

# PROJEKTRAPPORT

---

## PRISTJEK220

DATO FOR AFLEVERING: 27-05-2016

AARHUS UNIVERSITET - INGENIØRHØJSKOLEN

| VEJLEDER: TROELS FEDDER JENSEN (TFJ)

GRUPPE 7



## 1 RESUMÉ

Denne rapport beskriver et 4. semesterprojekt på Ingeniørhøjskolen Aarhus Universitet. Formålet med denne rapport er at redegøre for udviklingsforløbet af Pristjek220.

Pristjek220 er realiseret gennem to computerapplikationer; en forbruger- og en administrationsapplikation. I forbrugerapplikationen kan brugeren søge efter et produkt, og se hvilke forretninger produktet kan købes i, samt til hvilken pris. Derudover kan han lave en indkøbsliste, hvor han kan tilføje produkter, vælge antal samt vælge, hvilke forretninger han vil handle i. Derefter kan applikationen generere en indkøbsliste, hvor der bliver udregnet, hvor produkterne kan købes billigst. Der kan også, inde i den genererede indkøbsliste, skiftes, hvilken forretning han vil handle hvert enkelt produkt i. Den genererede indkøbsliste viser endvidere, hvor meget brugeren sparer, i forhold til at købe det hele i én forretning. Computerapplikationen gør det muligt for forbrugeren at overføre indkøbslisten til sine mobile enheder via mail, så den kan tages med ud at handle.

Administrationsapplikationen er bestående af to enheder; en til forretningsmanagerne og en til administratoren. Administratoren styrer Pristjek220 ved at tilføje og fjerne forretninger fra applikationen. Som forretningsmanager administrerer man én forretning, hvor man kan tilføje, fjerne og ændre produkter, der skal være i forretningens sortiment. Når man åbner administrationsapplikationen, kommer der en loginskærm, hvor man skal indtaste brugernavn og kode, hvis man skal lave ændringer til Pristjek220.

Applikationerne kommunikerer med en database, hvor informationerne gemmes. Forbrugerapplikationen kan ikke skrive i databasen, men kan imidlertid kun læse informationerne fra den. Det er derimod administrationsapplikationen, der bestemmer, hvad der skal stå af informationer i databasen.

Gruppen har arbejdet iterativt gennem projektet, og der er blevet brugt Scrum som udviklingsproces. Projektet er opsat til en Jenkins CI server, som er blevet brugt til at køre testene af produktet. Derudover er der blevet lavet en brugertest for at teste brugergrænsefladen.

## 2 ABSTRACT

This report describes a 4<sup>th</sup> semester project on Aarhus University School of Engineering. The purpose of this report is to describe the development process of Pristjek220.

Pristjek220 consists of two computer applications: a consumer- and an administrations application. In the consumer application, the user is able to search for a product and can then see where the product is available, along with its price. Furthermore, he is able to create a shopping list, where he needs to fill in which, and how many, products he needs, and which stores he wishes to visit. After this, the application can generate a shopping list, which contains the cheapest stores to purchase the products. In the generated shopping list, the user can also change the store in which he wishes to buy the individual products. The generated shopping list will furthermore show the total price, and how much there can be saved by visiting several stores. The shopping list can be sent as an e-mail, for easy transportability.

The administrations application consists of a store manager- and an administrator section. An administrator is able to create, and remove, stores from Pristjek220. A store manager is managing one store, where he is capable of adding and removing products and changing their prices. Upon starting the administrations application, the user will be met by a login screen, which requires both username and password, in order to access Pristjek220.

The applications are communicating with a database in which the information is stored. The consumer application has read-only permission. Meanwhile the administrations application has both read and write permission.

The team developed the product through an iterative development process where Scrum was used. The project is connected to a Jenkins CI server, which is used to run the tests of the product. Finally, a user test was made to test the graphical user interface from the user's point of view.

## INDHOLDSFORTEGNELSE

1	RESUMÉ.....	1
2	ABSTRACT .....	2
3	INDLEDNING .....	5
3.1	LÆSEVEJLEDNING .....	5
3.2	TERMLISTE .....	6
3.3	ARBEJDSFORDELING .....	7
4	PROJEKTFORMULERING .....	8
4.1	AFGRÆNSNING .....	9
5	SYSTEMBESKRIVELSE.....	10
5.1	KRAV .....	10
5.1.1	UDFORMNING AF KRAV .....	10
5.1.2	AKTØRER.....	11
5.1.3	USER STORY BESKRIVELSER.....	11
5.1.4	KVALITETSKRAV .....	13
6	PROJEKTGENNEMFØRELSE .....	14
6.1	ITERATIV UDVIKLING.....	14
6.2	DOKUMENTATION AF KODEN .....	14
6.3	DOKUMENTATION AF SYSTEMARKITEKTUR .....	15
7	SYSTEMDESIGN.....	16
7.1	ARKITEKTUR.....	16
7.2	GUI DESIGN OVERVEJELSER .....	18
7.3	PRISTJEK220 DATABASE.....	19
7.3.1	DESIGN AF DATABASEN .....	19
7.3.2	DATABASEADGANG .....	20
8	PRODUKTBESKRIVELSE .....	21
8.1	DELTE FUNKTIONALITETER.....	21
8.2	FORBRUGER .....	21
8.3	ADMINISTRATION .....	22
8.3.1	ADMINISTRATOR.....	22
8.3.2	FORRETNINGSMANAGER.....	23
9	TESTNING AF PRISTJEK220 .....	23
9.1	UNITTESTNING.....	24
9.2	INTEGRATIONSTESTNING .....	24
9.3	CODE METRICS .....	25

9.4	CI .....	25
9.5	BRUGERTEST .....	26
10	RESULTATER OG DISKUSSION .....	26
11	FREMTIDIGT ARBEJDE .....	28
12	KONKLUSION .....	29
13	REFERENCER .....	30
14	UNDERSKRIFTER .....	32

### 3 INDLEDNING

I denne rapport beskrives et 4. semesterprojekt på Ingeniørhøjskolen Aarhus Universitet. 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 integrationstests i koden, hvor der er benyttet NUnit Frameworket. Til sidst i projektet er der udført brugertests, hvor der er testet brugervenligheden af produktet, på folk der er i produktets målgruppe, for at se om brugergrænsefladen er intuitiv 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.

#### 3.1 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 redegørelse for 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 "Projektgennemførelse" følger herefter, hvor den iterative udviklingsmetode, der er benyttet i projektet, forklares. Samtidig fremlægges 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 i dette afsnit designet af databasen, med de overvejelser der er gjort, og de mønstre der er benyttet, for at løse nogle af de problemstillinger der opstod.

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

Herefter følger en gennemgang 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 forklares der, hvordan der er blevet udført brugertests for at teste brugervenligheden af produktet.

Til sidst fremlægges 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 forklares bagefter, hvad der kan laves af videreudvikling på produktet, i afsnittet "Fremtidigt arbejde". Afslutningsvist bliver der fremlagt en konklusion på projektet, i afsnittet "Konklusion".

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

## 3.2 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

SysML = Systems Modeling Language

TDD = Test Driven Development

VPN = Virtual Private Network

UX = User Experience

### 3.3 ARBEJDSFORDELING

TABEL 1: TABEL OVER ARBEJDSFORDELINGEN

	Data Access Layer	Business Logic Layer	Presentation Layer
US1 – Tilføj produkt til forretning	NN	NN	CW, RH
US2 – Finde den billigste forretning for et produkt i Pristjek220	NN	NN	CW, RH
US3 – Indtast indkøbsliste	RH	RH	RH
US4 – Find ud af hvor produkterne fra indkøbslisten kan købes billigst	MG	MG	CW, RH
US5 – Finde hvilke forretninger der har et produkt	NN	NN	RH
US6 – Sammenligning af billigste indkøb og indkøb i én forretning	NN	NN	RH
US7 – Tilføj en forretning til Pristjek220	NN	NN	RH
US8 – Autofuldførelse	AM, MG	AM, MG	AM
US9 – Send indkøbsliste på mail	-	AM	AM
US10 – Ændre prisen på et produkt i en forretning	-	AM	CW, MG
US11 – Fjern et produkt fra en bestemt forretning	MG	MG	CW, MG
US12 – Fjern et produkt fra Pristjek220	<i>Ej implementeret</i>	<i>Ej implementeret</i>	<i>Ej implementeret</i>
US13 – Fjern en forretning fra Pristjek220	MG	MG	MG
US14 – Find åbningstider for en forretning	<i>Ej implementeret</i>	<i>Ej implementeret</i>	<i>Ej implementeret</i>
US15 – Indstillinger for indkøbsliste	-	RH	RH
US16 – Juster hvor produktet skal købes efter Pristjek220 er kommet med et forslag	-	AM	AM
US17 – Kunne bestemme afstanden der skal tilbagelægges for at købe produkterne fra forslaget	<i>Ej implementeret</i>	<i>Ej implementeret</i>	<i>Ej implementeret</i>
US18 – Kunne vise en kørselsvejledning mellem de forskellige forretninger, som der skal handles i	<i>Ej implementeret</i>	<i>Ej implementeret</i>	<i>Ej implementeret</i>
US19 – Bekræftelse af oprettelse/sletning af produkt	-	-	RH

På Tabel 1 ses, hvordan fordelingen af gruppemedlemmerne har været på de forskellige user stories. Nedenfor er der uddybet, hvor gruppemedlemmerne har haft deres fokusområder.

**Anders(AM):** Har primært stået for Business Logic Layer og Presentation Layer, samt omstrukturering til MVVM.

**Christian(CW):** Er udgået i løbet af projektforløbet.

**Mette(MG):** Har arbejdet lidt over det hele, men primært i Business Logic Layer og Presentation Layer.

**Nicklas(NN):** Har primært stået for Data Access Layer, samt omstrukturering til Repository Pattern.

**Rasmus(RH):** Har primært stået for Presentation Layer.

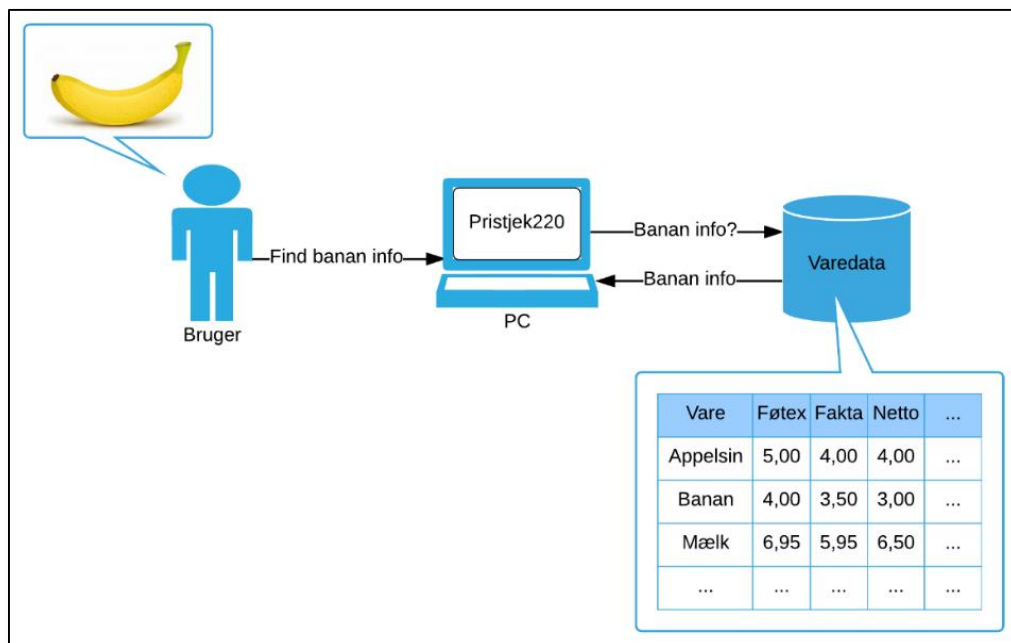


## 4 PROJEKTFORMULERING

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 mod hinanden, og dette resulterer i et stort udbud af både forskellige og ens produkter med forskellige priser. Der er derfor blevet fremstillet et produkt, kaldet Pristjek220, som tilstræber at give forbrugeren et let og simpelt overblik over, hvor de forskellige produkter kan handles billigst.

Pristjek220 har tre forskellige slags brugere; en forbruger, en forretningsmanager og en administrator. Forbrugeren er den person, der bruger Pristjek220 til at organisere sine daglige indkøb. Forretningsmanageren holder Pristjek220 opdateret med korrekte informationer om de produkter 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 applikationer; Pristjek220 Forbruger, til forbrugeren og Pristjek220 Forretning, som er en fælles applikation til både forretningsmanageren og administratoren. I Pristjek220 Forretning kan administratoren benytte sig af sine funktionaliteter ved at logge ind med et administratorlogin.

I Pristjek220 Forbruger er den grundlæggende funktionalitet, som forbrugeren har, at kunne finde den billigste forretning for et produkt. Han kan samtidig også se hvilke forretninger, der har produktet, og hvad det koster de forskellige steder, når han søger efter et produkt. Konceptet bag at slå et produkt 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 et produkt i Pristjek220, vil han blive foreslået produkter, efter to indtastede tegn, for at effektivisere brugen af Pristjek220.



FIGUR 1: RIGT BILLEDE OVER OPSLAG AF ET PRODUKT I PRISTJEK220.

For at gøre Pristjek220 mere effektivt, i forbindelse med forbrugerens indkøb, kan han indtaste en indkøbsliste, og Pristjek220 informerer så forbrugeren om, hvor produkterne 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 produkterne i én enkelt forretning, i modsætning til at købe produkterne, der hvor de er billigst. Forbrugeren har ligeledes mulighed for, at kunne justere, hvor produkterne skal købes, efter Pristjek220 er kommet med listen over, hvor de er billigst. Dette giver forbrugeren mulighed for, at vælge at flytte ét enkelt produkt på listen til en anden forretning, f.eks. i det tilfælde

at han skal handle i tre forretninger, men kun skal have ét produkt i den ene forretning. Yderligere kan det angives, hvis der er nogle forretninger, han ikke ønsker at handle i. Når forbrugeren har fået genereret sin indkøbsliste af Pristjek220, som han vil have den, har han mulighed for at få den sendt til sin egen E-mail. Dette gør, at han kan tilgå listen via sin mobiltelefon.

I Pristjek220 Forretning kan forretningsmanageren logge ind med sit forretningsmanagerlogin. Derefter har han mulighed for at tilføje og fjerne produkter, samt ændre prisen på et produkt, 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 produkterne fra hans forretning 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.

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.

#### 4.1 AFGRÆNSNING

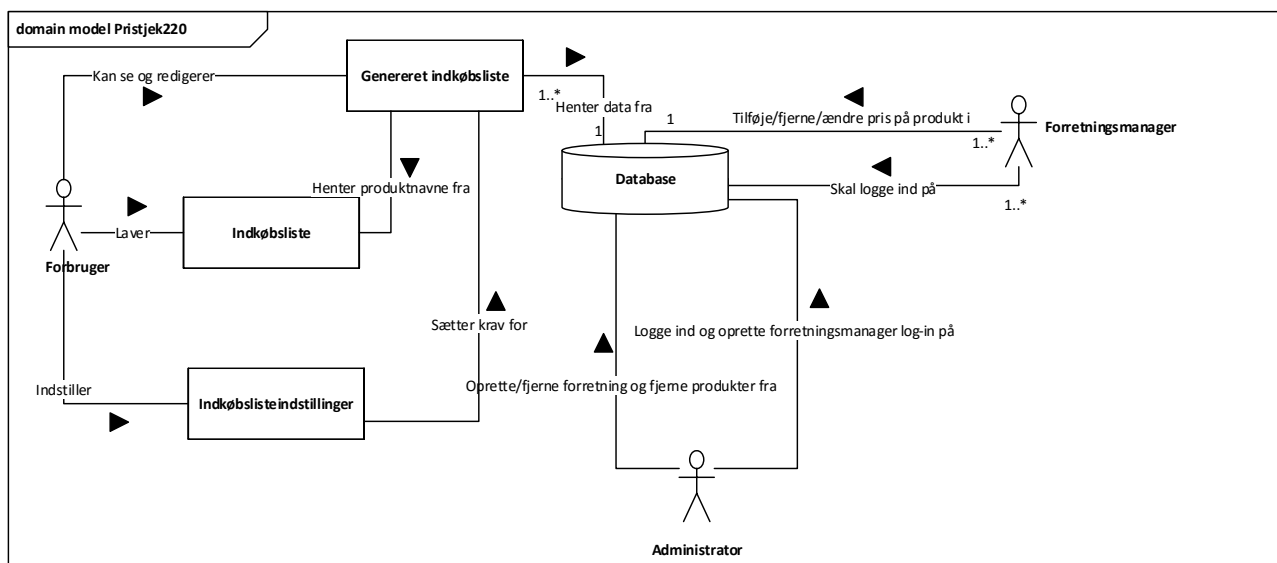
Pristjek220 består af to applikationer, hvor det ene er tiltænkt forretningerne, og det andet er til forretningernes forbrugere. Gennem disse applikationer kan forretningerne tilføje, fjerne og redigere informationer i deres sortiment i Pristjek220's database via internettet. Databasen kan dog kun tilgås via en VPN forbindelse. Forbrugerne kan sammenligne forretningernes sortiment med andre forretninger, og derved finde ud af hvor det er billigst at handle. Da forretningerne 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 der på smartphones kan modtages og læses E-mails.

Forbrugeren skal have adgang til en computer, for at kunne benytte Pristjek220. For at få størst muligt udbytte af produktet, skal han også have adgang til en smartphone eller en anden mobil enhed.

## 5 SYSTEMBESKRIVELSE

Systemet består af en forbrugerapplikation og en administrationsapplikation, som er to individuelle applikationer, således at det, ved udvidelser af den ene applikation, ikke er nødvendigt at opdatere den anden samtidig. I administrationsapplikationen er der mulighed for at logge ind som en forretningsmanager eller som administrator. Begge applikationer kobler op til en Microsoft SQL database. For at koble op er det nødvendigt at være på en VPN forbindelse. Databasen indeholder oplysninger om, hvor det er muligt at købe forskellige produkter, og hvad prisen er på produkterne, i de forskellige forretninger. Desuden indeholder den loginoplysningerne, der tjekkes op mod, når en administrator eller en forretningsmanager logger ind. På Figur 2 ses domæne modellen for systemet, hvor de forskellige interaktioner mellem aktørerne og entiteterne er vist.



FIGUR 2: DOMÆNE MODEL FOR PRISTJEK220

### 5.1 KRAV

Ud fra afsnittet "Projektformulering" er der opstillet en række user stories, der forklarer brugerens interaktion med systemet. User stories danner grundlaget for kravspecifikationen, og bliver brugt til at fastsætte systemets funktionalitet.

#### 5.1.1 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 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 skal dokumenteres en masse på skrift, der alligevel ikke bliver benyttet i implementeringen.

Der blev derfor valgt at benytte user stories [1] 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 blev valgt at benytte user stories, hvor en stor del af arbejdet med dem ligger i diskussionen med kunden, uden at projektet har en reel kunde, blev dette håndteret på en alternativ måde. Det var gruppen selv, der stod for produktets vision og agerede kunde, og diskussionen blev derfor håndteret ved, at gruppen sammen med vejlederen diskuterede i starten af hver iteration omkring ønskerne og detaljerne for de specifikke user stories, der skulle implementeres i iterationen.

### 5.1.2 AKTØRER

Systemet har tre forskellige slags aktører, som kort beskrives her:

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

**Forretningsmanager:** Forretningsmanageren er en primær aktør. Forretningsmanageren kan tilføje/fjerne produkter fra hans forretning, samt ændre priserne på produkterne.

**Administrator:** Administratoren er en primær aktør. Administratoren kan tilføje og fjerne forretninger fra Pristjek220.

### 5.1.3 USER STORY BESKRIVELSER

Der er her udvalgt nogle relevante user stories fra Kravspecifikationen [2]. Det er disse user stories, der bliver brugt til at gennemgå systemet i løbet af rapporten. De er rangeret efter hvilken prioritering, de har haft gennem udviklingsfasen.

#### **US2: Finde den billigste forretning for et produkt i Pristjek220:**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne finde den billigste forretning for et produkt 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 foretage billige indkøb.

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

**Som** en forbruger af Pristjek220

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

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

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

**US6: Sammenligning af billigste indkøb og indkøb i én forretning**

**Som** en forbruger af Pristjek220

**Vil** jeg kunne se en sammenligning af, hvad prisen er i én enkelt forretning, og hvor produkterne er billigst,

**Så** jeg kan vælge at køre i flere forretninger, fremfor bare at handle det hele i én forretning.

User storyen er relevant, fordi forbrugeren gerne vil kunne se, om det er besværet værd at tage i mere end én forretning for at handle.

**US8: Autofuldførelse:**

**Som** en forbruger af Pristjek220

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

**Så** der kan 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 sin 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 han så kan se på sin telefon.

#### 5.1.4 KVALITETSKRAV

Der er taget et uddrag af de mest relevante kvalitetskrav [2].

##### **1a: Pristjek220 skal leve op til de krav, som Microsoft stiller til UX design [3].**

Dette er valgt 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 [4], som de har udgivet, blevet sammenlignet med produktet under udviklingen.

##### **1b: Pristjek220 skal leve op til "Three-Click rule" [5].**

For at få en brugergrænseflade, som er let at navigere i, er der taget udgangspunkt i "Three-Click rule", som normalt bruges til hjemmesidenavigation. Denne regel angiver, at der skal kunne navigeres til forskellige funktionaliteter, ved brug af tre klik.

Der er i Pristjek220 i navigationen levet op til "Three-Click rule", 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" [5].**

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 angiver, at man skal prøve at begrænse antallet af elementer, man kan vælge fra en menu, til fem elementer.

Der er i Pristjek220 i de forskellige menuer levet op til "Rule of Five", ved at der ingen steder er mere end fem forskellige valgmuligheder.

##### **3a: Kodeord, der ligger i databasen, skal være krypterede**

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

Der er i Pristjek220 valgt at kryptere kodeord med SHA256 ud fra idéen om, at det ikke skulle være muligt at hacke databasen og derved få adgang til ikke-krypterede kodeord.

## 6 PROJEKTGENNEMFØRELSE

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

### 6.1 ITERATIV UDVIKLING

Det var et krav til projektet, at der skulle arbejdes iterativt. Derfor har gruppen måtte tage et valg, omkring hvilken arbejdsmetode der skulle benyttes. Valget lå mellem Scrum [6] og Kanban [7].

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. Derimod bliver arbejdet i Scrum opdelt i sprints, og hvert sprint har sit eget taskboard. Disse sprints bliver udfyldt med stories fra en product backlog, som nedbrydes til tasks, inden sprintet opstartes. Hvis alle tasks er udført, inden sprintet er omme, kan man tilføje flere fra product backloggen. Hvis man ikke når alt i sprintet, føres det resterende videre til næste sprint med højeste prioritet. Generelt set egner Scrum sig bedre til udvikling af systemer, og Kanban fungerer bedst til vedligeholdelse af systemer. Scrum sprintene er designet efter, at der skal være et færdigt produkt ved hvert sprint retrospektiv. Hvis kunden så ikke har flere penge at smide i projektet lige pludselig, kan han stadigvæk gå derfra med et produkt, der har nogle færdige funktionaliteter implementeret. Det giver samtidig mulighed for, at fremvise produktet, med de færdige funktionaliteter, for kunden ved en demonstration, i slutningen af hvert sprint. I forhold til vedligeholdelse af systemer, vil der højst sandsynligt dukke bugs op i produktet, og med Kanban 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 fremstillet. Det var givet fra starten, hvornår deadline for projektet faldt, og det blev samtidig besluttet, at køre sprints af 14 dages længde. Derfor var det kendt viden, hvor mange sprints der ville forekomme. Gruppen kunne derved 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 samtidig også gennemført et Scrum kursus ved Systematic A/S [8].

Gruppen har afvejet fra Scrum standarden og ikke tildelt noget gruppemedlem Scrum master rollen. Dette blev besluttet, da der hverken var nogen kunde, eller product owner, der skulle kommunikeres 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 indeholde mange små tasks, hvorved man ofte kunne rykke en task fra in progress til review / done. Men denne gang var det essentielt, at hver task skulle give værdi for historien. Fra starten af projektets forløb var det begrænset, hvor meget værdi de individuelle tasks gav historien, såsom tasks der beskrev, at der skulle skrives unittests. Men efter et par sprints begyndte det at give mening, at hver task skulle have værdi for historien, fordi det føltes mere naturligt, at arbejde med denne 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' beskrivelsesfelt, og derigennem kunne gruppen arbejde på deres foretrukne metode, imens deres tasks skabte værdi for historien.

### 6.2 DOKUMENTATION AF KODEN

En god dokumentation af source koden er værdifuld for projektet. Den gør, at det er let at danne sig et overblik over koden, så det derfor er let at sætte sig ind i produktet. Dermed er det også let at overskue, hvilke funktionaliteter der er tilgængelige. Manuel dokumentation af klasser og metoder i koden er tidskrævende. Det 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. Derved behøvede man kun at skrive lidt beskrivende tekst til, og på den måde skulle man ikke selv til at skrive, hvilke klasser, metoder og variabler der eksisterer i koden.

Der er derfor benyttet Sandcastle [9] 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. Udviklerens eneste ansvar er derfor, at skrive sigende kommentarer i koden, til den metode eller lignende, der er blevet tilføjet. Dokumenteringen af projektet kan findes i dokumentationen [10].

### 6.3 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" [11] 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 [12], hvor selve diagrammerne for projektet også kan findes.

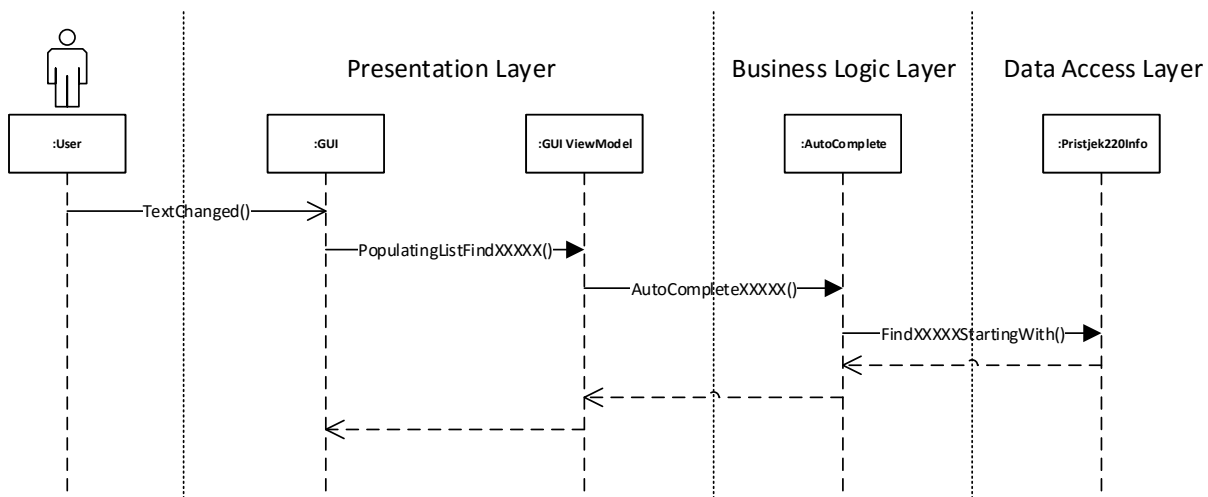


## 7 SYSTEMDESIGN

### 7.1 ARKITEKTUR

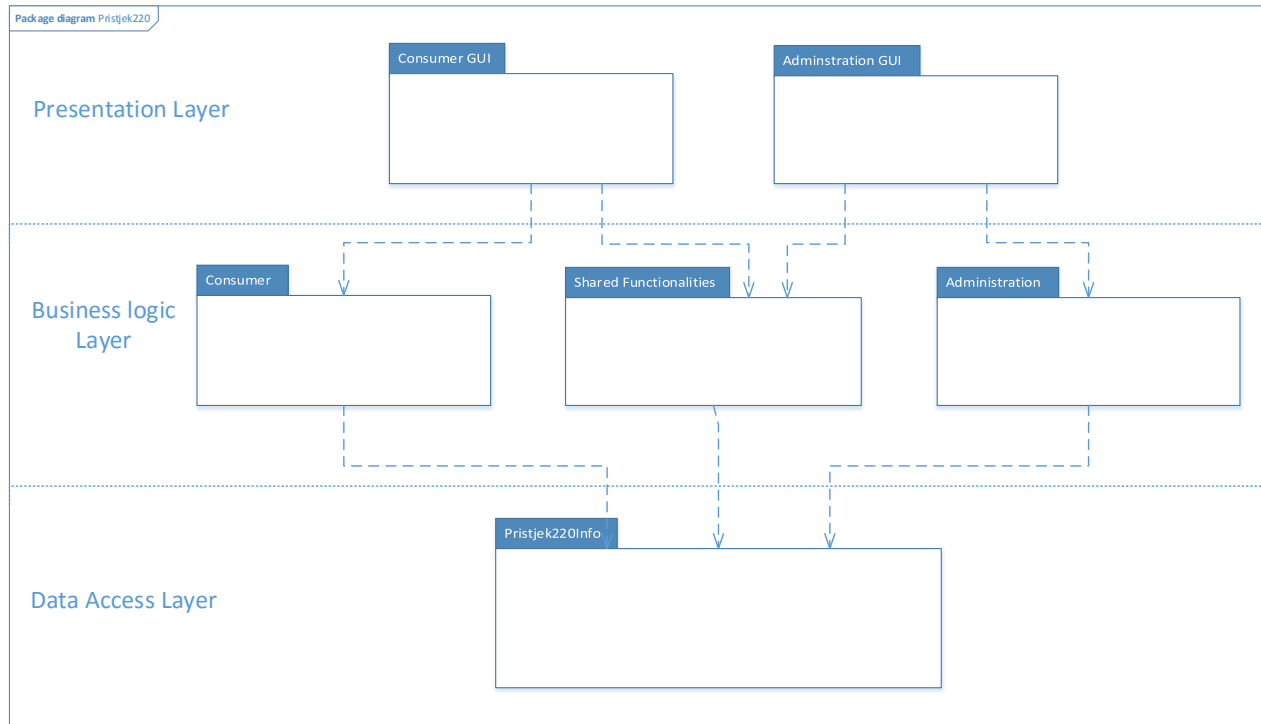
Da arkitekturen af projektet skulle besluttet, blev der valgt at bruge en lagdelte model. Grunden, til at valget faldt på den, er, at der ved den lagdelte model er en gruppering af klasser og pakker. Herved har de klasser, der befinder sig i ét lag, et sammenhængende ansvar for et vigtigt aspekt i systemet i den logiske separation, der er lavet i projektet. Ved at gruppere systemet på denne måde, er det nemt at finde ud af, hvilke pakker der bliver påvirket, hvis der laves en ændring eller tilføjelse til systemet. Med valget af den lagdelte model, skulle der også tages en beslutning om, hvor mange lag der skulle inddeles i. Ud fra størrelsen af systemet gav det mest mening at bruge en 3 lagdelte model [13]. 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 3: SEKVENSDIAGRAM FOR AUTOFULDFØRELSE, MED LAG OPDELING

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



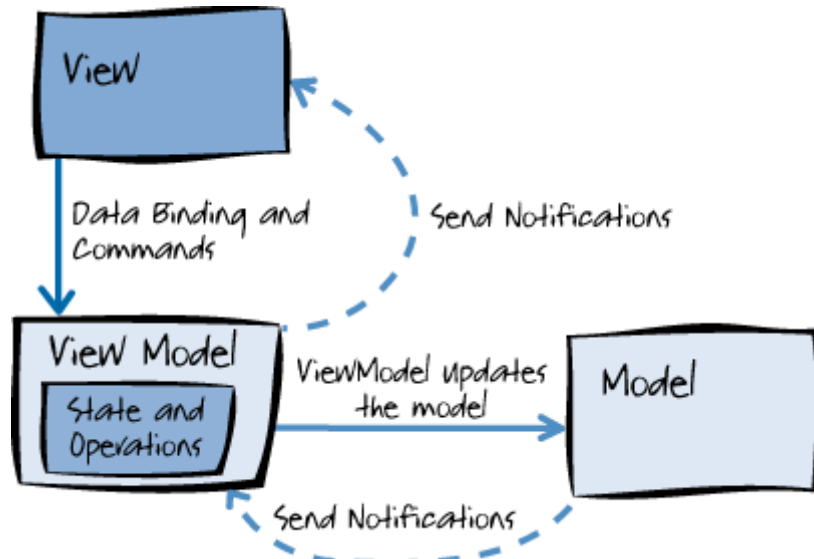
**FIGUR 4: PACKAGE DIAGRAM FOR PRISTJEK220**

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 at 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, og der kommer en høj samhørighed. En af de vigtigste ting ved at bruge 3 lags modellen er dog, at der er mulighed for at genbruge kode, sådan at man for eksempel kan genbruge de nederste lag af koden, 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.

## 7.2 GUI DESIGN OVERVEJELSER

Da GUI'en skulle designes, opstod der et problem op i form af, at det er svært at unitteste den. Dette gør at kodens funktionalitet, ikke kan verificeres, medmindre der anvendes et MVVM [14] pattern.

Ved brug af MVVM bliver bindingen mellem GUI'en og forretningslogikken løst, 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 5: MVVM MODEL [14]

For eksempel kan en forretningsmanager tilføje et nyt produkt til sin forretning i Pristjek220. Når produktets navn og pris er indtastet, og der bliver klikket på 'Tilføj produkt', tager view modellen navnet og prisen, og forsøger at sende dette videre til modellen. Hvis dette er succesfuldt, bliver produktet gemt i databasen. Der bliver så sat en bekræftelsestext, som et label i GUI'en er bindet til, og derved kan forretningsmanageren se, at hans produkttilføjelse 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 også muligt at unitteste GUI'en.

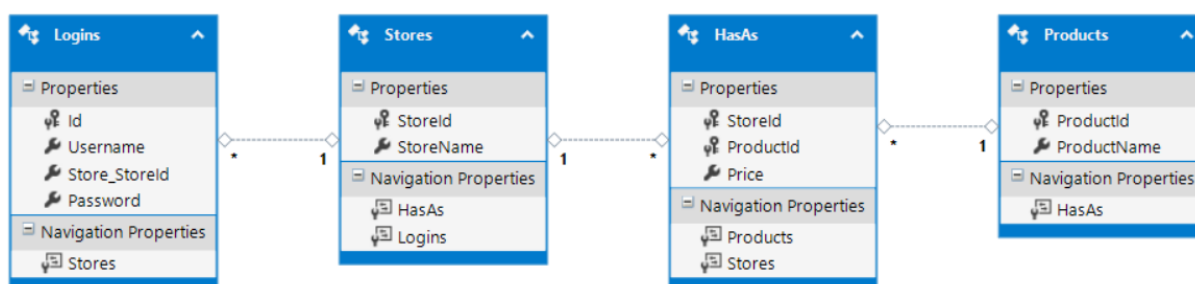
En anden fordel ved MVVM er, at hvis der i Pristjek220 ønskes, at den generede indkøbsliste skal vise prisen i både kroner og euro, så er view modellen oplagt til dette. En view model's primære opgave er at tage dataen fra modellen og formatere det, således at det tilpasser viewet's ønske. Så i stedet 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 genererede indkøbsliste.

## 7.3 PRISTJEK220 DATABASE

### 7.3.1 DESIGN AF DATABASEN

I Pristjek220's database er der fire forskellige entiteter kaldet 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, er det i Pristjek220 nødvendigt at have en property til produktets pris på relationen mellem forretningen og produktet. Denne property indeholder, hvad prisen for produktet er i lige præcis den forretning, det tilhører.

Løsningen på dette problem blev erfaret gennem undervisningen i DAB. Der blev her 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 6: UML DIAGRAM FOR DATABASEN.

TABEL 2: FØRSTE UDKAST AF OPSÆTTNINGEN 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

Oprindeligt var planen med databasen, at den skulle indeholde en enkel tabel, hvor man kunne se produktets navn i rækkerne, og de forskellige forretninger hen ad kolonnerne. På den måde ville man kunne finde prisen for produktet i den enkelte forretning. Denne opbygning er illustreret på Tabel 2. 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.

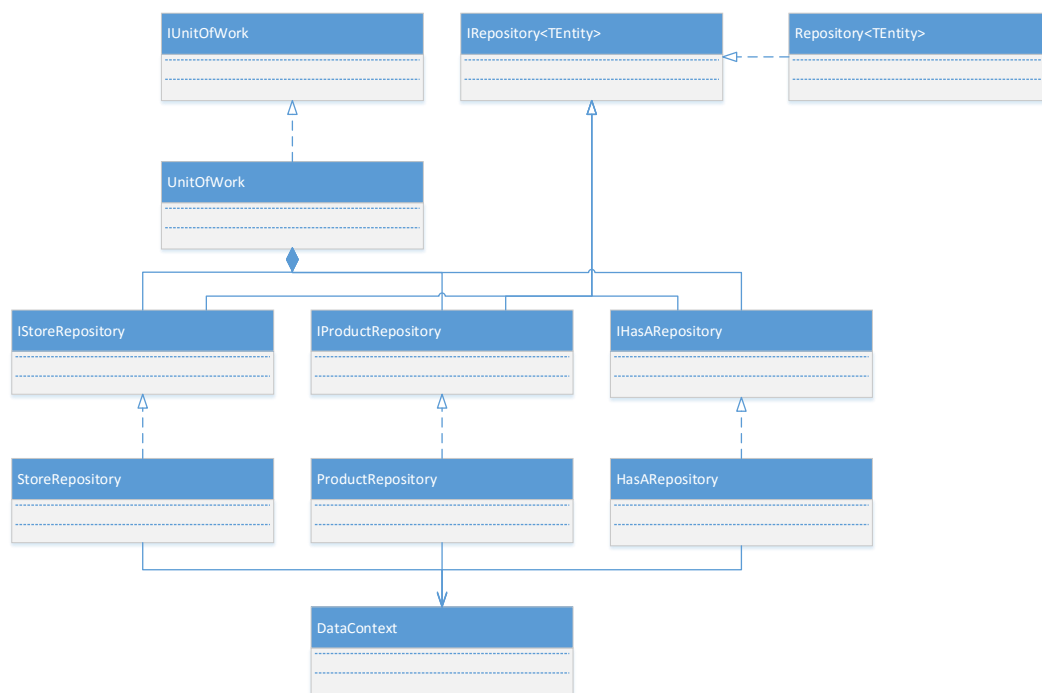
For at forretningerne ikke kan ændre prisen på et produkt i en anden forretning, eller ændre på andre forretningers sortiment, er der lavet et login til hver forretning, og alle logins gemmes i databasen. Denne entitet har et brugernavn, kodeord, og en reference til den forretning, den har kontrol over. Når der logges ind for en forretningsmanager, har han derved kun mulighed for at ændre, hvordan hans egen forretning skal fremstå. Derudover er det opbygget således at

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 applikationen højde for, hvilken retning administrationsapplikationen skal tage efter login.

### 7.3.2 DATABASEADGANG

Gennem Pristjek220's udvikling blev der erfaret, at det var besværligt at unitteste 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 unittests 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 den kode, som blev skrevet, blev duplikeret, da der var brug for den flere forskellige steder.

Der er derfor implementeret et Repository pattern [15] for at separere BLL fra DAL. Det gjorde det muligt at isolere BLL, så det kunne unittestes, da repositoryet kan substitueres ud. Repository pattern kommer dog stadig med den ulempe, at repositoryet har en hård binding ned til databasen, og ikke kan isoleres fra den. Derfor er det ikke blevet unittestet, men er i stedet blevet integrationstestet med databasen. Udover at lave en separation mellem BLL og DAL giver repositoryet også den fordel, at den laver et abstraktionslag til databasen. På den måde kan man samle al adgang ned til databasen, og undgår derved, at samme kode skal skrives flere steder.



FIGUR 7: IMPLEMENTERING AF REPOSITORY PATTERN I PRISTJEK220.

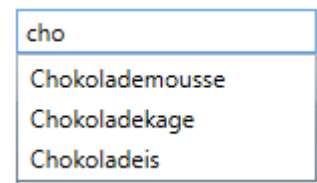
På Figur 7 kan det ses, hvordan Repository pattern'et er blevet implementeret i Pristjek220. De forskellige repositories indeholder CRUD funktionerne, for den tabel de hører til. ProductRepository indeholder derved funktionerne til Product tabellen i databasen. I Repository klassen, som de specifikke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

## 8 PRODUKTBEKRIVELSE

I dette afsnit vil der blive gennemgået funktionaliteterne i produktet. Der vil først blive redegjort for den fælles funktionalitet, der er mellem forbruger- og administrationsapplikationen. Derefter bliver forbrugerapplikationen, og efterfølgende administrationsapplikationen, uddybet. Under administrationsapplikationen bliver der først forklaret omkring administratoren, og derefter omkring forretningsmanageren. Der kan findes en brugermanual i dokumentationen [16], som beskriver, hvordan applikationerne skal bruges.

### 8.1 DELTE FUNKTIONALITETER

Der er nogle delte funktionaliteter mellem administrations- og forbrugerapplikationen. En af funktionaliteterne er autofuldførelse, som ligger i tekstboksene. Funktionen tjekker, det som brugeren allerede har skrevet, og sammenligner det med, hvad der ligger i databasen. Derved hjælper autofuldførelse brugeren til at se, om produktet eller forretningen findes i Pristjek220. Figur 8 viser, at der er blevet skrevet "cho", hvor autofuldførelse så foreslår de tre første produkter, der starter med "cho". Alle produkterne, som bliver foreslået, er produkter, der ligger i Pristjek220. Autofuldførelse giver kun tre produktforslag, for ikke at forvirre brugeren, med for mange muligheder. Det er blevet valgt efter brugertests, hvor brugerne synes, det var for uoverskueligt at have fem forslag, som var det første antal, gruppen havde sat. For administratoren bliver der også lavet autofuldførelse, de steder hvor der skal skrives en forretnings navn.



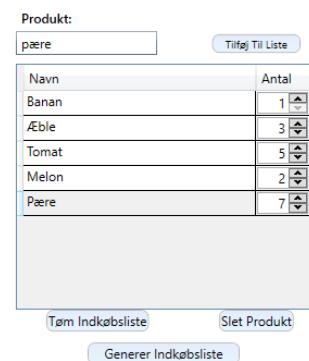
FIGUR 8: AUTOFULDFØRELSE

### 8.2 FORBRUGER

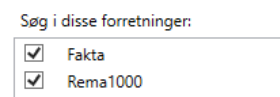
Hovedessensen for forbrugeren er, at kunne lave og generere en indkøbsliste, hvor der så bliver udregnet, hvor det er billigst at købe de forskellige produkter. På Figur 9 og Figur 10 kan der ses et udsnit af brugerinterfacet for indkøbslisten, som det ser ud i Pristjek220.

På Figur 9 kan det ses, at produkter kan indskrives, og tilføjes til indkøbslisten. Derefter kan der vælges det ønskede antal af produkterne. Herudover kan der slettes et produkt fra listen, og hele indkøbslisten kan også tømmes. På Figur 10 ligger indstillingsmuligheder, hvor der kan vælges hvilke forretninger, der skal handles i. Derudover er der mulighed for at generere 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 produkterne skal købes. På Figur 12 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 12 kan man se, hvor det er billigst at købe hvert enkelt produkt, og på Figur 11 vises, hvad det koster at købe det hele i én forretning. Her kan man se, at det koster 6,95 kr. mere at handle det hele i Rema1000, end det gør at tage i forskellige forretninger for 2 af produkterne. På Figur 12 kan der klikkes på de enkelte produkters forretninger, hvor der



FIGUR 9: PRISTJEK220  
INDKØBSLISTE



FIGUR 10: PRISTJEK220  
INDKØBSLISTE  
INDSTILLINGSMULIGHEDER

kan skiftes til en anden forretning at handle det produkt i. Derved udregner Pristjek220 en ny genereret indkøbsliste. Der er i den genererede indkøbsliste mulighed for, at sende sin indkøbsliste som E-mail, som kan ses på Figur 11.

Forretning	Produkt	Stk. pris	Antal	Pris
Føtex	Æble	2,00 kr	1	2,00 kr
Lidl	Banan	1,00 kr	1	1,00 kr
Rema1000	Melon	8,00 kr	1	8,00 kr
Rema1000	Tomat	2,00 kr	1	2,00 kr

FIGUR 12: GENERERET INDKØBSLISTE

Sum: 13 kr

Spar: 6,95 kr

I forhold til køb af alle produkter i Rema1000 hvor det koster 19,95 kr.

Indtast din E-mail her:

Send mail

FIGUR 11: BESPAREELSE I GENERERET INDKØBSLISTE OG SENDE E-MAIL

I forbrugerapplikationen er der også mulighed for at søge efter et produkt, uden at lave en indkøbsliste. Pristjek220 viser så, hvilke forretninger der sælger det, og hvad prisen er i hver forretning. Som det kan ses på Figur 13, er der blevet søgt efter produktet "banan", hvor Pristjek220 viser, hvor "banan" sælges, og hvad den koster disse steder.

Produkt:

banan

Forretning	Pris
Fakta	4,95 kr
Rema1000	6,00 kr

FIGUR 13: PRISTJEK220 SØG EFTER PRODUKT

## 8.3 ADMINISTRATION

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

Login

**Pristjek220 - Login**

Velkommen til Pristjek220 Forretning!

Her kan du logge ind med dit forretningsnavn og det tilhørende kodeord.

Brugernavn:

Kodeord:

FIGUR 14: PRISTJEK220 LOGIN

### 8.3.1 ADMINISTRATOR

Administratoren kan tilføje og fjerne forretninger fra Pristjek220. For at tilføje en forretning skal man angive et forretningsnavn, samt et kodeord, hvorefter man kan trykke "Tilføj forretning", hvorved forretningen tilføjes til Pristjek220. På Figur 15 ses, at Fakta bliver tilføjet til Pristjek220, når der trykkes "Tilføj forretning". Man kan derved efterfølgende logge ind med forretningsnavnet "Fakta" og det tilhørende kodeord.

Forretningsnavn:

Kodeord:

Bekræft kodeord:

FIGUR 15: PRISTJEK220 TILFØJ FORRETNING

Som administrator kan man også fjerne en forretning, som kan ses på Figur 16. Man indtaster en forretning, der skal fjernes, og klikker "Fjern forretning". På Figur 16 er det forretningen "Fakta", der er ved at blive fjernet fra Pristjek220.

Forretning:

FIGUR 16: PRISTJEK220 FJERN FORRETNING

### 8.3.2 FORRETNINGSMANAGER

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

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

Produkt:  Pris:  kr.

FIGUR 17: PRISTJEK220 TILFØJ PRODUKT

Produkt:  Pris:  kr.

Banan

FIGUR 18: PRISTJEK220 ÆNDRE PRIS PÅ PRODUKT

Forretningsmanageren har også muligheden for at fjerne produkter fra sin forretning, hvis f.eks. et produkt udgår af produktion. Hvis man prøver at fjerne et produkt, der ikke findes i forretningen, bliver der informeret om dette. Figur 19 viser, at for at fjerne et produkt fra en forretning, skal produktet indtastes, og der skal efterfølgende klikkes på "Fjern produkt". Herefter vil et pop-up vindue komme frem, hvor det skal bekræftes, at produktet skal fjernes.

Produkt:

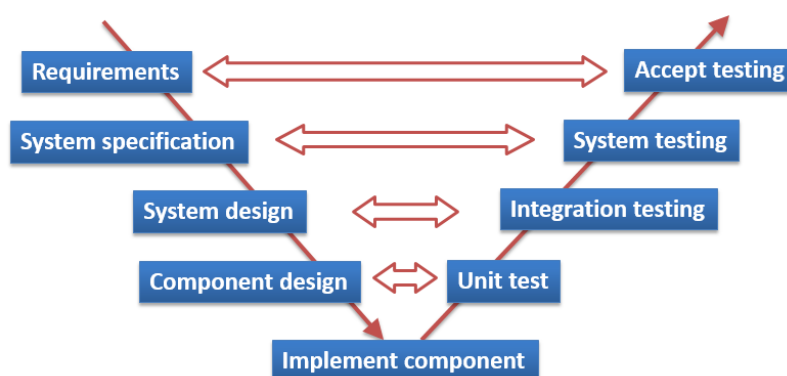
Banan

FIGUR 19: PRISTJEK220 FJERN PRODUKT

## 9 TESTNING 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 tests. Dette er illustreret på Figur 20, hvor de forskellige teststadier kan ses, som er unittest, integrationstest, systemtest og accepttest. Gennem udviklingen af Pristjek220 er der brugt FDD, hvilket giver den fordel, at man kan udvikle en feature til applikationen, 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 bliver der sparet en masse tid på de funktioner, som Product Owner beslutter sig for, at han ikke vil have. TDD er modsat, idet der skrives tests, før man laver features, hvilket giver den fordel, at koden lever op til testkravene med det samme.

I dette projekt er brugertesten brugt som systemtest, og accepttesten ligger i dokumentationen [17].



FIGUR 20: V-MODEL FOR UDVIKLINGEN AF ET PRODUKT. [18]

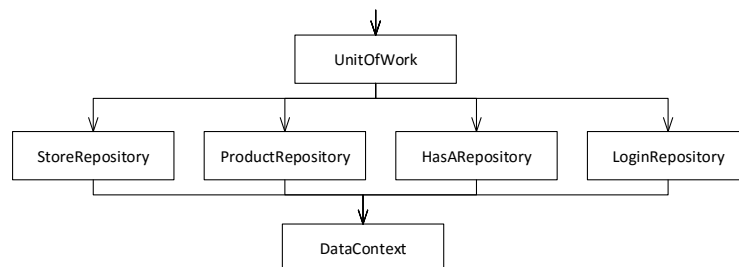


## 9.1 UNITTESTNING

Unittestning er det første teststadiet, som applikationen kommer igennem, da det tester de individuelle klasser isoleret fra resten af systemet. Dette betyder, at når en klasse får tilføjet en funktion, kan den testes uafhængigt af, hvor langt udviklingen af resten af systemet er. Gennem unittestning sikres der, at hver klasse individuelt opfører sig på den forventede måde, ved de givne inputs. Derved bliver applikationen kvalitetssikret, så den lever op til kundens forventninger. Til unittestning af Pristjek220 er der blevet benyttet NUnit frameworket samt NSubstitute, til at kunne opstille de forskellige testscenarier, og lave unittestene automatiske. NSubstitute er ligeledes et framework, som bruges til at substituere de forskellige klasser ud, som klassen, der testes, har relationer til. Derved isoleres klassen fra resten af systemet. Det at automatisere testene giver den fordel, at udvikleren ikke skal bruge tid på at teste manuelt flere gange. Derudover giver automatiske tests bedre mulighed for, at koden kan optimeres og refaktoreres undervejs. Denne optimering kan så blive kørt igennem de automatiserede tests, uden at der skal bruges for lang tid på at teste manuelt, for at sikre at funktionaliteten stadig fungerer, som den skal. Automatiske unittests er dog ikke den perfekte metode, der løser alle problemer, da de automatiserede tests kun tester, det de er lavet til at teste. Det vil sige, at nogle ting kan blive overset, hvis ikke man specifikt sørger for, at skrive nogle tests, der tester det. Derudover er der nogle ting, som unittests ikke kan teste, såsom brugervenlighed og hvordan den enkelte klasse opfører sig, når den sættes sammen med resten af systemet.

## 9.2 INTEGRATIONSTESTNING

Da unittests tester hver klasse isoleret fra resten af systemet, tester man kun, om 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 integrationstestning, skal man finde ud af, hvordan klasserne er afhængige af hinanden, og derved hvor man har brug for at lave integrationstest henne. På Figur 21 kan man se et udsnit af Pristjek220's dependency tree, hvor det kan ses, at produktet overholder DIP, da højniveau kalder lavniveau og ikke omvendt. For at se hele Pristjek220's dependency tree henvises der til dokumentationen [12].



FIGUR 21: UDSNIT AF DEPENDENCY TREE FOR PRISTJEK220.

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

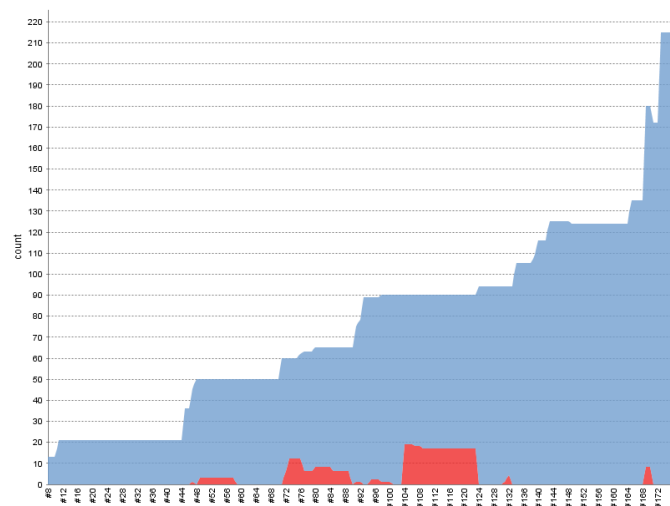
### 9.3 CODE METRICS

For at gøre koden mere vedligeholdelsesvenlig, og sørge for at de forskellige metoder ikke bliver for komplekse, er der gennem projektet gjort brug af code metrics [19], hvor der er benyttet vedligeholdelsesindekset, samt den cyklomatiske kompleksitet, af de forskellige metoder. Code metrics er et værktøj, der kan give en idé til, om koden burde omskrives eller refaktoreres. Det gør det også muligt at identificere de funktioner, der har en høj potentiel risiko. Derudfra kan det vurderes, om funktionen skal ændres, så risikoen for den bliver mindre. Dette vurderes ud fra, om ændringen giver en stor nok fordel, i forhold til den tid, der skal bruges på det. Der kan også besluttes, at funktionen bare skal testes godt igennem på grund af, at den har en øget risiko. Der er gennem projektet forsøgt at leve op til de ønskede værdier for vedligeholdelsesindekset, som er 80+. Ved tilføjelse af ny kode, er der prøvet på at gøre den så vedligeholdelsesvenlig som muligt, ved at holde linjeantallet og den cyklomatiske kompleksitet nede. Der er forsøgt at holde den cyklomatiske kompleksitet nede, ved at undgå unødvendige loops. Det har medført, at projektets metoder er blevet mindre komplekse.

### 9.4 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 under udvikling, bliver testene kørt igennem flere gange om dagen, og derfor vil udvikleren 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 [20] til rådighed, som er blevet benyttet gennem udviklingen af Pristjek220. Jenkins er et rigtig godt værktøj, da der spares tid på tests, som forklaret tidligere. 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.



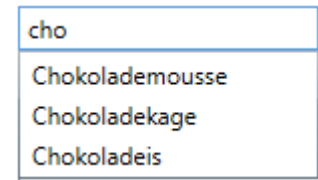
FIGUR 22: PRISTJEK220'S CI SERVER TEST/BUILD GRAF FRA JENKINS. [21]

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 at tjekke hvert 15. minut, om der er lavet en ændring i projektet. Hvis der er en ændring, bygger den projektet, og kører 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 der ikke var overensstemmelse i tegnsætningen i decimaltal, for produkternes priser. Som det kan ses ud fra Figur 22, gik der noget tid, før dette blev opdaget. Dette skyldes, at testene var hurtigere at køre

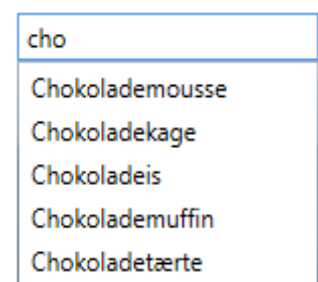
på computeren, og ikke fejlede der. Da dette så blev opdaget, ændrede gruppen den specifikke kultur for decimaltal, så det stemte overens, både på computerne og CI serveren.

## 9.5 BRUGERTEST

Brugergrænsefladen kan ikke testes ved automatiske tests. Derfor måtte der benyttes andre metoder for at teste brugergrænsefladen, og det er her, en brugertest [22] kan benyttes. En brugertest er en manuel test, som bliver lavet for at teste UX, hvor der fås feedback fra personer, der skal bruge programmet. Essensen af brugertestning er, at få brugerne til at afprøve produktet, og give værdifuld indsigt i, hvad der udfordrer brugerne, og besværliggør deres brug af produktet. Brugertests viser, om brugergrænsefladen er intuitiv for brugeren at benytte. Hvis brugeren ikke kan følge programmets flow, skal det noteres, så der kan laves ændringer i brugergrænsefladen, for at forbedre oplevelsen for brugeren. Brugertesten er blevet udført ved at udvælge en gruppe personer, og bede dem om at udføre nogle opgaver, som er blevet opstillet af gruppen. Et eksempel, på en opgave de skal igennem, lyder som følgende: "Du står i en forretning og kigger på chokolademousse, og syntes det er lidt dyrt. Kig derfor om du kan få det billigere i en anden forretning, og se om du kan spare nogle penge ved at gå til en anden forretning." Resten af testen, med beskrivelsen af opgaverne og resultatet herfra, kan ses i dokumentationen [23]. Ud fra den feedback, som der er kommet ud af brugertesten, er der blevet observeret, at folk synes, at autofuldførelsen kommer med for mange forslag, så det bliver uoverskueligt. Derfor har gruppen valgt at ændre antallet af produkter, som autofuldførelsen foreslår, fra fem forslag til tre forslag.



**FIGUR 23: AUTOFULDFØRELSE MED 3 ANBEFALINGER EFTER BRUGERTESTEN**



**FIGUR 24: AUTOFULDFØRELSE MED 5 ANBEFALINGER FØR BRUGERTESTEN**

På Figur 24 ses det, hvordan autofuldførelsen så ud inden brugertesten, hvor der var fem forslag til produkter, der findes i Pristjek220. Figur 23 viser, hvordan autofuldførelse er kommet til at se ud, efter brugertesten. Det er en vigtig detalje, der gør produktet mere overskueligt for en bruger, som gruppen ikke har tænkt over under udviklingen, men som brugertesten fremhæver, som en ting der gik ud over brugeroplevelsen.

## 10 RESULTATER OG DISKUSSION

Projektet er endt ud med, at der er udviklet et produkt bestående af to applikationer; én til forbrugeren og én samlet til administratoren og forretningsmanageren. Der er i projektets forløb prioriteret efter hvilke funktionaliteter, der var mest værdifulde for produktet at få implementeret, for at det var brugbart.

Forbrugeren har derfor fået implementeret muligheden for at søge efter et produkt, for at se hvilke forretninger der har det, og hvad det koster i hver enkelt forretning. Yderligere har forbrugeren muligheden for at indtaste sin indkøbsliste, og få en genereret indkøbsliste, der indeholder informationer om, hvor han skal handle henne. Han kan også fravælge nogle forretninger, hvis han ikke ønsker at handle i dem. Der kan også sammenlignes med, hvad det koster at købe alle produkter i én forretning, frem for den billigste mulighed, hvor der skal handles i flere forskellige forretninger. Når indkøbslisten er genereret, har forbrugeren ligeledes mulighed for at få sendt indkøbslisten til sin mail, så han kan tage den med ud at handle.

Administratoren kan tilføje og fjerne en forretning til Pristjek220. Forretningsmanageren kan så tilføje produkter til hans forretnings sortiment, og kan også ændre prisen på produkter, samt slette et produkt fra hans forretning.

Begge applikationer har autofuldførelse, for at øge brugervenligheden af produktet. Det er også alle disse funktionaliteter, der er beskrevet i afsnittet "Produktbeskrivelse", som beskriver det endelige produkt.

Der er blevet benyttet user stories til at opstille de funktionelle krav for projektet, hvilket har fremmet den iterative udviklingsproces. Hver user story kan meget nemt oversættes til en feature i en product backlog, så de kan benyttes som stories i et sprint. Det har også været meget effektivt, at have den diskussion inden implementeringen, som hører med til user stories. Det har gjort, at alle i gruppen var klar over, hvad de forskellige stories indebar, inden de skulle implementeres, hvorefter det var nemt at nedbryde historien til mindre tasks.

Det blev i starten af projektet aftalt, at man skulle sikre sig, at alle tests stadig kørte succesfuldt igennem, når man havde lavet ændringer eller tilføjelser i koden. Det skete dog i løbet af udviklingen af projektet, at medlemmer i gruppen glemte at gøre dette. Samtidig forekom det ikke gruppen naturligt at holde øje med CI serveren, og derfor tog det nogle gange lidt tid, før det blev opdaget, at flere af testene fejlede. Det har dermed også medført, at der har været perioder på CI serveren, hvor testene fejlede. Hvis gruppen havde integreret brugen af CI serveren mere i udviklingsprocessen, havde dette været undgået.

Gennem de udførte brugertests, blev der erfaret, at brugerne havde let ved at navigere, og udføre de stillede opgaver, i både administratordelen og forretningsmanagerdelen i administrationsapplikationen. Menuerne og hensigterne med funktionerne var lette at forstå, og intuitive at interagere med.

I brugertestningen af forbrugerapplikationen synes forsøgspersonerne, at der var for mange forslag til autofuldførelse. Derfor blev der efterfølgende taget en beslutning om, at mindske antallet af forslag som autofuldførelsen kom med, fra de oprindelige fem forslag til tre forslag i stedet. Det blev besluttet at lave denne ændring, da det ikke var en stor ændring, der krævede meget arbejde, så den var let at udføre.

Forsøgspersonerne havde også svært ved at finde ud af, hvad de skulle gøre, når de blev bedt om at lave en indkøbsliste, efter de havde søgt efter et produkt. De forstod ikke ud fra brugergrænsefladen, at de skulle klikke over i menuen, der hedder indkøbsliste, for at gøre det. De troede derimod, at de kunne tilføje til indkøbslisten fra samme sted, hvor de havde søgt efter et produkt før. Dette blev der besluttet ikke at gøre noget ved i produktet, da det ikke nødvendigvis bare var en lille ændring, men muligvis krævede lidt mere tid at udføre.

Brugertesten blev udført, da udviklingen af produktet var afsluttet, og arbejdet med at færdiggøre rapporten og dokumentationen for projektet var af højeste prioritet. De ændringer, der kunne laves i forbindelse med feedbacken fra brugertestene, blev der derfor valgt at ikke gøre noget ved, medmindre det var små ting, som nemt kunne rettes. For at resultaterne fra brugertesten kunne bruges til at gøre brugergrænsefladen mere brugervenlig, var det nødvendigt med mere tid til at udvikle i. Desværre var der kun en uge tilbage af projektløbet, da Pristjek220 var klar til brugertest.

## 11 FREMTIDIGT ARBEJDE

Til videreudvikling af produktet, vil det være vigtigt at få implementeret de ændringer, som brugertesten lagde op til, at der skulle laves. Det blev erfaret, at det ikke var intuitivt at tilføje produkter til sin indkøbsliste, da brugerne troede, at dette skulle ske under menuen "Søg efter produkt". Det kunne derfor ændres, så der f.eks. kunne trykkes "Tilføj til indkøbslisten", efter man havde søgt efter et produkt, hvorefter der blev skiftet vindue til indkøbslisten, hvor produktet samtidig var blevet tilføjet til listen.

Yderligere ville det være oplagt at gøre forbrugerapplikationen til en smartphone app, således at brugeren får mulighed for at lave ændringer til sin indkøbsliste, uden at skulle være hjemme ved sin PC. Dette ville desuden give mulighed for, at integrere systemet med telefonens GPS, og derved planlægge den korteste rute til alle forretningerne, samt give forbrugerne en kørselsanvisning. Forbrugeren kunne også have mulighed for, at begrænse afstanden han ønsker at køre hjemmefra, for at nå alle forretningerne, samt bestemme det maksimale antal forretninger, han ønsker at handle ind i. Derudover ville det være en hjælp, hvis en bruger, gennem Pristjek220, kunne se forretningernes aktuelle tilbudsaviser og åbningstider. Det ville samtidig også være smart, hvis Pristjek220 kunne tage højde for mængdetilbud. Dette vil sige, at hvis en forbruger skal have 10 bananer, og der er en forretning, der har et tilbud på 10 stk. for 30 kr., skal Pristjek220 kunne håndtere dette.

For 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 mulighed for at planlægge en madplan for ugen igennem applikationen.

Til selve forretningsmanagerapplikationen af produktet, ville det være oplagt at synkronisere databasen med forretningens egen database. Produktsortimentet ville således blive opdateret automatisk, og derved gøre det let at styre tilbudspriserne. Derudover vil en metode til at se hele produktsortimentet i databasen være anvendelig. Med en mulighed for at forretningsmanagerne kan sætte specifikke produkter til udsolgt, ville det også kunne forbedre produktet for forbrugerne.

På tværs af systemet ville det være oplagt at kunne angive produkter som økologiske eller glutenfri. Således kunne forbrugerens applikation have mulighed for kun at søge efter produkter, hvor disse kriterier er opfyldt. Derved kunne Pristjek220 virkelig lette arbejdet med dagligvareindkøb, for allergikere og folk med disse krav.

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

## 12 KONKLUSION

Formålet med projektet er at udvikle et program, der kan hjælpe folk med at handle så billigt som muligt. Visionen med produktet er, at hjælpe den fattige studerende med at lave en indkøbsliste, og kunne finde de billigste steder at handle produkterne fra listen. Den iterative arbejdsproces har været en rigtig god måde at arbejde på, da man har kunne ændre på projektet, i takt med at den samlede forståelse af produktet øgedes. Arbejdsprocessen virker rigtig godt sammen med Scrum, som er den udviklingsproces, som er benyttet gennem projektet, fordi man gennem hvert sprint får færdiggjort en eller flere user stories.

Gennem projektforsløbet er det lykkedes at fremstille et produkt, der kan håndtere problemet, med at finde det billigste sted at handle. Produktet består af to computerapplikationer; en administrationsapplikation og en forbrugerapplikation. Forbrugeren kan søge efter produkter, og se hvilke forretninger der har produkterne. Derudover kan forbrugeren lave en indkøbsliste, og få Pristjek220 til at regne ud, hvor det er billigst at handle produkterne på listen. Der er blevet implementeret, så man kan sende sin indkøbsliste til sin mail. På den måde kan man få den printet ud, eller åbnet på sin smartphone.

Administrationsapplikationen består af en administrator og en forretningsmanager. Forretningsmanageren kan tilføje og fjerne produkter, samt ændre deres pris, i hans forretning. Administratoren kan tilføje og fjerne forretninger. På den administrative del er der lagt et login ind, så man skal have et brugernavn og kodeord, for at kunne lave de handlinger. Forretningerne og deres sortiment ligger i Pristjek220's database, som forbrugeren søger i, ved at benytte den grafiske brugergrænseflade.

Applikationerne er implementeret til at kunne interagere med databasen. I databasen ligger de informationer, som der er tilføjet gennem administrationsapplikationen. Forbrugerapplikationen kan ikke tilføje informationer til databasen, men kan derimod kun læse fra den.

Der er blevet lavet en brugertest på applikationerne, for at teste for UX. Testen belyste nogle problemer, som brugerne havde, i deres anvendelse af applikationen. Nogle af problemerne kunne nemt rettes, mens andre er problemer, der skal rettes, hvis der bliver arbejdet videre på Pristjek220. Et lille problem, som brugerne kommenterede på, var, at autofuldførelse foreslog for mange produkter. Det problem blev rettet med det samme, da det var et gentagende problem i testen, og en lille ting at rette i produktet.

Det er lykkedes at lave et produkt, som lever op til den vision, der blev sat i starten af projektet. Produktet kan godt komme på markedet, i den tilstand som det er i nu. Men Pristjek220 ville være en langt bedre oplevelse for forbrugeren og forretningsmanageren, hvis produktet får flere funktionaliteter, som der er beskrevet i fremtidigt arbejde. Med implementeringen af disse funktionaliteter, kunne Pristjek220 have potentialet til, at gøre de billige indkøb let tilgængelige for alle.

## 13 REFERENCER

- [1] W. Nazzaro og C. Suscheck, »New to User Stories?,« 2010. [Online]. Available: <https://www.scrumalliance.org/community/articles/2010/april/new-to-user-stories>. [Senest hentet eller vist den 19 5 2016].
- [2] Gruppe7, »Kravspecifikation,« AU, Aarhus, 2016.
- [3] Microsoft, »Design applications for the Windows desktop,« 2016. [Online]. Available: <https://dev.windows.com/en-us/desktop/design>. [Senest hentet eller vist den 19 5 2016].
- [4] Microsoft, »UX checklist for desktop applications,« 2016. [Online]. Available: <https://msdn.microsoft.com/library/windows/desktop/dn742479.aspx>. [Senest hentet eller vist den 19 5 2016].
- [5] J. Zeldman, »Taking Your Talent to the Web,« May 2001. [Online]. Available: <http://takingyourtalenttotheweb.com/Taking%20Your%20Talent%20to%20the%20Web.pdf>. [Senest hentet eller vist den 19 5 2016].
- [6] K. Schwaber og J. Sutherland, »The Scrum Guide,« 7 2013. [Online]. Available: <http://www.scrumguides.org/scrum-guide.html>. [Senest hentet eller vist den 19 5 2016].
- [7] D. Peterson, »What is Kanban?,« 2015. [Online]. Available: <http://kanbanblog.com/explained/>. [Senest hentet eller vist den 19 5 2016].
- [8] Systematic, »Systematic,« 2016. [Online]. Available: <https://da.systematic.com/>. [Senest hentet eller vist den 19 5 2016].
- [9] EWSsoftware, »Sandcastle Help File Builder,« 2016. [Online]. Available: <https://github.com/EWSsoftware/SHFB>. [Senest hentet eller vist den 19 5 2016].
- [10] Gruppe7, *Pristjek220 Documentation*, Aarhus: AU, 2016.
- [11] P. Kruchten, »Architectural Blueprints - The "4+1" View,« November 1995. [Online]. Available: <http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf>. [Senest hentet eller vist den 19 5 2016].
- [12] Gruppe7, »Systemarkitektur,« AU, Aarhus, 2016.
- [13] Microsoft, »Three-Layered Services Application,« 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/ff648105.aspx>. [Senest hentet eller vist den 19 5 2016].
- [14] Microsoft, »The MVVM Pattern,« 2012. [Online]. Available: <https://msdn.microsoft.com/en-us/library/64959690.aspx?f=255&MSPPError=-2147217396>. [Senest hentet eller vist den 19 5 2016].
- [15] Microsoft, »The Repository Pattern,« 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/ff649690.aspx?f=255&MSPPError=-2147217396>. [Senest hentet eller vist den 19 5 2016].

- [16] Gruppe7, »Brugermanual,« AU, Aarhus, 2016.
- [17] Gruppe7, »Accepttest,« AU, Aarhus, 2016.
- [18] T. F. Jensen og F. B. Jakobsen, *Integration Test Patterns*, Aarhus: AU, 2016.
- [19] Microsoft, »Code Metrics Values,« 2016. [Online]. Available: <https://msdn.microsoft.com/en-us/library/bb385914.aspx>. [Senest hentet eller vist den 19 5 2016].
- [20] Jenkins, »Jenkins,« 2016. [Online]. Available: <https://jenkins.io/>. [Senest hentet eller vist den 19 5 2016].
- [21] Gruppe7, »Projekt Grp. 7 Projekt - Pristjek220,« 2016. [Online]. Available: <http://ci1.ase.au.dk:8080/job/Grp.%207%20Projekt%20-%20Pristjek220/>. [Senest hentet eller vist den 19 5 2016].
- [22] NIELSEN NORMAN GROUP, »Turn User Goals into Task Scenarios for Usability Testing,« 2014. [Online]. Available: <https://www.nngroup.com/articles/task-scenarios-usability-testing/>. [Senest hentet eller vist den 19 5 2016].
- [23] Gruppe7, »Brugertest,« AU, Aarhus, 2016.



## 14 UNDERSKRIFTER

ANDERS MEIDAHL MÜNSBERG



201404246

CHRISTIAN SLOT WINKEL

201370493

METTE GRØNBECH



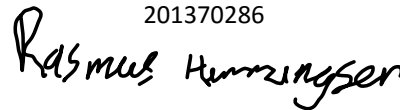
201305561

NICKLAS NIELSEN



201370286

RASMUS BRÆDDER HEMMINGSEN



201404469