SYSTEMARKITEKTUR

INDHOLDSFORTEGNELSE

1 INTRODUKTION

I dette dokument beskrives systemarkitekturen for 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. Produktet er dokumenteret ved brug af "4+1" view modellen, hvor der er lagt mest fokus på Logisk View. De generelle designbeslutninger er også beskrevet i dette dokument.

2 LÆSEVEJLEDNING

Systemarkitekturen er opbygget således, at der først er nogle indledende afsnit, hvor systemet beskrives. Dette sker i "Systemoversigt".

Efter introduktionen til projektet, følger der, i afsnittet "User stories", en henvisning til, hvor de funktionelle krav, som systemet er udarbejdet efter, kan findes.

Derefter kommer "Logisk View", hvor der først beskrives Data Access Layer, og derefter Shared Functionalities, som beskriver de funktionaliteter, som er delte mellem Consumer og Administration. Der uddybes så om Consumer og Consumer GUI, og de sekvensdiagrammer der hører til dem. Efter det uddybes der om Administrationen og Admin GUI, hvorefter de relevante sekvensdiagrammer vises. Endeligt beskrives Storemanager, med de tilhørende sekvensdiagrammer.

Der beskrives så de resterende views, med begrundelse for hvorfor de er valgt/fravalgt, og hvad de indeholder, hvis de er valgt.

Dernæst følger afsnittet "Generelle designbeslutninger", hvor der først redegøres for, hvilken arkitektur der er valgt, og efterfølgende hvilke mønstre der er brugt. Der følger hernæst afsnittet "Test", hvor der forklares, hvordan systemet er testet. Derefter beskrives der, hvilke udviklingsværktøjer der er brugt gennem projektet, i afsnittet "Udviklingsværktøjer". Til sidst følger afsnittet "Frameworks og packages", hvor der gennemgås, hvilke frameworks og packages der er brugt, i løbet af projektet.

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

3 TERMLISTE

BLL = Business Logic Layer

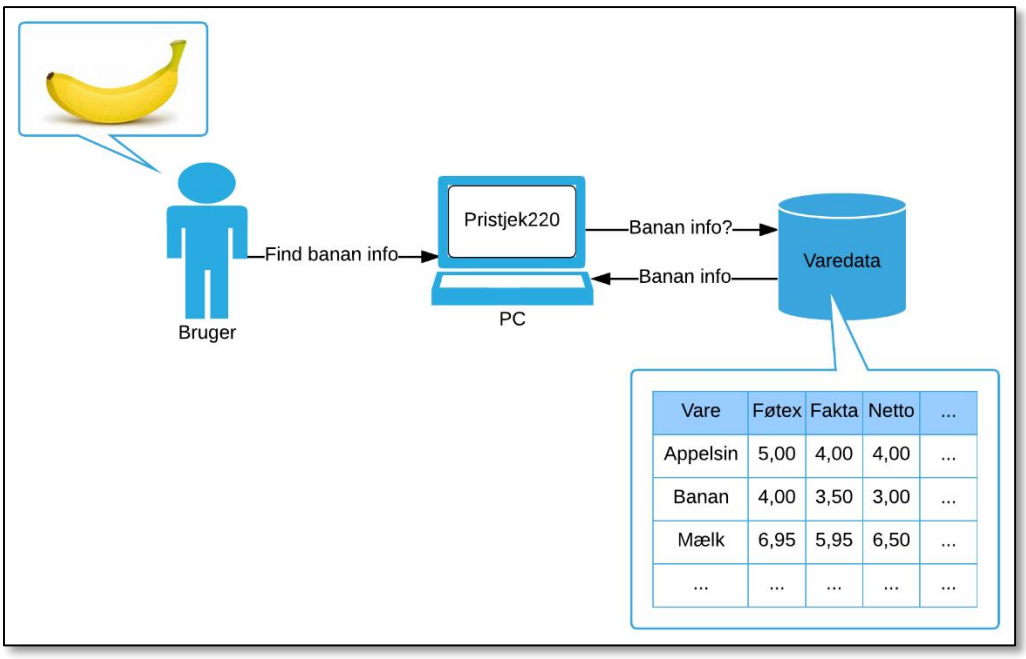
CRUD = Create, Read, Update and Delete

DB = Database

SD = Sekvensdiagram

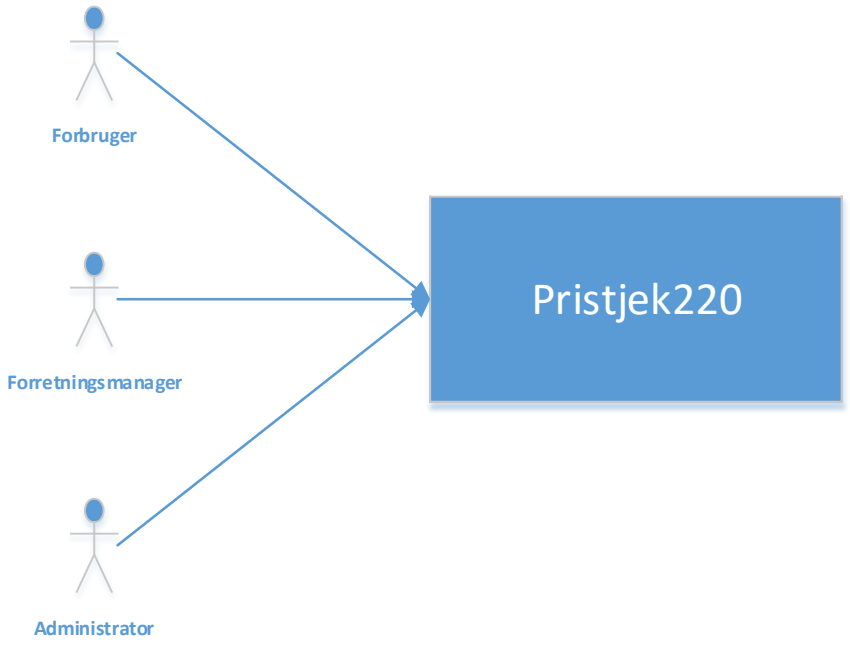
4 SYSTEMOVERSIGT

Konceptet bag, at slå et produkt op i Pristjek220, er illustreret på Figur 1. Det er den grundlæggende idé bag produktet, at en bruger skal kunne slå et produkt op, og finde ud af hvor det er billigst at købe. Pristjek220's andre funktionaliteter bygger på denne idé.



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

4.1 SYSTEM KONTEKST



FIGUR 2: AKTØR-KONTEKST DIAGRAM FOR PRISTJEK220

På Figur 2 ses aktør-kontekst diagrammet for Pristjek220, som viser de forskellige aktører. Aktørerne beskrives kort herunder:

- **Forbruger**
  - Forbrugeren er den almindelige bruger af Pristjek220, som til daglig bruger produktet, til at finde ud af hvor han skal handle ind henne.
- **Forretningsmanager**
  - Forretningsmanageren er bestyreren af en forretningskæde, som sørger for, at forretningens sortiment stemmer overens med informationerne i Pristjek220.
- **Administrator**
  - Administratoren styrer, hvilke forretninger der er i Pristjek220, og kan slette produkter, som ikke længere ønskes i Pristjek220.

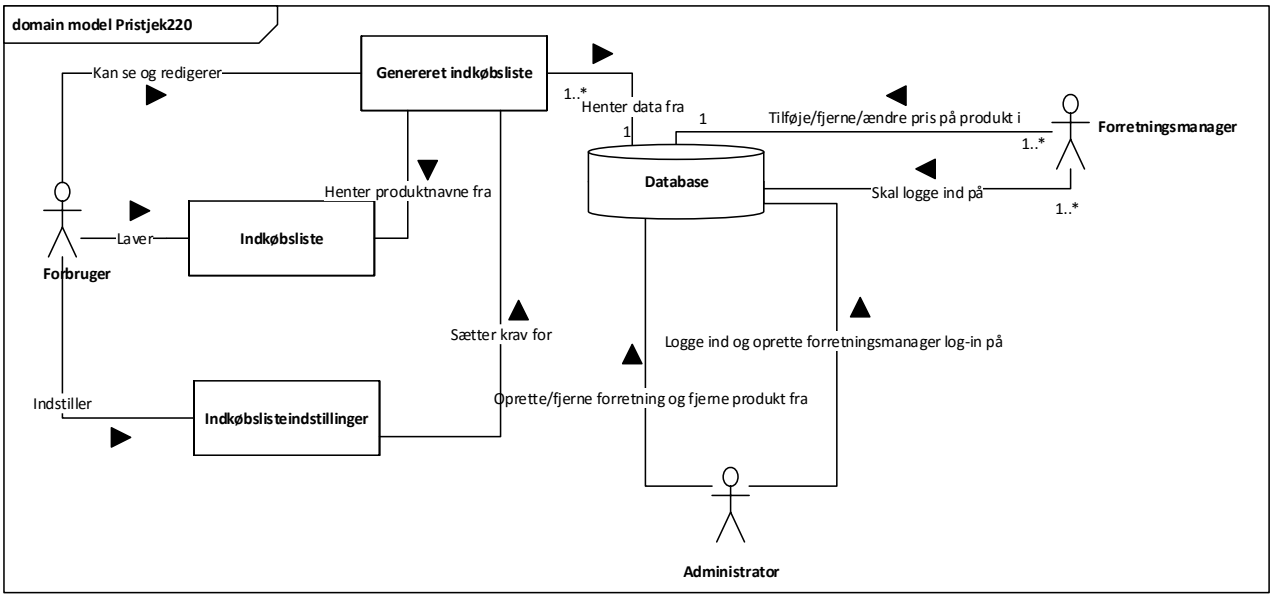
4.2 SYSTEM INTRODUKTION

Pristjek220 er et produkt, som tilstræber, at give forbrugeren et let og simpelt overblik over, hvor han kan handle sine dagligvarer billigst. Pristjek220 vil have tre forskellige 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.

Pristjek220 har en funktionalitet, sådan at en forbruger kan indtaste hans indkøbsseddel, og derefter kan han lave forskellige indstillinger, for hvilke forretninger han ønsker at handle i. Ud fra disse indstillinger, kan applikationen så generere en liste, der beskriver, hvor han billigst køber de forskellige produkter, han ønsker.

Forretningsmanageren kan tilføje og fjerne produkter fra hans forretning. Administratoren står for at oprette nye forretningsmanagere med deres tilhørende forretning. Der findes en brugermanual for Pristjek220 i dokumentationen [1].

4.2.1 DOMÆNEMODEL



FIGUR 3: DOMÆNEMODEL AF PRISTJEK220

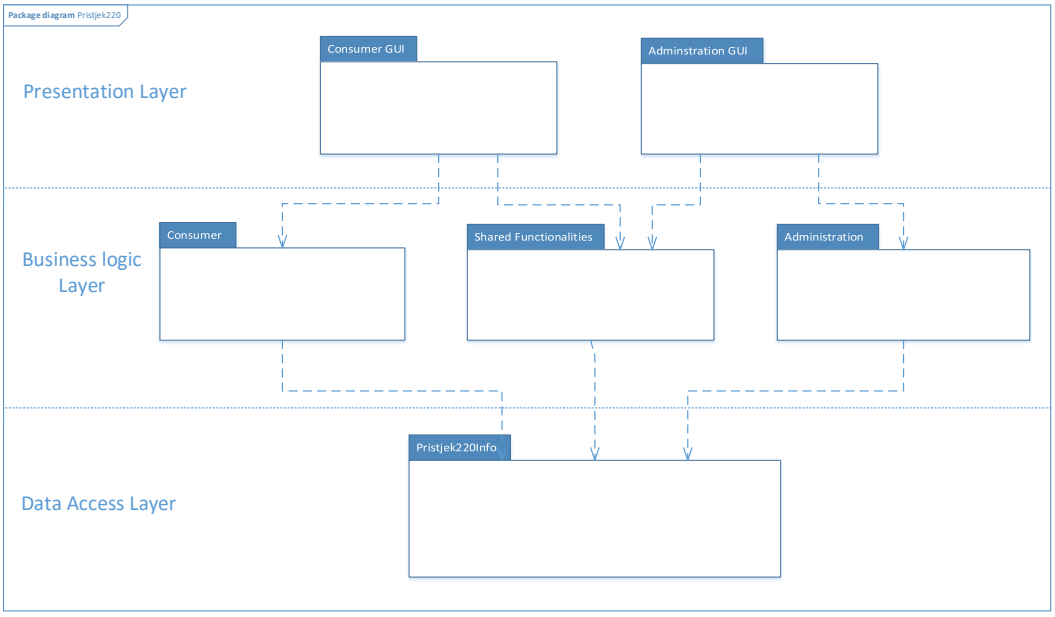
Figur 3 viser en domænemodel over Pristjek220, hvor der kan ses, hvordan de forskellige aktører interagerer med entiteterne. Databasen indeholder logins og en produktdatabase. Produktdatabasen indeholder de forskellige produkter, samt hvor man kan købe dem, og hvad deres pris er. Forbrugeren kan lave en indkøbsliste og, ved hjælp af indkøbslisteindstillinger, beslutte, hvilke forretninger der må handles i. Den detaljerede indkøbsliste, der genereres, skal så overholde disse indstillinger. Den detaljerede indkøbsliste genereres ud fra indkøbslisten ved at tjekke i databasen, hvor produkterne kan fås billigst, samtidig med at indstillingerne stadig overholdes.

5 USER STORIES

De funktionelle krav er formuleret i form af user stories. Disse user stories kan findes i Kravspecifikationen [2], hvor der også er opgivet nogle kvalitetskrav for Pristjek220.

6 LOGISK VIEW

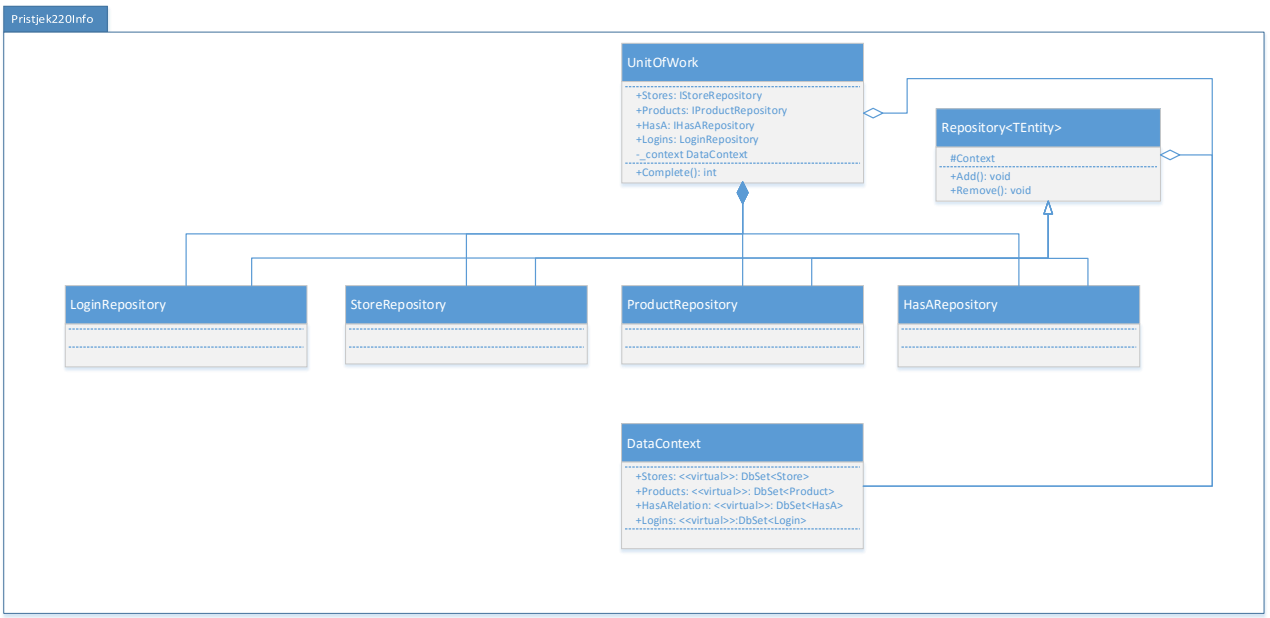
Det logiske view vil blive forklaret ud fra package diagrammet for systemet, hvor der vil blive kigget længere ind i hver pakke i dette afsnit. Det logiske view vil blive gennemgået først fra Data Access Layer, og derefter fra Shared Functionalities, da disse to bliver brugt af både Consumer og Administration, som package diagrammet på Figur 4 viser. Herefter vil Consumer blive gennemgået sammen med Consumer GUI, hvor der så vil blive vist sekvensdiagrammer, for de relevante funktioner som Consumeren har. Til sidst vil Administration blive gennemgået sammen med Administation GUI, hvor deres relevante funktioner præsenteres, i form af sekvensdiagrammer.



FIGUR 4: PACKAGE DIAGRAM FOR PRISTJEK220.

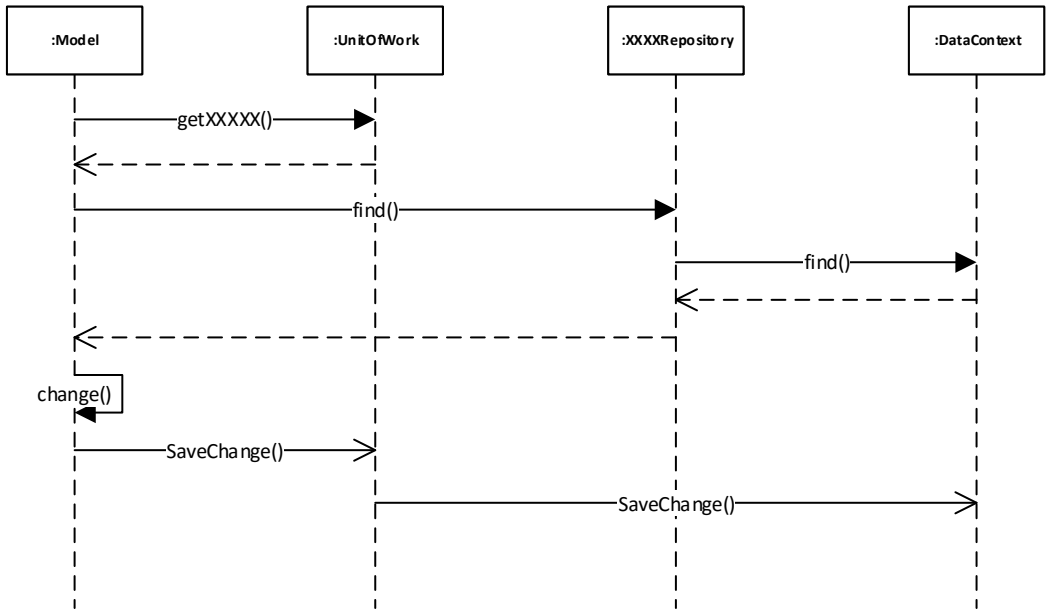
Package Diagrammet, som er vist på Figur 4, viser, hvordan koden for Pristjek220 er inddelt i forskellige lag og namespaces. Her er det tydeligt at se 3-lags modellen, da pakkerne er delt op i Data Access Layer, Business Logic Layer og Presentation Layer.

6.1 DATA ACCESS LAYER – PRISTJEK220INFO



FIGUR 5: PRISTJEK220INFO PACKAGE.

På Figur 5 kan klassediagrammet for Pristjek220Info ses, med de relevante funktioner på. Disse funktioner vil blive forklaret gennem SD'et, som vises på Figur 6. Her er der lavet et SD, for at kunne ændre værdien på en entitet. Der er kun lavet ét SD for Pristjek220Info, da det meste af funktionaliteten er det samme i de forskellige funktioner, med en meget lille variation. De forskellige Repositories på Figur 5 indeholder de forskellige CRUD-funktioner ned til databasen.



FIGUR 6: SD BESKRIVELSE AF HVORDAN REPOSITORY PATTERN VIRKER

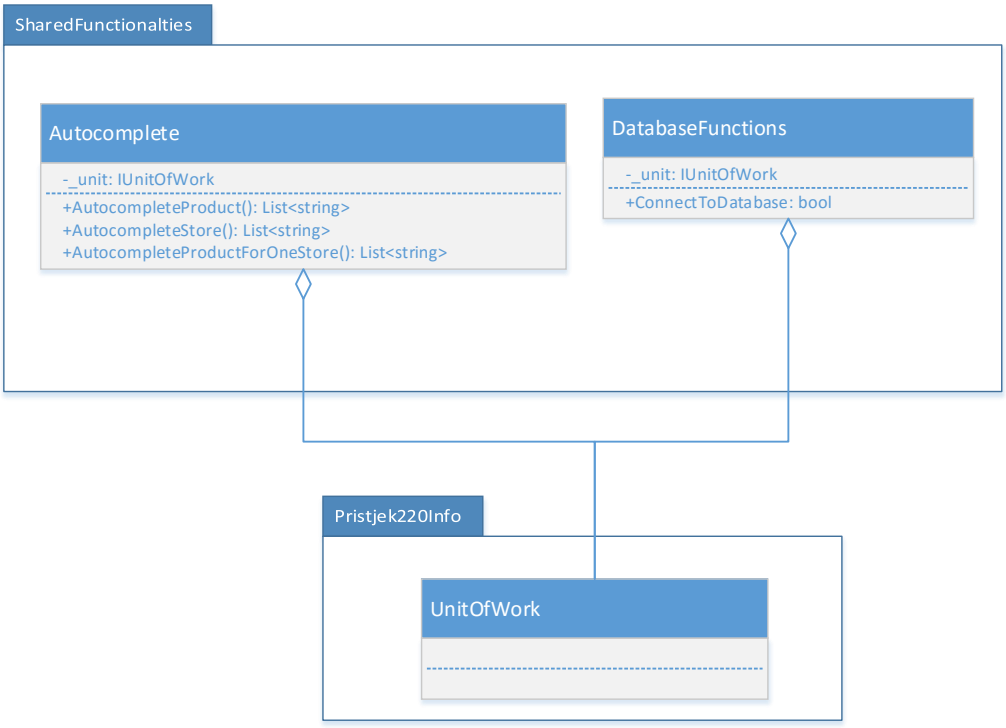
Figur 6 viser, hvordan repository pattern'et virker. Det står for at kalde de forskellig metoder på databasen, fra de forskellige models. I eksemplet er der taget udgangspunkt i, hvordan man kan ændre prisen på et produkt. Først kan

modellen lave et get-kald på det ønskede repository gennem UnitOfWork. Derefter kan der laves en find på det modtagende repository. Efterfølgende laver repositoriet en find ned i dataContexten, som så sender den entitet, der skal ændres, med tilbage til modellen. Så kan prisen ændres, og til slut kaldes der SaveChanges, for at det sendes til databasen.

På Figur 6 er der blevet valgt kun at illustrere én funktion for repository pattern'et, da funktionaliteten af de forskellige repositories er meget ens. Det er vist på Figur 6, hvor XXXX repræsenterer de forskellige repositories. Derudover er der kun valgt at vise ét diagram hvor der ændres på en entitet, da det at tilføje og fjerne har samme sekvens. Det med at gå ud til databasen og tilføje eller fjerne noget, og derefter gemme, er ens for dem alle, og det sker også gennem sekvensen af en ændring.

6.2 Business Logic Layer - Shared Functionalities

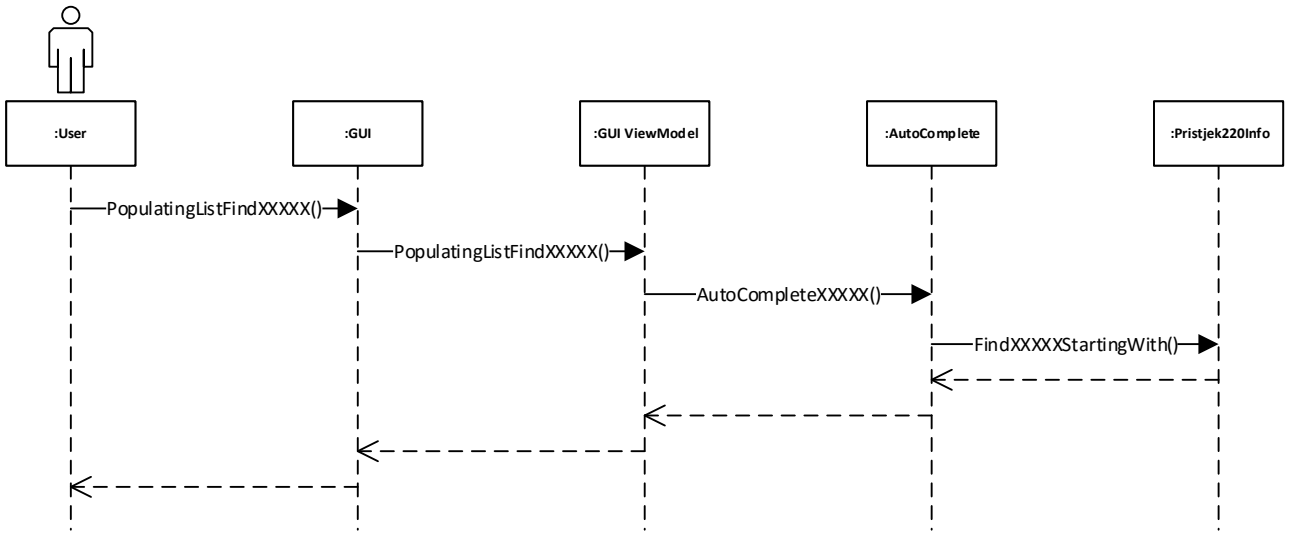
Shared Functionalities indeholder de funktioner, som både Consumer GUI og Administrations GUI bruger. Dette involver Autocomplete og DatabaseFunctions klasserne, som kan ses på Figur 7. DatabaseFunctions har kun én funktion. Denne funktion bruges til at etablere en forbindelse til databasen, når programmet startes op. Dette er valgt, for at det ikke skal tage lang tid, første gang der laves en søgning, eller en anden handling ned til databasen.



FIGUR 7: SHAREDFUNCTIONALITIES PACKAGE

Figur 7 viser indholdet af pakken Shared Functionalities, hvor klasserne Autocomplete og DatabaseFunctions er. Autocomplete klassen bruges til at lave programmets autofuldførelse, hvilket hjælper brugeren til at vide, hvad der er nede i databasen, når han skriver i de forskellige tekstbokse.

6.2.1 AUTOFULDFØRELSE

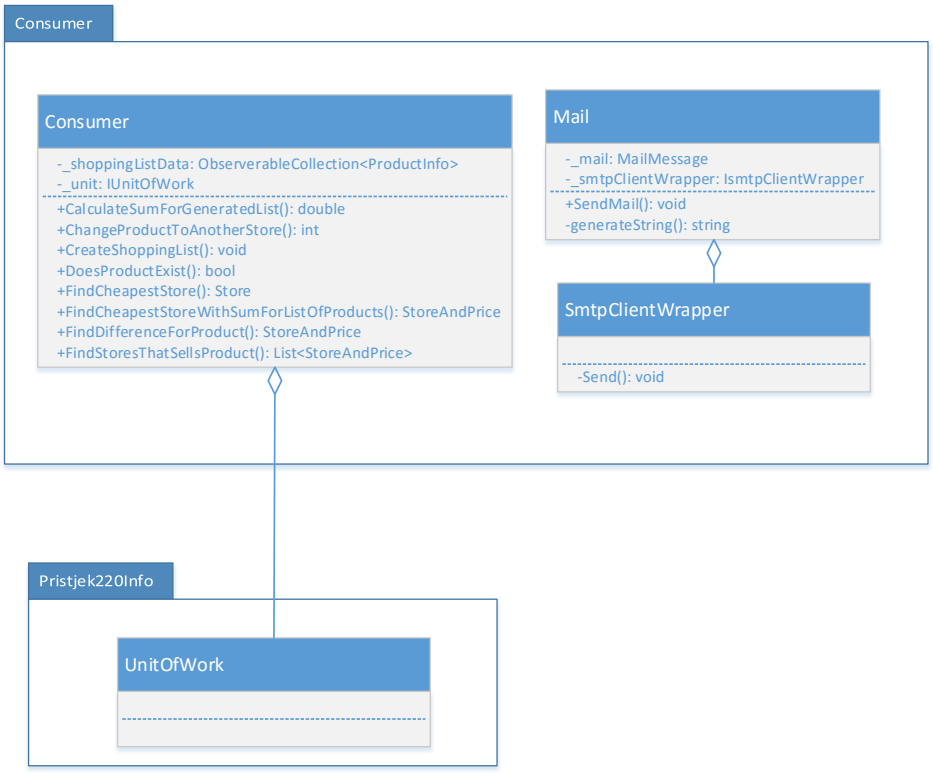


FIGUR 8: SEKVENSDIAGRAM FOR AUTOFULDFØRELSE

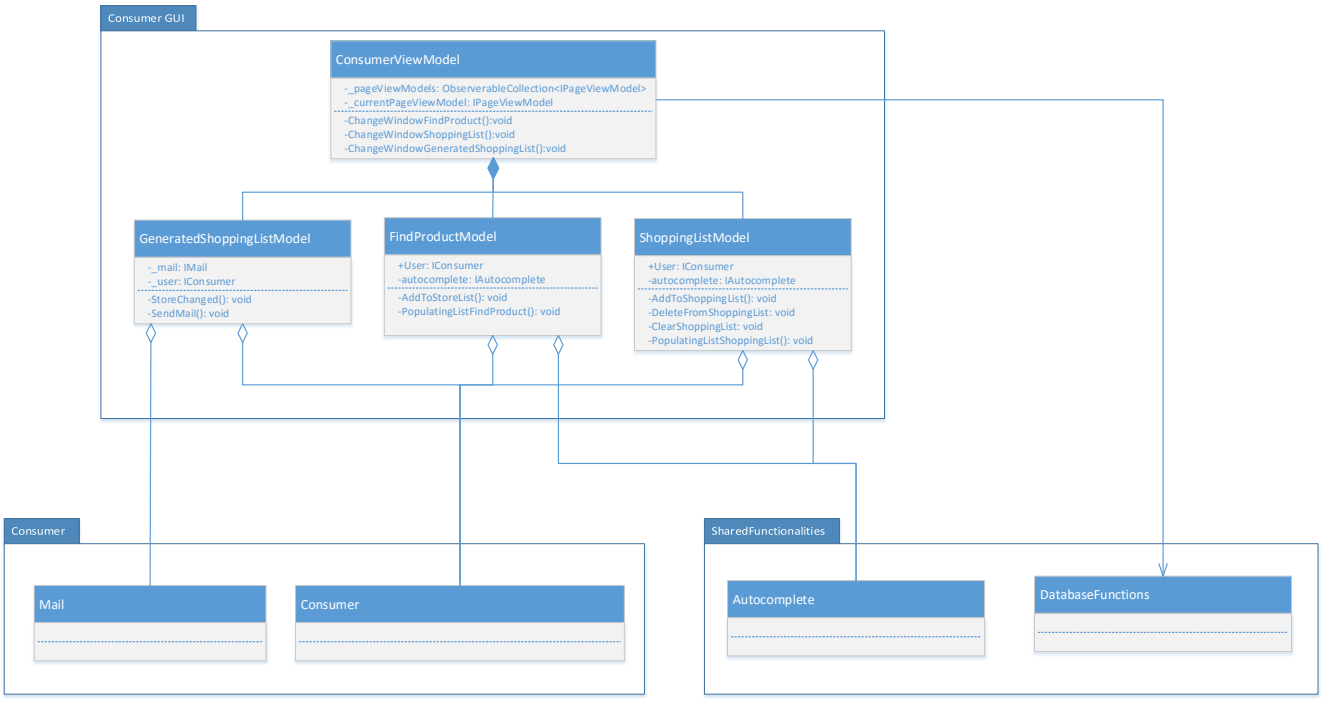
På Figur 8, ses sekvensdiagrammet for autofuldførelse, som viser, hvad der sker, når en User (forbruger, administrator eller forretningsmanager) begynder at indtaste i en autofuldførelsesboks. Der er forskellige metoder til autofuldførelse, afhængig af hvad der ønskes forslået, som er vist ved at bruge ”XXXXX”. Der er i diagrammerne valgt at skrive GUI, fordi autofuldførelse sker i alle tre GUI’er.

6.3 CONSUMER

Pakken Consumer indeholder klassen Consumer, Mail og SmtpClientWrapper. SmtpClientWrapper er en klasse, som er blevet lavet for at kunne teste Mail-klassen, da der ikke er noget interface ned til SmtpClient, som derfor ikke kunne substitueres ud i unittestene. Consumeren indeholder de funktionaliteter, som forbrugeren har brug for, for at kunne lave sin indkøbsliste, og få Pristjek220 til at genere en liste, der viser, hvor de forskellige produkter er billigst. Klasserne kan ses på Figur 9.



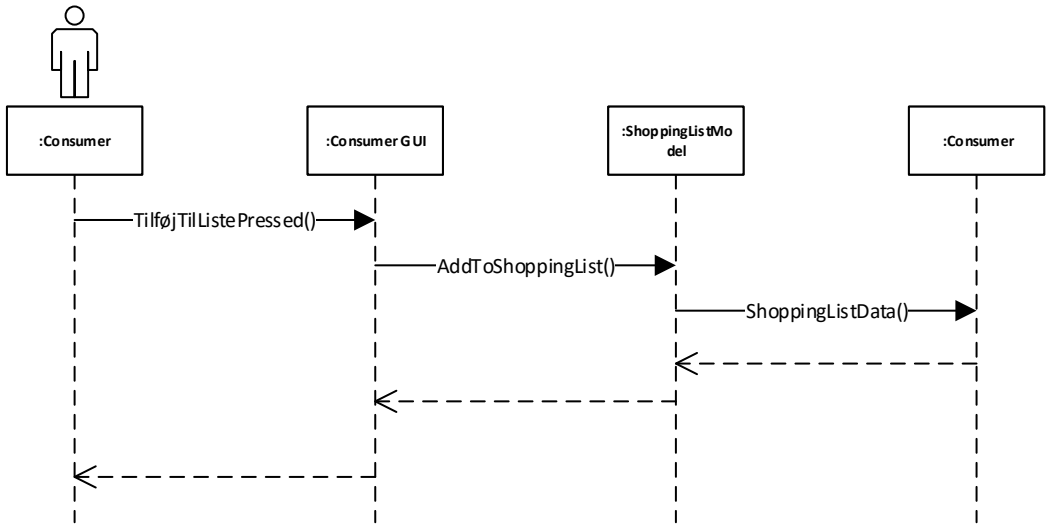
FIGUR 9: CONSUMER PACKAGE



FIGUR 10: CONSUMER GUI PACKAGE

Figur 10 viser indholdet af Consumer GUI pakken, og de relationer den har til de andre pakker. Inde i Consumer GUI pakken ligger ConsumerViewModellen, der er det overordnede vindue, som de andre viewmodeller ligger under. Der kan skiftes imellem disse viewmodeller gennem menuen.

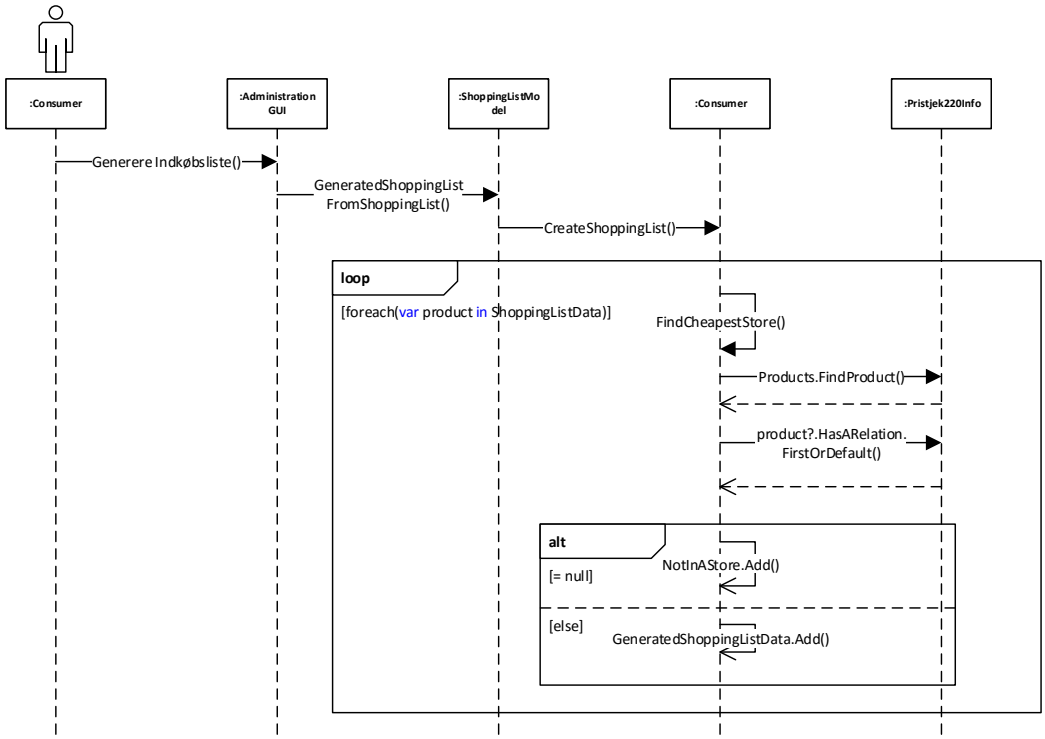
6.3.1 INDTAST INDKØBSLISTE



FIGUR 11: SD FOR AT INDTASTE INDKØBSLISTEN

Figur 11, er simplificeret, sådan at den kun viser, når der er indtastet noget i feltet, til at tilføje et produkt. Funktionen ShopppingListData er en set/get, som sætter Consumers ShoppingList, til at stemmeoverens med den aktuelle indkøbsliste.

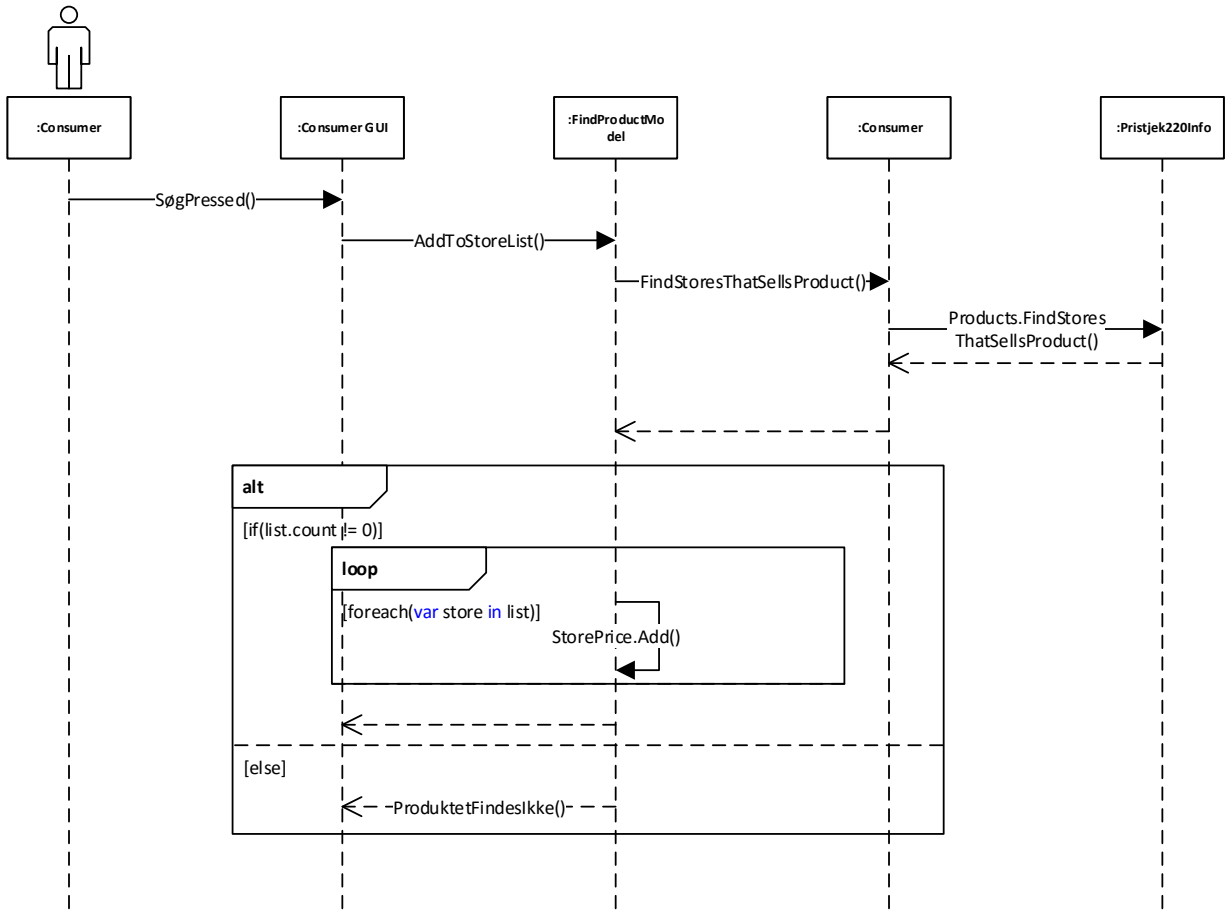
6.3.2 FIND UD AF HVOR PRODUKTERNE FRA INDKØBSLISTEN KAN KØBES BILLIGST



FIGUR 12: SD FOR AT FINDE UD AF HVOR PRODUKTENE FRA INDKØBSLISTEN KAN KØBES BILLIGST

Figur 12 viser, hvad der sker, når en bruger ønsker at få genereret en indkøbsliste. Consumer tjekker om produktet findes, og hvis det ikke findes, tilføjes det til listen med en ukendt forretning. Findes produktet derimod, løber den alle priser igennem, og returnerer den billigste, som den tilføjer til GeneratedShoppingListData.

6.3.3 FINDE HVILKE FORRETNINGER DER HAR ET PRODUKT

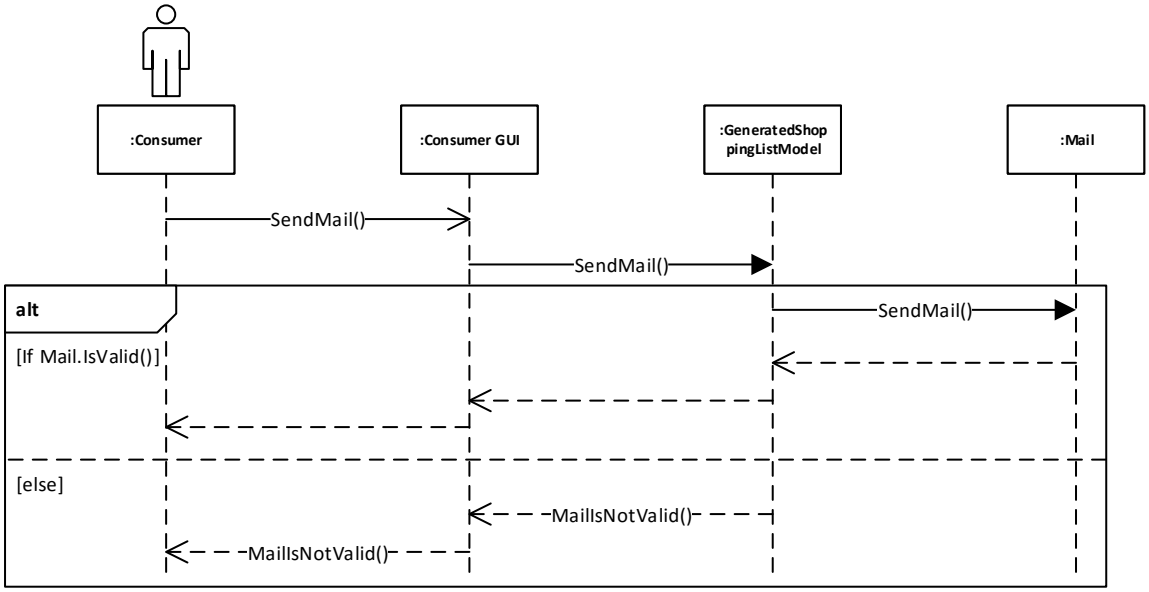


FIGUR 13: SD FOR AT FINDE HVILKE FORRETNINGER DER HAR ET PRODUKT

Figur 13 viser, hvad der sker, når en bruger ønsker at se, hvilke forretninger der har et produkt. Først tjekkes, om produktet findes i Pristjek220. Hvis det gør, returneres der en liste med de forretninger, der har produktet, samt prisen på produktet i disse forretninger. Denne liste bliver så løbet igennem, for at informationerne bliver tilføjet til den liste, som brugeren kan se.

6.3.4 SEND INDKØBSLISTE PÅ MAIL

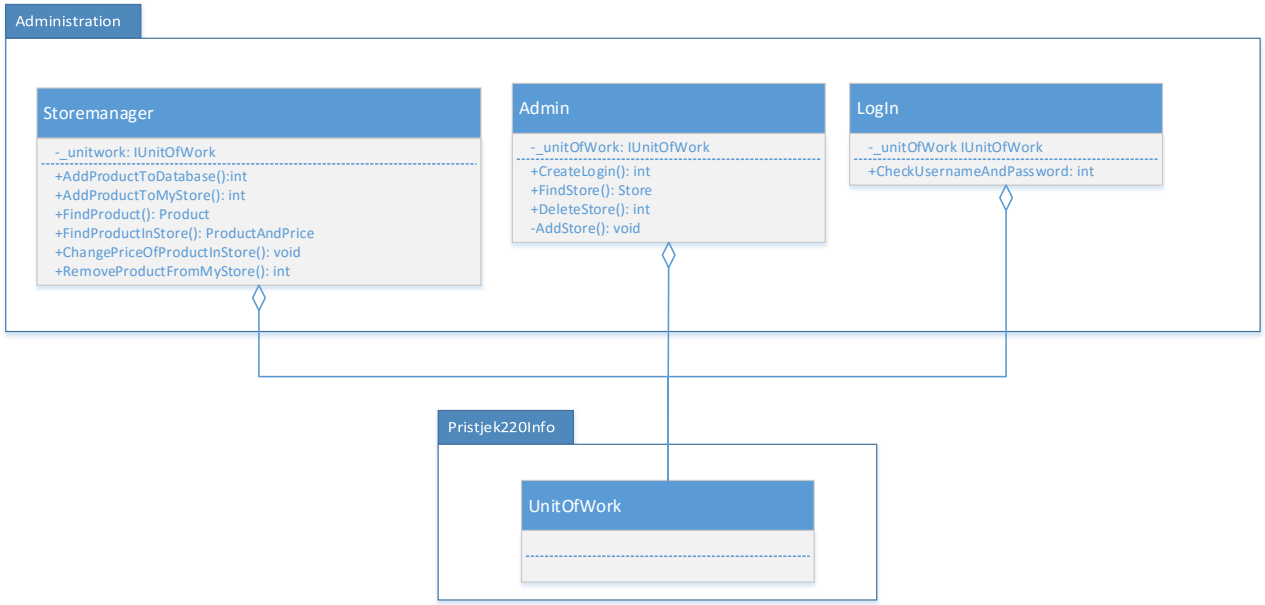
For at forbrugerne kan få deres indkøbsliste med, når de skal ud at handle, er der blevet implementeret en funktion, til at sende listen som en E-mail. Sekvensen for afsendingen af mailen kan ses på Figur 14.



FIGUR 14: SD FOR AT SENDE INDKØBSLISTE PÅ MAIL

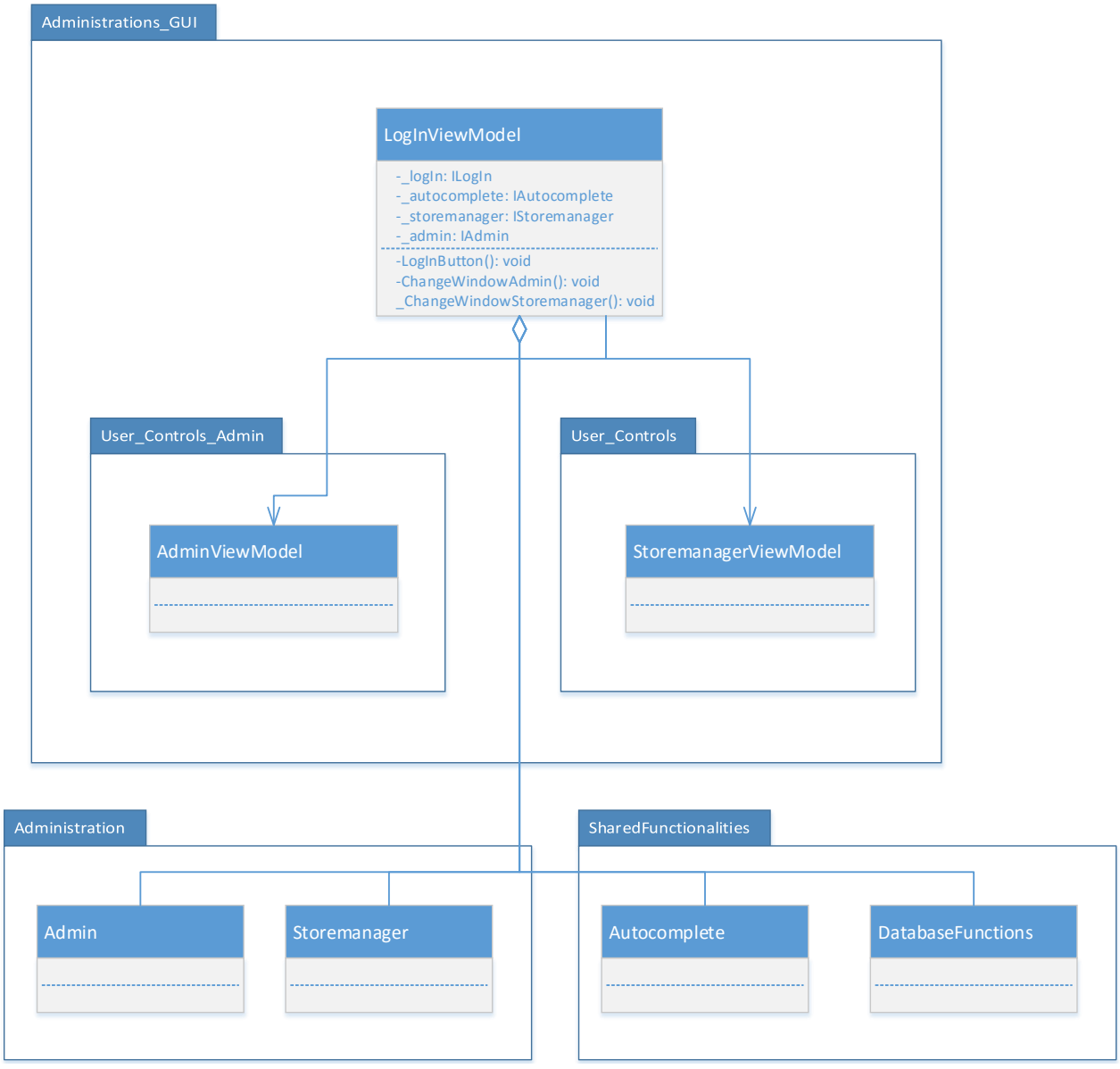
6.4 ADMINISTRATION

Administration-pakken indeholder de forskellige administrative klasser til Pristjek220, hvilket er forretningsmanageren og administratoren. For at kunne bruge disse to klasser, skal man logge ind. Her er det LogIn-klassen, der sørger for, at man har et gyldigt brugernavn og kodeord. Forretningsmanageren kan administrere sortimentet i hans egen forretning, og har derfor funktioner til dette. Administratoren er ham, der administrer, hvilke forretninger der er i Pristjek220, og har derfor funktioner, der tilfredsstiller dette behov. De forskellige klasser og funktioner kan ses på Figur 15, sammen med relationerne imellem dem.



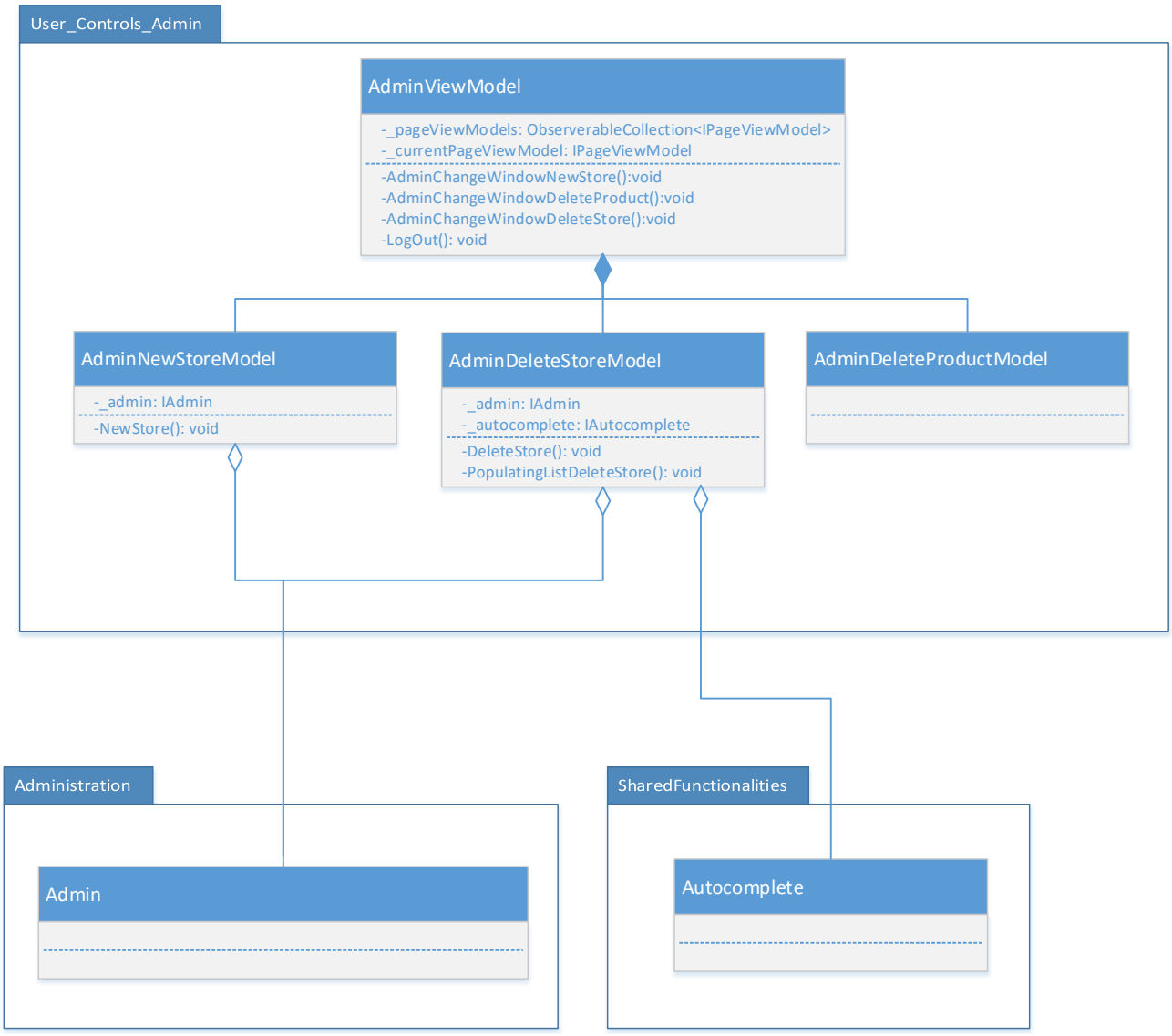
FIGUR 15: ADMINISTRATION PACKAGE

LogInViewModel er den første viewmodel, som bliver kørt, når administrationsapplikationen startes. Herfra kan man logge ind som administrator eller som forretningsmanager. Afhængigt af hvilket login der bliver indtastet, bliver AdminViewModel eller StoremanagerViewModel kørt. Når den startes op, laver LogInViewModel en forbindelse til databasen, for at der ikke skal bruges tid på at oprette forbindelsen, første gang der laves en søgning, eller en anden handling ned til databasen. LogInViewModel, AdminViewModel og StoremanagerViewModel ligger alle i pakken Administrations_GUI. Dette kan ses på Figur 16, hvor relationerne til andre pakker og deres klasser også er illustreret.



FIGUR 16: ADMINISTRATION GUI PACKAGE

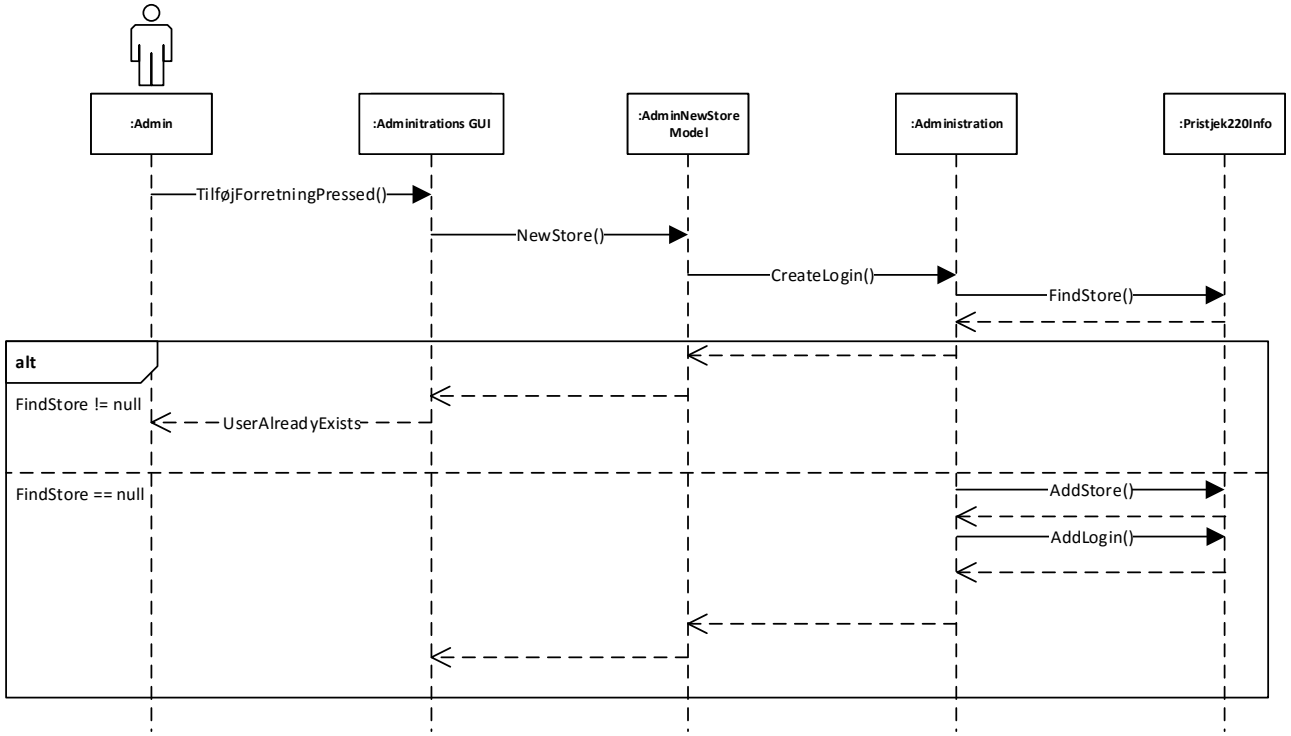
6.4.1 ADMIN



FIGUR 17: USER_CONTROLS_ADMIN PACKAGE

På Figur 17 kan man se, at AdminViewModel har tre Models. Disse models er de forskellige undermenuer, der er, når man er logget ind som Administrator. Disse models er koblet sammen med Admin og Autocomplete klasserne nede i BLL, for at kunne give administratoren mulighed for, at kunne udføre hans user stories. Dog er AdminDeleteProductModel ikke koblet sammen med BLL, da funktionaliteten ikke er blevet implementeret.

6.4.1.1 Tilføj en forretning til Pristjek220

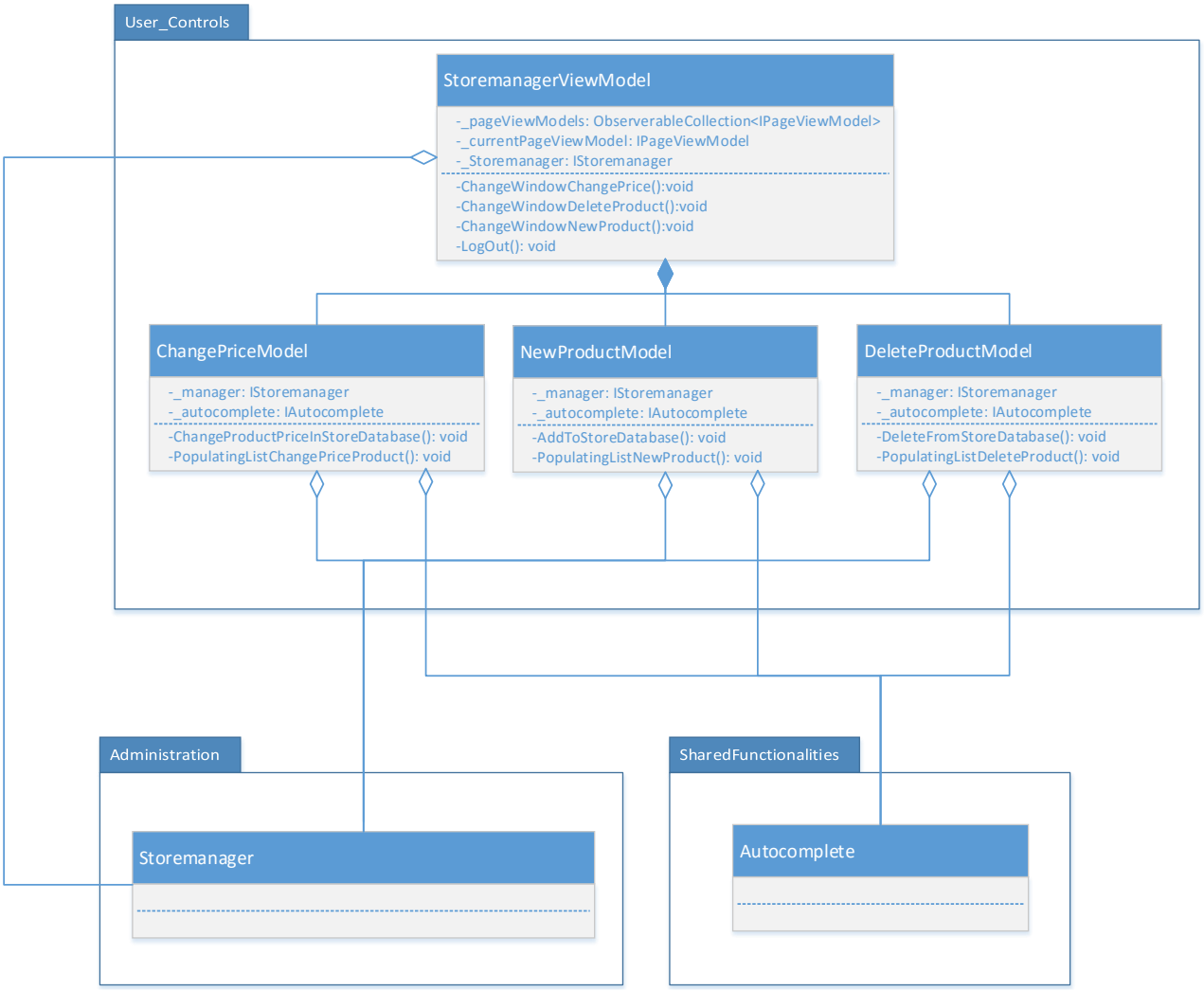


FIGUR 18: SD FOR TILFØJ EN FORRETNING TIL PRISTJEK220

Figur 18 viser sekvensen for at tilføje en forretning. I diagrammet er der taget udgangspunkt i, at Administratoren allerede har indtastet et brugernavn, samt et kodeord. Forretningen, som bliver tilføjet, får samme navn som brugernavnet. Brugernavnet til storemanageren for ”Fakta” vil altså dermed også være ”Fakta”.

Der er kun blevet lavet et SD for at tilføje en forretning, da sekvensen af at fjerne en forretning er det samme, og derfor ikke ville tilføje noget værdifuld viden.

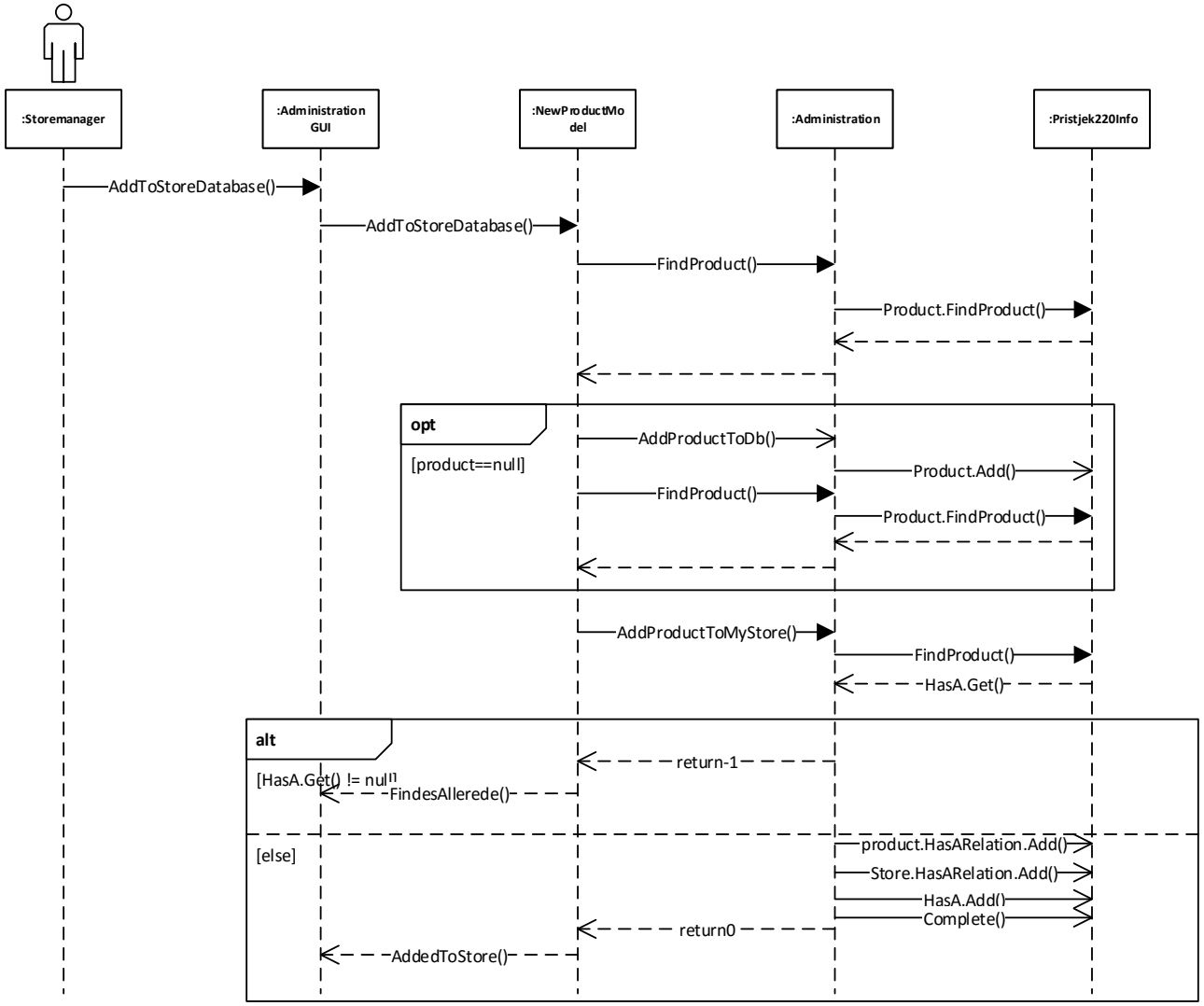
6.4.2 STOREMANAGER



FIGUR 19: USER_CONTROLS PACKAGE

På Figur 19 kan man se, at StoremanagerViewModel har tre forskellige Models, hvilket er de forskellige menuer, som forretningsmanageren kan navigere rundt imellem. Disse models har relationer ned til Storemanager- og Autocomplete-klasserne, for at forretningsmanageren kan udføre hans user stories.

6.4.2.1 TILFØJ PRODUKT TIL FORRETNING



FIGUR 20: SD FOR TILFØJ ET PRODUKT

Figur 20 viser kodesekvensen for at tilføje et produkt, til forretningsmanagerens forretning. Det kan ses, at der tjekkes på, om produktet findes i forvejen. Hvis det gør det, tilføjes det bare til hans forretning, ved at lave en relation mellem forretningen og produktet, med en HasA entitet i databasen. Hvis produktet ikke findes i databasen i forvejen, bliver produktet oprettet, hvorefter relationen mellem forretningen og produktet laves.

Diagrammerne for at ændre eller fjerne et produkt fra forretningsmanagerens forretning er ikke lavet, da Figur 20 viser overordnet, hvordan funktionaliteten virker.

7 DEVELOPMENT VIEW

Development View beskæftiger sig med at opdele softwaren i mindre dele, som subsystemer og lag. Denne opdeling sker ved at udarbejde component og package diagrammer. Disse to diagrammer beskrives kort i det følgende, hvor der samtidig også begrundes for, hvorvidt diagrammet er brugt i dette projekt.

Der er valgt at implementere package diagrammer, da de er gode til at danne et overblik, over de forskellige dele der indgår i programmet. Derudover er de gode til at dele klassediagrammerne op, og kun vise de klassediagrammer der hører til en bestemt pakke. Dette er en overskuelig måde at vise klassediagrammerne på, og nemt vise hvilke klasser der har afhængigheder til en anden pakke. Det er valgt at placere package diagrammerne i Logisk View, pga. den tætte sammenkobling med klassediagrammerne.

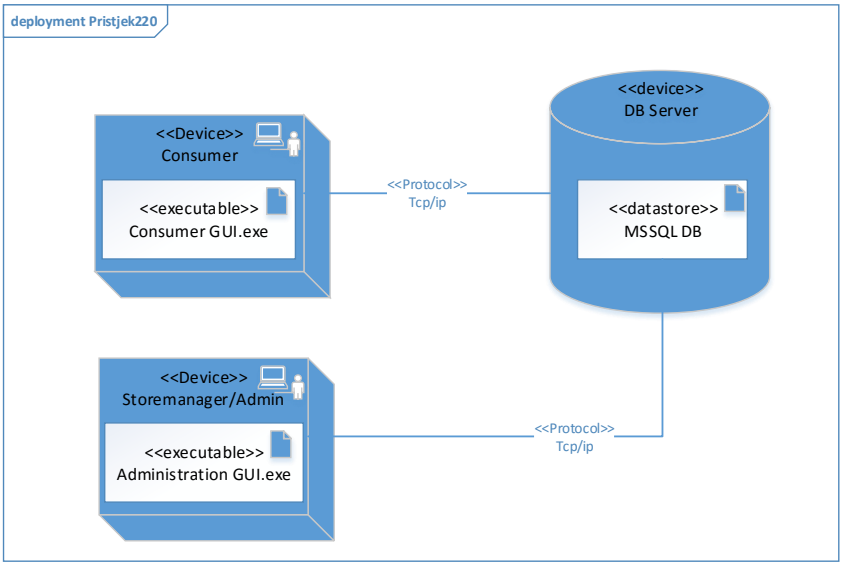
Component diagram er fravalgt at implementere, da det er et diagram, der blev vurderet til ikke at give værdi til projektet. Et component diagram viser, hvilke komponenter der er, samt de forskellige interfaces der forventes herimellem. Diagrammet bliver typisk brugt til Component-Based Development (CBD) [3], hvilket går ud på at genbruge komponenter, som andre allerede tidligere har lavet, og blot sætte de forskellige komponenter sammen i ens system. Det vil derfor ikke være relevant i dette projekt, da der ikke benyttes Component-Based Development, og projektets størrelse ikke er stor nok, til at det tilfører nogen værdi.

8 PROCESS VIEW

Process view fokuserer på dynamikken i systemet, og viser de forskellige processer, og hvordan de kommunikerer. Her vil der typisk blive benyttet flow charts til at illustrere dette. Gruppen var dog enige om, at flow charts ikke viste noget, som sekvensdiagrammerne i det logiske view ikke allerede viste. Der er derfor valgt ikke at udarbejde flow charts, da sekvensdiagrammerne lettere kan relateres til koden. Selve process view er derfor heller ikke implementeret i arkitekturen, da der ikke var noget, gruppen følte var relevant for projektet at vise her.

9 Deployment View

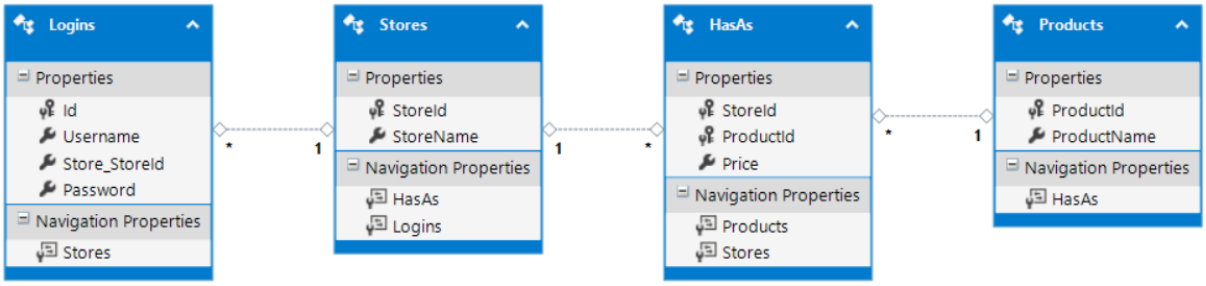
Deployment view viser, hvordan softwarekomponenter er distribueret på moduler i det fysiske plan, samt hvordan der kommunikeres mellem disse komponenter. Der er valgt at benytte dette view, da det viser, hvordan der kommunikeres mellem applikationerne, der kører på en computer og serveren.



FIGUR 21: DEPLOYMENT DIAGRAM FOR PRISTJEK220

Figur 21 viser deployment diagrammet for Pristjek220. Diagrammet viser på hvilke hardware elementer, som de forskellige software implementeringer skal placeres. Kommunikationen, mellem de forskellige enheder og DB Serveren, foregår ved brug af Tcp/Ip, som er den protokol, der overføres til og fra databasen med. De forskellige executables er applikationer, som kører på de to devices.

10 Data View



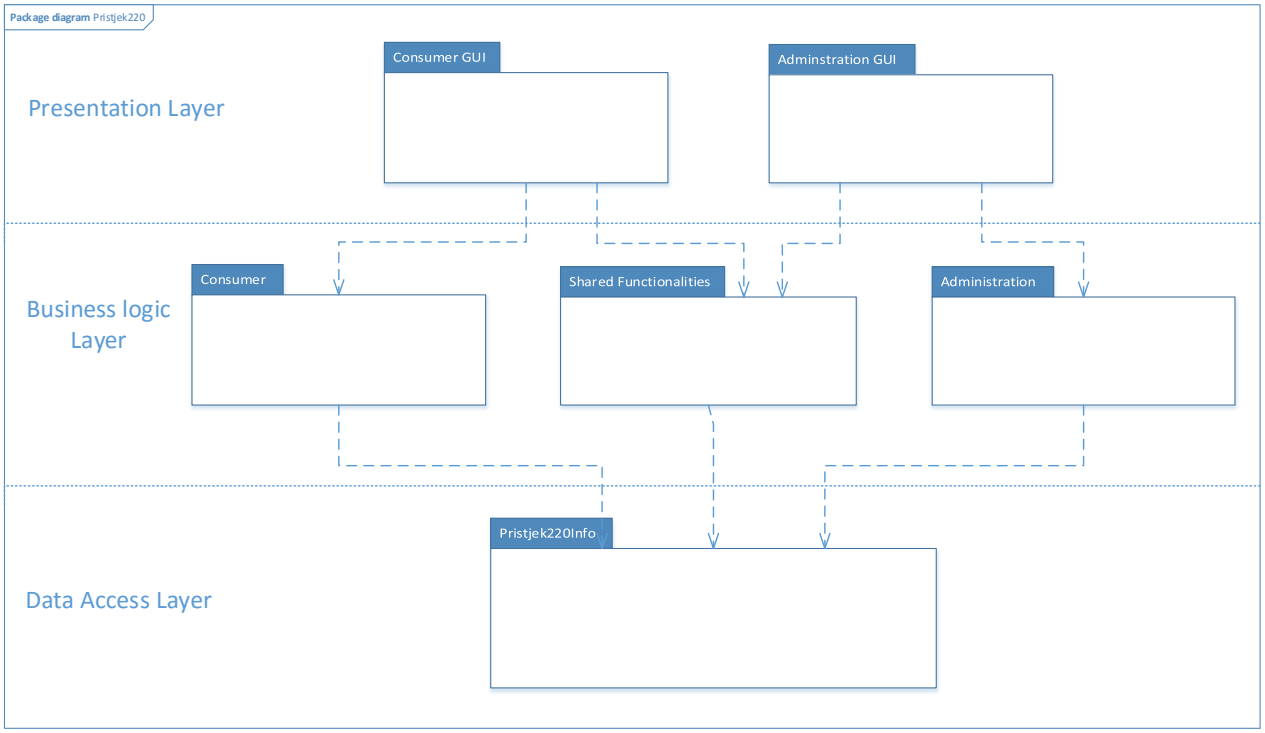
FIGUR 22: UML DIAGRAM FOR DATABASEN.

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. UML diagrammet for databasens opbygning kan ses på Figur 22.

11 GENERELLE DESIGNBESLUTNINGER

11.1 ARKITEKTUR MØNSTRE

Der er i projektet brugt 3-layer model, som det kan ses illustreret på Figur 23.

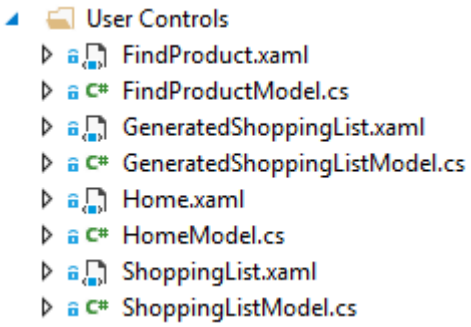


FIGUR 23: PACKAGE DIAGRAM FOR PRISTJEK220

11.2 DESIGN MØNSTRE

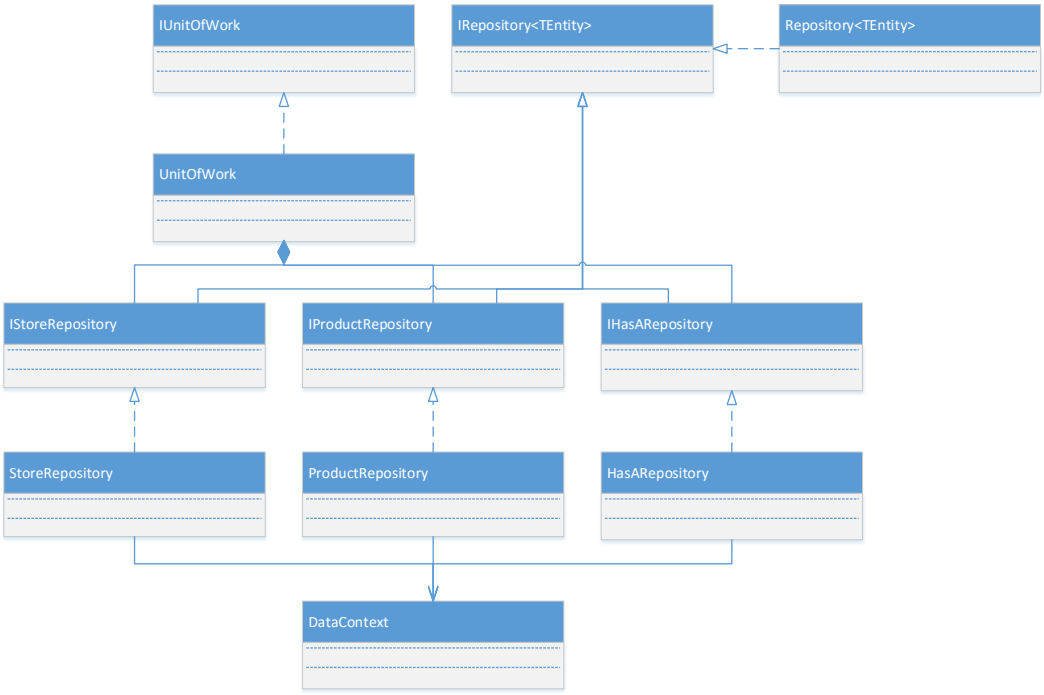
11.2.1 MVVM PATTERN

Gennem Logical view, kan der ses, at der i programmet er brugt MVVM, ved at de forskellige views har en tilhørende viewmodel. Det kan også ses på Figur 24, som viser de forskellige views (.xaml filerne) med deres viewmodel (.cs filerne). Der kan findes yderligere eksempler på det i source koden.



FIGUR 24: SCREENSHOT AF OPBYGNING AF VIEW OG VIEWMODEL

11.2.2 REPOSITORY PATTERN



FIGUR 25: IMPLEMENTERING AF REPOSITORY PATTERN I PRISTJEK220

På Figur 25 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. 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.

11.3 GENERELLE BRUGERGRÆNSEFLADEREGLER

De generelle brugergrænsefladeregler, som produktet er udarbejdet efter, kan ses i kravspecifikationen [2] under kvalitetskrav.

12 TEST

12.1 UNITTEST

For at dokumentere kvaliteten af Pristjek220, er produktet blevet unittestet, for at sikre at klasserne i programmet lever op til de forventede krav, og kan udføre de formulerede user stories.

Til at udføre unittestene i Pristjek220, er der blevet benyttet NUnit, hvilket er et framework i C#, som gør det let at lave opsætningen af testcases, og derved tjekke at funktionerne virker. NSubstitute er samtidig også blevet benyttet til at unitteste med. Nsubstitute er ligeledes et framework, som gør det lettere at lave Stubs og Mocks, til at tjekke tilstand og adfærd, når man tester funktionerne.

Alle funktioner i Pristjek220 er blevet unittestet, med nogle få undtagelser:

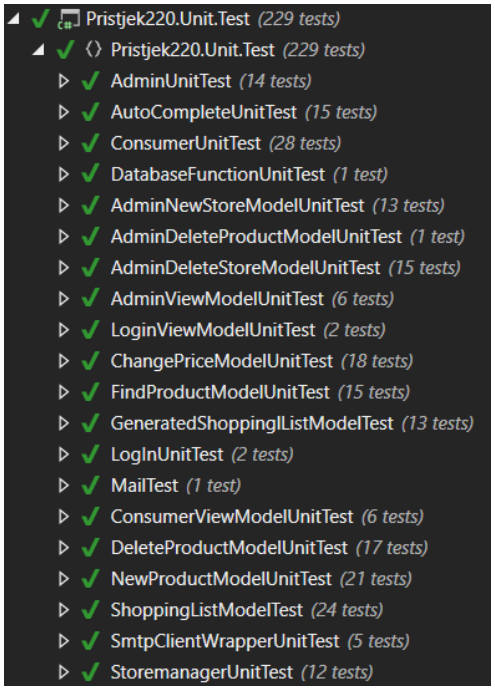
- SmtpClientWrapper – Send()
- EnterKeyPressed Funktionen
- LogInViewModel – ChangeWindow funktionerne

Disse funktioner er ikke blevet testet, da der ved hvert tilfælde har været en situation, der har gjort, at de ikke umiddelbart var mulige at unitteste.

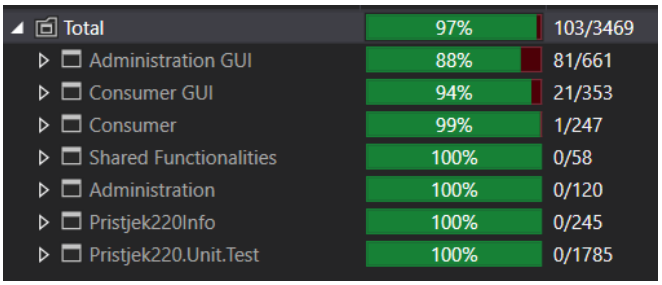
Send-funktionen har vi ikke kunne teste, da den smider en exception, da klassen bliver substitueret ud, og derfor ikke bliver instantieret, og ikke kan sende en mail.

EnterKeyPressed er ikke blevet testet, da der ikke er fundet en metode at lave inputtet, som er et KeyEventArgs, og derfor ikke har kunne kalde funktionen i testene.

LogInViewModel ChangeWindow funktionerne er ikke blevet testet, da den kalder en applikation, som ikke er instantieret, på grund af at der igennem testene ikke kører nogen applikation.



FIGUR 27: UDFØRELSE AF UNITTEST I PRISTJEK220



FIGUR 26: COVERAGE AF PRISTJEK220 UNITTESTS

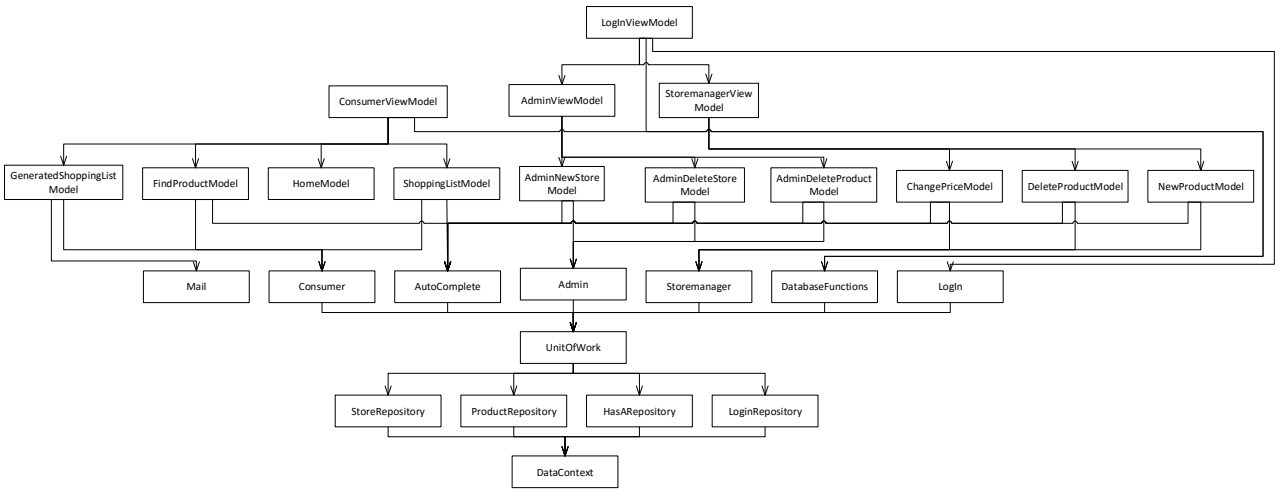
Figur 27 viser resultatet af Pristjek220's 229 automatiserede unittests, hvor det kan ses, at de alle er godkendt. Figur 26 viser coveragen af unittestene, hvor det kan ses, at der ikke opnås 100% coverage. Dette skyldes, som der blev forklaret før, at der er nogle ting, der ikke er blevet testet. De klasser, som gruppen ikke selv har skrevet, er blevet ekskluderet.

12.2 INTEGRATIONSTEST

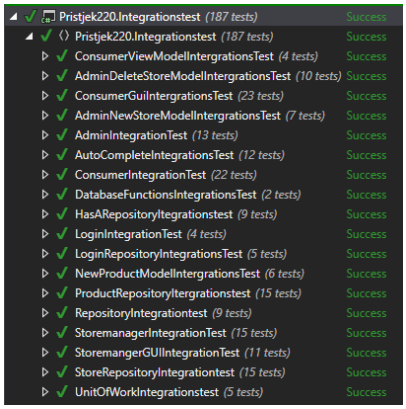
Da unittests tester hver klasse isoleret fra resten af systemet, er der nogle ting, som ikke bliver testet igennem unittestning, såsom hvordan de forskellige klasser interagerer med hinanden. Dette kan man derimod teste med integrationstestning. Her bliver interfacene mellem klasserne testet, da der bliver lavet funktionskald ned til klassen igennem disse. Her kan der ske fejl, hvis der gennem udviklingen ikke har været den samme forståelse for, hvordan de skal kommunikere sammen.

Til udførelse af integrationstest er der også blevet benyttet NUnit, da dette framework gør det nemt at opsætte test cases.

Integrationstestene er blevet lavet gennem en bottom-up strategi, for at sikre at funktionaliteten mellem de forskellige klasser fungerer helt. Der er derfor skrevet testdrivers til hver klasse, der interagerer med en anden klasse, for at teste denne integrering. Der er testet ud fra Pristjek220's dependency tree, som kan ses på Figur 28.



FIGUR 28: PRISTJEK220'S DEPENDENCY TREE



FIGUR 29: UDFØRELSE AF INTEGRATIONSTEST I PRISTJEK220

Figur 29 viser resultatet af Pristjek220 229 automatiserede integrationstest, hvor der kan ses de alle er godkendt.

13 Udviklingsværktøjer

13.1 Visual Studio [4]

Applikationerne er programmeret i C# i Visual Studio, som er Microsofts programmeringsværktøj, med tillægsværktøjet ReSharper og forskellige NuGet packages.

13.2 Microsoft Visio [5]

Microsoft Visio er Microsofts program til at lave diagrammer. Programmet giver et godt interface til at få produceret de ønskede diagrammer. Der er i Visio udviklet diagrammer i UML.

13.3 Microsoft Word [6]

Til redigering af rapporten og andre dokumenter i projektet er der valgt at bruge Word, som har en grafisk brugergrænseflade, der er let at gå til. Word har gode og lette funktionaliteter, der gør det let at lave kildehenvisninger og billedtekster. Derudover indeholder den som standard en velfungerende stave- og grammatikkontrol.

13.4 Scrumwise [7]

Scrumwise er et værktøj til at styre et Scrum board, som kan tilgås online af alle teamets medlemmer, samt af vejlederen.

13.5 GitHub [8]

Github er en hjemmeside, hvor der kan oprettes Git repositories, som bruges til versionsstyring. Der er gennem projektet brugt git på produktet og dokumenterne.

13.6 TortoiseGit [9]

TortoiseGit er den Git klient, der er benyttet til at interagere med GitHub repositoriet.

14 Frameworks og packages

Her følger en kort beskrivelse af, hvilke frameworks og packages der er brugt, og hvorfor de er brugt.

14.1 Generelle frameworks

EntityFramework [10]

Er et framework fra Microsoft, som bruges til at automatisere databaserelaterede aktiviteter til applikationer. Det er gennem projektet brugt til at oprette en database, tilgå den og ændre i den.

Newtonsoft.Json [11]

Er en package, til at lave Json objekter, som fungerer bedre end den indbyggede i C#. Den er brugt til at konvertere til og fra Json, når der skrives til og fra en fil.

14.2 Test Frameworks

NUnit [12]

Er et framework til at automatisere tests med, således at en applikation kan teste dens funktionaliteter, ved at opsætte test cases, som frameworket kører.

NSubstitute [13]

Er et framework til at substituere klasser ud med, når der laves unittests, og i nogle tilfælde integrationstests. Det gøres for at kunne isolere problemet, til hvad der ønskes testet.

14.3 Gui frameworks

WPFToolkit [14]

Er brugt for at få en autocomplete boks, som lever op til de krav, som der ønskes gennem projektet.

MvvmLight & MvvmLightLibs & CommonServiceLocator [15]

Er brugt for at kunne sende et taste tryk, sammen med en command, så der kan tjekkes, om der er trykket enter, på en autocomplete boks.

15 REFERENCER

[1] Gruppe7, »Brugermanual,« Au, Aarhus, 2016.

[2] Gruppe7, »Kravspecifikation,« AU, Aarhus, 2016.

[3] Techopedia, »Component-Based Development (CBD),« 2016. [Online]. Available: https://www.techopedia.com/definition/31002/component-based-development-cbd. [Senest hentet eller vist den 22 5 2016].

[4] Microsoft, »Visual Studio,« 2016. [Online]. Available: https://www.visualstudio.com/. [Senest hentet eller vist den 22 5 2016].

[5] Microsoft, »Visio,« 2016. [Online]. Available: https://products.office.com/en-us/visio/. [Senest hentet eller vist den 22 5 2016].

[6] Microsoft, »Word,« 2016. [Online]. Available: https://products.office.com/en-us/word. [Senest hentet eller vist den 22 5 2016].

[7] Scrumwise, »Scrumwise,« 2016. [Online]. Available: https://www.scrumwise.com/. [Senest hentet eller vist den 22 5 2016].

[8] Github, »Github,« 2016. [Online]. Available: https://github.com/. [Senest hentet eller vist den 22 5 2016].

[9] TortoiseGit, »TortoiseGit,« 2016. [Online]. Available: https://tortoisegit.org/. [Senest hentet eller vist den 22 5 2016].

[10] Microsoft, »Entity Framework,« Marts 2015. [Online]. Available: https://www.nuget.org/packages/EntityFramework/6.1.3. [Senest hentet eller vist den 26 5 2016].

[11] J. Newton-King, »Json.NET,« 3 2016. [Online]. Available: https://www.nuget.org/packages/Newtonsoft.Json/8.0.3. [Senest hentet eller vist den 26 5 2016].

[12] C. Poole, »NUnit,« 4 2016. [Online]. Available: https://www.nuget.org/packages/NUnit/3.2.1. [Senest hentet eller vist den 26 5 2016].

[13] A. Egerton og D. Tchepak, »NSubstitute,« 3 2016. [Online]. Available: https://www.nuget.org/packages/NSubstitute/1.10.0. [Senest hentet eller vist den 26 5 2016].

[14] J. og S. , »WPF Toolkit,« 01 2012. [Online]. Available: https://www.nuget.org/packages/WPFToolkit/3.5.50211.1. [Senest hentet eller vist den 26 5 2016].

[15] L. Bugnion, »Mvvm Light,« 4 2016. [Online]. Available: https://www.nuget.org/packages/MvvmLight/5.2.0/. [Senest hentet eller vist den 26 5 2016].