)	(basmati ris)
ostindisk karry	(karrypasta)	kinaradise	sod sherry	sod sherry	(karry)
frisk salvie	(salvie)	(salvie)	fisksauce	fisksauce	(salvie)
koncentreret tomatpure	(tomatpure)	(tomatpure)	soltørrede tomater	soltørrede tomater	(tomatpure)
hakket svinelever	hakket svinekod	(svinelever)	hakket svinekod	hakket svinekod	(svinelever)
store kapers med stilk	(kapers)	syltede artiskokhjerter	trefarvet is	trefarvet is	(kapers)
hvidvin fx rieslin	(hvidvin)	(hvidvin)	hvidvinseddike	hvidvinseddike	(hvidvin)
friskpresset limesaft	(limesaft)	(limesaft)	appelsinsaft	appelsinsaft	(limesaft)
kartoffel	(kartofler)	(kartofler)	(kartofler)	(kartofler)	kartoffelmel
	11	8	3	2	10

Figur E.1: Informanterne satte ring om de mappings, de mente var korrekte.

F Usabilitytest

Usabilitytest med Merete er blevet optaget[13].

Usabilitytest med Keld er blevet optaget[12].

I forbindelse med en usabilitytest foretaget på de 2 informenter, benyttede vi følgende case:

F.1 Case

1. Gå på Foodl.dk
2. Find en opskrift som indeholder “gulerødder” og “piskefløde”.
3. Gå ind på opskriften: “Lakridsfisk på fennikelbund”
4. Gå tilbage til siden du kom fra, hvor du ser søgeresultatet
5. Favoriser opskriften “Lakridsfisk på fennikelbund”
6. Skalér opskriften, så ingredienserne passer til 8 personer
7. Fra opskriften “Lakridsfisk på fennikelbund”, tilføj lakridspulver til indkøbslisten
8. Tilføj alle ingredienser fra “Lakridsfisk på fennikelbund” til indkøbslisten
9. Gå ind på din indkøbsliste
10. Lakridspulver er der 2 gange - fjern den første
11. Du mangler også noget mælk - tilføj “mælk” til indkøblisten
12. Print indkøbslisten ud
13. Indkøbslisten er nu udskrevet. Slet alle varer på din indkøbsliste
14. Gå ud på forsiden
15. Opret en bruger på Foodl med brugernavn: test@test.dk, kodeord: “test123”.
16. Skift kodeordet til at være “1234567”.
17. Lav en ny søgning på “Baconskiver” og “Hakket oksekød”.
18. Sortér opskrifter efter alfabetisk orden.
19. Gør sådan at du kun ser de opskrifter, der er hurtigst at lave.
20. Send en mail til Foodl-udviklerne
21. Find ud af hvor mange, der har været med til at udvikle Foodl.
22. Du er nu færdig på foodl.dk - log ud
23. Du har ikke været på Foodl i lang tid. Prøv nu at finde den opskrift du favoriserede.
24. Du kan ikke længere lide opskriften, fjern den fra dine favoritter.