Navne/fødselsdatoer: Patrick Ekberg(15/03-1990), Peter Hemmingsen(05/10-1989), Rasmus Hecter(21/02-1979), Simon Christiansen(10/05-1992)
1. Års Eksamensprojekt for Datamatiker uddannelsen på Københavns Erhvervs Akademi
Lygten 37, Nørrebro, København

Datamatikerprojekt 1.år - 2017

Indholdsfortegnelse

LitteraturListe **107**

Problemformulering

overordnet spørgsmål: hvordan kan FoodCare udvide sin position på madmarkedet?

hvorfor skal FoodCare bruge vores system (Foodie-log)?
hvilke fordele/ulemper er der ved indførsel Foodie-log?
hvordan vil det nye system udvikle kundernes forhold til FoodCare?

Indledning

Følgende er rapporten for 1.års eksamensprojektet, for datamatikeruddannelsen på Københavns erhvervsakademi. Rapporten indeholder en beskrivelse af forløbet med udviklingen af vores logbogs-app, kaldet Foodie-log, for firmaet Foodcare. Foodcare er et firma der sælger sous vide udstyr og er ejet af Mads Hecter. Efter forespørgsel, blev det afgjort at vi kunne lave en logbogs-app for dem, hvori den enkelte bruger kan oprette sous vide opskrifter til sin private liste. Der vil også være et forum hvori adskillige brugere kan oprette indlæg og kommentere på disse indlæg.
Selve rapporten er delt op i fire dele svarende til de fire fag og kravs områder rapporten skal opfylde. Under disse dele vil man kunne se de metoder,strategier og overvejelser vi har gjort os gennem forløbet, i form af representative billeder, diagrammer, udvalgt kildekode og tilhørende beskrivelser. Sidst i rapporten vil vi komme med en konklusion på projektet samt en vurdering af fremtidige udviklingsmuligheder for Foodie-log.

Software design

Krav-specifikationer:

Foodcare (kundens) overordnede krav:
Applikationen skal fungere som en log bog over sous vide opskrifter, hvor man skal have muligheden for at dele sine opskrifter og erfaringer med andre brugere af sous vide appen i et forum. user af appen skal også kunne stille spørgsmål til andre brugere omkring deres erfaringer med opskrifter. Foodcare skal være i stand til at redigere i forummet og slette indlæg, hvis der er nogen der skriver irrelevante beskeder.
Alle brugere (også dem der ikke har en konto) der går ind på appens forside har adgang til en liste med et udvalg af opskrifter.

Funktionelle krav:

- Usere skal have adgang til en opskrifts liste uden log-in.
- Usere skal kunne registrere sig ved at indtaste email, username og password.
- user skal kunne logge sig ind ved at indtaste sit specifikke username og password.
- user skal kunne se og opdatere sine konto detaljer(Email,username og password).
- user skal være i stand til at slette sin egen konto.
- user skal kunne oprette en opskrift til sin egen opskrifts liste.
- user skal have adgang til sine egne opskrifter via en liste på sin homepage.
- user skal kunne bedømme sine opskrifter på en skala fra 1-10,
- user skal kunne skrive en vurdering af den enkelte opskrift.
- user skal kunne opdatere sine opskrifter.
- user skal kunne slette sine opskrifter.
- user skal kunne oprette et indlæg i et forum.
- user skal kunne kommentere til indlæg i forum.
- (Admin skal kunne slette beskeder i forum)
- (Admin skal kunne CRUD forside opskrifterne)

Non-funktionelle krav

Sikkerhed:
users data skal kun være tilgængelig for user selv.
users data skal beskyttes af spring-security.
Databasen skal beskyttes af prepared statements.

Database:
Databasen skal bestå af tables i 3.normalform

Design:

Generelt design:
Vi vil bruge elementer fra Foodcare's hjemmeside såsom de samme skrifttyper, farver, layout. Nedenfor et par tidlige eksempler på hvordan vores design skulle se ud.
Forum.html:

Home **Logout**

Foodie-Log

FOODIE FORUM:

Overskrift
Text

TILFØJ

Hummer sous vide

Hvor lang tid skal en hummer sous vides?	USERNAME 1

SVAR SLET (kun admin der kan se delete)

RE: Hummer sous vide --- Du skal koge den i en time med hovedet nedad når månen er halv og de 7 jomfruer spiller "Nothing else matters"	USERNAME 2

SVAR SLET (kun admin der kan se delete)

RE:RE: Hummer sous vide --- Ok så tror jeg at jeg springer den opskrift over da jeg hader Metallica	USERNAME 1

SVAR SLET (kun admin der kan se delete)

Homepage.html:

Foodie-Log

Tilføj opskrift

Navn på opskrift	emne	udskæring	vægt	tykkelse	tid	temperatur	detaljer

TILFØJ

Mine opskrifter

Oksekød- navn på opskrift (Karakter 1-10) SLET BEDØM HVIS OPSKRIFT

Svinekød -navn på opskrift (Karakter 1-10) SLET BEDØM HVIS OPSKRIFT

opskrift.html:

Foodie-Log

opskrift: Culottesteg
Emne: Oksekød
Udskæring: Culottesteg
Vægt: 1,2 kg
Tykkelse: 10 cm
Tid: 23 timer
Detaljer:

Første skal man huske at…… ….. Herefter skal man være opmærksom på….

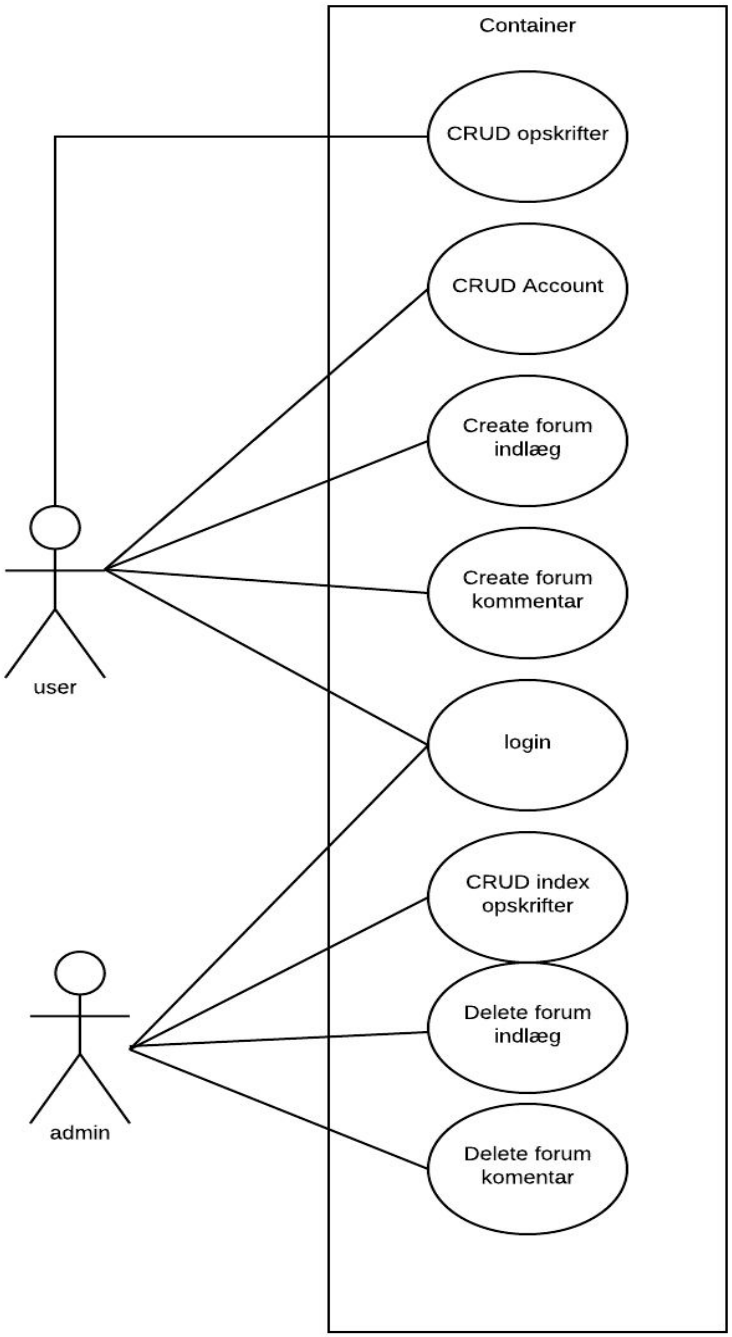
Vurdering:

Min steg blev rigtig fin ….blablablablablablablablablabla Blablablablablablablablablablablablablabla

OPDATER

Usecase diagram

Det følgende use case diagram viser et overblik af de forskellige Actors og deres tilhørende usecases.


Container
CRUD opskrifter
CRUD Account
Create forum indlæg
Create forum kommentar
login
CRUD index opskrifter
Delete forum indlæg
Delete forum komentar
user
admin

Use cases

Title - Display opskriftsliste

Actor: User

beskrivelse:
Når user åbner appen, kommer de ind på index html. Her vil user kunne se en liste med et udvalg af opskrifter. Der vil være knapper som user kan trykke på for at blive sendt hen til specifikke steder på listen.

Title - Registrering

Actor: User

beskrivelse:
user skal kunne registrere sig ved at trykke på “registrer”. user bliver bedt om at angive brugernavn, password og email. user skal derefter trykke “submit” for at færdiggøre processen og bliver sendt tilbage til index.html.

Title - Log in

Actor: User

beskrivelse:
user skal kunne logge ind ved at trykke på knappen “log in. user bliver bedt om at angive username og password og skal derefter trykke “submit” for at blive logget ind og sendt videre til sin egen homepage.

Title - Update Account Details

Actor: User

beskrivelse:
user kan opdatere sine bruger information, ved at trykke på “account”.
user kommer derefter ind på en ”account” side. user ville få valget mellem at ændre password eller Email eller returnere til homepage.
Ved valget af email/password ændring, skal user så indtaste sit gamle email/Password og et nyt og skal derefter trykke “submit” for at gemme ændringerne. når ændringen er blevet bekræftet bliver user sendt tilbage til sin homepage.

Title - Delete Account

Actor: user

beskrivelse:
user skal kunne slette sin account. Dette gøres ved at trykke på en “account” knap på homepage. user skal så indtaste sit password og trykke på “submit” for at bekræfte at man vil slette sin account. users account bliver slettet og user bliver sendt til index.html.

Title - Create opskrift

Actor: user

beskrivelse:
user skal være i stand til at tilføje en opskrift til sin food-log. På homepage.html er der indtastningsfelter for en nye opskrifter. I disse felter skal user indtaste henholdsvis title til opskriften, emne, udskæring samt vægt, tykkelse og tid. user kan også angive tips og tricks under “detaljer” feltet. user skal så trykke “submit” som derefter tilføjer den nye opskrift til listen over users opskrifter.
Alternative scenario:
når user vil bekræfte sin opskrift og ikke har udfyldt alle felterne, ville user blive gjort opmærksom på hvilke felter som ikke er udfyldt, og spurgt om de er sikker på om man ville fortsætte.
Alternative scenario
Hvis user skriver en ukorrekt værdi i et felt, så kommer der en error message der gør user opmærksom på at værdien ikke er korrekt, og at den skal ændres før man kan fortsætte.

Title - Read opskrift

Actor: user

beskrivelse:
user skal kunne læse sine opskrifter. Der ligger en oversigt af users opskrifter på homepage.html .user kan trykke på knappen “vis opskrift” ud fra hver opskrift, for at komme til opskrift.html

Title - Write Evaluation

Actor: user

beskrivelse:
user skal kunne give en skriftlig vurdering af sine opskrifter samt en karakter.
user klikker på “bedøm” ud fra en allerede eksisterende opskrift på homepage.html. user ville derefter blive ført til opskrift.html hvor user giver opskriften en karakter mellem 1-10. Der **kan** også gives en detaljeret skriftlig beskrivelse af opskriften. user kan trykke “submit” for at gemme og returnere til homepage.html.

Alternative scenario:
user angiver karakteren men giver **ikke** en detaljeret beskrivelse. user kan derefter trykke “submit” for at gemme og returnere til homepage.

Title - Update opskrift

Actor: user

beskrivelse:
user skal kunne opdatere/ændre i sine opskrifter. users liste af opskrifter ligger på homepage.html. user kan trykke på knappen “vis opskrift” ud fra opskriften, for at komme til opskrift.htm. Herefter kan user trykke på “update” for at kunne ændre i indtastningsfelterne og herefter trykke “submit” for at gemme ændringerne. user vil så blive sendt tilbage til opskrift.html.

Title - Delete opskrift

Actor: user

beskrivelse:
user skal kunne slette sine opskrifter fra sin liste.
user kan trykke på “delete” ud fra den specifikke opskrift man ønsker at slette. user bliver spurgt om en bekræftelse og ja/nej om man ønsker det eller ej.

Title - Create Forum indlæg

Actor: user

beskrivelse:
user kan trykke på “forum” fra homepage for at komme til forum.html.

user kan her oprette en tråd, ved at skrive en overskrift og tekst i indtastningsfelterne på siden og trykke “submit”.

Title - Create comment on Forum indlæg

Actor: user

beskrivelse:
På forums siden kan user se oplagte tråde. Ved disse tråde er der en svarknap. Når user trykker på denne vil de kunne indtaste en kommentar og trykke submit for at tilføje kommentaren til den tråd. user bliver derefter returneret til forum.html

Title - Delete Comment/ forum indlæg

Actor: Admin

beskrivelse:
Admin skal kunne slette kommentarer og indlæg/tråde. Adminen trykker på en delete knap ud fra den kommentar eller tråd som han vil slette. Denne knap kan kun ses og bruges af adminen.

Title - Create index opskrifter

Actor: admin

beskrivelse:
Admin skal være i stand til at oprette opskrifter til udvalget på indexsiden. Dette sker fra admin.html siden. På denne side vil der være en “lav forside opskrift” knap som admin kan trykke på. Der vil så dukke indtastningsfelter op som admin skal udfylde med relevant input, hvorefter han kan trykke “submit” for at bekræfte og tilføje opskriften til forside opskrifterne.

Title - Update index opskrifter

Actor: Admin

beskrivelse:
Admin skal være i stand til at ændre i allerede eksisterende index.html opskrifter. Dette opnår han via en “vis forside opskrifter” knap som befinder sig på admin.html siden. Når admin klikker på den knap vil listen med alle forside opskrifterne blive vist. Ud fra hver opskrift vil der være en “update” knap. Når den bliver klikket på vil den pågældende opskrift blive vist og man kan ændre i indtastningsfelterne som man ønsker. Når man er færdig med ændringerne trykker man “submit” og ændringerne vil blive gemt og admin vil blive sendt tilbage til admin.html.

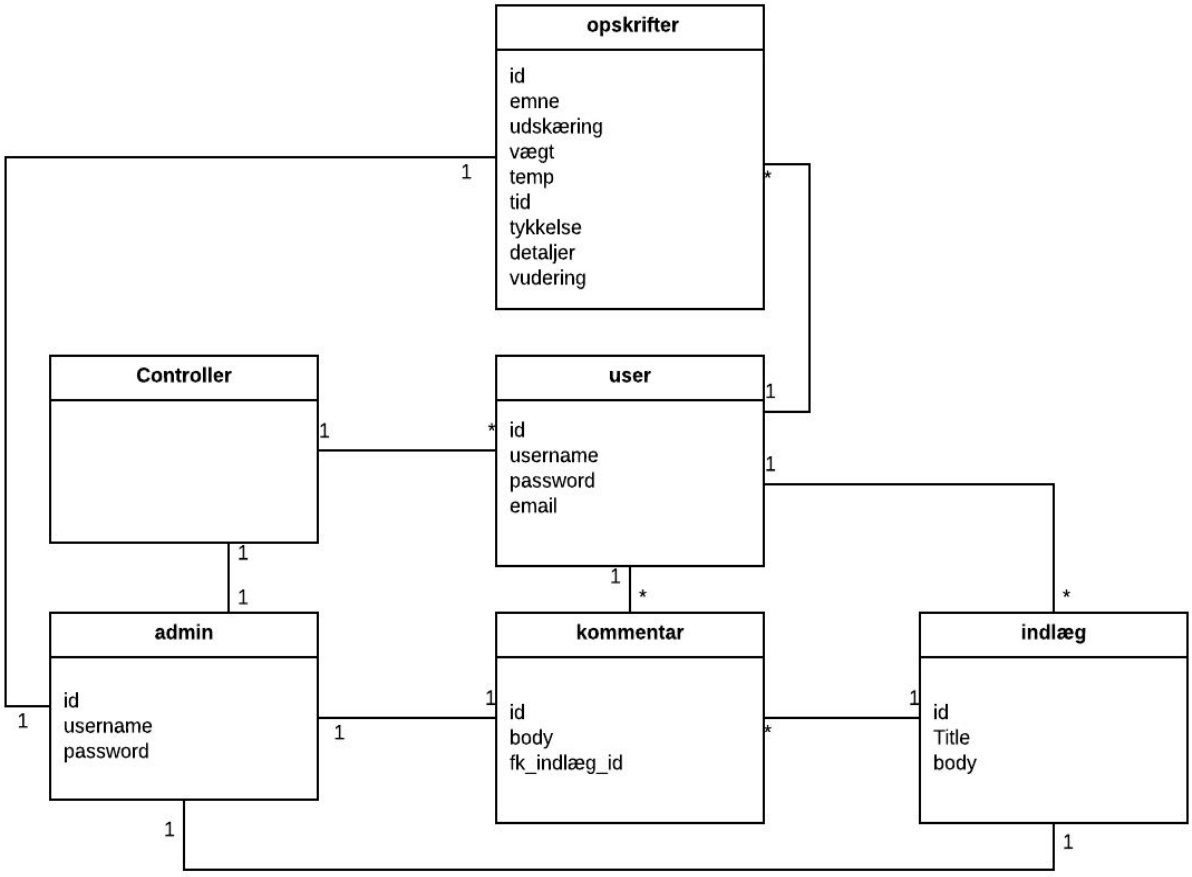
Title - delete index opskrifter

Actor: Admin

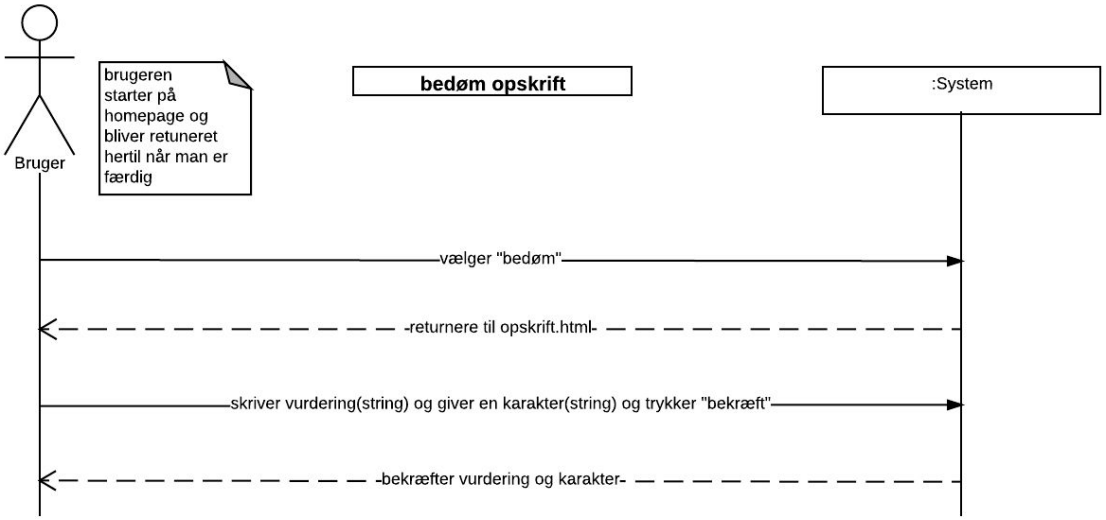
beskrivelse:
Admin skal være i stand til at slette forside opskrifter. Admin skal trykke “vis forside opskrifter” på admin.html. Listen af opskrifter vil blive vist. Ud fra hver opskrift vil der være en “delete” knap.
Når admin trykker på den bliver han spurgt om han er sikker eller ej. Han kan svare ja eller nej. Hvis han svarer ja bliver opskriften slettet, hvis nej vil der ikke ske noget.

Domain model

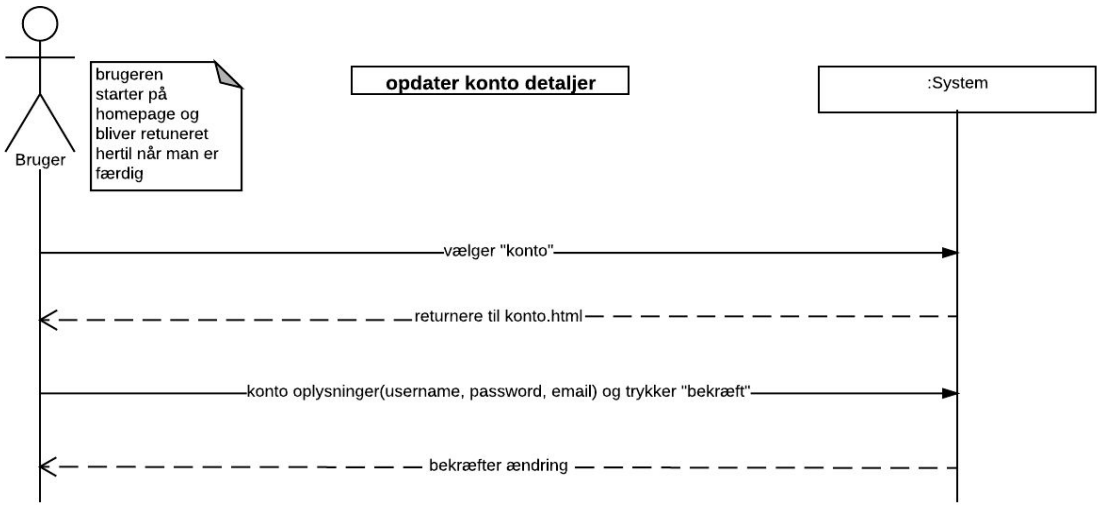
Nedenfor kan man se vores domain model. Her ses forholdet og multiplicities mellem de forskellige parter der indgår i vores projekt.
Som ses i diagrammet, har konceptet “user” relationer med næsten alle andre koncepter. Da useren kan lave og opdatere opskrifter og har mulighed for at have flere opskrifter, har han relation dertil. Useren har også også en relation både med indlæg og kommentar, da der er mulighed for at lave begge hvis man er en oprettet user på siden. Userens interaktion med programmet foregår igennem controlleren. Administratoren for systemet har mulighed for at ændre i opskrifterne på index.html. Der er også mulighed for at ændre i de kommentarer og indlæg som useren lægger på forumet.


opskrifter
id
emne
udskæring
vægt
temp
tid
tykkelse
detaljer
vudering
Controller
user
id
username
password
email
admin
id
username
password
kommentar
id
body
fk_indlæg_id
indlæg
id
Title
body
1
*

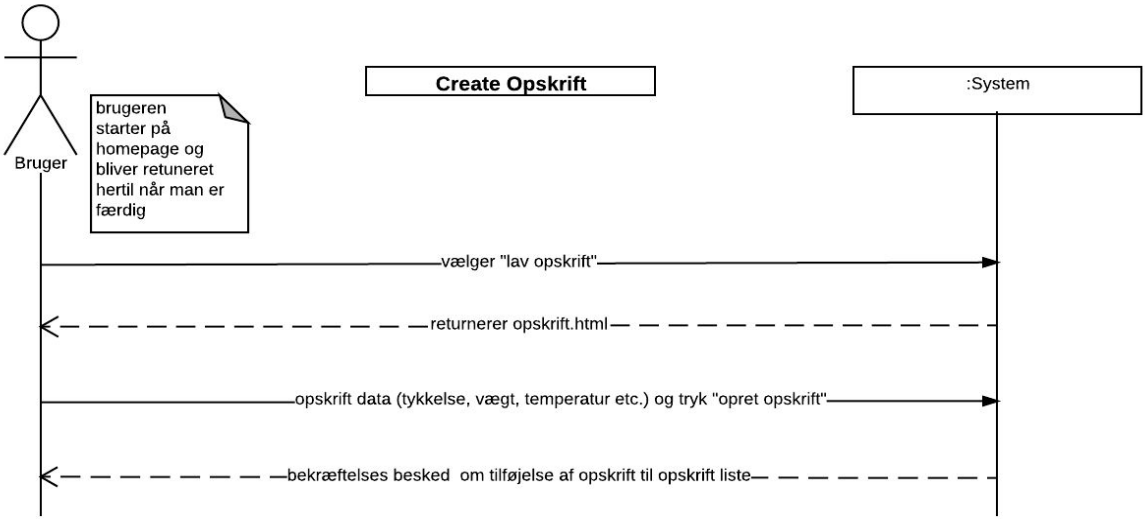
SSD - System Sequence Diagrams



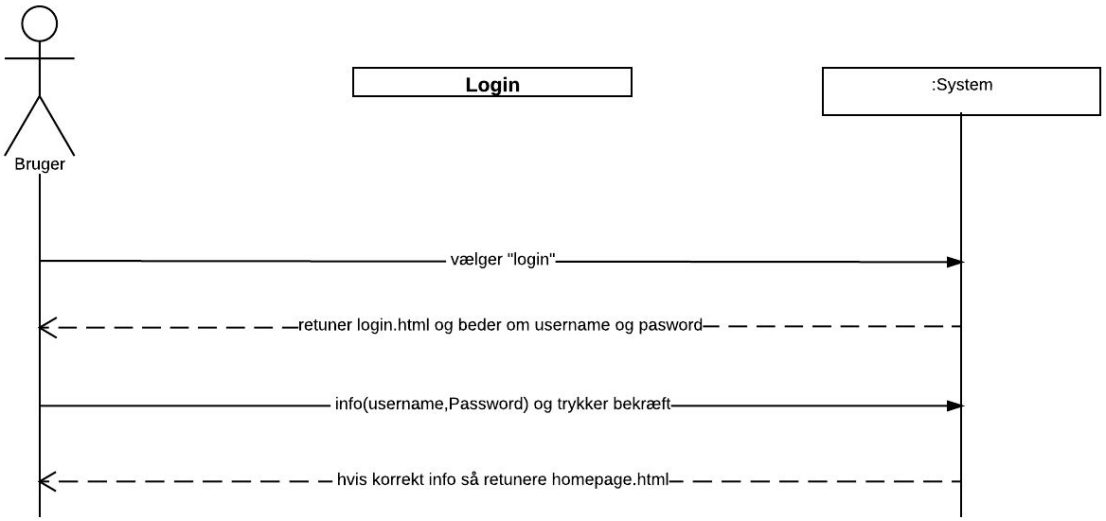
Vi har en bruger i denne SSD, som skal bedømme en opskrift. Han gør det ved at vælge knappen bedøm, systemet returnerer user til opskrift hvor han skriver en vurdering og en karakter af opskriften. Systemet bekræfter vurderingen.



Her opdaterer user sin konto. Han vælger konto hvorefter han bliver returneret til konto.html. Han skriver konto oplysninger og systemet bekræfter ændringerne.



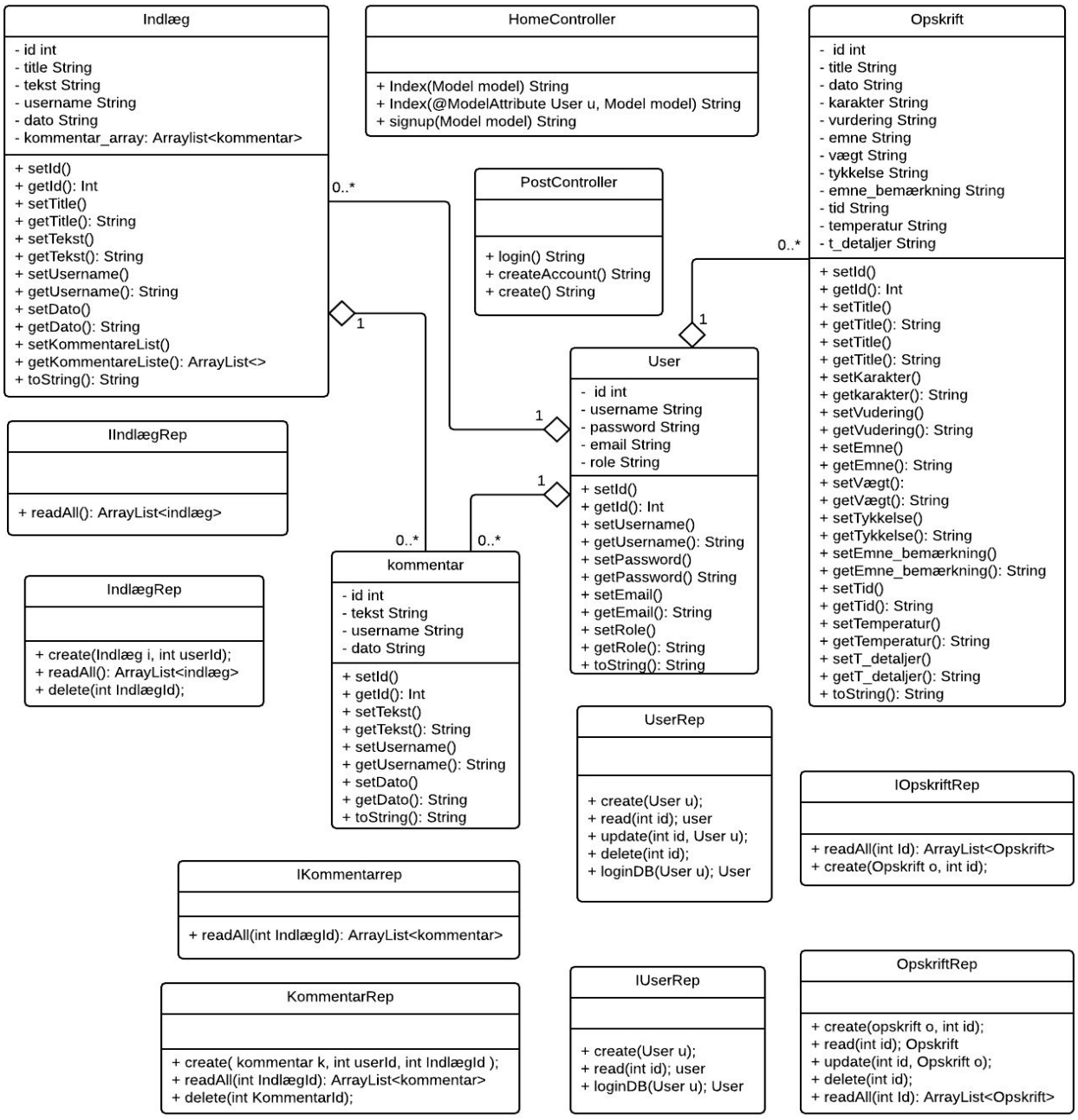
Her skal user lave en opskrift. Efter sit valg, bliver han returneret til opskrift.html. Her skriver han dataen som hører til opskriften. Systemet giver en bekræftelse.



user vælger at logge ind. Han bliver returneret til login.html og bliver bedt om username/password. user skriver info og trykker login. Hvis det stemmer overens, returnerer til homepage.html.

Class Diagram

Herunder ses vores endelige class diagram. Heri ses alle de klasser vi bruger, samt deres attributter og metoder. Man ser også relationerne mellem vores objekter,hvordan en user kan have flere indlæg, opskrifter og kommentare.



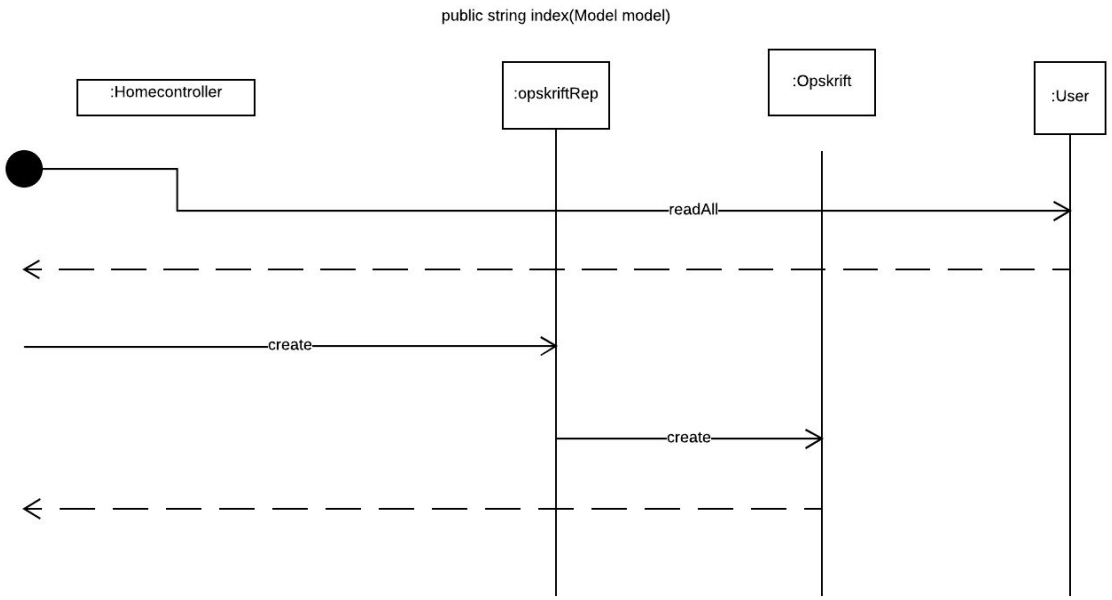
SD diagrammer:

Herunder ses vores Sekvens diagrammer for nogle udvalgte metoder vi bruger i vores program.

Det følgende SD er lavet over følgende metoder.

```
public String index(Model model){
```

```
  model.addAttribute("user", new User());
  model.addAttribute("opskrifter", oRep.readAll(1));
   return "index";
}
```



Her starter vi :Homecontroller, vi går hen til opskrift repository hvor vi læser alle opskrifter med tilhørende user. Derefter bliver der lavet en opskrift med den læste information så det tilhører den specifikke user.

Næste SD er for følgende metode, hvori en user tilføjer et nyt indlæg til forumet.

```
public ArrayList<Indlæg> readAll(){
```

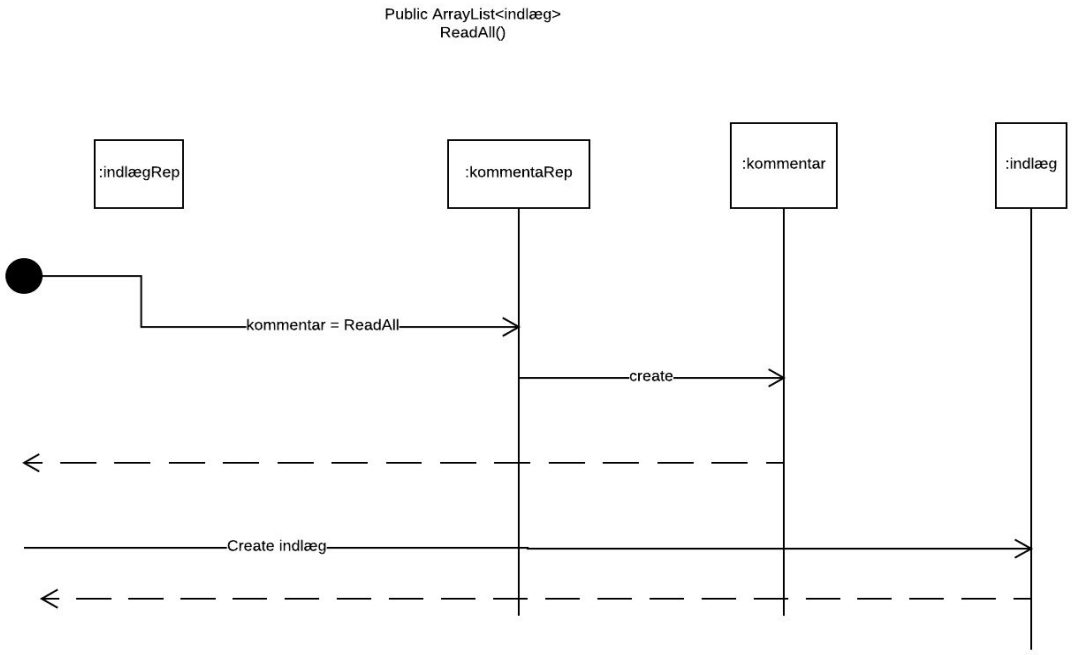
```
    ArrayList<Indlæg> forum = new ArrayList<>();

    SqlRowSet sqlRowSet1;
    sqlRowSet1 = jdbc.queryForRowSet("SELECT * FROM db.indlæg INNER
JOIN db.users ON fk_user_id = db.users.user_id");
    while(sqlRowSet1.next()) {

        int indlægId = sqlRowSet1.getInt("indlæg_id");

        ArrayList<Kommentar> kommentare = kRep.readAll(indlægId);

        forum.add(new Indlæg(sqlRowSet1.getInt("indlæg_id"),
sqlRowSet1.getString("title"),
                sqlRowSet1.getString("tekst"),
sqlRowSet1.getString("username"), sqlRowSet1.getString("dato"),
kommentare));
    }
    return forum;
}
```

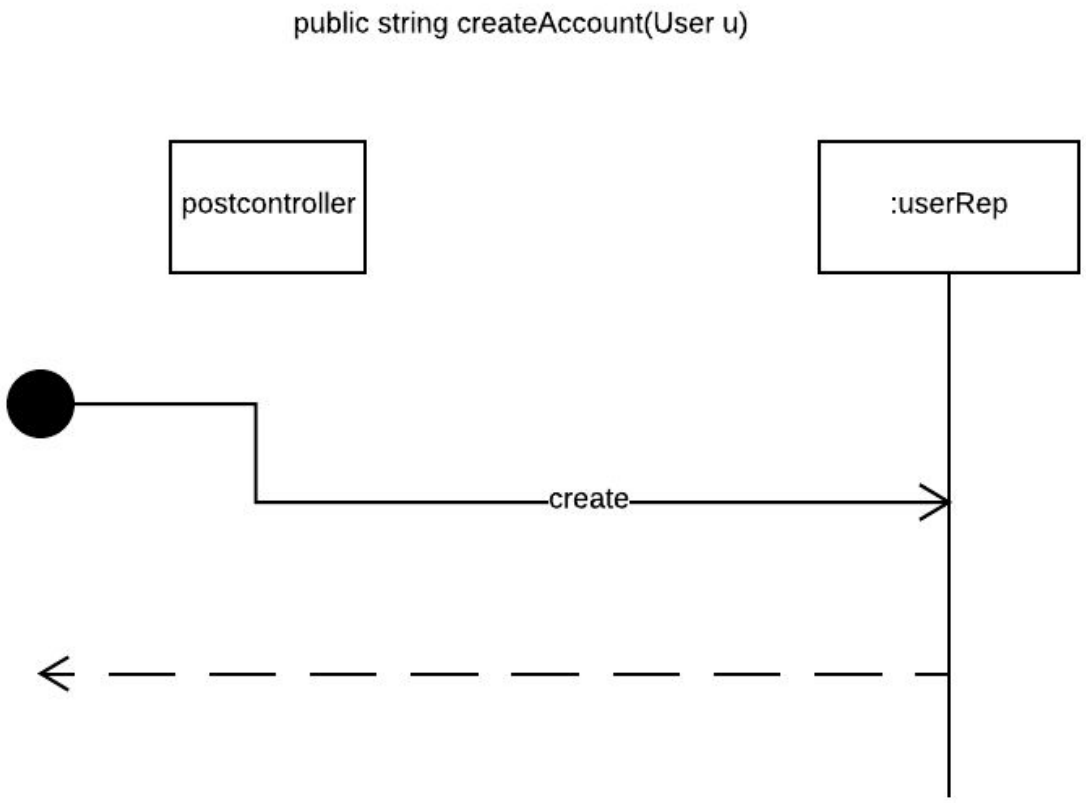


Sekvens diagrammet starter ud i indlægRep hvor den finder alle eksisterende indlæg.herefter går den videre til kommentarRep hvor den læser alle de tilhørende kommentarer og returener dem. Til sidst laver den et nyt indlæg og returner.

Tilsidst har vi et SD der viser metoden der opretter en bruger i systemet.

```
@PostMapping("/signUp")
public String createAccount(@ModelAttribute User u) throws
SQLException{

    uRep.create(u);
    return "redirect:/";
}
```



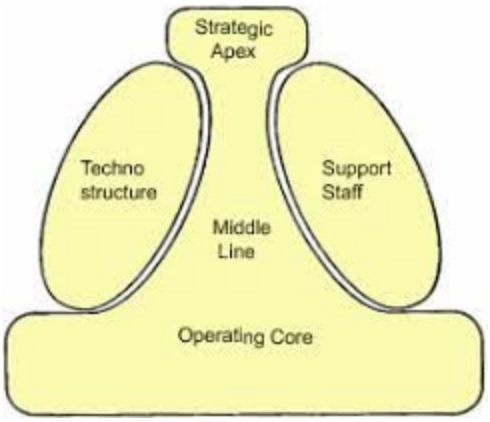
Metoden begynder i postcontroller og går over i userRep hvor den så laver en bruger og returnerer det.

ITO - InformationsTeknologi i Organisationer

Organisationsform

Foodcare er en enkeltmands virksomhed, som sælger varer indenfor vakuumpakning af fødevarer og tilbehør til sous vide tilberedning mad.
Foodcare har både vertikal og horisontal virksomhedsstruktur. Direktøren står i spidsen som beslutningstager, imens de ansatte har en vertikal struktur, som giver dem bedre mulighed for at kommunikere.

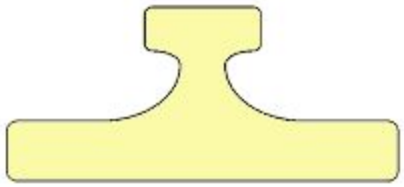
Mintzberg organisationsteori:



Mintzberg:
Mintzberg er arbejdspsykolog som har udarbejdet en teori som beskriver forskellige organisationsformer og strukturer ud fra forskellige punkter. Mintzberg beskriver også udviklingen af virksomheder ofte starter på et vist stadie, hvorefter de så udvikler sig i takt med at både indre og ydre faktorer udvikler og ændrer sig.
Teorierne tager udgangspunkt i, at der er to behov som er modstridende for en menneske drevet virksomhed, hvor den ene er at fordele arbejdsopgaver og derefter koordinere dem.
Modellen ovenfor, beskriver de grundlæggende elementer for strukturen i en virksomhed.
Her ses ledelsen som øverst med mellemleder og den operative(medarbejdere) længere nede og en mellemleder i midten. Ved hver side, kan man se de forskellige støtte afdelinger, såsom marketing, IT osv.

Ifølge Mintzbergs organisationsteori, ville en virksomhed som Foodcare være beskrevet som **"Simpel Struktur"**. Som beskrevet i organisationsformen, er virksomheden baseret på, at direktøren står i spidsen for beslutningstagning samt ansvar. Han fordeler også arbejde mellem medarbejdere, hvor det bedst kan betale sig. Kommunikationen mellem leder og medarbejder

er god, og det giver også grund for hurtig og effektiv beslutningstagning, da systemet samt varebeholdninger er af et overskueligt niveau.



Unge virksomheder plejer som regel også at have en Simpel Struktur ifølge Mintzberg, da den simple struktur har størst effekt i mindre virksomheder, da større virksomheder kan finde det svært, at holde styr på information hvis der ikke er en mere direkte struktur i arbejds - og informations fordeling.
Som vist i ovenstående model, har den strategiske direktør ansvar for det operationelle niveau med support fra medarbejdere, hvor at medarbejderne har direkte kontakt til ledelsen.
Dette gør at medarbejdere har et mere frit råderum på arbejdspladsen, samtidig med at det stadig er den ledende del, som står for styring af virksomheden.
Den beslutningstagende process er også meget fleksibel, da der er kort afstand mellem leder og medarbejder.

Foodcare hører delvist også til **Adhocratiet**, da det er baseret på nichê viden indenfor kokkerering, samt har samarbejde med flere eksperter i form af bloggere/køkkenchefer mm.
Foodcare har også god mulighed for at tilpasse sig ændringer i markedet. Dette kan argumenteres for, da det er en lille virksomhed som er baseret på onlinesalg, med lav lagerbeholdning og høj diversitet.

Forretningsgrundlag

idégrundlag:
Foodcare havde som basis, at ville sælge vakuumpakker til madlavning og senere inkluderede at sælge produkter til sous vide madlavning, som er brugervenlige, miljøvenlige og i høj kvalitet.
mission:
at formindske madspild i hjemmet/restauranter mm. vi håber at viden indenfor vakuumpakning ville øge danskernes opfattelse af bæredygtig madlavning.
værdier:

vi har fokus på god kommunikation og samarbejde med kunderne, både private som virksomheder.
Vi prøver at sælge produkter med en god garanti og med håbet om, at vores kunder ville være vores største form for reklame og samarbejdspartnere.

vision:

Vores (virksomhedens) vision er, at det danske madmarked bliver vendt på hovedet og der kommer mere fokus på kvalitetsmad som er lavet på en bæredygtig måde. Vi håber at kunne få samarbejde med lokale miljøer samt kommuner for at få vores værdier og visioner ført ud i livet.

Swot analyse af FoodCare:

Strengths:	**Weakness:**
Online butik. Lille virksomhed. Nichë marked. Kunder med interesse. bæredygtighed. samarbejder med restauranter/bloggere.	handler kun online. lille virksomhed. Nichë marked. dyre/eksklusive produkter. afhængig af postvæsen. begrænsede ressourcer.
Opportunities:	**Threat:**
mulighed for udvidelse/investorer. øget interaktion mellem kunde vha. app. samarbejde med kommunen om bla. madspild.	konkurrence fra detail/store virksomheder - (disrupted af coop og bilka.). utilfredse kunder/dårlige anmeldelser.

SWOT beskrevet:

både en styrke og svaghed ved Foodcare, er at det ikke er en fysisk butik, men at alle varerne bliver handlet online, hvilket betyder, at der ikke skal skal laves noget logistik omkring at eje en butik hvor de forskellige varer skulle sælges. Til gengæld bliver man afhængig af postvæsen og internetudbyder.
Foodcare sælger varer til et nichê marked, hvilket igen kan ses både som en styrke men også en svaghed. Kunder der handler inden for et nichê marked, er ofte engagerede og har valgt produkterne samt udbyderen med omhu, altså er der også en stor interesse fra kundernes side for at producenter inde for deres marked, klarer sig godt. Kunder er også bedre til at reflektere over køb når det er blevet gjort med et vist engagement, og kan derfor være mere tilbøjelige til at lave god omtale om sælgeren (Hvis altså det har været en positiv oplevelse). Det kan dog være svært som sælger at nå ud til et stort segment. Varerne er også lidt mere eksklusive og kan være svært for en del af segmentet at købe.

Foodcare har allieret sig i et samarbejde med både forskellige restaurationer og bloggere som bruger og fortæller om produkterne, hvilket ikke kun åbner for et mere professionelt marked, men også giver FoodCare feedback fra professionelle, hvilket derfor ville øge udviklingen af produkter hos FoodCare. Endvidere er der også stor mulighed for at tiltrække opmærksomhed fra diverse investorer, dette kan give lejlighed for at udvide virksomheden til salg i diverse andre lande som Tyskland eller England.
Foodcare er en lille virksomhed og med det følger et begrænset budget. Dette ville føre til en tæt afhængighed af at ens leverandør kan levere varer som er i orden og til tiden, da budgettet ofte ikke kan holde til fejl.
Foodcare start grundlag, var at sælge vakuum-pakker til detailhandlen.
I starten var der ikke nogen konkurrence og prisen kunne derfor sættes rimelig høj.
Et par år efter begyndte Coop og Bilka at sælge selv samme produkter, bare fra andre leverandører/firmaer og foodcare blev derfor nødt til at sænke priserne.
Foodcare mistede en del kunder og prøver nu at få de kunder tilbage ved at sælge bedre og flere maskiner med mere og specielt tilbehør.
Foodcare er også igang med at skifte retning til at handle mere med restauranter, delikatesser, slagtere etc . Fordelen ved dette er at Foodcare kan sælge større og dyrere maskiner og derved ikke skal sælge så mange enheder og derfor mindske arbejdet med reklamationer og ikke er så afhængig af postvæsenet.

Interessentanalyse

- **developers**

Udviklerne af systemet har en interesse i dets funktionelle niveau, da det har betydning for karakterniveau og vores fremtidige porteføljе.

- **slutbrugerne**

Fokus med programmet, er slutuser. Programmet skal fungere som en service hvor user kan reflektere og dokumentere sine oplevelser med sous vide.

- **kunde**

Kunden har interesse i at programmet virker som det skal, da det vil være med til at udbrede sous vides popularitet og dermed efterspørgelsen på det udstyr som kunden udbyder.

- **KEA**

KEA har en interesse i at programmet bliver en success, da et godt program er ensbetydende med, at de studerende med høj sandsynlighed vil fortsætte på studiet.

- **samarbejdspartnere (kokke der giver opskrifter etc.)**

Foodcare har et tæt samarbejde med forskellige restaurationer, bloggere og kokke, som både skriver om, og bruger, Sous Vide.

- **familiemedlemmer til projekt arbejdere**

da der er meget arbejde i at lave et projekt, kan det have små konsekvenser for diverse familiemedlemmer.

- **marketingafdeling**

kundens marketingafdeling har interesse i at projektet bliver en success, da det kan bruges som en måde at markedsføre de forskellige produkter på.

Feasibility study

- **Tekniske forhold:**
    1. Den nødvendige teknologi er tilgængelig, da alle har mulighed for at udvikle projektet. Ligeså er ressourcer til oprettelse af systemet, da vi har lært hvad vi skal bruge til udførelse.
    2. I fremtiden ville der er være en forventning om at der skal bruges økonomiske midler på at vedligeholde systemet ved udgifter til konsulenter/support. Yderligere ville admin skulle bruge tid/ressourcer på opdatering af systemet.

3. udviklerne af systemet tager forholdsregler mht. sikkerheden i systemet. Systemet kræver dog nok et eftertjek indenfor systemsikkerhed.

- **Økonomiske og Finansielle forhold**
    1. cost/benefit er positiv da udviklerne af systemet arbejder gratis.
    2. der ville løbende være udgifter vedr. drift af systemet da database/program skal placeres på en server.
    3. der er fare for tab af kunder hvis systemet ikke lever op til kundernes forventning.
    4. vi har en forventning om, at der ville være reklamer via systemet samt generere mere trafik på hans webshop pga. øget aktivitet indenfor emnet.
- **Organisatoriske og operationelle forhold**
    1. da det er en ekstra service til hans webshop er der ikke nogen påvirkning af de operationelle eller organisatoriske forhold.

- **Juridiske og retlige forhold**
    1. udvikler og kunde er ikke kommet til en afklaring om hvorvidt der er fremtidige forpligtelser.
    2. Administrator ville have adgang til slutbrugernes emails. Dette ville foregår ifht. persondatalovens retningslinjer.

Risikoanalyse

kravs specifikations risiko:
Kravene til systemet er ikke blevet gjort klare nok, og systemet bliver derfor ikke brugbart for kunden.
teknologisk risiko:
Der er en risiko for problemer med database vedligeholdelse/håndtering, som gør at systemet stopper med at fungere efter en periode.
personale risiko:

Hvis en person fra projektet bliver langtidssyg eller melder sig ud, kan det have store konsekvenser for systemets kvalitet og tidshorisont.

Kunde:

kunden mister sin forretning. Dette kan bl.a. Ske ved at kunden løber tør for ressourcer

økonomisk risiko:

kunden løber tør for penge, og kan ikke betale for at vedligeholde systemet.

sikkerhed:

sikkerheden er ikke i orden og det kan give adgang til uvedkommende som kan ødelægge systemet.

risiko moment	**sandsynlighed**	**konsekvens**	**produkt**
kravspecifikations risiko	3	3	9
teknologiske risici	1	7	7
personale risici	2	7	14
kunden mister forretning	2	10	20
økonomisk risiko	2	7	14
sikkerhed	2	3	6

Risiko Moment	Sandsynlighed	Konsekvens	Produkt	Præventive tiltag	Ansvarlig	Løsnings-forslag	Ansvarlig
Personale risici	2	7	14	Sørge for at alle er i stand til at udføre arbejdet.	Alle i gruppen	Der bliver brugt tid på at alle kan alt.	Alle i gruppen er ansvarlig
Kunde mister forretning	2	10	20	Kunden har en god forretnings-strategi	Kunden	Find ny kunde	Alle i gruppen

Økonomisk risici	2	7	14	Kunden sørger for at opspare ressourcer	Kunden	Lån penge	Kunden

Det skal dog siges, at de beskrevede risici er rent teoretiske.

Strategiske/Organisatoriske overvejelser ved indførsel af systemet

Familieforbindelse:

Følgende er de mulige overvejelser som firmaet Foodcare har gjort sig i henhold til projektet. Ejeren af Foodcare er bror til en af udviklerne af projektet og har kommunikeret et ønske om en logbogs app med et forum.

Investering i projektet:

Da dette projekt er et eksamensprojekt behøver Foodcare ikke at investerer penge i projektet, da udviklerne arbejder gratis.

Ansvar:

Efter at have givet sine specifikke ønsker om hvad projektet gerne skal resultere i, har de overladt alle overvejelser og ansvar for hvordan disse ønsker efterkommes.

Efter indførsel af systemet, ville der være bedre kommunikation med slutkunden:

ved indførsel af det nye system, ville kunden have mulighed for at bruge sin log-in til siden til at logge på appen. Her er der mulighed for at dele og diskutere ideer samt resultater med andre brugere af Foodcare's hjemmeside og dermed og brugere af deres produkter. Det ville også øge muligheden for, at kunden kan finde et behov, for at investere i flere af virksomhedens produkter, for at opnå nye resultater.

Der skal udpeges en administrator for at vedligeholde det nye system.

Da der ville være brug for en form for styring på bla. det oprettede forum for user, skal der findes en administrator til denne opgave. Administratoren kan også redigere i diverse andre dele af systemet, for at det kan køre optimalt for at tilfredsstille slutkunden.

Software Konstruktion

HTML tags

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
```

```
<head>
  <title>index</title>
```

Vi gør brug af fragments, så vi kan genbruge en masse fælles indhold og mindske risikoen for redundans. Fragments er en del af en af html side, hvor vi definerer det indhold der skal genbruges og derefter kalder det givne fragment med th:replace”fragment.html::fragment-navn”
I nedenstående tilfælde kalder vi fragments, der vedrører styling og deraf navnet header-css og footer-css

```
  <div th:replace="fragments/headers :: header-css"/>
  <div th:replace="fragments/footer :: footer-css"/>
```

Herunder kalder vi vores stylesheet style.css

```
  <link rel="stylesheet" href="style.css" />
```

Her definere vores styling på vores tags lokalt på vores side. Style tagget indikerer at nu skriver vi CSS kode.

```
  <style>
```

<Nav> definerer et sæt af navigationslinks, som vi bruger som en menu af links.

```
    nav{
        /*border: 1px solid red;*/
        position: fixed;
        top: 150px;
        width: 10%;

    }
```

 står for divison og bruges til at gruppere blok elementer. F.eks bruger vi denne   at lægge en svag grå farve under alt den tekst der er i html tag.

```
    div.tommelregel{
        width: 95%;
        margin: auto;
        background-color: whitesmoke;
        color: gray;

    }
```

<pre> er et tag der gør, at vi kan vise den tekst, eksakt som den er skrevet i vores html.

Altså når man skifter linie (line Break) skifter browseren også linie og når man laver mellemrum laver den det eksakt samme mellemrum

```
    pre {
        width: 100%;
        margin: auto;
    }
```

<a> Tag bruges til at definere hyperlinks. Som udgangspunkt er <a> født med:

- Understreget og blå for et ikke besøgt link.
- Understreget og lilla for et besøgt link.
- Understreget og rød for aktivt link.

Vi har lavet de ovenstående om så teksten er sort og at a.link fylder en hel block og at når man har musen over linket så skifter baggrundsfarven til lightgrey.

```
    a{
        text-decoration: none;
        color: black;
    }

    a.link{
        padding: 5px;
        display: block;


    }

    a:hover{
        background-color: lightgray;
    }
```

<class> Her vises en hele den “section”, hvor vi bruger <class> til at fortælle “section” hvilken styling den skal have fra vores style.css. Ligeledes gør vi med <indput class og <teaxarea class etc.

```
<section class="createOpskrift">

  <form th:action="@{/createO(id=${user.id})}" method="post" th:object="${opskrift}" >

    <div align="center">
    <input class="create" type="text" th:field="*{title}" placeholder="Opskrift Titel"/>
    </div>

    <div class="emne">
    <div align="center">
```

```
        <input class="create" type="text" th:field="*{emne}" placeholder="Opskrift Emne"/>
        </div>
        <div align="center">
        <input class="create" type="text" th:field="*{weight}" placeholder="Emne V&aelig;gt"/>
        </div>

        <div align="center">
        <input class="create" type="text" th:field="*{tykkelse}" placeholder="Emne Tykkelse"/>
        </div>
        <div align="center">
        <textarea class="create" type="text" th:field="*{e_bemærkning}"
placeholder="Bem&aelig;rkninger Til Emnet" />
        </div>
        </div>
        <div class="tilberedning">
        <div align="center">
        <input class="create" type="text" th:field="*{tid}" placeholder="Tiberednings Tid"/>
        </div>
        <div align="center">
        <input class="create" type="text" th:field="*{temperatur}" placeholder="Tilberednings
Temperatur"/>
        </div>
        <div align="center">
        <textarea class="create" type="text" th:field="*{t_detaljer}" placeholder="Detaljer Omkring
Tilberedning"/>
        </div>
        </div>

        <div align="left">
        <input class="create" type="submit" value="Tilf&oslash;j Opskrift" style="width: 20%;
margin-bottom: 20px"/>
        </div>

    </form>

</section>
```

Forum

```
    @GetMapping("/forum")
```

```
    public String getForum(@RequestParam("userId") int userId, Model model) throws
SQLException{
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("forum", iRep.readAll());
        model.addAttribute("indlaeg", new Indlæg());
        model.addAttribute("kommentar", new Kommentar());
        return "forum";
    }
```

Når useren har kaldet getForum metoden i homeContolleren for at få returneret forum.html filen, bliver iRep.readAll() metoden kaldet så han kan læse alle indlæg og kommentar- attributer der er i vores database.

```
    public ArrayList<Indlæg> readAll(){
        ArrayList<Indlæg> forum = new ArrayList<>();
        SqlRowSet sqlRowSet1;
        sqlRowSet1 = jdbc.queryForRowSet("SELECT * FROM db.indlæg ");
        while(sqlRowSet1.next()) {
            int indlægId = sqlRowSet1.getInt("indlæg_id");
            ArrayList<Kommentar> kommentare = kRep.readAll(indlægId);
            forum.add(new Indlæg(sqlRowSet1.getInt("indlæg_id"), sqlRowSet1.getString("title"),
                    sqlRowSet1.getString("tekst"), sqlRowSet1.getString("username"),
sqlRowSet1.getString("dato"), kommentare));
        }
        return forum;
    }
```

I vores iRep.readAll() metode, bliver der til at starte med lavet en ArrayListe, som får variabel-navnet “Forum”, som vi senere gerne vil returnere til getForum metoden

Vi laver derefter en SqlRowSet instance med variable navnet sqlRowSet1
Som vi sætter lig med jdbc.queryForRowSet("SELECT * FROM db.indlæg ");
Hvilket betyder at vi query’er igennem vores jdbc database og selecter alle ting fra de rows, som ligger i db.indlæg og gemmer dem i sqlRowSet1
Efter det, looper vi igennem vores variabel med et while loop while(sqlRowSet1.next()) { og for hvert indlæg den finder i variablen udføre den det der står i loopet og når den har gjort det, finder den det næste indlæg.
I loopet laver vi en int og giver den variabel navnet indlægId og sætter den lig sqlRowSet1.getInt("indlæg_id"); hvilket betyder at vi får id’et for det givne indlæg som loopet har fundet

Hvert indlæg har en ArrayListe med kommentarer som hører til indlægget selv. For at få dem kalder vi en anden metode kRep.readAll(indlægId); og tilføjer det indlægId vi fandt tidligere i

iRep.readAll() metoden, som et parameter og sætter kRep.readAll(indlægId); lig med en ArrayListe ArrayList<Kommentar> kommentare

I vores kRep.readAll(indlægId); laver vi som det første en ny ArrayListe med variabel navnet kommentare, som vi senere gerne vil returnere tilbage til iRep.readAll().
Ligesom i iRep.readAll() metoden query'er vi efter de rows hvor sql statementet ("SELECT * FROM db.kommentare WHERE fk_indlæg_id = '" + indlægId + "'"); stemmer overens. I dette tilfælde er det alle de rows, hvor kommentarens fk_key_id er lig med det indlægId parameter som vi som vi fandt i iRep.readAll()

Vi looper derefter igennem sqlRowSet2 og for hver row der er i variablen, laver vi et nyt kommentar object som vi add'er til ArrayList<Kommentar> kommentare og retunerer listen tilbage til iRep.readAll()

```
   public ArrayList<Kommentar> readAll(int indlægId){
       ArrayList<Kommentar> kommentare = new ArrayList<>();
       SqlRowSet sqlRowSet2;
       sqlRowSet2 = jdbc.queryForRowSet( "SELECT * FROM db.kommentare WHERE
fk_indlæg_id = '" + indlægId + "'");
       while (sqlRowSet2.next()){
           kommentare.add(new Kommentar(sqlRowSet2.getInt("kommentar_id"),
                   sqlRowSet2.getString("tekst"), sqlRowSet2.getString("username"),
sqlRowSet2.getString("date")));
       }
       return kommentare;
   }
```

Når kRep.readAll(indlægId); har returneret en liste med kommentare tilbage til iRep.readAll() laver vi inde i while loopet (som er inde i iRep.readAll())et nyt Indlæg object som bliver add'et til forum listen når alle indlæg med hver deres kommentare er add'et til forum retunerer vi forum listen til getForum metoden i homeContolleren

I getForum metoden kalder vi model.addAttribute("forum", iRep.readAll()); som gør at listen ryger videre til forum.html filen med variabel navnet "forum".

```
<section class="readIndlæg">
    <div class="readIndlæg1" th:each="indlaeg : ${forum}">
        <div th:if="${user.role} eq 'admin'" align="right">
            <a th:href="@{/deleteI(inlaegId=${indlaeg.getId()}, userId=${user.id})}" onclick="return
confirm('Er du sikker på at du vil slette dette Indlæg?');">X</a>
        </div>
```

Inde i vores forum.html modtager vi vores forum ArrayListe som vi looper igennem med et each loop: `th:each="indlaeg : ${forum}"`

For hvert indlæg der bliver fundet i ArrayListen udskriver den alle indlægges attributter:

```
        <h4 class="username1" th:text="${indlaeg.username}"/>
        <p th:text="${indlaeg.dato}"/>
        <div class="readIndlæg2">
            <div class="readIndlæg3">
                <h3 th:text="${indlaeg.title}"/>
                <pre th:text="${indlaeg.tekst}"/>
            </div>
```

Hvert indlæg har en kommentarListe som også skal udskrives. For at vi kan få udskrevet alle kommentare der er i listen. Looper vi igen med et each loop og udskriver alle attributter for hver kommentar der er i listen

```
            <div class="readKommentare1" th:each="kommentar :
${indlaeg.getKommentareListe()}">
                <div th:if="${user.role} eq 'admin'" align="right">
                    <a th:href="@{/deleteK(kommentarId=${kommentar.getId()}, userId=${user.id})}"
onclick="return confirm('Er du sikker på at du vil slette denne Kommentar?');">X</a>
                </div>
                <h4 class="username2" th:text="${kommentar.username}"/>
                <p th:text="${kommentar.dato}" />
                <pre class="kommentar" th:text="${kommentar.tekst}"/>
            </div>
```

For hvert indlæg der bliver loopet ud, bliver der tilføjet en form med det specifikke indlægs id. Det gør at hver kommentar der bliver skrevet i vores forum, bliver tilføjet til et specifik indlæg. Det indlægs id Requester vi i post metoden, hvor der er en create metode som modtager id'et samt den nye kommentar og merger dem sammen i databasen:

```
            <form class="writeKommentar" th:action="@{/kommentar(userId=${user.id},
indlaegId=${indlaeg.id})}" method="post" th:object="${kommentar}">
                <textarea class="writeKommentar" type="text" th:field="*{tekst}" placeholder="Skriv
en kommentar"/>
                <div>
                <input type="submit" value="Post Kommentar" style="width: 25%; margin: 0; " />
                </div>
            </form>
        </div>
    </div>
  </section>
```

Hvad er JDBC?

JDBC er en Java API (application Program Interface) som hjælper med at skrive Java applikationer som kan bruges til 3 forskellige aktiviteter indenfor programmering:
1 at skabe en connection til en bestemt data source - dette kunne være en database
2 at sende queries og opdatere statements til databasen
3 modtage og forarbejde diverse resultater fra databasen

JDBC bruger vi til at skabe en forbindelse til vores database inde i application.properties således:

```
spring.datasource.url=jdbc:mysql://localhost:3306
```

vi opretter forbindelse til mysql databasen på localhost:3306.

For at kunne bruge de metoder der findes i JDBC, skal det først istansieres. Det gør vi ved at skrive:

```
@Autowired
private JdbcTemplate jdbc;
```

når dette er gjort, kan vi begynde at bruge de metoder, som JDBC har. et eksempel på en metode kunne være:

```
public void delete(int id){
```

```
    jdbc.update("DELETE FROM db.users WHERE user_id = '" + id + "'");
}
```

Her bruger vi jdbc.update til at delete en user fra db.user databasen hvor user_id = id

Normalisering af databasen

Databasen i dette projekt skulle være på den 3.normalform. Formålet med normalisering er primært at reducere redundans. Måden man opnår 3.normalform er ved at gå systematisk igennem sin database og rette til så det først opnår 1.form så 2.form og til sidst 3.form, da de forudsætter at man har opnået den forrige form for at opnå den næste.

For at opnå 1.normalform skal databasen opfylde et simpelt krav. Der må for hver attribut i relationen kun være en værdi. Et eksempel på hvordan det ikke må være er hvis en attribut kaldet farve har værdien {grøn,orange}. Der må kun være en værdi af gangen, altså enten {grøn} eller {orange}.
Som det kan ses nedenfor i modellen af vores database tabeller, så opfylder vores Database dette krav, da der for hver attribut kun er én værdi.

For at opnå 2.normalform forudsættes det at man allerede er på 1.normalform. Derudover så kræves der fuld afhængighed af primærnøglen for attributterne. Dette er især relevant hvis man har en sammensat primærnøgle. Hvis man eksempelvis har en relation med en sammensat primærnøgle XY bestående af attributterne X og Y, så må der ikke findes en attribut A der kun er funktionel afhængig af enten X eller Y.
I vores tabeller er der dog ingen sammensatte primærnøgler hvilket gør det nemt at sørge for at 2.normalform er opnået. Alle attributterne afhænger af deres tilhørende primærnøgle så fuld funktionel afhængighed er opnået.

3.normalform forudsætter at ens relationer er på 2.normalform. Derudover kræves der at der ikke findes nogen transitiv afhængighed. Transitiv afhængighed er når en attribut har funktionel afhængighed af en anden attribut, som ikke er en primærnøgle eller indgår i primærnøglen. Eksempelvis, hvis man har en primærnøgle P, som har en attribut X der er afhængig af P, og en anden attribut Y som er funktionel afhængig af X men ikke P, så er Y transitiv afhængig af P. Vores tabeller er på 3.normalform , da der ikke findes nogen transitiv afhængighed.

Emner					
emne_id	type	vægt	tykkelse	bemærkning	fk_opskrift_id
1	Ko	500g	2cm	"økologisk ko"	1

Indlæg				
inlæg_id	title	tekst	dato	fk_user_id
1	"Kogt Ko"	"smager godt"	xx/yy-zzzz	1

Kommentare				
kommentar_id	tekst	date	fk_indlæg_id	fk_user_id
1	"det var godt"	xx/yy-zzzz	1	1

Opskrifter				
opskrift_id	karakter	vurdering	dato	fk_user_id
1	9	"skide godt"	xx/yy-zzzz	1

Tilbedredninger				
Tilbedredninger_id	tid	temp	e_detaljer	fk_emne_id
1	2 timer 30min	57 grader	"vær agtpågivende"	1

Users				
user_id	username	password	email	role
1	dellekaj420	12345	ko@mail.dk	admin

Interface

Vi bruger interfaces ved, at vi implementerer interfacet i vores repositories og derved får skabt nogle konventioner om hvad metoderne i vores repositories skal hedder og hvilke data typer de skal indeholde.

Interface kan være optimalt, hvis man programmere mange på samme program, da konventionerne om hvad metoderne skal indeholde, vil holde alle programmører på samme sti og derved få mindre fejl.

Eksempel fraUserRep klassen i Foodie Log:

```
public class UserRep implements IUserRep
```

Eksempel fra vores controller i Foodie Log,

```
IUserRep uRep = new UserRep();
```

Eksempel fra UserRep og IUserRep i vores Foodie Log, hvor vi overskriver metoden men holder os inden for den konvention, der er lavet i interfacet, om at metoden skal indeholde “User u”.
UserRep klassen:

```
@Override
public void create(User u)
```

IUserRep klassen:

```
void create(User u)
```

Vi påtænker på sigt at udvide vores software og have flere aktører der vil gøre brug af samme inface og derved gøre brug af polymorphism.
Polymorphism er et græsk ord og betyder “mange typer/former”.

Et fiktivt eksempel på brug af polymorphism er, at hvis man skulle lave et computerspil og havde besluttet, at alle de aktører, der er med i det gældende software, skal kunne råbe,men at de skulle ikke råbe på samme måde.

Infacet ville så se cirka sådan ud:

```
Public interface Crud
{
        Void shout(String shout)
}
```

Og et eksempel på de metoder i de respektive klasser for aktørerene ville se cirka således ud:

```
Public class Rubber implements Crud
{
```

```
@override
        public void shout ()
        {
        system.out.println(“I’m gonna rub you”)
        }

Public class Police implements Crud
{
        @override
        public void shout()
        {
        system.out.println(“STOOOOOP - Police”)
}

Public class Run
{
        Public static void main
        {
        Crud p = Police;
        Crud r = Rubber;
        p.shout()
        r.shout()
        }

}
```

Model

Model er et interface der implementere 4 klasser: BindingAwareCurrentModel, BindingAwareModelMap,ConcurrentModel, ExtendedModelMap og RedirectAttribuesModelMap og indeholder følgende metoder: addAllAttributes(), addAttribute(), asMap(), containsAttribute() og mergeAttributes().

Vi har i Foodie Log kun gjort brug af addAtribute(), som vi har brugt til at få spring mvc til at forstå, hvad vi gerne vil have sendt til vores front end.

Eksempel:

```
@GetMapping("/homepage")
```

```
public String getHomepage(@RequestParam("id") int id, Model model){
```

Her sender vi et nyt objekt af opskrift som skal læses i homepage.html.
Objektet bliver læst ved, at den grønne tekst der står i citationstegn, bliver sat i tuborgklammer med et $ foran.
Se html (a)

```
  model.addAttribute("opskrift", new Opskrift());
```

Her læser vi en specifik bruger og sender den videre, som matcher det id vi fanger med @RequestParam.
Se html (b)

```
  model.addAttribute("user", uRep.read(id));
```

Her kalder vi en metode som reader alle opskrifter for en specifik bruger hvor opskriftens fk matcher users id og returnere derefter en ArrayListe, hvor opskrifterne ligger i. Model.addtribute sender derefter denne ArrayListe videre til homepage med variabelnavnet “opskrifter” users id for vi i gennem @RequestParam .
Se html (c)

```
  model.addAttribute("opskrifter", oRep.readAll(id));

  return "homepage";
}
```

Her er et udsnit af vores homepage.html

```
      <section class="createOpskrift">
```

Det er her vi fanger (a) og (b) i ”${user.id}” og “${opskrift}”

```
          <form th:action="@{/createO(id=${user.id})}" method="post" th:object="${opskrift}">
```

Og så tilføjer vi attributter til vores opskrift object ved at skrive *{title} som så bliver udfyldt af user vi af en form i vores homepage.html

```
<input class="signup" type="text" th:field="*{title}" placeholder="Opskrift Titel"/>
```

Det er her vi fanger (c) ”${opskrifter}” og udskriver de valgte attributter via en Arrayliste vi har lavet inde i vores readAll methode og iterere så igennem listen med et for each loop.

```
              <tr th:each="opskrift : ${opskrifter}">
                  <td th:text="${opskrift.title}"/>
                  <td th:text="${opskrift.emne}"/>
                  <td th:text="${opskrift.karakter}"/>
                  <td >
```

Teknik

Prepared Statements

Når en database modtager et SQL statement (f.eks SELECT * FROM db.user_table;) tjekker dens engine først for syntax fejl og derefter finder den en måde at eksekvere det sendte SQL statement, på den mest effektive måde.
Når databasens engine har lavet en forespørgelse plan, kan den skrive til databasen og det er her Prepared statements kommer ind i billedet.

Et Prepared statement sender nemlig det samme SQL statement afsted hver gang (?,?,?,?) så det er også den samme forespørgelse plan der bliver kørt, hver gang et Prepared statement bliver eksekveret.
Efterfølgende bliver spørgsmålstegnene udskiftet med værdier via nogle SetMetoder der findes i Prepared statement klassen som java har defineret.

I et almindeligt SQL statement (f.eks (“Update blogDB.user_table SET password = '" + newPassword + "' WHERE id = '" + id + “”), ville man skulle finde en ny vej, hver gang på grund af variablen newPassword, der vil have en ny værdi hver gang SQL statement bliver eksekveret og derved bruge mere CPU kraft og gøre programmet langsommere.

Prepared statements kan også bruges i forhold til SQL Injections og derved beskytte vores indhold i databasen.
SQL injections kan kun foregå hvis man sætter en værdi direkte ned databasen, som når en bruger af programmet tilføjer input information, så som når user indtaster sit nye password for at opdatere det (f.eks.(“Update blogDB.user_table SET password = '" + newPassword + "' WHERE id = '" + id + “”)
SQL injection kan foregå fordi den SQL streng vi sender afsted ikke er lukket (“ ‘ “) og man vil derved kunne fortsætte strengen og skrive mere til det allerede eksisterende SQL statement, som f.eks DROP databasenavn.

Her er et eksempel på Prepared statement vi bruger i vores opgave når vi opretter en bruger.

```
public void create(User u) throws SQLException {
  Connection myConn = null;
  PreparedStatement myStmt = null;


  try {
     // 1. Get a connection to database
     myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306", "root", "");

     // 2. Prepare statement
     myStmt = myConn.prepareStatement("INSERT INTO db.user_table(user_name,
user_email, user_password, user_role) VALUES(?,?,?,?,?)");


     // 3. Set the parameters
     myStmt.setString(1, u.getName());
     myStmt.setString(2, u.getPassword());
     myStmt.setString(3, u.getEmail());
     myStmt.setString(4, u.getRole());


  } catch (Exception exc) {
     exc.printStackTrace();
  } finally {

     if (myStmt != null) {
        myStmt.close();
     }

     if (myConn != null) {
        myConn.close();
     }
  }
}
```

Spring Security

Vi har prøvet, at implementer spring security, da vores program ikke er helt sikkert.
Hvis man F.eks har viden om hvilket id en user har, kan det lade sig gøre at komme ind på hans profil uden at logge ind, ved at skrive følgende for den bruger der har et userid på 23:

http://localhost:8000/updateU?userId=23

Vi nåede til at sætte en klasse op der konfigurerede hvem der måtte komme ind på hvilke sider, hvor vi hardcodede en bruger og så kaldte den bruger med nogle get-metoder.
Det så sålede ud:

```
@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

  User u = new User();

  @Autowired //her definere vi hvem der skal identificeres og hvilken rolle de skal have

  public void configureGlobalSecurity(AuthenticationManagerBuilder auth)

        throws Exception {

    auth.inMemoryAuthentication().withUser(u.getUsername()).password(u.getPassword())

          .roles(u.getRole());


  }
```

```
    @Override
    protected void configure(HttpSecurity http) throws Exception //Throws smider answaret videre til den der kalder metoden
    {
        http.authorizeRequests()
                // her difineres login url
                .antMatchers("/login").permitAll()
                // her defineres hvilke html sites der IKKE må tilgås uden et login og den rigtige rolle

.antMatchers("/*homepage*/**","/*updateOpskrift*/**","/*updateUser*/**").access("hasRole('USER')").and()
                // HVIS DU IKKE HAR DEN RIGTIGE ROLLE BLIVER MAN SENDT TIL .formlogin()
                .formLogin()
                //Start siden efter login
                .defaultSuccessUrl("/homepage.html").and().exceptionHandling()
                .accessDeniedPage("/access-denied");
    }
}
```

Vi vil på et senere tidspunkt få implementeret Spring Security, men da det ikke er en del af pensum har vi prioriteret andre ting, da vi kunne se, at det ville tage lidt længere tid end forventet.
Det vi formoder, vi mangler er en forbindelse til vores brugere i databasen, hvorefter de skal godkendes og så til sidst få user gemt i en form for hukommelse, så man er godkendt i den tid man er logget ind.

Vi har slettet vores SecurityConfig klasse igen, da den ville spærre for de sider, som den allerede godkendte bruger (via vores nuværende login metode) skal have adgang til.

Amazon RDS og Elastik bean stalk

Vi har uploadet vores software til elastik bean stalk og vores database til RDS til dette link:
http://foodielog-env-1.zawy22dxmp.eu-central-1.elasticbeanstalk.com/
Da vi testede programmet fandt vi ud af at vores forum ikke virkede som det skulle. Det var fordi at vores local server port ikke gjorde forskel på store og små bogstaver, hvilket AWS serveren gjorde. Vi rettede selvfølge fejlen og programmet kørte nu fejlfrit.
Vi kan desværre ikke lade programmet blive online da den server connection vi har lært at lave på AWS og de security groups vi havde i vores database "RDS" var heller ikke sikre da vi var nødt til at åbne for inbound og outbound. Vi vil dog demonstrere vores software online til eksamen.

Vejledning til Foodie-log

Her er et link til en lille video om hvordan man kommer i gang med vores software.
Software og databasen er desværre ikke sikker nok til at vi kan lade det blive oppe i skyen.

https://youtu.be/xbhF1Y9TzTc

Konklusion

Brugen af Foodie-log hos FoodCare vil være med til at effektivisere og udbrede viden og bedre opskrifter for sous vide madlavning. Hvilket ifølge vores observeringer vil føre til øget salg af Foodcare's sous vide produkter, da kundernes engagement forøges i takt med at Foodie-log bliver mere populært.

Ud fra vores analyser kan vi konkludere, at en nem og overskuelig samling af ens opskrifter og erfaringer med sous vide madlavning, som giver slut user en god erfaring med netop det som FoodCare kan tilbyde, kan styrke firmaets image og salg.
På forummet kan man sammenligne ens egne oplevelser med andre. Vi mener at dette er positivt, da det styrker kommunikationen mellem brugerne.

Da systemet er baseret på at der er eksisterende brugere, kræver det at der er brugere for at opnå den optimale effekt. Uden brugere er det svært at sammenligne erfaringer.
FoodCare er en ung virksomhed som har ressourcestærke konkurrenter. Der kan opstå finansielle og salgs forhindringer, hvis populariteten af sous vide produkter øges.

Når man hører om andre folks erfaringer vil man blive inspireret til at prøve det selv. Det udstyr man så har brug for vil Foodcare glædeligt sælge til en.
FoodCare's samarbejde med bloggere/restauranter er en del af en marketing strategi som i fremtiden vil gøre det nemmere for FoodCare at sælge sine produkter. Salg til virksomhed som skal bruge større ordre, er også med til at sørge for at der er plads til innovation i virksomheden. Vi regner med at indførslen af Foodie-log, vil have en positiv effekt på firmaets popularitet og salg.

Reflektioner

I løbet af projektarbejdet har vi delt diverse arbejdsopgaver op efter behov og individuelle styrker. Dette har fungeret fint, dog har der opstået få problemer ved sammenførelse af de forskellige dele af projektet.
Vi blev enige om rimelig tidligt, at opgaven skulle skrives på dansk. Det var nemmere for os at skrive og ville gøre forståelsen nemmere for alle indblandede og at hjemmesiden vi skriver for, også er dansk. Desværre medførte det, at koden også delvist skulle skrives på dansk. Det gav en masse compile fejl, som kunne være undgået, hvis det var skrevet på engelsk.
Som tidligere beskrevet, har vi prøvet at implementere noget security i vores system. Desværre har vi ikke fået det til at virke, men hvis vi ville at, vores system skulle implementeres på FoodCare.dk ville det være et vigtigt krav.
For at få vores database online, har vi brugt amazone's AWS system. Vi havde nogle problemer til sidst i forløbet, hvor vi havde fået assistance til opsætningen, så vi var sikre på, at det var gjort korrekt. Desværre var det sat op på som en ikke-gratis server - det kostede os en del ressourcer, og er en fejl, som vi ikke gør igen.
Vi har også haft problemer med den browser vi har brugt, som er google chrome. Vi fandt ud af, at hvis styling tags er tomme, så virker de ikke på browseren. Vi har været nødsaget til at bruge firefox for at kunne få stylingen i systemet til at virke. Vi har også erfaret, at den samme browser, som f.eks google chrome er giver at meget forskelligt output på Version 62.0.3202.94 på Version 63.0.3239.84.
Efter en længerevarende dialog med Foodcares ejer er vi blevet bevidst om, at han var i den tro ,at der er også var en applikation til telefonmarkedet. Vi har nok ikke været helt klare i vores udmelding med, hvad en web-applikation lige nøjagtig er. Vi brugte termen app flere gange i dialogen med Foodcare ejeren og det har fået ham til at tror at det også var en telefon applikation.

Vi vil meget gerne på sigt lave en applikation til telefonmarkedet, da det ville være meget mere brugbart for ejeren af Foodcare.

KildeKode

Følgende er alt vores kildekode:

Homecontroller.java (ansvarlig: Patrick)

```
package dk.kea.controllers;

import dk.kea.model.Interface.IInlægRep;
import dk.kea.model.Interface.IKommentarRep;
import dk.kea.model.Interface.IOpskrifterRep;
import dk.kea.model.Interface.IUserRep;
import dk.kea.model.entities.Indlæg;
import dk.kea.model.entities.Kommentar;
import dk.kea.model.entities.Opskrift;
import dk.kea.model.entities.User;
import dk.kea.model.repository.IndlægRep;
import dk.kea.model.repository.KommentarRep;
import dk.kea.model.repository.OpskriftRep;
import dk.kea.model.repository.UserRep;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestParam;

import java.sql.SQLException;

// her getter vi alle vores html sider
@Controller
public class HomeController {

// her Autowire vi vore repositories med vores interfaces så vi kan
//bruge vores metoder fra repositorierne
   @Autowired
   IInlægRep iRep = new IndlægRep();
```

```
    @Autowired
    IOpskrifterRep oRep = new OpskriftRep();

    @Autowired
    IUserRep uRep = new UserRep();

    @Autowired
    IKommentarRep kRep = new KommentarRep();

// getter til vores index side
    @GetMapping("/")
    public String index(Model model){

        model.addAttribute("opskrifter", oRep.readAll(1));
         return "index";
    }
// getter til vores homepage
    @GetMapping("/homepage")
    public String getHomepage(@RequestParam("userId") int id, Model
model){

         model.addAttribute("opskrift", new Opskrift());
         model.addAttribute("user", uRep.read(id));
         model.addAttribute("opskrifter", oRep.readAll(id));

         return "homepage";
    }
// getter til vores signup side
    @GetMapping("/signUp")
    public String signup(Model model){

         model.addAttribute("user", new User());
         model.addAttribute("opskrifter", oRep.readAll(1));
         return "signUp";
    }

    @GetMapping("/deleteO")
    public String deleteO(@RequestParam("opskriftId") int opskriftId,
@RequestParam("userId") int userId, Model model){
         oRep.delete(opskriftId);
         model.addAttribute("opskrift", new Opskrift());
         model.addAttribute("user", uRep.read(userId));
         model.addAttribute("opskrifter", oRep.readAll(userId));
```

```
        return "homepage";
    }

    @GetMapping("/updateO")
    public String getUpdateO(@RequestParam("opskriftId") int
opskriftId, @RequestParam("userId") int userId, Model model){

        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("opskrift", oRep.read(opskriftId));
        return "updateOpskrift";
    }

    @GetMapping("/updateU")
    public String getUpdateU(@RequestParam("userId") int userId, Model
model){

        model.addAttribute("user", uRep.read(userId));
        return "updateUser";
    }

    @GetMapping("/deleteU")
    public String deleteU(@RequestParam("userId") int userId){

        uRep.delete(userId);
        return "index";
    }

    @GetMapping("/forum")
    public String getForum(@RequestParam("userId") int userId, Model
model) throws SQLException{



            model.addAttribute("user", uRep.read(userId));
            model.addAttribute("forum", iRep.readAll());
            model.addAttribute("indlaeg", new Indlæg());
            model.addAttribute("kommentar", new Kommentar());
            return "forum";
        }


    @GetMapping("/logout")
    public String logout(){
```

```
        return "redirect:/";
    }

    @GetMapping("/deleteI")
    public String deleteI(@RequestParam("inlaegId") int indlægId,
@RequestParam("userId") int userId, Model model){

        iRep.delete(indlægId);
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("forum", iRep.readAll());
        model.addAttribute("indlaeg", new Indlæg());
        model.addAttribute("kommentar", new Kommentar());
        return "forum";
    }

    @GetMapping("/deleteK")
    public String deleteK(@RequestParam("kommentarId") int
kommentarId, @RequestParam("userId") int userId, Model model){

        kRep.delete(kommentarId);
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("forum", iRep.readAll());
        model.addAttribute("indlaeg", new Indlæg());
        model.addAttribute("kommentar", new Kommentar());
        return "forum";
    }


}
```

postController.java(ansvarlig: Patrick)

```
package dk.kea.controllers;

import dk.kea.model.Interface.IInlægRep;
import dk.kea.model.Interface.IKommentarRep;
import dk.kea.model.Interface.IOpskrifterRep;
import dk.kea.model.Interface.IUserRep;
import dk.kea.model.entities.Indlæg;
import dk.kea.model.entities.Kommentar;
import dk.kea.model.entities.Opskrift;
import dk.kea.model.entities.User;
```

```
import dk.kea.model.repository.IndlægRep;
import dk.kea.model.repository.KommentarRep;
import dk.kea.model.repository.OpskriftRep;
import dk.kea.model.repository.UserRep;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.sql.SQLException;

// Controller til alle vores post metoder
@Controller
public class PostController {

   @Autowired
   IUserRep uRep = new UserRep();

   @Autowired
   IOpskrifterRep oRep = new OpskriftRep();

   @Autowired
   IInlægRep iRep = new IndlægRep();

   @Autowired
   IKommentarRep kRep = new KommentarRep();

   @PostMapping("/login")
   public String login(@ModelAttribute User u, Model model) throws
SQLException{

           u = uRep.loginDB(u);
           if (u != null) {

               model.addAttribute("opskrift", new Opskrift());
               model.addAttribute("user", u);
               model.addAttribute("opskrifter",
oRep.readAll(u.getId()));
               return "homepage";
           }
       return "redirect:/";
```

```
    }

    @PostMapping("/signUp")
    public String createAccount(@ModelAttribute User u) throws
SQLException{

        uRep.create(u);
        return "redirect:/";
    }

    @PostMapping("/createO")
    public String createO(@RequestParam("id") int id, Opskrift o,
Model model) throws SQLException{

        oRep.create(o, id);
        model.addAttribute("opskrift", new Opskrift());
        model.addAttribute("user", uRep.read(id));
        model.addAttribute("opskrifter", oRep.readAll(id));
        return "homepage";
    }

    @PostMapping("/updateO")
    public String updateO(@RequestParam("userId") int userId,
@RequestParam("opskriftId") int opskriftId, Opskrift o, Model model)
throws SQLException{

        oRep.update(opskriftId, o);
        model.addAttribute("opskrift", new Opskrift());
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("opskrifter", oRep.readAll(userId));
        return "homepage";
    }

    @PostMapping("/updateU")
    public String UpdateU(@RequestParam("userId") int userId, User u,
Model model) throws SQLException{

        uRep.update(userId, u);
        model.addAttribute("opskrift", new Opskrift());
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("opskrifter", oRep.readAll(userId));
        return "homepage";
    }
```

```
    @PostMapping("/indlaeg")
    public String createI(@RequestParam("userId") int userId, Indlæg
i, Model model) throws SQLException {


            iRep.create(i, userId);
            model.addAttribute("user", uRep.read(userId));
            model.addAttribute("forum", iRep.readAll());
            model.addAttribute("indlaeg", new Indlæg());
            model.addAttribute("kommentar", new Kommentar());
            return "forum";
        }


    @PostMapping("/kommentar")
    public String createK(@RequestParam("userId") int userId,
@RequestParam("indlaegId") int indlaegId, Kommentar k, Model model)
throws SQLException{

        User u = uRep.read(userId);

        kRep.create(k, u.getUsername(), indlaegId);
        model.addAttribute("user", uRep.read(userId));
        model.addAttribute("forum", iRep.readAll());
        model.addAttribute("indlaeg", new Indlæg());
        model.addAttribute("kommentar", new Kommentar());
        return "forum";
    }


}
```

indlæg.java(Ansvarlig Peter & Simon)

```
package dk.kea.model.entities;

import java.util.ArrayList;

public class Indlæg {

    private int id;
```

```
    private String title;
    private String tekst;
    private String username;
    private String dato;
    private ArrayList<Kommentar> kommentare;

    public Indlæg() {
    }

    public Indlæg(int id, String title, String tekst, String username,
String dato, ArrayList<Kommentar> kommentare) {
        this.id = id;
        this.title = title;
        this.tekst = tekst;
        this.username = username;
        this.dato = dato;
        this.kommentare = kommentare;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getTekst() {
        return tekst;
    }

    public void setTekst(String tekst) {
        this.tekst = tekst;
    }
```

```
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getDato() {
        return dato;
    }

    public void setDato(String dato) {
        this.dato = dato;
    }

    public ArrayList<Kommentar> getKommentareListe() {
        return kommentare;
    }

    public void setKommentareListe(ArrayList<Kommentar> kommentare) {
        this.kommentare = kommentare;
    }

    @Override
    public String toString() {
        return id + title + tekst + username + dato + kommentare;
    }
}
```

kommentar.java(Ansvarlig: Peter & Simon)

```
package dk.kea.model.entities;

public class Kommentar {

    private int id;
    private String tekst;
    private String username;
    private String dato;

    public Kommentar() {
```

```
    }

    public Kommentar(int id, String tekst, String username, String
dato) {
        this.id = id;
        this.tekst = tekst;
        this.username = username;
        this.dato = dato;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTekst() {
        return tekst;
    }

    public void setTekst(String tekst) {
        this.tekst = tekst;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getDato() {
        return dato;
    }

    public void setDato(String dato) {
        this.dato = dato;
    }

    @Override
```

```
    public String toString() {
        return id + tekst + username + dato;
    }
}
```

opskrift.java(Ansvarlig Peter & Simon)

```
package dk.kea.model.entities;



import java.util.Date;


public class Opskrift {

    private int id;
    private String title;
    private Date dato;
    private String karakter;
    private String vurdering;
    private String emne;
    private String weight;
    private String tykkelse;
    private String e_bemærkning;
    private String tid;
    private String temperatur;
    private String t_detaljer;

    public Opskrift() {
    }

    public Opskrift(int id, String title, Date dato, String karakter,
String vurdering, String emne, String weight, String tykkelse, String
e_bemærkning, String tid, String temperatur, String t_detaljer) {
        this.id = id;
        this.title = title;
        this.dato = dato;
        this.karakter = karakter;
        this.vurdering = vurdering;
        this.emne = emne;
        this.weight = weight;
```

```
        this.tykkelse = tykkelse;
        this.e_bemærkning = e_bemærkning;
        this.tid = tid;
        this.temperatur = temperatur;
        this.t_detaljer = t_detaljer;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public Date getDato() {
        return dato;
    }

    public void setDato(Date dato) {
        this.dato = dato;
    }

    public String getKarakter() {
        return karakter;
    }

    public void setKarakter(String karakter) {
        this.karakter = karakter;
    }

    public String getVurdering() {
        return vurdering;
    }
```

```
public void setVurdering(String vurdering) {
    this.vurdering = vurdering;
}

public String getEmne() {
    return emne;
}

public void setEmne(String emne) {
    this.emne = emne;
}

public String getWeight() {
    return weight;
}

public void setWeight(String weight) {
    this.weight = weight;
}

public String getTykkelse() {
    return tykkelse;
}

public void setTykkelse(String tykkelse) {
    this.tykkelse = tykkelse;
}

public String getE_bemærkning() {
    return e_bemærkning;
}

public void setE_bemærkning(String e_bemærkning) {
    this.e_bemærkning = e_bemærkning;
}

public String getTid() {
    return tid;
}

public void setTid(String tid) {
    this.tid = tid;
}
```

```
    public String getTemperatur() {
        return temperatur;
    }

    public void setTemperatur(String temperatur) {
        this.temperatur = temperatur;
    }

    public String getT_detaljer() {
        return t_detaljer;
    }

    public void setT_detaljer(String t_detaljer) {
        this.t_detaljer = t_detaljer;
    }

    @Override
    public String toString() {
        return id + title + dato + karakter + vurdering + emne + weight
+ tykkelse + e_bemærkning + tid + temperatur + t_detaljer;
    }
}
```

user.java(ansvarlig: Simon & Peter)

```
package dk.kea.model.entities;

public class User {

    private int id;
    private String username;
    private String password;
    private String email;
    private String role = "user";

    public User() {
    }

    public User(int id, String username, String password, String
email, String role) {
        this.id = id;
```

```
        this.username = username;
        this.password = password;
        this.email = email;
        this.role = role;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
```

```
        this.role = role;
    }

    @Override
    public String toString() {
        return id + username + password + email + role;
    }
}
```

iIndlægRep.java(ansvarlig: Patrick & Rasmus)

```
package dk.kea.model.Interface;

import dk.kea.model.entities.Indlæg;

import java.sql.SQLException;
import java.util.ArrayList;

public interface IInlægRep {

    ArrayList<Indlæg> readAll();

    void create(Indlæg i, int userId) throws SQLException;

    void delete(int indlægId);
}
```

iKommentarRep(Ansvarlig: Patrick & Rasmus)

```
package dk.kea.model.Interface;

import dk.kea.model.entities.Kommentar;

import java.sql.SQLException;
import java.util.ArrayList;

public interface IKommentarRep {

    ArrayList<Kommentar> readAll(int indlægId);

    void create(Kommentar k, String username , int indlægId) throws
SQLException;
```

```
    void delete(int kommentarId);
}
```

iOpskrifterRep.java(Ansvarlig: Rasmus & Patrick)

```
package dk.kea.model.Interface;

import dk.kea.model.entities.Opskrift;

import java.sql.SQLException;
import java.util.ArrayList;

public interface IOpskrifterRep {

    ArrayList<Opskrift> readAll( int id);

    void create(Opskrift o, int id) throws SQLException;

    void delete(int id);

    Opskrift read(int id);

    void update(int id, Opskrift o) throws SQLException;
}
```

iUserRep.java(Ansvarlig Rasmus & Patrick)

```
package dk.kea.model.Interface;

import dk.kea.model.entities.User;

import java.sql.SQLException;

public interface IUserRep {

    void create(User u) throws SQLException;

    User read(int id);

    User loginDB(User u) throws SQLException;
```

```
    void update(int id, User u) throws SQLException;

    void delete(int id);
}
```

indlægRep.java(Ansvarlig Patrick & Rasmus)

```
package dk.kea.model.repository;

import dk.kea.model.Interface.IInlægRep;
import dk.kea.model.Interface.IKommentarRep;
import dk.kea.model.Interface.IUserRep;
import dk.kea.model.entities.Indlæg;
import dk.kea.model.entities.Kommentar;
import dk.kea.model.entities.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.rowset.SqlRowSet;
import org.springframework.stereotype.Repository;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;

@Repository
public class IndlægRep implements IInlægRep{

    @Autowired
    IKommentarRep kRep = new KommentarRep();

    @Autowired
    IUserRep uRep = new UserRep();

    @Autowired
    private JdbcTemplate jdbc;

    public void create(Indlæg i, int userId) throws SQLException{

        Connection myConn = null;
        PreparedStatement myStmt = null;
```

```
        User u = uRep.read(userId);

        try {
            // 1. Get a connection to database
            myConn =
DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            // 2. Prepare statement
            myStmt = myConn.prepareStatement("INSERT INTO db.indlæg
(title, tekst, username) VALUES (?,?,?) ");


            // 3. Set the parameters
            myStmt.setString(1, i.getTitle());
            myStmt.setString(2, i.getTekst());
            myStmt.setString(3, u.getUsername());

            myStmt.executeUpdate();


        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
        }

    }

    public ArrayList<Indlæg> readAll(){


        ArrayList<Indlæg> forum = new ArrayList<>();
```

```
        SqlRowSet sqlRowSet1;
        sqlRowSet1 = jdbc.queryForRowSet("SELECT * FROM db.indlæg ");
        while(sqlRowSet1.next()) {

            int indlægId = sqlRowSet1.getInt("indlæg_id");

            ArrayList<Kommentar> kommentare = kRep.readAll(indlægId);

            forum.add(new Indlæg(sqlRowSet1.getInt("indlæg_id"),
sqlRowSet1.getString("title"),
                    sqlRowSet1.getString("tekst"),
sqlRowSet1.getString("username"), sqlRowSet1.getString("dato"),
kommentare));
        }
        return forum;
    }

    public void delete(int indlægId){

        jdbc.update("DELETE FROM db.indlæg WHERE indlæg_id = '" +
indlægId + "'");

    }


}
```

kommentarRep.java(Ansvarlig: Patrick & Rasmus)

```
package dk.kea.model.repository;

import dk.kea.model.Interface.IKommentarRep;
import dk.kea.model.Interface.IUserRep;
import dk.kea.model.entities.Kommentar;
import dk.kea.model.entities.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.rowset.SqlRowSet;
import org.springframework.stereotype.Repository;

import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;

@Repository
public class KommentarRep implements IKommentarRep{

   @Autowired
   private JdbcTemplate jdbc;

   IUserRep uRep = new UserRep();

   public void create(Kommentar k, String username, int indlaegId)
throws SQLException{

        Connection myConn = null;
        PreparedStatement myStmt = null;


        //User u = uRep.read(userId);

        System.out.println(k + "gooooooddaaaag");

        try {
            // 1. Get a connection to database
            myConn =
DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            // 2. Prepare statement
            myStmt = myConn.prepareStatement("INSERT INTO db.kommentare
(tekst, fk_indlæg_id, username) VALUES (?,?,?) ");


            // 3. Set the parameters
            myStmt.setString(1, k.getTekst());
            myStmt.setInt(2, indlaegId);
            myStmt.setString(3, username);

            myStmt.executeUpdate();


        } catch (Exception exc) {
```

```
            exc.printStackTrace();
        } finally {

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
        }

    }

    public ArrayList<Kommentar> readAll(int indlægId){

        ArrayList<Kommentar> kommentare = new ArrayList<>();

        SqlRowSet sqlRowSet2;
        sqlRowSet2 = jdbc.queryForRowSet( "SELECT * FROM db.kommentare
WHERE fk_indlæg_id = '" + indlægId + "'");
        while (sqlRowSet2.next()){

            kommentare.add(new
Kommentar(sqlRowSet2.getInt("kommentar_id"),
                    sqlRowSet2.getString("tekst"),
sqlRowSet2.getString("username"), sqlRowSet2.getString("date")));
        }

        return kommentare;
    }

    public void delete(int kommentarId){

        jdbc.update("DELETE FROM db.kommentare WHERE kommentar_id = '"
+ kommentarId + "'");

    }
}
```

opskriftRep.java(Ansvarlig Rasmus & Patrick)

```
package dk.kea.model.repository;

import dk.kea.model.Interface.IOpskrifterRep;
import dk.kea.model.entities.Opskrift;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.rowset.SqlRowSet;
import org.springframework.stereotype.Repository;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;

@Repository
public class OpskriftRep implements IOpskrifterRep {

    @Autowired
    private JdbcTemplate jdbc;

    @Override
    public void create(Opskrift o, int id) throws SQLException
    {
        Connection myConn1 = null;
        PreparedStatement myStmt1 = null;


        try
        {
            // 1. Get a connection to database
            myConn1 = DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            // 2. Prepare statement
            myStmt1 = myConn1.prepareStatement("INSERT INTO db.opskrifter(title, vurdering,
karakter, fk_user_id) VALUES (?,?,?,?)");

            // 3. Set the parameters
            myStmt1.setString(1, o.getTitle());
```

```
            myStmt1.setString(2, o.getVurdering());
            myStmt1.setString(3, o.getKarakter());
            myStmt1.setInt(4, id);

            // 4. Execute SQL query
            myStmt1.executeUpdate();



            myStmt1 = myConn1.prepareStatement("INSERT INTO db.emner(type, vægt, tykkelse,
bemærkning, fk_opskrift_id) VALUES (?, ?, ?, ?, LAST_INSERT_ID())");
            myStmt1.setString(1, o.getEmne());
            myStmt1.setString(2, o.getWeight());
            myStmt1.setString(3, o.getTykkelse());
            myStmt1.setString(4, o.getE_bemærkning());
            myStmt1.executeUpdate();



            myStmt1 = myConn1.prepareStatement("INSERT INTO db.tilberedninger(tid, temp,
detaljer, fk_emne_id) VALUES (?,?,?, LAST_INSERT_ID())");
            myStmt1.setString(1, o.getTid());
            myStmt1.setString(2, o.getTemperatur());
            myStmt1.setString(3, o.getT_detaljer());
            myStmt1.executeUpdate();


        }
        catch (Exception exc)
        {
            exc.printStackTrace();
        }
        finally
        {

            if (myStmt1 != null)
            {
                myStmt1.close();
            }

            if (myConn1 != null)
            {
                myConn1.close();
```

```
        }
    }
}

public Opskrift read(int id){

    SqlRowSet sqlRowSet;
    sqlRowSet = jdbc.queryForRowSet("SELECT * FROM db.tilberedninger " +
                        "INNER JOIN db.emner ON fk_emne_id = emne_id " +
                        "INNER JOIN db.opskrifter ON fk_opskrift_id = opskrift_id " +
                        "WHERE opskrift_id = '" + id + "'");

    if(sqlRowSet.next()){
        return new Opskrift(sqlRowSet.getInt("opskrift_id"), sqlRowSet.getString("title"),
sqlRowSet.getDate("dato"), sqlRowSet.getString("karakter"),
                    sqlRowSet.getString("vurdering"), sqlRowSet.getString("type"),
sqlRowSet.getString("vægt"), sqlRowSet.getString("tykkelse"),
                    sqlRowSet.getString("bemærkning"), sqlRowSet.getString("tid"),
sqlRowSet.getString("temp"), sqlRowSet.getString("detaljer"));
    }

    return null;
}

public void update(int id, Opskrift o) throws SQLException{
    Connection myConn1 = null;
    PreparedStatement myStmt1 = null;


    try {
        // 1. Get a connection to database
        myConn1 = DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

        // 2. Prepare statement
        myStmt1 = myConn1.prepareStatement("UPDATE db.opskrifter SET title = ?, karakter =
?, vurdering = ? WHERE opskrift_id = ? ");

        // 3. Set the parameters
        myStmt1.setString(1, o.getTitle());
        myStmt1.setString(2, o.getKarakter());
        myStmt1.setString(3, o.getVurdering());
        myStmt1.setInt(4, id);
```

```
            // 4. Execute SQL query
            myStmt1.executeUpdate();



            myStmt1 = myConn1.prepareStatement("UPDATE db.emner SET type = ?, vægt = ?,
tykkelse = ?, bemærkning = ? WHERE fk_opskrift_id = ?");

            myStmt1.setString(1, o.getEmne());
            myStmt1.setString(2, o.getWeight());
            myStmt1.setString(3, o.getTykkelse());
            myStmt1.setString(4, o.getE_bemærkning());
            myStmt1.setInt(5, id);

            myStmt1.executeUpdate();



            myStmt1 = myConn1.prepareStatement("UPDATE db.tilberedninger SET tid = ?, temp =
?, detaljer = ? WHERE fk_emne_id = ?");

            myStmt1.setString(1, o.getTid());
            myStmt1.setString(2, o.getTemperatur());
            myStmt1.setString(3, o.getT_detaljer());
            myStmt1.setInt(4, id);

            myStmt1.executeUpdate();

        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {

            if (myStmt1 != null) {
                myStmt1.close();
            }

            if (myConn1 != null) {
                myConn1.close();
            }
        }
    }
```

```
    public void delete(int id){
        jdbc.update("DELETE FROM db.opskrifter WHERE opskrift_id = '" + id + "'");
    }

    public ArrayList<Opskrift> readAll(int id){

        ArrayList<Opskrift> opskrifter = new ArrayList<>();

        SqlRowSet sqlRowSet;
        sqlRowSet = jdbc.queryForRowSet("SELECT * FROM db.tilberedninger " +
                "INNER JOIN db.emner ON fk_emne_id = emne_id " +
                "INNER JOIN db.opskrifter ON fk_opskrift_id = opskrift_id " +
                "WHERE db.opskrifter.fk_user_id = '" + id + "'");

        while(sqlRowSet.next()){

            opskrifter.add(new Opskrift(sqlRowSet.getInt("opskrift_id"), sqlRowSet.getString("title"),
sqlRowSet.getDate("dato"), sqlRowSet.getString("karakter"),
                    sqlRowSet.getString("vurdering"), sqlRowSet.getString("type"),
sqlRowSet.getString("vægt"), sqlRowSet.getString("tykkelse"),
                    sqlRowSet.getString("bemærkning"), sqlRowSet.getString("tid"),
sqlRowSet.getString("temp"), sqlRowSet.getString("detaljer")));
        }

        return opskrifter;
    }


}
```

userRep.java(Ansvarlig: Rasmus & Patrick)

```
package dk.kea.model.repository;

import dk.kea.model.Interface.IUserRep;
import dk.kea.model.entities.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.rowset.SqlRowSet;
import org.springframework.stereotype.Repository;

import java.sql.*;
```

```
@Repository
public class UserRep implements IUserRep {

    @Autowired
    private JdbcTemplate jdbc;

    @Override
    public void create(User u) throws SQLException {
        Connection myConn = null;
        PreparedStatement myStmt = null;


        try {
            // 1. Get a connection to database
            myConn =
DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            // 2. Prepare statement
            myStmt = myConn.prepareStatement("INSERT INTO
db.users(username, password, email, role) VALUES(?,?,?,?)");


            // 3. Set the parameters
            myStmt.setString(1, u.getUsername());
            myStmt.setString(2, u.getPassword());
            myStmt.setString(3, u.getEmail());
            myStmt.setString(4, u.getRole());

            myStmt.executeUpdate();

        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
```

```
        }
    }

    @Override
    public User read(int id){

        SqlRowSet sqlRowSet;

        sqlRowSet = jdbc.queryForRowSet("SELECT * FROM db.users WHERE
user_id = '" + id + "'");
        if(sqlRowSet.next()){

            return new User(sqlRowSet.getInt("user_id"),
sqlRowSet.getString("username"), sqlRowSet.getString("password"),
                            sqlRowSet.getString("email"),
sqlRowSet.getString("role"));
        }
        return null;
    }

    public void update(int id, User u) throws SQLException{
        Connection myConn = null;
        PreparedStatement myStmt = null;


        try {
            // 1. Get a connection to database
            myConn =
DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            // 2. Prepare statement
            myStmt = myConn.prepareStatement("UPDATE db.users SET
username = ?,  password = ?, email = ? WHERE user_id = ? ");


            // 3. Set the parameters
            myStmt.setString(1, u.getUsername());
            myStmt.setString(2, u.getPassword());
            myStmt.setString(3, u.getEmail());
            myStmt.setInt(4, id);

            myStmt.executeUpdate();
```

```
        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
        }
    }

    public void delete(int id){
        jdbc.update("DELETE FROM db.users WHERE user_id = '" + id +
"'");
    }

    public User loginDB(User u) throws SQLException// kan laese alle
brugere
    {
        Connection myConn = null;
        PreparedStatement myStmt = null;


        try {

            myConn =
DriverManager.getConnection("jdbc:mysql://localhost:3306", "simon",
"simon003");

            myStmt = myConn.prepareStatement("SELECT * FROM db.users
WHERE username = ? AND password = ?");

            myStmt.setString(1, u.getUsername());
            myStmt.setString(2, u.getPassword());

            ResultSet rs = myStmt.executeQuery();

            while (rs.next()){
```

```
                return new User(rs.getInt("user_id"),
rs.getString("username"), rs.getString("password"),
                        rs.getString("email"), rs.getString("role"));
            }

        } catch (Exception exc) {
            exc.printStackTrace();
        } finally {

            if (myStmt != null) {
                myStmt.close();
            }

            if (myConn != null) {
                myConn.close();
            }
        }
        return null;
    }
}
```

Style.css(Ansvarlig: Patrick & Simon)

```
body{
    margin: 0 ;

}

section {
    width: 70%;
    margin: 20px auto;
    border: 1px solid lightgray;
    border-radius: 10px;
    /* border: 1px solid red; */
}

form{
    width: 70%;
    margin: auto;
}

input{
```

```
    margin-top: 20px;
    width: 100%;
}

textarea{
    margin-top: 20px;
    width: 100%;
    height: 3cm;
}

main{
    width: 70%;
    margin: 150px auto;
    height: 90%;
    /*border: 1px solid black;*/
}

header{
    width: 100%;
    margin: auto;
    padding: 0;
    position: fixed;
    top: 0;
    background-color: whitesmoke;

}

footer{
    width: 100%;
    margin: auto;
    background-color: whitesmoke;
}
```

forum.html(Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>

    <title>Foodie-log</title>
    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
```

```
<link rel="stylesheet" href="style.css" />
<style>

    div.readIndlæg1{
        width: 70%;
        margin: 10px auto;
        border: 1px solid lightgray;
        border-radius: 10px;
        padding: 5px;
    }

    section.readIndlæg{
        border: none;
    }

    div.readIndlæg2{
        width: 90%;
        margin: auto;
    }
    div.readIndlæg3{
        border-bottom: 1px solid lightgray;
    }

    div.readKommentare1{

        width: 90%;
        margin: 10px 0 0 50px;
    }

    p{
        font-size: x-small;
        color: gray;
        margin: 0;
        display: inline-block;
    }

    h4.username1{
        display: inline-block;
        margin: 0 0 0 34px;
    }

    h4.username2{
        margin: 0;
```

```
            display: inline-block;

        }

        pre.kommentar{
            width: 91%;
            margin: 0;
        }

        textarea.writeKommentar{
            width: 90%;
            height: 40px;
            margin: 0;
        }

        form.writeKommentar{
            width: 90%;
            margin: 10px 0 0 50px;
        }



    </style>
</head>
<body>
<header>
    <div th:replace="fragments/headers :: afterLogin"></div>
</header>
<main class="forum">

    <div style="width: 70%; margin: auto ">
        <h1> Forum </h1>
    </div>

    <section class="createIndlæg">

        <form th:action="@{/indlaeg(userId=${user.id})}" method="post"
th:object="${indlaeg}">

            <div>
                <input class="indlæg" type="text" th:field="*{title}"
placeholder="Indlæg Titel"/>
            </div>
```

```
            <div>
                <textarea class="indlæg" type="text"
th:field="*{tekst}" placeholder="Indlæg Tekst"/>
            </div>
            <div>
                <input class="indlæg" type="submit" value="Post Indlæg"
style="width: 20%; margin-bottom: 20px"/>
            </div>
        </form>

    </section>

    <section class="readIndlæg">

        <div class="readIndlæg1" th:each="indlaeg : ${forum}">

            <div th:if="${user.role} eq 'admin'" align="right">
                <a th:href="@{/deleteI(inlaegId=${indlaeg.getId()},
userId=${user.id})}" onclick="return confirm('Er du sikker på at du
vil slette dette Indlæg?');">X</a>
            </div>

            <h4 class="username1" th:text="${indlaeg.username}"/>
            <p th:text="${indlaeg.dato}"/>

            <div class="readIndlæg2">
                <div class="readIndlæg3">
                    <h3 th:text="${indlaeg.title}"/>
                    <pre th:text="${indlaeg.tekst}"/>
                </div>
                <div class="readKommentare1" th:each="kommentar :
${indlaeg.getKommentareListe()}">

                    <div th:if="${user.role} eq 'admin'" align="right">
                        <a
th:href="@{/deleteK(kommentarId=${kommentar.getId()},
userId=${user.id})}" onclick="return confirm('Er du sikker på at du
vil slette denne Kommentar?');">X</a>
                    </div>

                    <h4 class="username2"
th:text="${kommentar.username}"/>
                    <p th:text="${kommentar.dato}" />
```

```
                    <pre class="kommentar"
th:text="${kommentar.tekst}"/>

                </div>

                <form class="writeKommentar"
th:action="@{/kommentar(userId=${user.id}, indlaegId=${indlaeg.id})}"
method="post" th:object="${kommentar}">
                    <textarea class="writeKommentar" type="text"
th:field="*{tekst}" placeholder="Skriv en kommentar"/>
                    <div>
                    <input type="submit" value="Post Kommentar"
style="width: 25%; margin: 0; " />
                    </div>
                </form>
            </div>
        </div>

    </section>
</main>

<footer>

    <div th:replace="fragments/footer :: footer"></div>

</footer>
</body>
</html>
```

Homepage.html (ansvarlig: Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>

    <title>Title</title>
    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
    <link rel="stylesheet" href="style.css" />
    <style>
```

```
        div{

        }

        th, td{
            width: 25%;
            margin: auto;
        }


        table, tr{
            width: 100%;
            margin: auto;

        }

        div.emne{

            margin-top: 30px;

        }

        div.tilberedning{

            margin-top: 30px;
            margin-bottom: 30px;

        }

    </style>
</head>
<body>

    <header>
        <div th:replace="fragments/headers :: afterLogin"></div>
    </header>
    <main >

        <div style="width: 70%; margin: auto ">
            <h1> Tilføj Opskrift </h1>
        </div>

        <section class="createOpskrift" >
```

```
            <form th:action="@{/createO(id=${user.id})}" method="post"
th:object="${opskrift}" >

                <div align="center">
                <input class="create" type="text" th:field="*{title}"
placeholder="Opskrift Titel"/>
                </div>

                <div class="emne">
                <div align="center">
                <input class="create" type="text" th:field="*{emne}"
placeholder="Opskrift Emne"/>
                </div>
                <div align="center">
                <input class="create" type="text" th:field="*{weight}"
placeholder="Emne V&aelig;gt"/>
                </div>
                <div align="center">
                <input class="create" type="text"
th:field="*{tykkelse}" placeholder="Emne Tykkelse"/>
                </div>
                <div align="center">
                <textarea class="create" type="text"
th:field="*{e_bemærkning}" placeholder="Bem&aelig;rkninger Til Emnet"
/>
                </div>
                </div>

                <div class="tilberedning">
                <div align="center">
                <input class="create" type="text" th:field="*{tid}"
placeholder="Tiberednings Tid"/>
                </div>
                <div align="center">
                <input class="create" type="text"
th:field="*{temperatur}" placeholder="Tilberednings Temperatur"/>
                </div>
                <div align="center">
                <textarea class="create" type="text"
th:field="*{t_detaljer}" placeholder="Detaljer Omkring Tilberedning"/>
                </div>
                </div>
```

```
                <div align="left">
                <input class="create" type="submit"
value="Tilf&oslash;j Opskrift" style="width: 20%; margin-bottom:
20px"/>
                </div>

            </form>

        </section>

        <section class="readOpskrifter">
            <table>
                <tr>
                    <th>Titel</th>
                    <th>Emne</th>
                    <th>Karakter</th>
                    <th>Knapper</th>
                </tr>
                <tr th:each="opskrift : ${opskrifter}">
                    <td align="center" th:text="${opskrift.title}"/>
                    <td align="center" th:text="${opskrift.emne}"/>
                    <td align="center" th:text="${opskrift.karakter}"/>
                    <td align="center" >
                        <a
th:href="@{/updateO(opskriftId=${opskrift.id}, userId=${user.id})}">Se
opskrift</a>
                        <a
th:href="@{/deleteO(opskriftId=${opskrift.id}, userId=${user.id})}"
onclick="return confirm('Er du sikker på at du vil slette denne
opskrift?');">Slet opskrift</a>
<!--(id=${blog.id})(user_id=${user.id})-->
                    </td>
                </tr>
            </table>
        </section>

    </main>
    <footer>

        <div th:replace="fragments/footer :: footer"></div>

    </footer>
```

```
</body>
</html>
```

Index.html (ansvarlig: Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <title>index</title>


    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
    <link rel="stylesheet" href="style.css" />


    <style>


        nav{
            /*border: 1px solid red;*/
            position: fixed;
            top: 150px;
            width: 10%;

        }

        table{
            width: 95%;
            margin: 10px auto;
            /*border: 1px solid blue;*/
        }


        h2{
            width: 95%;
            margin: auto;
            padding: 20px;
            /*border: 1px solid green;*/
        }
```

```
        div.tommelregel{
            width: 95%;
            margin: auto;
            background-color: whitesmoke;
            color: gray;

        }

        pre {
            width: 100%;
            margin: auto;
        }

        a{
            text-decoration: none;
            color: black;
        }

        a.link{
            /*border: 1px solid blue;*/
            padding: 5px;
            display: block;


        }

        a:hover{
            background-color: lightgray;
        }




    </style>

</head>
<body>
<header>
    <div th:replace="fragments/headers :: beforeLogin"></div>
</header>
<main>
    <nav>
```

```
            <a class="link" href="#beef">OkseK&oslash;d</a>

            <a class="link" href="#pig">SvineK&oslash;d</a>

            <a class="link" href="#lame">LammeK&oslash;d</a>

            <a class="link" href="#bird">Fjerkr&aelig;</a>

            <a class="link" href="#fish">Fisk</a>

            <a class="link" href="#green">Frugt og Gr&oslash;nt</a>


    </nav>
    <section id="beef">
        <h2 align="center"> OkseK&oslash;d </h2>
        <div class="tommelregel" align="center">
                    <pre><i>
Tommelfingerregel

Meget r&oslash;dt: 48 °C
R&oslash;dt: 54 °C
Medium-R&oslash;d: 57 °C
Medium: 62 °C
Medium-Gennemstegt: 65 °C
Gennemstegt: 72 °C
                    </i></pre>
        </div>
        <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i timer</th>
```

```
                </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                 <div th:if="${opskrift.title} eq 'oksekød'">
                     <td th:text="${opskrift.emne}"/>
                     <td th:text="${opskrift.weight}"/>
                     <td th:text="${opskrift.temperatur}"/>
                     <td th:text="${opskrift.tid}"/>
                 </div>
                 </tr>
        </table>
    </section>
    <section id="pig">
        <h2 align="center"> SvineK&oslash;d </h2>
        <div class="tommelregel" align="center">
                    <pre ><i>
Tommelfingerregel

Rosa: 59 °C
Egner sig bedst til m&oslash;rbrad, koteletter, kamsteg
Svag rosa: 61 °C
Egner sig til de fleste udsk&aelig;ringer
Gennemstegt 65 °C
Forarbejdede udsk&aelig;ringer som bacon, fl&aelig;sk, hamburgerryg
og lignende smager bedst hvis de bliver gennemstegt
                    </i></pre>
        </div>
        <table>
            <tr>
                <th>Udsk&aelig;ring</th>

                <th>V&aelig;gt i kg</th>

                <th>Temperatur i celsius</th>

                <th>Tid i timer</th>

            </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                <div th:if="${opskrift.title} eq 'svinekød'">
                <td th:text="${opskrift.emne}"/>
                <td th:text="${opskrift.weight}"/>
                <td th:text="${opskrift.temperatur}"/>
```

```
                <td th:text="${opskrift.tid}"/>
                </div>
            </tr>
        </table>
    </section>
    <section id="lame">
        <h2 align="center"> LammeK&oslash;d </h2>
        <div class="tommelregel" align="center">
                    <pre ><i>
Tommelfingerregel

Meget r&oslash;dt: 48 °C
R&oslash;dt: 54 °C
Medium-R&oslash;d: 57 °C
Medium: 62 °C
Medium-Gennemstegt: 65 °C
Gennemstegt: 72 °C
                    </i></pre>
        </div>
        <table>
            <tr>
                <th>Udsk&aelig;ring</th>

                <th>V&aelig;gt i kg</th>

                <th>Temperatur i celsius</th>

                <th>Tid i timer</th>

            </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                <div th:if="${opskrift.title} eq 'lammekød'">
                <td th:text="${opskrift.emne}"/>
                <td th:text="${opskrift.weight}"/>
                <td th:text="${opskrift.temperatur}"/>
                <td th:text="${opskrift.tid}"/>
                </div>
            </tr>
        </table>
    </section>
    <section id="bird">
        <h2 align="center"> Fjerkr&aelig; </h2>
        <div class="tommelregel" align="center">
```

```
                    <pre ><i>
Det anbefales ikke at man tilbereder en hel fugl, da der er en stor
luftlomme inde i fuglen.
Dette vil g&oslash;re det stort set umuligt at tilberede den med sous
vide-metoden.

Alle udsk&aelig;ringer er i medium st&oslash;rrelse

                    </i></pre>
        </div>
        <table>
            <tr>
                <th>Udsk&aelig;ring</th>

                <th>V&aelig;gt i kg</th>

                <th>Temperatur i celsius</th>

                <th>Tid i timer</th>

            </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                <div th:if="${opskrift.title} eq 'fjerkræ'">
                <td th:text="${opskrift.emne}"/>
                <td th:text="${opskrift.weight}"/>
                <td th:text="${opskrift.temperatur}"/>
                <td th:text="${opskrift.tid}"/>
                </div>
            </tr>
        </table>
    </section>
    <section id="fish">
        <h2 align="center"> Fisk </h2>
        <div class="tommelregel" align="center">
                    <pre ><i>
Fisk er meget delikat. Derfor er det vigtigt, at man ikke giver det
for meget.
Tider og temperaturer herunder, er sat efter at fisken skal
tilberedes under medium.
                    </i></pre>
        </div>
        <table>
            <tr>
```

```
                <th>Udsk&aelig;ring</th>

                <th>V&aelig;gt i kg</th>

                <th>Temperatur i celsius</th>

                <th>Tid i minutter</th>

            </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                <div th:if="${opskrift.title} eq 'fisk'">
                <td th:text="${opskrift.emne}"/>
                <td th:text="${opskrift.weight}"/>
                <td th:text="${opskrift.temperatur}"/>
                <td th:text="${opskrift.tid}"/>
                </div>
            </tr>
        </table>
    </section>
    <section id="green">
        <h2 align="center"> Frugt og Gr&oslash;nt </h2>
        <div class="tommelregel" align="center">
                    <pre ><i>
Alle udsk&aelig;ringer er i mellem st&oslash;rrelser
                    </i></pre>
        </div>
        <table>
            <tr>
                <th>Udsk&aelig;ring</th>

                <th>V&aelig;gt i kg</th>

                <th>Temperatur i celsius</th>

                <th>Tid i minutter</th>

            </tr>
            <tr th:each="opskrift: ${opskrifter}" align="center">
                <div th:if="${opskrift.title} eq 'frugt'">
                <td th:text="${opskrift.emne}"/>
                <td th:text="${opskrift.weight}"/>
                <td th:text="${opskrift.temperatur}"/>
                <td th:text="${opskrift.tid}"/>
```

```
                </div>
            </tr>
        </table>
    </section>
</main>
<footer>

    <div th:replace="fragments/footer :: footer"></div>

</footer>
</body>
</html>
```

signup.html(Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <title>Sign up</title>
    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
    <link rel="stylesheet" href="style.css" />

    <style>

        nav{
            /*border: 1px solid red;*/
            position: fixed;
            top: 150px;
            width: 10%;

        }

        table{
            width: 95%;
            margin: 10px auto;
            /*border: 1px solid blue;*/
        }


        h2{
```

```
            width: 95%;
            margin: auto;
            padding: 20px;
            /*border: 1px solid green;*/
        }

        div.tommelregel{
            width: 95%;
            margin: auto;
            background-color: whitesmoke;
            color: gray;

        }

        pre {
            width: 100%;
            margin: auto;
        }

        a{
            text-decoration: none;
            color: black;
        }

        a.link{
            /*border: 1px solid blue;*/
            padding: 5px;
            display: block;


        }

        a:hover{
            background-color: lightgray;
        }

    </style>
</head>
<body>

    <header>
        <div th:replace="fragments/headers :: signup"></div>
    </header>
```

```
    <main>
        <nav>

            <a class="link" href="#beef">OkseK&oslash;d</a>

            <a class="link" href="#pig">SvineK&oslash;d</a>

            <a class="link" href="#lame">LammeK&oslash;d</a>

            <a class="link" href="#bird">Fjerkr&aelig;</a>

            <a class="link" href="#fish">Fisk</a>

            <a class="link" href="#green">Frugt og Gr&oslash;nt</a>

        </nav>
        <section id="beef">
            <h2 align="center"> OkseK&oslash;d </h2>
            <div class="tommelregel" align="center">
                    <pre><i>
Tommelfingerregel

Meget r&oslash;dt: 48 °C
R&oslash;dt: 54 °C
Medium-R&oslash;d: 57 °C
Medium: 62 °C
Medium-Gennemstegt: 65 °C
Gennemstegt: 72 °C
                    </i></pre>
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>
```

```
                    <th>Temperatur i celsius</th>

                    <th>Tid i timer</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'oksekød'">
                    <td th:text="${opskrift.emne}"/>
                    <td th:text="${opskrift.weight}"/>
                    <td th:text="${opskrift.temperatur}"/>
                    <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
        <section id="pig">
            <h2 align="center"> SvineK&oslash;d </h2>
            <div class="tommelregel" align="center">
                    <pre ><i>
Tommelfingerregel

Rosa: 59 °C
Egner sig bedst til m&oslash;rbrad, koteletter, kamsteg
Svag rosa: 61 °C
Egner sig til de fleste udsk&aelig;ringer
Gennemstegt 65 °C
Forarbejdede udsk&aelig;ringer som bacon, fl&aelig;sk, hamburgerryg
og lignende smager bedst hvis de bliver gennemstegt
                    </i></pre>
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i timer</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'svinekød'">
```

```
                        <td th:text="${opskrift.emne}"/>
                        <td th:text="${opskrift.weight}"/>
                        <td th:text="${opskrift.temperatur}"/>
                        <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
        <section id="lame">
            <h2 align="center"> LammeK&oslash;d </h2>
            <div class="tommelregel" align="center">
                    <pre ><i>
Tommelfingerregel

Meget r&oslash;dt: 48 °C
R&oslash;dt: 54 °C
Medium-R&oslash;d: 57 °C
Medium: 62 °C
Medium-Gennemstegt: 65 °C
Gennemstegt: 72 °C
                    </i></pre>
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i timer</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'lammekød'">
                        <td th:text="${opskrift.emne}"/>
                        <td th:text="${opskrift.weight}"/>
                        <td th:text="${opskrift.temperatur}"/>
                        <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
```

```
        <section id="bird">
            <h2 align="center"> Fjerkr&aelig; </h2>
            <div class="tommelregel" align="center">
                    <pre ><i>
Det anbefales ikke at man tilbereder en hel fugl, da der er en stor
luftlomme inde i fuglen.
Dette vil g&oslash;re det stort set umuligt at tilberede den med sous
vide-metoden.

Alle udsk&aelig;ringer er i medium st&oslash;rrelse

                    </i></pre>
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i timer</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'fjerkræ'">
                        <td th:text="${opskrift.emne}"/>
                        <td th:text="${opskrift.weight}"/>
                        <td th:text="${opskrift.temperatur}"/>
                        <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
        <section id="fish">
            <h2 align="center"> Fisk </h2>
            <div class="tommelregel" align="center">
                    <pre ><i>
Fisk er meget delikat. Derfor er det vigtigt, at man ikke giver det
for meget.
Tider og temperaturer herunder, er sat efter at fisken skal
tilberedes under medium.
                    </i></pre>
```

```
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i minutter</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'fisk'">
                        <td th:text="${opskrift.emne}"/>
                        <td th:text="${opskrift.weight}"/>
                        <td th:text="${opskrift.temperatur}"/>
                        <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
        <section id="green">
            <h2 align="center"> Frugt og Gr&oslash;nt </h2>
            <div class="tommelregel" align="center">
                    <pre ><i>
Alle udsk&aelig;ringer er i mellem st&oslash;rrelser
                    </i></pre>
            </div>
            <table>
                <tr>
                    <th>Udsk&aelig;ring</th>

                    <th>V&aelig;gt i kg</th>

                    <th>Temperatur i celsius</th>

                    <th>Tid i minutter</th>

                </tr>
                <tr th:each="opskrift: ${opskrifter}" align="center">
                    <div th:if="${opskrift.title} eq 'frugt'">
                        <td th:text="${opskrift.emne}"/>
```

```
                        <td th:text="${opskrift.weight}"/>
                        <td th:text="${opskrift.temperatur}"/>
                        <td th:text="${opskrift.tid}"/>
                    </div>
                </tr>
            </table>
        </section>
    </main>

    <footer>
        <div th:replace="fragments/footer :: footer"></div>
    </footer>

</body>
</html>
```

opdateOpskrift.html (Ansvarlig: Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>

    <title>Title</title>

    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
    <link rel="stylesheet" href="style.css" />

    <style>


        th{
            width: 25%;
            margin: auto;
        }

        table, tr{
            width: 100%;
            margin: auto;
        }

        div.emne{
```

```
            margin-top: 30px;

        }

        div.tilberedning{

            margin-top: 30px;
            margin-bottom: 30px;

        }
    </style>
</head>
<body>
<header>
    <div th:replace="fragments/headers :: afterLogin"></div>
</header>
<main>

    <div style="width: 70%; margin: auto ">
        <h1> Opdater Opskrift </h1>
    </div>

    <section>

        <form th:action="@{/updateO(userId=${user.id},
opskriftId=${opskrift.id})}" method="post" th:object="${opskrift}">

            <div align="center">
            <input class="updateTitle" type="text" th:field="*{title}"
placeholder="Opskrift Titel"/>
            </div>



            <div class="emne">
            <div align="center">
            <input class="update" type="text" th:field="*{emne}"
placeholder="Opskrift Emne"/>
            </div>

            <div align="center">
```

```
            <input class="update" type="text" th:field="*{weight}"
placeholder="Emne V&aelig;gt"/>
            </div>

            <div align="center">
            <input class="update" type="text" th:field="*{tykkelse}"
placeholder="Emne Tykkelse"/>
            </div>

            <div align="center">
            <textarea class="updateBemærkning" type="text"
th:field="*{e_bemærkning}" placeholder="Bem&aelig;rkninger Til
Emnet"/>
            </div>
            </div>



            <div class="tilberedning">
            <div align="center">
            <input class="update" type="text" th:field="*{tid}"
placeholder="Tiberednings Tid"/>
            </div>

            <div align="center">
            <input class="update" type="text" th:field="*{temperatur}"
placeholder="Tilberednings Temperatur"/>
            </div>

            <div align="center">
            <textarea class="updateDataljer" type="text"
th:field="*{t_detaljer}" placeholder="Detaljer Omkring Tilberedning"/>
            </div>
            </div>



            <div class="vurdering">

            <div align="center">
                <input class="update" type="text"
th:field="*{karakter}" placeholder="Opskrift Karakter"/>
            </div>
```

```
            <div align="center">
                <textarea class="updateVurdering" type="text"
th:field="*{vurdering}" placeholder="Opksrift Vurdering"/>
            </div>
            </div>


            <div align="left">
            <input class="update" type="submit" value="Opdater
Opskrift" style="width: 20%; margin-bottom: 20px"/>
            </div>

        </form>

    </section>
</main>
<footer>

    <div th:replace="fragments/footer :: footer"></div>

</footer>
</body>
</html>
```

updateUser.html (Ansvarlig: Patrick & Simon)

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>

    <title>Title</title>
    <div th:replace="fragments/headers :: header-css"/>
    <div th:replace="fragments/footer :: footer-css"/>
    <link rel="stylesheet" href="style.css" />
    <style>

        input{
            margin: 0;
        }

        p{
```

```
            margin: 10px 0 0 0;
        }

    </style>


</head>
<body>
    <header>
        <div th:replace="fragments/headers :: afterLogin"></div>
    </header>
    <main>
        <div style="width: 70%; margin: auto ">
            <h1> Updater profil </h1>
        </div>

        <section>
            <form th:action="@{/updateU(userId=${user.id})}"
method="post" th:object="${user}">

                <p> Username </p>
            <div align="center">

                <input class="update" type="text"
th:field="*{username}" placeholder="Username"/>
            </div>

                <p> Password </p>
            <div align="center">
                <input class="update" type="text"
th:field="*{password}" placeholder="Password"/>
            </div>

                <p> Email </p>
            <div align="center">

                <input class="update" type="text" th:field="*{email}"
placeholder="Email"/>
            </div>

                <div align="left" style="margin: 10px 0;">
                <input class="update" type="submit" value="Gem"
style="width: 25%; margin: 0; "/>
```

```
                </div>
                <div align="right">
                    <!--<a
th:href="@{/deleteU(userId=${user.id})}">Slet profil</a>-->
                    <a th:href="@{/deleteU(userId=${user.id})}"
onclick="return confirm('Er du sikker på at du vil slette din
profil?');">Slet profil</a>
                </div>

            </form>

        </section>

    </main>
    <footer style="position: absolute; bottom: 0">

        <div th:replace="fragments/footer :: footer"></div>

    </footer>

</body>
</html>
```

LitteraturListe

Link til guide omkring organisationsteori:
http://forlaget94.dk/cms/wp-content/uploads/F94-TM-2012-B1-K1-Prove.pdf

Link til pdf på fronter hvori der findes en guide til ITO analyseredskaber:
https://fronter.com/kea/links/files.phtml/897219293$603682853$/2.+semester/ITO/kopier/uge+38/projekt_og_analyseredskaber.pdf

Links til sider der beskriver prepared statments:
http://www.theserverside.com/news/1365244/Why-Prepared-Statements-are-important-and-how-to-use-them-properly

https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html

https://www.youtube.com/watch?v=gU3DLOsw0Eg

https://docs.google.com/document/d/e/2PACX-1vS5A34jXa-Qy0HY7b4hAXqbswLpMWyQoOuFJY1WM8PZysKywwXECtj89Xv0lDv_f58K_BdD1yogvWOl/pub

http://www.theserverside.com/definition/Java-Database-Connectivity-JDBC

https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.htm

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/ui/Model.html