Projektrapport

Pristjek220

Dato for aflevering: 27-05-2016

|  |  |  |
| --- | --- | --- |
| Aarhus Universitet - Ingeniørhøjskolen | | | Vejleder: Troels Fedder Jensen (TFJ) |

|  |
| --- |
| Gruppe 7 |

# Resumé

Resumé af projektet med problemstilling, hvordan det er løst og konklusion

# Abstract

Resumé på engelsk

Indholdsfortegnelse

[1 Resumé 1](#_Toc451425712)

[2 Abstract 1](#_Toc451425713)

[3 Indledning 4](#_Toc451425714)

[4 Læsevejledning 4](#_Toc451425715)

[4.1 Termliste 5](#_Toc451425716)

[5 Projektformulering 5](#_Toc451425717)

[6 Afgrænsning 6](#_Toc451425718)

[7 Systembeskrivelse 7](#_Toc451425719)

[7.1 Krav 7](#_Toc451425720)

[7.1.1 Udformning af krav 7](#_Toc451425721)

[7.1.2 Aktører 8](#_Toc451425722)

[7.1.3 User story beskrivelser 8](#_Toc451425723)

[7.1.4 Kvalitetskrav 9](#_Toc451425724)

[8 Projektgennemførelse 10](#_Toc451425725)

[8.1 Iterativ udvikling 10](#_Toc451425726)

[8.2 Dokumentation af koden 11](#_Toc451425727)

[8.3 Dokumentation af systemarkitektur 11](#_Toc451425728)

[9 Systemdesign 12](#_Toc451425729)

[9.1 Arkitektur 12](#_Toc451425730)

[9.2 GUI design overvejelser 14](#_Toc451425731)

[9.3 Pristjek220 Database 15](#_Toc451425732)

[9.3.1 Design af databasen 15](#_Toc451425733)

[9.3.2 Databaseadgang 16](#_Toc451425734)

[10 Produkt beskrivelse 17](#_Toc451425735)

[10.1 Delte funktionaliteter 17](#_Toc451425736)

[10.2 Forbruger 17](#_Toc451425737)

[10.3 Administration 18](#_Toc451425738)

[10.3.1 Administrator 18](#_Toc451425739)

[10.3.2 Forretningsmanager 19](#_Toc451425740)

[11 Test af pristjek220 19](#_Toc451425741)

[11.1 Unittest 20](#_Toc451425742)

[11.2 Integrationstest 20](#_Toc451425743)

[11.3 (evt. Code Metrics) 21](#_Toc451425744)

[11.4 CI 21](#_Toc451425745)

[11.5 Forbrugertest 22](#_Toc451425746)

[12 Resultater og diskussion 22](#_Toc451425747)

[13 Fremtidigt arbejde 22](#_Toc451425748)

[14 Konklusion 23](#_Toc451425749)

[15 Referencer 23](#_Toc451425750)

[16 Underskrifter 24](#_Toc451425751)

# Indledning

I denne rapport beskrives et 4. semesterprojekt på Ingeniørhøjskolen i Aarhus. Der er fremstillet et produkt, hvis formål er at give forbrugeren mulighed for at lave sine indkøb så billigt som muligt. Produktet består af to applikationer, som har hver deres grafiske brugergrænseflade. Begge applikationer har adgang til den samme eksterne database. Under udviklingen af produktet er der arbejdet iterativt, hvor der er benyttet Scrum som en agil udviklingsmetode. Der er her arbejdet i sprints af to ugers varighed, hvor målet efter hvert sprint var at have fuldt ud implementeret nogle udvalgte user stories. Koden til projektet er styret ved at benytte versionsstyringsværktøjet Git i form af Git klienten TortoiseGit. Til designet af applikationen er der benyttet en 3-lagdelt arkitektur samt flere forskellige designmønstre, til bl.a. at abstrahere fra databasen, og afkoble den grafiske brugergrænseflade fra den bagvedliggende kode. Produktet er testet ved hjælp af unittests og integrationstest i koden, hvor der er benyttet NUnit Frameworket. Til sidst i projektet er der udført forbrugertests, hvor der er testet brugervenligheden af produktet, på folk der er i produktets målgruppe, for at se om det er intuitivt at benytte. Produktet er dokumenteret i den tilhørende dokumentation, hvor der både findes kravspecifikation for projektet, systemarkitektur med diagrammer over produktet og dokumentation for koden til produktet.

# Læsevejledning

Rapporten er opbygget således, at der først er nogle indledende afsnit, hvor baggrunden for projektet præsenteres samtidig med visionen og afgrænsningen for produktet. Dette sker i afsnittene ”Projektformulering” og ”Afgrænsning”.

Efter introduktionen til projektet følger der, i afsnittet ”Systembeskrivelse”, en beskrivelse af systemet samt de funktionelle krav og kvalitetskravene. Her forefindes overvejelser over, hvorfor der er valgt at formulere de funktionelle krav i form af user stories, samt et uddrag af nogle af de user stories, der er formuleret i kravspecifikationen.

Afsnittet ”Projektgennemførsel” følger herefter, hvor den iterative udviklingsmetode, der er benyttet i projektet, forklares. Samtidig beskrives nogle af de overvejelser, der er gjort i forhold til de valg, der er taget omkring dokumentation af koden og af systemarkitekturen.

I det efterfølgende afsnit ”Systemdesign” forklares systemets design. Her beskrives først arkitekturen af systemet, og herefter de overvejelser der er gjort i forbindelse med designet af den grafiske brugergrænseflade, og de mønstre der er brugt her. Endeligt beskrives designet af databasen, med de overvejelser der er gjort, og de mønstre der er benyttet, for at løse nogle af de problemstillinger der blev fundet.

Afsnittet ”Produktbeskrivelse” giver et overblik over, hvad det endelige produkt har af funktionaliteter. Afsnittet starter med kort at beskrive nogle af de funktionaliteter, der er delt mellem de to applikationer, som produktet er delt op i. Herefter beskrives forbrugerapplikationen, med de funktionaliteter den har. Efterfølgende beskrives administrationsapplikationen, hvor administratordelen beskrives først og derefter forretningsmanagerdelen, med de funktionaliteter de har hver især.

Herefter følger en beskrivelse af, hvordan produktet er testet, i afsnittet ”Testning af Pristjek220”. Først beskrives, hvordan koden er testet individuelt og derefter testet integreret, samt hvordan continuous integration er benyttet i projektet. Endeligt beskrives, hvordan der er blevet udført forbrugertests for at teste brugervenligheden af produktet.

Til sidst beskrives de resultater, der er opnået igennem projektet, og der følger herefter en diskussion af dem, samt af selve projektet, i afsnittet ”Resultater og diskussion”. Ud fra dette beskrives bagefter, hvad der kan laves af videreudvikling på produktet, i afsnittet ”Fremtidigt arbejde”. Endeligt følger der en konklusion på projektet, i afsnittet ”Konklusion”.

Det følgende afsnit er en termliste, hvor de forkortelser, der er brugt igennem rapporten, er opstillet.

## Termliste

BLL = Business Logic Layer

CI = Continuous Integration

CRUD = Create, Read, Update og Delete

DAB = Database fag på 4. semester

DAL = Data Access Layer

DIP = Dependency Inversion Principle

FDD = Feature Driven Development

PL = Presentation Layer

SRP = Single Responsibility Principle

TDD = Test Driven Development

UX = User Experience

# Projektformulering

Som forbruger er det svært at danne sig et overblik over, hvor det er billigst at handle ind. Der er i dag mange forskellige forretningskæder, der konkurrerer med hinanden, og dette resulterer i et stort udbud af forskellige varer med forskellige priser. Der er derfor blevet fremstillet et program, kaldet Pristjek220, som tilstræber at give forbrugeren et let og simpelt overblik over, hvor de forskellige varer kan handles billigst.

I Pristjek220 Forbruger er den grundlæggende funktionalitet, som forbrugeren har at kunne finde den billigste forretning for en vare. Han kan samtidig også se hvilke forretninger, der har varen, og hvad den koster de forskellige steder, når han søger efter en vare. Konceptet bag at slå en vare op i Pristjek220 er illustreret på Figur 1. Det er denne grundidé, som resten af funktionaliteterne for Pristjek220 Forbruger bygger på. Når forbrugeren søger efter en vare i Pristjek220, bliver han, så snart han har indtastet to bogstaver, foreslået varer, der indeholder de bogstaver, han har indtastet for at effektivisere brugen af Pristjek220.

For at gøre Pristjek220 mere effektivt i forbindelse med forbrugerens indkøb kan han indtaste en indkøbsliste, og Pristjek220 informerer så forbrugeren om, hvor varerne fra indkøbslisten kan findes billigst, samt hvad de koster. Her har forbrugeren samtidig mulighed for at se en sammenligning af, hvad det vil koste at købe alle varerne i én forretning i modsætning til at købe varerne, der hvor de er billigst. Forbrugeren har ligeledes mulighed for at kunne justere, hvor varerne skal købes, efter Pristjek220 er kommet med listen over, hvor det er billigst. Dette giver forbrugeren mulighed for at vælge at flytte én enkelt vare på listen til en anden forretning, f.eks. i det tilfælde at han skal handle i tre forretninger, men kun skal have én vare i den ene forretning. Yderligere kan det angives, hvis der er nogle forretninger, man ikke ønsker at handle i. Når forbrugeren har fået genereret sin indkøbsliste af Pristjek220, som han vil have den, har han mulighed for at ændre den og derefter få den sendt til sin egen E-mail. Dette gør, at han kan tilgå listen via hans mobiltelefon.

Pristjek220 har tre forskellige brugere; en forbruger, en forretningsmanager og en administrator. Forbrugeren er ham, der bruger Pristjek220 til at organisere sine daglige indkøb. Forretningsmanageren holder Pristjek220 opdateret med korrekte informationer om de varer og priser, der findes i netop hans forretningskæde. Administratoren servicerer Pristjek220, så der kan oprettes og fjernes forretninger. Baseret på disse tre brugere er Pristjek220 opdelt i to programmer; et program, Pristjek220 Forbruger, til forbrugeren og et fælles program, Pristjek220 Forretning, til både forretningsmanageren og administratoren. I Pristjek220 Forretning kan administratoren benytte sig af sine funktionaliteter ved at logge ind med et administratorlogin, som han har.

I Pristjek220 Forretning kan forretningsmanageren logge ind med sit forretningsmanagerlogin. Derefter har han mulighed for at tilføje og fjerne varer, samt ændre prisen på en vare, fra den forretning, han styrer. Han bliver samtidig bedt om at bekræfte sine valg, så sandsynligheden for, at han gør noget ved en fejl, er minimeret. Ved at forretningsmanageren holder informationerne om varerne fra hans butik opdaterede, holder han forbrugerne oplyste og giver dem de bedste vilkår for at få fuldt udbytte af Pristjek220. Administratoren kan i Pristjek220 Forretning logge ind med sit administratorlogin, hvorefter han kan tilføje en forretning med tilhørende forretningsmanager eller fjerne en forretning fra Pristjek220.



Figur : Rigt billede over opslag af en vare i Pristjek220.

Pristjek220 henvender sig til det moderne menneske, som er økonomisk anlagt og gerne vil spare penge, når der handles dagligvarer i hverdagen. Det henvender sig altså både til studerende uden mange penge på lommen, samt familier der ønsker at få mest muligt ud af deres penge.

# Afgrænsning

Pristjek220 består af to programmer hvor det ene er tiltænkt til forretningerne og det andet til forretningernes forbruger. Gennem disse programmer kan forretningerne tilføje og fjerne informationer, om deres sortiment af forretningen i pristjek220 database via internettet. Pristjek220 database kan kun tilgås lokalt eller via VPN, derfor er det nødvendigt at oprette en VPN forbindelse. Forbrugerne kan sammenligne forretningernes sortiment med andre forretninger, og derved finde ud af hvor det er billigst at handle. Da forretninger selv står for at opdatere deres sortiment vil forbrugerne ikke have mulighed for at sammenligne med de forretninger som vælger ikke at være med i Pristjek220.

Pristjek220 er udviklet som computerapplikationer, og der er af den grund udviklet en mulighed til forbrugerne så de kan sende deres indkøbsseddel på E-mail, da de fleste mobiltelefoner understøtter E-mails.

# Systembeskrivelse

Systemet består af en forbrugerapplikationer og en forretningsapplikationen, som er to individuelle programmer, således at ved udvidelser af det ene program, er det ikke nødvendigt at opdatere det andet. I forretningsenheden er der mulighed for at logge ind som en forretningsmanager eller som administrator. Begge programmer kobler op til en Microsoft SQL database, for at koble op er det nødvendigt at være på en VPN forbindelse. Databasen indeholder beskrivelser af hvor der er mulighed for at købe forskellige produkter og hvad deres pris er i de forskellige forretninger. Derudover indeholder den loginoplysningerne, der tjekkes op mod når en administrator eller en forretningsmanager logger ind. På Figur 2 ses domæne modellen for systemet, hvor der kan ses de forskellige interaktioner mellem aktørerne og entiteterne.



Figur : Domæne model for Pristjek220

## Krav

Ud fra *Projektformuleringen* er der opstillet en række User Stories, der beskriver brugerens interaktion med systemet. User Stories danner grundlag for kravspecifikationen, og bruges til at fastsætte systemets funktionalitet.

### Udformning af krav

For at opbygge en kravspecifikation for Pristjek220 skulle der i starten af projektet formuleres funktionelle krav i form af forskellige scenarier. Gruppen havde tidligere kun arbejdet med use cases, hvor der var erfaring med, at de ikke fungerede særlig effektivt i en iterativ proces. Use cases laves typisk ved, at man følger en skabelon, og her er der en fare for, at man er mere opmærksom på, om man følger skabelonen frem for, om de vigtige detaljer er med. Ved at følge denne skabelon bliver der også formuleret rigtig meget på skrift for en use case, hvilket ikke nødvendigvis tilføjer noget værdi. Der ligger derfor her en del arbejde i at formulere dem på skrift, uden at det egentlig kan benyttes i implementeringen. Gruppen ønskede derfor at finde en metode til at udforme de funktionelle krav, som lagde mere op til en iterativ udvikling, hvor der ikke skulle dokumenteres en masse på skrift, der ikke blev benyttet i implementeringen.

Der er derfor valgt at benytte user stories til at formulere de scenarier, der udgør de funktionelle krav. En user story er en kort beskrivelse af scenariet, samt hvad der skal til, før implementeringen af det kan godkendes. Detaljeringen af scenariet er udskudt, indtil det skal designes og implementeres. Til den tid hører der en diskussion med kunden til, hvor han fortæller, hvordan han ønsker, den specifikke funktionalitet for scenariet skal være, og dette diskuteres i forhold til, hvad der er muligt. Det er ved denne diskussion, at detaljeringen fastsættes.

Fordelen ved at benytte user stories er, at der ikke skal formuleres en masse detaljer på skrift. Detaljerne fastlægges i stedet mundtligt med kunden inden implementeringen af den. User storien fungerer her i stedet som en reminder om den diskussion, der har været med kunden, samt de detaljer der blev fundet frem til, at der skulle gælde. Dette kræver dog, at der er en vis tillid mellem kunden og udvikleren, eftersom der ikke skriftligt formuleres en masse detaljer, der gælder som en aftale, som der gøres ved use cases.

Da der er valgt at benytte user stories, hvor en stor del af arbejdet med dem ligger i diskussionen med kunden, men projektet ikke har en egentlig kunde, er dette håndteret på en alternativ måde. Det er gruppen selv, der har stået for produktets vision og ageret kunde, og diskussionen er derfor håndteret ved, at gruppen sammen med vejlederen har diskuteret i starten af hver iteration omkring ønskerne og detaljerne for de specifikke user stories, der skulle implementeres i iterationen.

### Aktører

**Forbruger:** Forbrugeren er en primær aktør. Forbrugeren ønsker at gøre hans indkøb så billigt og let som muligt, ved at få Pristjek220 til at finde ud af hvor han skal købe hans forskellige varer.

**Forretningsmanager:** Forretningsmanageren er en primær aktør. Forretningsmanageren kan tilføje/fjerne varer fra deres forretning, samt ændre priserne.

**Administrator:** Administratoren er en primær aktør. Administratoren kan tilføje og fjerne forretninger fra pristjek220.

### User story beskrivelser

Der er her udvalgt nogle relevante User Stories fra Kravspecifikationen (Gruppe7, 2016), ud fra hvilke User Stories der bliver brugt i løbet af rapporten, de er rangeret efter hvilken prioritering de har haft gennem udviklingsfasen.

//DER ER PÅ NUVÆRENDE TIDSPUNKT, LAVET NOGLE EKSEMPLER PÅ US, MEN DA JEG IKKE VED HVILKE VI BIL BRUGE, KAN DER GODT FOREKOMME ÆNDRINGER SENERE.

**US2: Finde den billigste forretning for en vare i Pristjek220:**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne finde den billigste forretning for en vare fra Pristjek220,

**Så** der kan laves en indkøbsliste med den mindste pris.

User storyen er relevant, fordi det er en fundamental egenskab, for at pristjek220 kan hjælpe forbrugeren med at fortage billige indkøb.

**US4:** **Find ud af hvor varerne fra indkøbslisten kan købes billigst:**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne se hvor det er muligt at købe varerne på indkøbslisten billigst ud fra de indtastede indstillinger for indkøbslisten,

**Så** jeg kan spare penge på mine indkøb.

User storyen er relevant, fordi uden indkøbsliste funktionaliteten ville det blive for besværligt at bruge Pristjek220, hvis der skulle handles flere varer.

**US8: Autofuldførelse:**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne se hvilke vare og forretninger, der allerede findes i Pristjek220, når jeg sidder og søger,

**Så** der skal komme forslag op som jeg kan trykke på.

User storyen er relevant, fordi brugeren gerne vil kunne slippe for at skrive hele den ønskede ting og samtidig være sikker på at produktet er i Pristjek220, uden at skulle slå det op først.

**US9:** **Send indkøbsliste på mail:**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne modtage min indkøbsliste/forslag til indkøbssteder på mail,

**Så** min indkøbsliste bliver mobil.

User storyen er relevant, fordi brugeren gerne vil kunne slippe for at have hans computer med ud og handle, og derfor ønsker en mere transportabel måde at have sin indkøbsliste med på. Dette kan opnås ved at han sender en E-mail til sig selv, som så kan se på en telefon.

### Kvalitetskrav

Der er taget et uddrag af de mest relevante kvalitetskrav (Gruppe7, 2016).

**1a: Pristjek220, skal leve op til de krav, som Microsoft, stiller til UX design** (Microsoft, 2016)**.**

For at få en brugergrænseflade der er brugervenlig og som giver en intuitiv ide om hvad der skal gøres, som bruger. Der er derfor taget udgangspunkt i de krav, som Microsoft stiller til en desktop applikation, for at overholde en generel standard.

For at opfylde de krav, som der stilles fra Microsoft UX design, er tjeklisten (Microsoft, 2016) som de har udgivet, blevet sammenlignet med produktet under udviklingen.

**1b: Pristjek220 skal leve op til 3-click rule** (Zeldman, 2001)**.**

For at få en brugergrænseflade som er let at navigere i er der taget udgangspunkt i 3-Click rule, som normalt bruges til hjemmeside navigation, som siger at der skal kunne navigeres til forskellige funktionaliteter ved bruge at tre klik.

Der er i Pristjek220 i navigationen levet op til ” til 3-click” ved at der ingen steder er mere end 3 klik fra hvor man starter til man er ved den ønskede funktionalitet.

**1c: Pristjek220 skal leve op til rule of five** (Zeldman, 2001)**.**

For at få en brugergrænseflade som er let at navigere i er der taget udgangspunkt i rule of five, som normalt bruges til at få en hjemmeside der er overskuelig. Denne regel siger at man skal prøve at begrænse antallet af elementer man kan vælge fra en menu til fem.

Der er i Pristjek220 i de forskellige menu’er levet op til ”rule of five” ved at der ingen steder er mere end 5 forskellige valgmuligheder.

**4b: Kodeord der ligger i databasen skal være krypteret**

For at få et produkt, som er mindre sårbart over for angreb er der valgt at kodeordene der ligger i databasen skal være krypteret.

Der er i Pristjek220 valgt at krypterer kodeord med SHA256, ud fra ideen om at det ikke skulle være muligt at hack databasen og derved få adgang til ikke krypterede kodeord.

# Projektgennemførelse

I dette afsnit vil der blive forklaret hvordan gruppen har gennemført projektet.

## Iterativ udvikling

Der er besluttet at der skal arbejdes iterativt, derfor har gruppen måtte tage et valg til hvilken arbejdsmetode, der skulle benyttes. Valget lå mellem Scrum (Schwaber & Sutherland, 2013)og Kanban (Peterson, 2015).

Begge metoder anvender et taskboard til at organisere det arbejde, der skal laves, så alle medlemmer på teamet kan se, hvad der bliver lavet og af hvem. Den store forskel er dog, at i Kanban er der en begrænsning på hvor mange tasks, der må være aktive ad gangen, og når der er plads, bliver der bare fyldt på fra en product backlog. Hvorimod i Scrum bliver arbejdet opdelt i sprints, og hvert sprint har sit eget taskboard. Disse sprints bliver udfyldt med stories, som nedbrydes til tasks, fra en product backlog, inden de opstartes, og hvis alle tasks er udført, kan man tilføje flere fra product backloggen, og hvis man ikke når alt, føres de videre til næste sprint med højeste prioritet. Generelt set egner Scrum sig bedre til udvikling af systemer, og Kanban fungere bedst til vedligeholdelse af systemer fordi. Scrum sprintene er designet efter et færdigt produkt ved hvert sprint retrospektiv, så hvis kunden ikke har flere penge at smide i projektet kan han stadigvæk gå derfra med et produkt, med nogle funktioner implementeret. Med hensyn til vedligeholdelse af andre systemer vil der højst sandsynligt dukke bugs op i produktet, og med et Kanban system kan bugs hurtigt blive givet en prioritet og blive fixet.

Udviklingen af Pristjek220 er foregået med Scrum, fordi det er et nyt system der bliver udviklet. Det var givet fra starten hvornår deadline for projektet faldt, og med en beslutning på at køre sprints af 14 dages længde. Så det var kendt viden, hvor mange sprints der ville forekomme, og derved kunne gruppen danne et overblik over hvor meget tid der ville være til rådighed, hvis et nyt aspekt fra kravspecifikationen, skulle implementeres.

Yderligere har gruppen fra starten hældt mere mod Scrum, både fordi at sådan har arbejdsfaconen været på tidligere semesterprojekter, og alle gruppens medlemmer har gennemført et Scrum kursus på Systematic A/S (Systematic, 2016).

Gruppen har afveget fra Scrum standarden og ikke tildelt nogen Scrum master rollen, fordi der hverken var nogen kunde, eller product owner, som han skulle stå for kommunikationen med. Yderligere er der gjort erfaringer med opsætningen af product backloggen, især med fokus på hvordan taskene skulle opskrives. Fra tidligere semestre var gruppen oplært i at et taskboard skulle bestå af mange små opgaver, hvori man ofte kunne rykke en opgave fra in progress til review / done. Men denne gang var det essentielt at taskene skulle give værdi for storien. Fra starten af projektet forløbet var det begrænset hvor meget værdi de individuelle tasks gav storien, såsom task der beskrev at der skulle skrives unittests. Men efter et par sprints begyndte det at give mening, fordi det føltes mere naturligt at arbejde på den metode. Man var færdig når man kunne rykke sin task, og ikke når man havde rykket 5-6 stykker. Idéen med de små opgaver blev dog ikke helt udraderet, de blev beskrevet i de individuelle tasks’ beskrivelses felt, og derigennem kunne gruppen arbejde på deres foretrukne metode, imens deres tasks skabte værdi for storien.

## Dokumentation af koden

En god dokumentation af source koden er værdifuld for projektet, for let at kunne danne sig overblik over koden, så det er let at sætte sig ind i, og dermed også let at overskue hvilke funktionaliteter der er tilgængelige. Manuel dokumentation af klasser og metoder i koden er tidskrævende, og egner sig ikke godt, når der laves ændringer i koden, da der derved er en masse dokumentation, der skal rettes for at holde dokumentationen opdateret. Det ville derfor være optimalt, hvis klasser, metoder og variabler i koden automatisk kunne registreres, så man kun behøvede at skrive lidt beskrivende tekst til, og derved ikke selv skulle til at skrive hvilke klasser, metoder og variabler, der eksisterer i koden.

Der er derfor benyttet Sandcastle (EWSoftware, 2016) som dokumentationsgenerator i dette projekt. Den er valgt, da den er godt integreret med Microsoft Visual Studio, C# og .Net Frameworket. Den største og vigtigste fordel ved at benytte Sandcastle er, at opsætningen af dokumentationen sker automatisk, og der derved er mindre vedligehold for at holde dokumentationen opdateret. Programmørens eneste ansvar er derfor at skrive sigende kommentarer i koden, til den metode eller lignende der er blevet skrevet.

## Dokumentation af systemarkitektur

Til at beskrive systemarkitekturen for projektet er der i de tidligere semesterprojekter anvendt flere forskellige SysML diagrammer. Fælles for disse er, at de egner sig bedst til systemer, hvor der indgår hardware og software. Dette projekt er derimod kun et softwareprojekt, og der var derfor et behov for at finde en anden metode til at beskrive softwarearkitekturen, som fokuserede mere på softwareintense projekter.

Af denne grund bliver ”4+1” (Kruchten, 1995) view modellen benyttet til at beskrive systemarkitekturen. Modellen består af fire views, der bruges til at beskrive systemet fra forskellige synspunkter. De fire views er logical view, development view, process view og deployment view. Deployment view og process view er ingeniørens tilgangsvinkel til systemet. Kundens tilgang til systemet er fra logical view. Projektledere bruger development view til at se projektet fra. Derudover består modellen også af et sidste view, som omhandler scenarierne i systemet. Disse scenarier beskriver de interaktioner, der finder sted i systemet, og er beskrevet i form af use cases eller user stories. Scenarierne er frigjorte fra de andre views, og bruges til at finde frem til elementerne i arkitekturen, og de fire andre views er derfor illustreret ved hjælp af disse scenarier.

Fordelen ved at benytte denne model er, at man får en systemarkitektur, der er godt beskrevet fra flere forskellige relevante synspunkter. En stakeholder i systemet har derved nem adgang til de informationer, der er behov for, for at forstå systemet på hans niveau. Når man har tilføjet en funktion, er modellen samtidig god til at sørge for, at man kommer rundt omkring alle views, og får overvejet, om man har ændret noget i forhold til de forskellige views. Derved er man sikker på, at man får dokumenteret det, hvis man f.eks. har lavet ændringer i samtidigheden. Ulempen ved modellen er derimod også, at der nemt kan bruges alt for meget tid i systemets udviklingsproces på at dokumentere igennem en masse forskellige diagrammer i de forskellige views. Det er vigtigere at have et fungerende system frem for en komplet dokumentation, da dette sløver udviklingsprocessen mere, end det gavner. Derfor er der prioriteret efter hvilke views og diagrammer, der giver mest mening at benytte i projektet, og de resterende diagrammer er derfor udeladt. Dette betyder, at der i Pristjek220 ikke er benyttet process view, samtidig med at det kun er nogle bestemte diagrammer, der er benyttet i de andre views, da det var dem, der blev vurderet til at være mest værdifulde. For en gennemgang af hvilke diagrammer der er benyttet, henvises der til dokumentationen under 4+1, hvor selve diagrammerne for projektet også kan findes.

# Systemdesign

## Arkitektur

Da arkitekturen af projektet skulle besluttes, blev der valgt at bruge en lagdelt model. Grunden til at valget faldt på den, var at ved den lagdelte model er der en gruppering af klasser og pakker. Herved har de klasser, der befinder sig i ét lag, et sammenhængende ansvar for et vigtigt aspekt i systemet i den logiske separation, der er lavet i projektet. Ved at gruppere systemet på denne måde, kan der hurtigt findes ud af, hvor der skal ændres ting, hvis der er noget, der skal ændres. Med valget af at bruge den lagdelte model skulle der så tages en beslutning om, hvor mange lag der skulle inddeles i. Ud fra størrelsen af systemet gav det mest mening at bruge en 3 lagdelt model (Microsoft, 2016). Den 3 lagdelte model, består af; DAL, BLL og PL.

Et eksempel på hvordan et normalt scenarie kunne se ud, kan ses på Figur 3, som viser hvad der sker, når en bruger indtaster i et felt, der kan autofuldføre. Først ændrer brugeren på teksten på GUI’en, hvorefter GUI’en kalder ned på GUI viewmodel, for at den skal udfylde listen. Dette sker i PL, hvorefter der så kaldes en funktion i klassen AutoComplete i BLL. Denne funktion kalder så en funktion i en klasse i DAL, som laver et database udtræk ud fra den indtastede tekst.



Figur : Sekvens diagram for Autofuldførelse, med lag opdeling

Hver af disse lag kan så bestå af flere klasser og pakker, som det kan ses på Figur 4, som viser et package diagram for Pristjek220. Hver pakke er så inddelt under de forskellige lag (PL, BLL og DAL). Hver pakke indeholder så klasser, som har de egenskaber, som pakken beskriver. I programmet ligger de forskellige klasser under pakkens namespace.



Figur : Package diagram for Pristjek220

Ved at der er valgt at bruge en lagdelt model og under hvert lag have forskellige klasser, kommer der en separation af applikationsspecifikke tjenester fra de mere generelle tjenester, som gør SRP bliver overholdt. Derudover kommer der en separation af højniveaushandlinger fra lavniveaushandlinger, som gør, at DIP bliver overholdt. Ved at overholde SRP fås et system, som er mindre sårbart over for ændringer på et senere tidspunkt. Koblingen og afhængighederne formindskes deraf mellem de forskellige klasser, samt der kommer høj samhørighed. En af de vigtigste ting ved at bruge 3 lags modellen er dog, at der er muligheden for at genbruge kode, sådan at man for eksempelvis kan genbruge de nederste lag af kode til en applikation med en anden brugergrænseflade. Det gøres endnu mere simpelt ved, at de forskellige lag er implementeret med interfaces, som simplificerer en ændring eller udskiftning af dele af systemet. Ved at lave den logiske segmentering mellem de forskellige lag er klarheden af koden øget, for andre der skulle ønske at arbejde videre med koden.

## GUI design overvejelser

Under designet af GUI’en dukkede et problem op i form af det er svært at unitteste den, og derfor kan kodens funktionalitet ikke verificeres, medmindre der anvendes et MVVM (Microsoft, 2012) pattern.

Ved brug af MVVM bliver bindingen mellem GUI’en og buisness logikken, løsnet idet at viewet primært er defineret i XAML filen, med en begrænset code-behind. Ved at binde viewet til en ICommand, der ligger i view modellen, kan der kaldes funktioner fra viewet nede i view modellen og ved databindings kan der deles data. Dette er illustreret på Figur 5



Figur - MVVM model (Microsoft, 2012)

For eksempel, i Pristjek220 kan en forretningsmanager tilføje en ny vare til sin forretning. Når varens navn og pris, er indtastet og der bliver klikket på ’Tilføj vare’, tager view modellen navnet og prisen og forsøger at sende dette videre til modellen. Hvis dette er succesfuldt, bliver varen gemt i databasen og en bekræftelsestekst bliver sat, som et label i GUI’en er bindet til, og derved kan forretningsmanageren se at hans vare tilføjelse var en succes.

Fordi at MVVM laver en opdeling af GUI’en i et view og en viewmodel, er det muligt for teamet at arbejde på begge ting, uden at skulle frygte at ødelægge hinandens kode. Ved at indsætte en driver imellem viewet og viewmodellen, bliver det muligt at unit teste GUI’en.

En anden fordel ved MVVM er, at hvis der i Pristjek220 ønskes at den generede indkøbsliste skal vise prisen i både kroner og euro, så er view modellen oplagt til dette. En view model’s primære opgave er at tage dataen fra modellen og formatere det således at det tilpasser viewet’s ønske. Så i stedet for at tilføje prisen i euro helt nede i databasen, bliver prisen bare konverteret til euro i view modellen og derfra bindet til en ny en kolonne i den generede indkøbsliste.

## Pristjek220 Database

### Design af databasen

I Pristjek220’s database er der fire forskellige entiteter, og de hedder Store, Product, HasA og Login. Mellem entiteterne Store og Product er der en mange-til-mange relation, da én forretning kan sælge mange produkter, og ét produkt kan blive solgt i mange forretninger. Denne relation bliver normalt selv oprettet, hvis relationen ikke har nogle andre properties. Da en forretning ikke nødvendigvis sælger et produkt til den samme pris som i andre forretninger, var det i Pristjek220 nødvendigt at have en property til produktets pris på relationen mellem forretningen og produktet. Denne property skal indeholde, hvad prisen for produktet er i lige præcis den forretning, det tilhører.

Løsningen på dette problem blev erfaret gennem undervisningen i DAB. Her blev der fundet frem til, at en mange-til-mange relation, hvor der er brug for properties på relationen, skal have oprettet en entitet til relationen. Af denne grund blev entiteten HasA oprettet i Pristjek220. HasA har derfor en property til prisen, samt en relation til en forretning og et produkt, for at binde de to entiteter sammen. Derved kan Pristjek220 håndtere, at en ny forretning åbner og oprette HasA entiteter til de produkter, som er i den nye forretnings sortiment. Ligeledes, hvis en forretning får et nyt produkt i deres sortiment, vil der blive oprettet en HasA entitet mellem forretningen og produktet. Denne håndtering gør samtidig, at der kun skal være én udgave af hvert produkt og af hver forretning, da der kan bindes mange HasA entiteter mellem dem. Modellen for databasen kan ses på Figur 6.



Figur UML diagram for databasen.

Tabel : Første udkast af opsætningen af databasen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Aldi | Fakta | Føtex | Kiwi | Rema1000 |
| Banan | 2.95 | 1.95 | 2.50 | 2.55 | 2.70 |
| Tomat | 2.90 | 2.40. | 1.95 | 2.60 | 2.30 |

Inden denne erfaring var planen med den første udgave af databasen, at den skulle indeholde en enkel tabel, hvor man kunne se produktets navn i rækkerne og de forskellige forretninger hen ad kolonerne, og på den måde finde prisen for produktet i den enkelte forretning som kan ses på Tabel 1. Dette viste sig at have nogle problemer, da Product klassen skulle laves med en variabel for hver forretning, som prisen kunne gemmes i. Det resulterede i, at databasen ikke ville være åben for udvidelser, hvilket er ineffektivt, når forretninger kan åbne og lukke, og forretningerne får nye produkter i deres sortiment.

For at forretningerne ikke kan ændre prisen på et produkt i en anden forretning, eller ændre på andre forretningers sortiment, er der lavet et login til hver forretning, og alle logins gemmes i databasen. Denne entitet har et brugernavn, kodeord, og en reference til den forretning, den har kontrol over. Når der logges ind for en forretningsmanager, har han derved kun mulighed for at ændre, hvordan hans egen forretning skal fremstå. Derudover er der lavet så Administratoren, som kan tilføje og slette forretninger, også har et login. Dette login er specielt i forhold til de andre, da det giver adgang til administrationsdelen af forretningerne. Derfor er Administratorens login blevet lavet med en forretning, der hedder Admin, og på den måde tages der i programmet højde for, hvilken retning administrationsprogrammet skal tage efter login.

### Databaseadgang

Gennem Pristjek220’s udvikling blev der erfaret, at det var besværligt at unit teste BLL da der var for hård binding mellem BLL og DAL. Det vil sige at det ikke var muligt at isolere BLL fra DAL, og da unit tests kræver at klassen, som skal testes, skal isoleres fra resten af koden, var det ikke muligt at teste BLL på denne måde. Et andet problem var, at meget af det kode, som blev skrevet, blev duplikeret, da der var brug for det forskellige steder.

Der blev derfor implementeret et Repository pattern for at separere BLL fra DAL. Derved blev det muligt at isolere BLL, så det kunne unit testes da repositoriet kan substitueres. Repository pattern kommer dog stadig med den ulempe, at det så har en hård binding ned til databasen, og ikke kan isoleres fra den. Derfor er det ikke blevet unit testet, men er i stedet blevet integrationstestet med databasen. Udover at lave en separation mellem BLL og DAL giver repositoriet også den fordel, at den laver et abstraktionslag til databasen, hvor man på den måde kan samle al adgang ned til databasen, og derved undgå at samme kode skal skrives flere steder.



Figur : Implementering af Repository pattern i Pristjek220.

På Figur 7 kan der ses, hvordan Repository pattern’et er blevet implementeret i Pristjek220. De forskellige repositories indeholder CRUD funktionerne, for den tabel de hører til. ProductRepository indeholder derved funktionerne til Product tabellen i databasen. I Repository klassen, som de specifikke repositories nedarver fra, ligger de generelle funktioner som Add og Remove for at undgå duplikeret kode. Unit of Work er lavet som et access point til repositoriesne fra BLL. Det samler alle repositoriesene i en klasse så administrationen og forbruger ikke skal have alle repositoriesene med når de oprettes. Derudover giver Unit Of Work også den fremtidsmulighed, at der kan implementeres funktioner hvor der kan tilføjes eller fjernes mange ting på én gang, uden at gemme efter hver tilføjelse.

# Produkt beskrivelse

I dette afsnit vil der blive gennemgået funktionaliteterne i produktet, hvor der først vil blive gennemgået den fælles funktionalitet mellem forbruger- og administrationsapplikationen. Derefter vil blive forklaret om forbrugeren og derefter om administrationsdelen, hvor der først bliver forklaret om administratoren og derefter forretningsmanageren.

## Delte funktionaliteter

Der er delt funktionalitet mellem administrations- og forbrugerapplikationen. En af funktionaliteterne er autofuldførelse, som ligger i tekstboksene. Funktionen tjekker hvad der ligger i databasen og sammenligner det, med det som brugeren allerede har skrevet. Derved hjælper autofuldførelse brugeren til at se om produktet eller forretningen findes i Pristjek220. Figur 8 viser er der blevet skrevet ”cho”, hvor autofuldførelse så anbefaler de tre første produkter der starter med ”cho”, alle produkterne som bliver anbefalet er produkter der ligger i Pristjek220. Autofuldførelse anbefaler kun tre produkter, for ikke at forvirre brugeren, med for mange muligheder. Det er blevet valgt efter forbruger test, hvor brugerne syntes det var for uoverskueligt at have fem anbefalinger, som var det første antal som gruppen havde sat.



Figur : Autofuldførelse

## Forbruger

Hovedessensen er at kunne lave og generere en indkøbsliste, hvor der så bliver udregnet hvor det er billigst at købe de forskellige produkter. På Figur 9 og Figur 10 kan der ses et udsnit af brugerinterfacet for indkøbslisten, som det ser ud i Pristjek220.

Figur 9: Pristjek220 indkøbsliste



På Figur 9 ses, at produkter kan indskrives, og tilføjes til indkøbslisten. Derefter kan der vælges det ønskede antal af det produkt. Herudover kan der slettes et produkt, eller hele indkøbslisten. På Figur 10 ligger indstillingsmuligheder, hvor der kan vælges hvilke butikker, der skal handles i. Derudover er der mulighed for at generer en indkøbsliste, hvor Pristjek220 udregner hvor det er billigst at handle, med den udfyldte indkøbsliste, hvor der tages højde for de forretninger der er tjekket af.

Den genererede indkøbsliste viser, hvor varerne skal købes. På Figur 13 kan man se at indkøbslisten fra Figur 9 er indsat, med de produkter der findes i forretningerne. Produkterne der ikke findes i en forretning, bliver lagt i en separat liste. På Figur 13 kan man se hvor det er billigst at købe hvert enkelte produkt, og på Figur 11 vises hvad det koster at købe det hele i en butik. Her kan man se at det koster 1,05 kr. mere at handle det hele i Rema1000, end at tage i Fakta for 2 af produkterne. På Figur 13, kan der klikkes på de enkelte produkters forretninger og derved skifte til en anden forretning at handle det produkt i. Derved udregner Pristjek220 en ny genereret indkøbsliste. Der er i den genereret indkøbsliste mulighed for at sende sin indkøbsliste på mail som kan ses på Figur 11.



Figur : Pristjek220 indkøbsliste indstillingsmuligheder



Figur 11: Pristjek220 genereret indkøbsliste sum, besparelse og E- mail



Figur : Pristje220 søg efter produkt



Figur : Pristjek220 Genereret indkøbsliste

I forbrugerapplikationen er der også mulighed for at søge efter et produkt. Pristjek220 viser så hvilke butikker der sælger det, og hvad prisen er for hver forretning. Som der kan ses på Figur 12 er der blevet søgt efter ”banan”, hvor Pristjek220 viser hvor banan sælges.

## Administration



Figur : Pristjek220 Login

Administrationsapplikationen består af to dele, administrator og forretningsmanager. Når Administrationsapplikationen bliver startet, kommer der en login skærm. Derfra kan man logge ind som Administrator eller Forretningsmanager, login vinduet ses på Figur 14.

### Administrator

Administratoren kan tilføje til og fjerne forretninger fra Pristjek220. For at tilføje en forretning skal man angive forretningsnavn og kodeord, derefter kan man trykke tilføj forretning, som kan ses på Figur 15. Hvorefter Fakta bliver tilføjet til Pristjek220, og man kan logge ind med Fakta og det tilhørende kodeord.



Figur : Pristjek220 tilføj forretning

Som administrator kan man også fjerne en forretning, som kan ses på Figur 16. Hvor man indtaster en forretning der skal fjernes og klikker fjern forretning, på Figur 16 er det Fakta der er ved at blive fjernet.



Figur : Pristjek220 fjern forretning

### Forretningsmanager

Forretningsmanageren har ansvaret for sin egen forretning og kan derfor tilføje, ændre og slette produkter. For at tilføje et produkt til forretningen skal der indtastes et produkt og en pris, som der kan ses på Figur 17. Hvorefter et pop-up vindue bliver vist, hvor man skal bekræfte om oplysningerne er korrekte. Hvis produktet allerede findes i forretningen, bliver der gjort opmærksom på at det allerede findes.



Figur : Pristjek220 tilføj vare

Efter et produkt er tilføjet til forretningen kan der til enhver tid ændres på prisen for produktet, hvis der for eksempelvis er tilbud. Herefter vil der blive vist et pop-up vindue med bekræftelse af ændringen af prisen.



Figur 18: Pristjek220 ændre pris på vare

Forretningsmanageren har også muligheden for at fjerne produkter fra sin forretning, hvis f.eks. et produkt udgår af produktion. Hvis man prøver at skrive et produkt der ikke findes i forretningen, bliver der informeret om det. Figur 19 viser at for at fjerne et produkt for en forretning skal produktet indtastes og klikkes på fjern. Her efter vil et pop-up vindue med bekræftelse af fjernelse af produktet.

Figur : Pristjek220 fjern produkt



# Test af pristjek220

For at lave et godt produkt, og sikre at produktet lever op til product owner’s forventninger, bliver produktet udsat for nogle forskellige former for test. Dette er illustreret på Figur 20, hvor der kan ses de forskellige teststadier, som er unittest, integrationstest, systemtest, accept test. Gennem udviklingen af Pristjek220 er der brugt FDD, hvilket giver den fordel at man kan udvikle en feature til programmet, hvorefter der kan bestemmes om den skal sættes i produktion. Derefter kan der opstilles tests, således at den lever op til kravene. På den måde spares der en masse tid på de funktioner, som product owner beslutter sig for, han ikke vil have. TDD er modsat, der skrives test før man laver features, hvilket giver den fordel, at koden lever op til testenes kravene med det samme.



Figur : V-model for udvikling af produkt. (Jensen & Jakobsen, 2016)

## Unittest

Unit testning er det første teststadie, som programmet kommer igennem, da det tester de individuelle klasser isoleret fra resten af systemet. Det vil sige at når en klasse får tilføjet en funktion, kan den testes uafhængigt af, hvor langt udviklingen af resten af systemet er. Gennem unit testning sikres der, at hver klasse individuelt opfører sig på den forventede måde ved de givne inputs, og derved bliver programmet kvalitetssikret, så det lever op til forbrugernes forventninger. Til unittest af Pristjek220 er der blevet benyttet Nunit frameworket og Nsubstitute, til at kunne opstille de forskellige testscenarier, og lave unittestene automatiske. Nsubstitute er også et framework, som bruges til at substituere de forskellige klasser, som klassen, der testes, har relationer til. Derved isoleres klassen fra resten af systemet. Det at automatisere testene giver den fordel at programmøren ikke skal bruge tid på at teste manuelt flere gange. Derudover giver automatiske test muligheden for, at koden kan optimeres efterhånden. Denne optimering kan så blive kørt igennem de automatiserede tests, uden at skulle bruge for lang tid på at teste manuelt, at funktionaliteten stadig fungere, som den skal. Automatiske unittests er dog ikke den gyldne hammer, der løser alle problemer, da de automatiserede tests kun tester, det de er lavet til at teste, hvilket vil sige, at nogle ting, kan blive overset. Derudover er der nogle ting som unittests ikke kan teste, så som brugervenlighed og hvordan den enkelte klasse opfører sig, når den sættes sammen med resten af systemet.

## Integrationstest

Da unit test tester hver klasse isoleret fra resten af systemet, tester man kun på at den enkelte klasse fungerer som forventet. Der kan dog forekomme nogle problemer, når de sættes sammen med andre klasser, hvis udviklerne har forstået interaktionen mellem klasserne på forskellige måder. Derfor bliver produktet integrationstestet så disse fejl bliver opdaget og kan blive rettet. Når man skal lave integrationstest skal man finde ud af hvordan klasserne er afhængige af hinanden og derfor hvor man har brug for at lave integrationstest henne. På Figur 21 kan man se et udsnit af Pristjek220’s dependency tree, hvor man kan se at produktet overholder DIP, da højniveau kalder lavniveau og ikke omvendt.



Figur : Udsnit af dependency tree for Pristjek220.

Der er forskellige måder at klare integrationstest på for at komme rundt om det hele. Integrationstest af Pristjek220 er gjort med strategien ”Bottom-up”, for let at kunne dække alle interfaces mellem klasserne. ”Bottom-up” kræver dog man skriver mange drivers til hvert lag når der testes, men der behøves dog ingen stubs når det testes med nogle få undtagelser som fx generering af tilfældige tal.

## (evt. Code Metrics)

## CI

Gennem udviklingen af projekter kan man komme ud for, at de automatiserede tests kan tage lang tid at køre igennem. Derfor kan man bruge en CI server til at køre testene, så der kan arbejdes videre, imens testene kører igennem. I større projekter der udvikles på, bliver testene kørt igennem flere gange om dagen, og derfor ville programmøren bruge lang tid på at sidde og vente på, at testene er færdige, før han kan arbejde videre.

Ingeniørhøjskolen har stillet en Jenkins CI server til rådighed, som er blevet benyttet gennem udviklingen af Pristjek220. Jenkins er et rigtig godt værktøj, da der spares tid på tests som forklaret før. Dog er det ikke blevet benyttet så meget til udviklingen af Pristjek220, da det ikke er et stort system, og der fås svar på testene inden for få sekunder, når det køres på computeren.



Figur 22: CI server Test/build graf.

Figur 22 viser antallet af tests pr. byg fra CI serveren for Pristjek220, hvor det blå er succesfulde tests, og de røde er tests, der er fejlet, CI serveren er indstillet til hvert 15 minut til at tjekke om der er lavet en ændring i projektet, hvis der er bygger den projektet og køre alle tests igennem. Gennem brugen af CI serveren fandt gruppen ud af, at Jenkins havde nogle uoverensstemmelser med dansk/engelsk. Nogle tests virkede fint på computeren, mens CI serveren fejlede på testene, da den ikke kunne punktum og komma for priserne for produkterne. Som det kan ses ud fra Figur 22, gik der noget tid, før dette blev opdaget. Dette skyldes, at testene var hurtigere at køre på computeren, og ikke fejlede der. Da dette så blev opdaget, ændrede gruppen den specifikke kultur for decimaltal, så det stemte overens både på computerne og CI serveren.

## Forbrugertest

# Resultater og diskussion

# Fremtidigt arbejde

Til videreudvikling af produktet vil det være oplagt at gøre forbrugerapplikationen til en smartphone app, således at brugeren får mulighed for at lave ændringer til sin indkøbs, uden at skulle være hjemme ved sin PC. Yderligere ville dette give mulighed for at integrere systemet med telefonens GPS, og derved planlægge den korteste route til alle butikkerne. Derudover ville være en hjælp hvis en bruger, gennem Pristjek220, kunne se butikkers aktuelle tilbudsaviser, samt åbningstider. Forbrugeren vil også have mulighed for at begrænse afstanden han ønsker at køre hjemmefra, for at nå alle butikkerne, samt bestemme det maksimale antal butikker han ønsker at handle ind i.

For at hurtigt at udfylde indkøbslisten ville det være en mulighed at oprette opskrifter. Disse opskrifter vil indeholde en ingrediensliste. Andre brugere vil således være i stand til at se opskrifterne, og skulle de føle sig fristet, kan de tilføje alle ingredienserne til indkøbslisten. Dette ville også være en oplagt metode at planlægge en madplan for ugen.

Til selve forretningsmanagerapplikationen af produktet, ville det være oplagt at synkronisere databasen med forretnings egen database, således at vare sortimentet bliver opdateret automatisk, og derved gøre det let at styre tilbudspriserne. Derudover vil en metode til at se hele vare sortimentet i databasen være anvendeligt, samt mulighed for at sætte specifikke varer til udsolgt.

På tværs af systemet ville det være oplagt at angive varer som økologisk eller gluten fri. Således kunne forbrugerens applikation have mulighed for at kun søge efter varer hvor disse kriterier er opfyldt. Derved kunne Pristjek220 virkelig lette arbejdet for folk med disse krav, eller allergier, til deres dagligvare indkøb.

For at kunne implementere produktet i andre lande end Danmark, ville det være oplagt at kunne få brugergrænsefladen fremvist på engelsk.

# Konklusion

# Referencer

# Underskrifter

|  |  |
| --- | --- |
| Anders Meidahl Münsberg |  |
|  | 201404246 |
| Christian Slot Winkel |  |
|  | 201370493 |
| Mette Grønbech |  |
|  | 201305561 |
| Nicklas Nielsen |  |
|  | 201370286 |
| Rasmus Brædder Hemmingsen |  |
|  | 201404469 |