

Projektrapport
4. Semester - Gruppe 1



Rieder, Kasper
201310514

Jensen, Daniel V.
201500152

Clausen, Ole
20115758

Konstmann, Mia
201500157

Kloock, Michael Møller
201370537

Høj, Christoffer Søby
201407641

Vejleder:
Poul Ejnar Røvsing

Indhold

1	Forord	5
2	Termliste	7
3	Indledning	8
3.1	Koncept	8
3.2	Systembeskrivelse	9
4	Kravs specifikation	10
4.1	Aktørbeskrivelse	10
4.2	Epics	10
4.3	Userstories	11
4.4	Ikke funktionelle krav	11
4.5	Accepttestspecifikation	11
5	Projektafgrænsning	12
6	Metode	13
6.1	UML	13
6.1.1	Pakkediagram	13
6.1.2	Sekvensdiagram	13
6.1.3	Tilstandsmaskinediagram	13
6.1.4	Klassediagram	13
6.2	Scrum	13
6.2.1	Roller	14
6.2.2	Kommunikation	14
6.2.3	Sprint planlægning	14
6.3	GitHub	16
7	Analyse	17
7.1	Arkitektur	17
7.1.1	Desktop applikation	17
7.1.2	Central server	17
7.1.3	Database	18
7.1.4	Web applikation	18
7.1.5	Programmeringssprog	18
7.1.6	HTTP	19
7.1.7	Bootstrap	19
7.1.8	jQuery	19
7.2	Test	19
7.2.1	NUnit	19
7.2.2	NSubstitute	19
8	Systemarkitektur	20
8.1	Domænemodel	20
8.2	Database	21
8.2.1	Brugerdatabase	21
8.2.2	Butiksdatabase	21
8.3	Database API	22
8.4	Brugergrænseflade arkitektur	23
8.4.1	Desktop applikation	24
8.4.2	Web applikation	25
9	Design & Implementering	26
9.1	Butiksdatabase	26

9.2	Database API	26
9.3	Desktop applikation	29
9.4	Web applikation	30
10	Test	33
10.1	Unit test	33
10.1.1	Database API	33
10.1.2	Desktop applikation	33
10.1.3	Web applikation	34
10.2	Integrationstest	34
11	Resultater	36
12	Fremtidig arbejde	37
12.1	Ikke implementeret funktionaliteter	37
12.2	Brugergrænseflader	37
12.3	Plantegninger	37
12.4	Integrering med eksisterende applikationer	37
13	Konklusion	38
14	Bilagsliste	39

Abstract

This paper describes the progression of a project for a group of students during their fourth semester. The endproduct for this development cycle is Locatiles, which consists of two major components; a webapplication for store customers and a desktop application for administrators. After using the MoSCoW method to prioritize system functionalities the final product will include a store database, a webapplication, a desktop application and a database API.

Locatiles has been developed for customers to satisfy their need for locating items in a store. The customers can access the webapplication from their smartphones, and use it to locate items in the store.

The desktop application's appearance and functionality has been created by using WPF. From the desktop application it is possible for the store administrator to administrate item, their placements and the floorplan which the items are placed on. Changes made by the store administrator will then be visible for the store customers on the webapplication, which has been developed using ASP.NET.

The store database is common datasource for the desktop application and the web application. The applications utilize the database API to access the store database, which is hosted the central server. The central server also hosts the webapplication which can be found at <http://locatiles.azurewebsites.net>.

Resume

Denne rapport beskriver udviklingsforløbet af et semesterprojekt for 4. semester studerende på IKT linjen, hvor Locatiles er slutproduktet. Produktet består af to hoveddele; en web applikation til butikskunder og en desktop applikation til administratorer. Ved brug af MoSCoW metoden til prioritering af systemets funktionaliteter vil det endelige produkt omfatte en butiksdatabase, en web applikation, en desktop applikation og en database API.

Locatiles er udviklet til at være en løsning for kunder, der ikke kan lokalisere varer i en butik. De kan via deres smartphones gøre brug af produktet til at finde eftersøgte varer og få dem vist på butikkens plantegning.

Desktop applikationens udseende og funktionalitet er udviklet ved brug af WPF. Fra desktop applikationen kan butiksadministratorer administrere varer, deres placeringer samt den plantegning som varene placeres på. De ændringer som butiksadministratoren foretager vil herefter blive synlige for butikskunder på web applikationen, som er udviklet ved brug af ASP.NET.

Bindeleddet mellem desktop applikationen og web applikationen er en central server, hvor der med en fælles database API kan tilgås data. Denne centrale server står også for at hoste web applikationen og butikkens database. Web applikationen kan tilgås via <http://locatiles.azurewebsites.net>.

1 Forord

Denne rapport beskriver et udviklingsforløb, som resulterer i slutproduktet Locatiles. Locatiles er en varesøgnings løsning, som er udviklet i forbindelse med 4. semester projektet på IKT linjen Ingeniørhøjskolen Aarhus Universitet.

Rapporten beskriver visionen for produktet samt dets afgrænsning, så det passer til tiden afsat til et 4. semester projekt. Efterfølgende præsenteres resultaterne af udviklingsforløbet, hvori der indgår en butiksdatabase samt web- og desktop applikationer.

Gruppen som har deltaget i udviklingen af Locatiles er følgende personer: Daniel Jensen, Mia Konstmann, Christoffer Høj, Kasper Rieder, Michael Kloock og Ole Clausen. På tabel 1 ses de ansvarsområder de nævnte medlemmer har haft under projektføreløbet.

På tabellen angives et primært ansvar for en del af produktet med et *P* og et *S* angiver et sekundært ansvar.

Ansvarsområder	Daniel V. J.	Mia P. K.	Christoffer S. H.	Kasper R.	Michael M. K.	Ole C.
Database						
Design	P	P	P	P	P	P
Deployment			P			P
Database API						
Design	P	P	S	P		S
Implementering	P	P	P	P		P
Desktop Applikation						
Item View	P					
ItemGroup View	P					
Floorplan View				P	S	
StoreSection View	S	P		S		
Web Applikation						
Implementering	S		P	S		P
Test						
Unit test	P		P	P	P	P
Integrationstest	P	P	P	P	S	P

Tabel 1: Oversigt over ansvarsområder

Ved slutningen af udviklingsforløbet bestod Locatiles af følgende dele:

- En butiksdatabase, som indeholde informationer omkring butikkens varebestand og vare placeringer
- En desktop applikation til administrering af en butiks varebestand, vare placeringer, samt plantegning
- En web applikation til søgning af varer samt fremvisning af deres placeringer på butikkens plantegning
- En database API til tilgang af databasens data, anvendt af både desktop og web applikationen

Læsevejledning

I rapporten dokumenteres udviklingsforløbet på dansk, dog er diagramteksterne skrevet på engelsk for at stemme overens med koden. For oversættelserne henvises der til termlisten afsnit 2 side 7.




I løbet af rapporten samt dokumentationen vil der jævnligt refereres til implementeringen af database API. Med dette menes specifikt “Store Database API” delen af database API.

Aktøren systemadministrator er ikke implementeret i dette projekt, men for at fremme forståelsen for det endelige produkt er systemadministratoren inkluderet tilogmed systemarkitekturen.

2 Termliste



Main Window	Hovedmenuen i desktop applikationen vil i flere diagrammer blive refereret til som "Main Window".
Manage Items	Vinduet hvor varer administreres bliver igennem rapporten referet til som "Manage Items" eller "Administrer varer"
Create Item	Vinduet i desktop applikationen, hvor varer oprettes bliver i rapporten refereret til enten som "Create item" eller "Opret varer".
Manage Storesections	Vinduet hvor sektioner administreres, bliver igennem rapporten refereret til som "Manage Storesections" eller "Administrer sektioner".
Edit Storesection	Vinduet hvor sektioner redigeres, bliver igennem rapporten refereret til som "Edit storesection" eller "Rediger sektion".
Create storesection	Vinduet hvor sektioner oprettes, bliver igennem rapporten refereret til som "Create Storesection" eller "Opret sektioner".
Manage Itemgroups	Vinduet hvor varegrupper administreres, bliver igennem rapporten refereret til som "Manage Itemgroups" eller "Administrer varegrupper".
Create Itemgroup	Vinduet hvor varegrupper oprettes, bliver igennem rapporten refereret til som "Create itemgroup" eller "Opret varegrupper".
Edit Itemgroup	Vinduet hvor varegrupper redigeres, bliver igennem rapporten refereret til som "Edit Itemgroup" eller "Rediger varegruppe"
Manage floorplan	Vinduet hvor plantegninger administreres, bliver igennem rapporten referet til som "Manage floorplan" eller "Administrer plantegninger"
Plantegning	En plantegning er et billede af butikkens indretning, hvor reoler og gangarealer ses oppefra. Oversættelse: Floorplan
Sektion	En sektion er et punkt på plantegningen hvorpå der kan placeres et vilkårligt antal varer. Oversættelse: Store Section
Vareplacering	Lokationen af den sektion, som varer er tildelt
Reol	En reol er en fysisk reol som varer står på i en butik
Varegruppe	En varegruppe er en samling af varer der hører under samme kategori, f.eks. mejeri. Oversættelse Item Group
Over varegruppe	En varegruppe kan tilhøre en over varegruppe, f.eks. kan varegruppen "ost" tilhøre over varegruppen "mejeri". Oversættelse: Parent Item Group
Admin	Admin er en forkortelse for administrator. Igennem rapporten begge begreber bruges omskifteligt
Epics	Ordet dækker over en samling af relaterede userstories for systemets funktionalitet
CRUD	CRUD er et udtryk for operationerne "Create", "Read", "Update" og "Delete"
API	API er et udtryk for "Application Programming Interface", hvilket er et software modul grænseflade som en klient kan gøre brug af for at tilgå modulets funktionaliteter
Bruger Database	Bruger databasen indeholder bruger oplysninger som tilknytter brugere af systemet til specifikke butiksdata-baser
DbAPI	En term for "database API"

3 Indledning

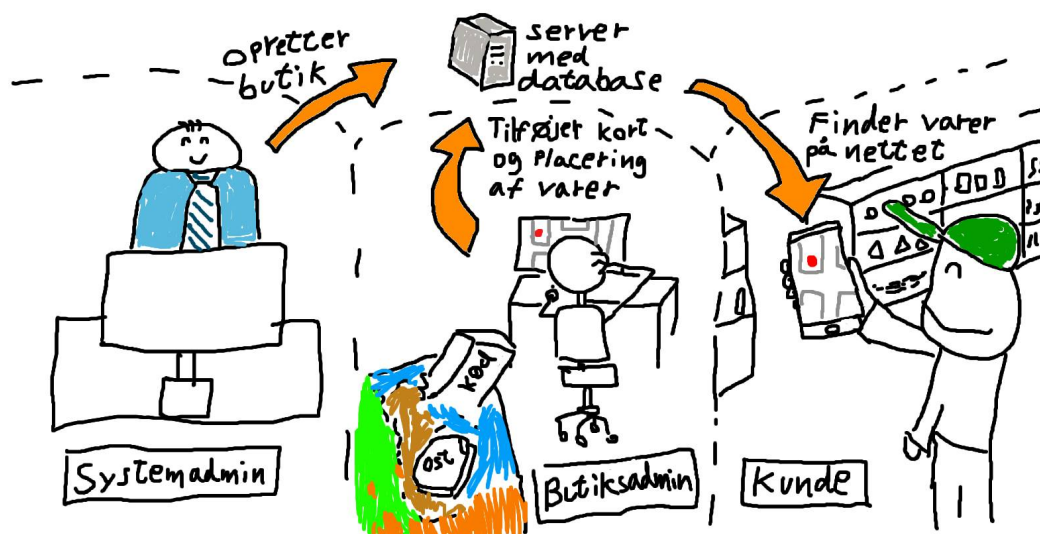
Denne projektrapport omhandler udviklingen af produktet Locatiles. Produktet er inspireret af det scenarie, hvor man som kunde i en forretning ikke kan finde en ønsket vare. Her er kundens eneste muligheder at finde en medarbejder  at spørge om hjælp, eller undvære den ønskede vare. For at afhjælpe dette problem udvikles Locatiles, et produkt, hvor kunder via en applikation får mulighed for at søge efter en vares placering i en given forretning, for at få varen vist visuelt på butikkens plantegning.

Produktet består af to grafiske brugergrænseflader: en web applikation som butikskunden bruger til at lokalisere varer og en desktop applikation som butiksadministratoren bruger til at administrere varerne og deres placering i butikken. Derudover skal desktop applikationen bruges af en systemadministrator til at oprette nye butikker og butiksadministratører.

Det endelige mål med produktet er at gøre indkøbsoplevelsen bedre for butikskunden og spare butikmedarbejdernes tid, idét de ikke længere skal vise kunder rundt.

Locatiles er let inspireret af eksisterende lokationstjenester såsom Google Maps [1] eller Ikea's eget varelokationssystem. De eneste reelle konkurrenter til Locatiles  butikkernes egne applikationer, som kun indeholder tjenester til oprettelse af indløbslister og fremvisning af tilbudsaviser. Det kunne dog være i butikkernes interesse, at gøre brug af Locatiles til at forbedre deres kunders indkøbsoplevelse. 

3.1 Koncept



Figur 1: Koncepttegning af Locatiles

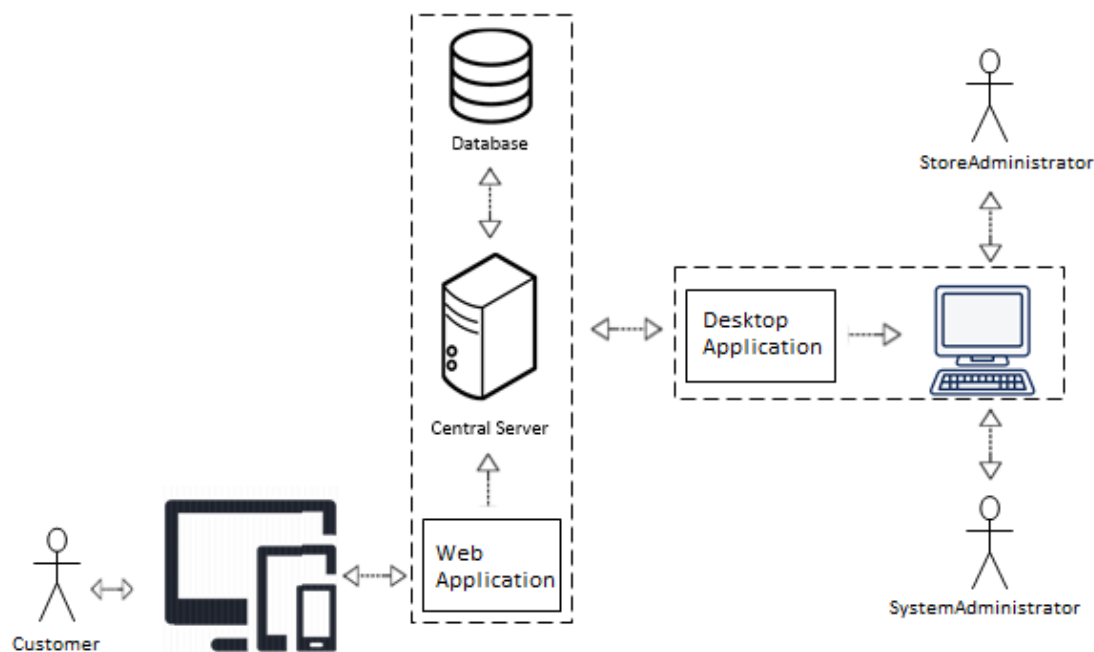
På figur 1 ses en oversigt af systemets brugere og hvordan de anvender systemet.

En systemadministrator opretter en butiksdatabase til butikken samt login for en butiksadministrator på en server. Butiksadministratoren indsætter en plantegning af butikken og sektioner på plantegningen. Disse sektioner kan for eksempel repræsentere en reol eller en køler. Sektioner kan tildeles varer som derved får en lokation i butikken. Butiksadministratoren har også mulighed for at tildele varegrupper til varer. Derefter har kunden mulighed for at søge efter en bestemt vare, ved brug af en web applikation, som tilgås via internettet. Hvis varen eksisterer og er tildelt en sektion, får kunden varens placering vist som et punkt afmærket på butikkens plantegning.

3.2 Systembeskrivelse

Herunder beskrives systemet samt typiske brugerscenarier hvori produktet indgår.

Figur 2 viser de centrale dele som indgår i systemet. Den stiplede afmærkning omkring den centrale server angiver at databasen og web applikationen bliver hosted på denne. På PC'en hostes desktop applikationen som også angives med en stiplede afmærkning. For at web og desktop applikationen kan tilgå databasens data udvikles der et Application Programming Interface (API), som beskrives nærmere i afsnit 8.3, s. 22.



Figur 2: Illustration af systemets centrale dele

Et typisk brugerscenarie for kunden, ville være at kunden står i forretningen og leder efter en vare. Kunden kan ikke finde varen og vælger derfor at gøre brug af Locatiles web applikation via egen smartphone eller en af butikkens standere.

I web applikationen anvender kunden søgefeltet ved at indtaste varens navn. Herefter vises en liste af varer som indeholder søgeteksten, hvor kunden vælger den vare som søges efter, hvorefter dennes placering vises på plantegningen. Kunden kan nu finde varen ud fra markeringen på plantegningen.

Et typisk brugerscenarie for butiksadministratoren er når der skal oprettes en ny plantegning for butikken efter en omrokering af samtlige af butikkens varer.

Butiksadministratoren logger ind på systemet, vælger administrer plantegning og indsætter en ny plantegning. Derefter placerer butiksadministratoren nye sektioner på plantegningen. Herefter kan varer tilføjes til de ny oprettede sektioner. Butiksadministratoren gemmer de nye ændringer så disse kan ses af kunden.

Et typisk brugerscenarie for systemadministratoren er når en ny klient køber produktet. Systemadministratoren opretter en ny butik i databasen samt en ny butiksadministrator.

Systemadministratoren logger ind i systemet, vælger at oprette en butik i menuen og herefter at oprette en butiksadministrator. Systemadministratoren informerer klienten om de nye login detaljer.

4 Kravspecifikation

Ud fra systembeskrivelsen er der formuleret en række krav til produktet. Disse er formuleret som userstories og et antal ikke funktionelle krav. Følgende afsnit beskriver de aktører som interagerer med systemet, de opstillede krav til produktet samt en accepttestspecifikation, som specificerer den ønskede funktionalitet til når et krav er opfyldt.

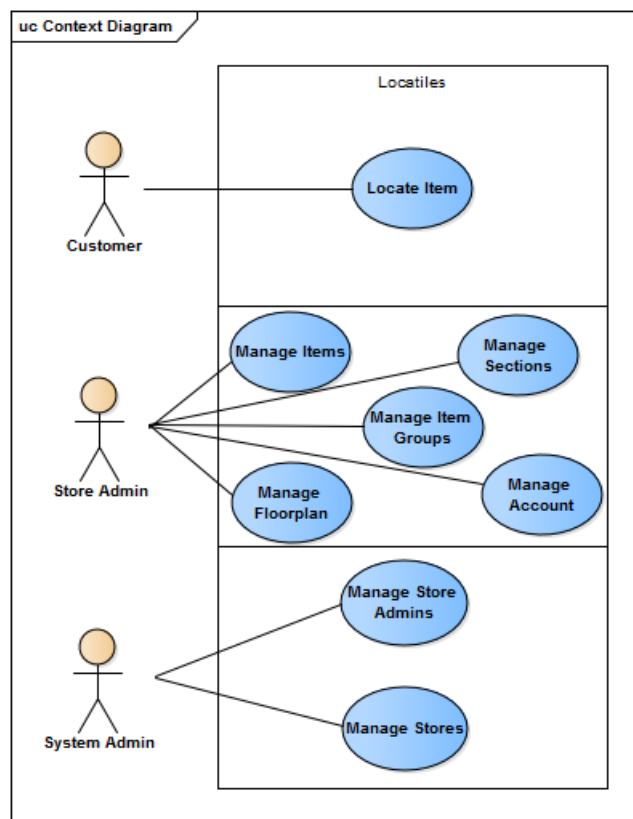
4.1 Aktørbeskrivelse

Følgende er en beskrivelse af de aktører der interagerer med systemet.

Kunde	Denne aktør vil interagere med systemet igennem en web applikation, for at finde en vare i butikken
Butiksadministrator	Denne aktør vil tilgå butikkens database, for at vedligeholde butikkens information, såsom vareliste, varegrupper, plantegninger og sektioner
Systemadministrator	Denne aktør vil oprette butikkernes databaser, samt vedligeholde butik administratorernes login informationer.

4.2 Epics

Til beskrivelse af systemets ønskede funktionaliteter er der blevet opstillet en række epics[2] som tager udgangspunkt i hvad de tre brugere ønsker af funktionalitet. I dette afsnit er disse epics opstillet i et kontekst diagram, figur 3, med en tilhørende beskrivelse. De følgende beskrivelser beskriver kun de epics, som implementeres under dette projektförløb. For en fyldestgørende beskrivelse af samtlige epics henvises der til dokumentationen afsnit 2.2, s. 7.



Figur 3: Kontekst diagram over Locatiles' epics

Locate Item	Som kunde vil jeg kunne søge efter en vares placering i butikken.
Manage Items	Som butikksadministrator vil jeg kunne administrere (tilføje, fjerne og redigere) varer i butikkens databasen.
Manage Item Groups	Som butikksadministrator vil jeg kunne tildele varegrupper til varer for at kategorisere disse.
Manage Floorplan	Som butikksadministrator vil jeg kunne tilføje og opdatere plantegninger for at kunne placere varer.
Manage Sections	Som butikksadministrator vil jeg kunne tilføje og fjerne varer fra sektioner for at give en vare en lokation.

Epics som værktøj gav gruppen meget værdi i opstartsfasen af projektet. I tidligere semesterprojekter havde gruppens medlemmer haft primær erfaring med brugen af use cases med tilhørende fully dressed use cases. På grund af typen af projekter under uddannelsen, hvor gruppemedlemmer altid har været i tæt kontakt, er det erfaret at fully dressed use cases ikke har givet meget værdi til projektet, fordi de ofte er endt med at blive alt for tekniske eller formelle i deres beskrivelse alt for tidligt i projektets forløb. Derfor valgte gruppen til udvikling af Locatiles at gøre brug af epics samt user stories, da disse værktøjer fokuserede mere på at identificere produktets funktionalitet uden detaljerede beskrivelser af hvordan funktionaliteten burde implementeres. Dette gav et godt og effektivt resultat, da energien i gruppen blev brugt på at opnå en fælles forståelse for hvad produktet skulle kunne.



4.3 Userstories

De beskrevne epics fra forrige afsnit er blevet delt ud i user stories, som mere specifikt beskriver de funktionaliteter der ønskes af det endelige system. Disse user stories er kategoriseret efter hvilke aktører de relaterer til og er prioriteret efter MoSCoW princippet [3] i projektafgrænsningen, afsnit 5 s. 12. For at se samtlige userstories for systemet henvises der til dokumentationen afsnit 2.3, s. 8.

4.4 Ikke funktionelle krav

Til at beskrive produktet yderligere er der udarbejdet ikke funktionelle krav til systemet. Disse stiller krav til det styresystem som desktop applikationen skal køres på samt plantegningens filtype. De ikke funktionelle krav findes i dokumentationen afsnit 2.5, s. 11.

4.5 Accepttestspecifikation

Ud fra de specificerede userstories er der udarbejdet en accepttestspecifikation for at verificere systemets funktionaliteter. Ved enden af udviklingsforløbet blev systemet holdt op imod accepttesten. Alle opstillede accepttests kunne gennemføres og forløb som forventet. For en udfyldt accepttestspecifikation henvises der til dokumentationen afsnit 6, s. 61.

5 Projektafgrænsning

Ud fra de specificerede userstories i dokumentationen afsnit 2.3, s. 8. er der blevet foretaget en MoSCoW analyse. Denne analyse prioriterer de definerede userstories, som skal opfyldes til at kunne implementere det endelige produkt. I følgende afsnit præsenteres de userstories, som dette projekt forløb omfatter. For en fyldstgørende analyse af samtlige user stories henvises der til dokumentationen afsnit 2.6, s. 12.

Must have:

- Som butiksadministrator vil jeg kunne opdatere plantegningen til butikken, for at placere sektioner på den
- Som butiksadministrator vil jeg kunne tilføje, redigere samt fjerne sektioner på butikkens plantegning, så jeg kan bestemme hvor varer kan placeres
- Som butiksadministrator vil jeg kunne oprette og slette varer i butikkens database, for at vedligeholde kundens søgedatabase
- Som butiksadministrator vil jeg kunne tilføje og fjerne varer fra en sektion, for at ændre lokationen på en vare i butikken
- Som kunde vil jeg kunne søge efter en af butikkens varer, for at kunne se varens placering

Should have:

- Som butiksadministrator vil jeg kunne oprette, redigere og slette varegrupper for at kategorisere butikkens varer

Punkterne foroven viser de højst prioriterede user stories som er udvalgt til dette projektforløb. Under udviklingen af produktet vil der blive implementeret funktionaliteter som sørger for at de udvalgte userstories er realiseret.

6 Metode

Under projektforløbet er der anvendt en række arbejdsmetoder til at gennemføre projektet. Disse metoder er anvendt til at dokumentere de design valg, som under projektforløbet er blevet taget, og fremme udviklingen af det færdige produkt. Følgende underafsnit beskriver de metoder som gruppen har gjort brug af til gennemførelse af dette projekt.

6.1 UML

Til dokumentering af Locatiles' software er modelleringssproget UML [4] blevet brugt. Valget faldt på UML da det er et anerkendt værktøj brugt i industrien. Dette medfører at dokumentationen er læsbar for andre ingeniører, der har kendskab til UML, og dermed kan få en forståelse af systemet. Da modelleringssproget er universielt og er blevet introduceret i undervisningen er de følgende diagramtyper blevet anvendt til at beskrive systemet.

6.1.1 Pakkediagram

Til at beskrive systemets arkitektur, design og implementering er typen pakkediagram blevet anvendt. Pakkediagrammer bruges til at identificere moduler, med en ønsket funktionalitet, som senere bliver udpenslet til en eller flere specifikke og konkrete klasser.

6.1.2 Sekvensdiagram

For at beskrive interaktionerne mellem modulerne som udgør systemet er der anvendt sekvensdiagrammer. Med disse diagrammer bliver specifikke forløb, som for eksempel hvordan StoreDatabaseService opretter dets tabel objekter, beskrevet ved hjælp af en sekvens af handlinger over en tidsperiode.



6.1.3 Tilstandsmaskinediagram



Til at beskrive navigeringen mellem vinduer i desktop applikationen er der anvendt tilstandsmaskinediagram. Her repræsenterer de forskellige vinduer hver sin tilstand, som skifter når brugeren interagerer med brugergrænsefladen.

6.1.4 Klassediagram

For at kunne beskrive relationerne mellem systemets klasser er der gjort brug af klassediagrammer. På denne måde er det muligt at få en forståelse for hvilke klasser der indgår i systemet samt deres relationer med hinanden.

6.2 Scrum

Til processtyring af dette projekt er udviklingsmetoden scrum [5] blevet anvendt. Scrum er et agilt og iterativt værktøj til software udvikling, som sætter fokus på kommunikation og planlægning under et projektforløb. Gruppens medlemmer har erfaring med denne procesmetode fra tidligere semesterprojekter, hvor dele af scrum, samt hvordan scrum er blevet implementeret, har resulteret i en hæmmet og ineffektiv proces. Navnlige de daglige stå-op møder, manglende faciliteter og avancerede planlægningsprogrammer, såsom Pivotal Tracker [6], bidrog til at hæmme processen og introducerede meget spildtid. Ud fra disse erfaringer, har gruppen forsøgt at gøre bedre brug af scrum, ved at tilpasse implementeringen af scrum til at tage højde for disse udfordringer. Eksempelvis er det avancerede program Pivotal Tracker skiftet ud med et simple





planlægningsprogram, og i stedet for at holde daglige stå-op møder, har gruppen opsat en Facebook samtale, så fysisk tilstedeværelse ikke længere var nødvendigt. Følgende afsnit vil beskrive gruppens brug af hjælpemidler fra scrum, der er blevet brugt igennem udviklingsforløbet.



6.2.1 Roller

I traditionel scrum er der tre roller et individ kan påtage sig; medlem i udviklingsholdet, scrum master eller product owner. Under forløbet har alle gruppens medlemmer indgået som en del af udviklings holdet samt fungeret som product owner. Gruppen, som helhed, har selv sat de krav der skulle stilles til produktet, reviewet produktet ved sprint afslutning samt planlægge de forløbne sprints. Ved hver sprint opstart er der blevet udpeget en ny scrum master til at sørge for at bevare processen samt holde udviklingsholdet opdateret omkring projektets status.

Sprint nr.	Scrum master
1	Daniel Jensen
2	Ole Clausen
3	Kasper Rieder
4	Mia Konstmann

Tabel 2: Oversigt af scrum masters

Tabel 2 viser en oversigt af hvem der har fungeret som scrum master under de pågældende sprints.

6.2.2 Kommunikation

Gruppens primære kommunikationsforum har bestået af en gruppe chat på Facebook og en Facebookgruppe. Gruppe chatten blev valgt fremfor fysiske stå op møder da alle kunne deltage i samtalen på trods af fysisk tilstedeværelse. På denne måde kunne gruppens medlemmer holde sig opdateret omkring projektets status på en måde der var let tilgængeligt. Projektets Facebook-gruppe blev brugt til at aftale arbejdsdage, notificering omkring vigtige ændringer i projektet og generel information omkring projektets forløb. Ved at anvende en platform, der i forvejen var integreret i gruppemedlemmernes hverdag, blev det muligt at holde kontakt til hinanden og opretholde god kommunikation.

6.2.3 Sprint planlægning

Det vigtigste element af scrum som gruppen har implementeret i processen er sprint planlægningen. Til at administrere sprintplanerne er projektsstyrings applikationen Trello [7] blevet anvendt. Dette er et web baseret værktøj som kan tilgås via en browser eller en app. På denne måde var sprintbackloggen lettilgængelig for alle medlemmer på alle platforme. Trello bruger virtuelle boards hvorpå man kan vedhæfte notes. Dette minder meget om et traditionelt scrumboard og applikationen var let at sætte sig ind i. Et eksempel på et sprint board fra projektet ses på figur 4.



Figur 4: Sprint board på Trello fra projektet


For hvert sprintopstart blev der oprettet et nyt board. Herpå blev målet for sprintet defineret ved hjælp af udvalgte userstories. Ud fra disse user stories blev der defineret en række opgaver, der skulle udføres, for at sprintet kunne betragtes som succesfuldt. På denne måde har der været en struktureret tilgang til hvordan møderne skulle forløbe. Dette har resulteret i at gruppen har kunne effektivt oprette et passende antal opgaver til hvert sprint, som medførte at målene for sprintene er blevet gennemført.

På figur 4 kan det yderligere ses at hver arbejdsopgave er blevet tildelt en af følgende farver: grøn, gul, eller rød.


Disse farver relaterer sig til tidsestimering af hver enkelt opgave. Gruppen har fra tidligere semesterprojekter erfaret at tidsestimering i præcise timeantal sjældent har givet gode praktiske resultater. Typisk blev der i disse tilfælde brugt for langt tid på at give præcist tidsestimering, hvor disse endte med at være langt fra det sande tidsforbrug. Derudover kan faktiske timeantal også være svære at forholde sig til, hvis der ikke er forhåndskendskab til opgavernes indhold.

På grund af disse erfaringer valgte gruppen at gøre brug af et simpelt pointsystem i form af de tre førnævnte farver. Hver farve refererer til et ca. antal af timer:

- Grøn: under 4 timer
- Gul: mellem 4 og 12 timer
- Rød: Over 12 timer

Ved at gøre brug af dette system fandt gruppen ud af at processen af tidsestimering blev mere effektivt gennemført, da det var nemmere at forholde sig til  valg. Det var desuden også nemmere at forholde sig til om en opgave lænte mere imod 4 timer end 8 timer. Dette system fungerede godt for gruppen igennem udviklingsforløbet. Det gav gode estimeringer og arbejdsopgaverne for alle sprints nåede at blive gennemført.



I starten af udviklingsforløbet lavede gruppen en sprintplan,  hvor tiden op til projektets deadline blev opdelt i flere forskellige sprints. Hvert sprint fik tildelt et primært mål. På denne måde blev det under udviklingsforløbet relativt nemt at se hvor godt projektet skred frem, da man altid kunne holde dets nuværende status op imod de mål der gerne skulle have været nået for hvert sprint.

Sprint nr.	Varighed	Mål
1	3 uger	Design og deployment af database samt første udkast til database API
2	3 uger	Design og implementering af desktop applikation samt påbegyndelse af test
3	3 uger	Færdiggørelse af alt software, herunder web og desktop applikation samt test
4	3 uger	Færdiggørelse af dokumentation og rapport

Tabel 3: Oversigt over sprint mål

Tabel 3 viser en oversigt af målene for hvert sprint samt deres varighed.

6.3 GitHub

Til projektet var det vigtigt at have samlet alle projektets filer og dokumenter online og centralt, således at alle gruppemedlemmer kunne tilgå, og redigere i filerne. Det var også af stor betydning, at der blev holdt en automatisk versionshistorik for alle filer, så alle ændringer blev dokumenterede, samt at det også blev muligt at vende tilbage til tidligere versioner hvis noget skulle gå galt.

Til dette brugte vi versionsstyrings softwaret Git, via GitHub [8, 9]. GitHub er en webbaseret løsning, der stiller en gratis Gitserver til rådighed, samt en brugergrænseflade for at kunne tilgå oprettede git repositories. Andre alternativer såsom Dropbox er blevet valgt fra grundet Dropbox' basale understøttelse af versionsstyring og konflikt håndtering. Under denne overvejelse er Teamfoundation også valgt fra, på trods af at denne service understøtter versionsstyring, da gruppen tidligere har stiftet bekendtskab med GitHub og gruppens størrelse forhindrede os i at gøre brug af Teamfoundations gratis services.

7 Analyse

Under udviklingsforløbet af Locatiles er der gjort brug af flere værktøjer og services til realiseringen af produktet. Disse har fremmet udviklingen af produktet og har været afgørende for projektets udviklingsforløb.

7.1 Arkitektur

Det er under arkitekturfasen at mange kritiske beslutninger for et produkt bliver taget, da disse beslutninger udgør produktets fundament og som alle valg senere i udviklingsforløbet bygger på. I sidste ende er disse beslutninger de dyreste at rette op på efter de er blevet taget. Derfor brugte gruppen tid i starten af projektforsløbet på at overveje samt diskutere hvilke teknologier der skulle bruges for at kunne realisere systemet. Følgende afsnit beskriver de teknologier og frameworks der bliver brugt under udviklingen af produktet samt begrundelser for disse valg.

På baggrund af figur 2 i projektets systembeskrivelse side 9, kan følgende centrale elementer identificeres:



- Desktop Applikation
- Central Server
- Database
- Web Applikation



Desktop applikationen og web applikationen udvikles ved brug af .NET frameworket [10] med hoveddelen af udviklingen foregående i Visual Studio. Dette framework tilbyder mange fordele til projektet, som vil blive yderligere specificeret i dette afsnit.

7.1.1 Desktop applikation

Desktop applikationen er i ikke funktionelle krav, afsnit 4.4, s. 11, specificeret til at skulle kunne afvikles på Windows 10.

På baggrund af dette blev Microsofts WPF [11] teknologi valgt til at designe udseendet og udvikle desktop applikationen i. WPF er en del af .NET frameworket og har nogle stærke fordele for projektet:

- Applikationens brugergrænseflade kan designes visuelt i XAML med yderligere drag and drop funktionalitet direkte inde fra Visual Studio
- Der eksisterer et stort sæt af prædefinerede kontroller som kan gøres brug af uden ekstra programmeringstid
- Der er mulighed for at designe WPF applikationer ved brug af GUI arkitekturmønstre såsom MVVM. Dette gør det muligt at skrive genanvendeligt forretningslogik

Ved at bruge WPF kan der spares meget tid på udvikling af brugergrænsefladen til desktop applikationen, samtidig med at den kan få et professionelt udseende. Dette giver mere tid til at kunne fokusere på udvikling af forretningslogik.

7.1.2 Central server

Til hosting af den centrale server blev en SQL Server [12] hos Microsoft Azure [13] brugt.

Hosting hos Microsoft Azure giver god mening for projektet, da:

- Det ligger indenfor Microsofts økosystem og er derfor velintegreret med Visual Studio, hvor man kan administrere tilknyttede databaser, web applikationer og sin Azure konto

- Microsoft tilbyder Developer Program Benefit, hvilket er et 12 måneders program, hvor der gratis gives 200 kr hver måned til services, såsom hostede databaser eller web applikationer
- Microsoft Azure tilbyder hosting af både SQL servere samt ASP.NET web applikationer, hvilket der skal bruges til projektet
- Microsoft Azure tilbyder en skalérbar service, i det tilfælde at forbruget pludseligt vokser

7.1.3 Database

Til projektet gøres brug af en MS SQL database. En relationel database giver god mulighed for at modellere butik- og brugerdatabaserne.

Til hosting af SQL databasen bruges Microsofts Azure MS SQL database, da det som førnævnt er det muligt at få en gratis SQL database hostet via Developer Program Benefit. Da denne er en del af Microsoft Azure, er det også velintegreret med Visual Studio, hvilket betyder at databasen kan administreres derfra.

Til kommunikation med databasen gøres brug af ADO.NET [14]. ADO.NET er også en del af .NET frameworket og tilbyder en nem måde at kommunikere med datakilder på.

7.1.4 Web applikation

Web applikationen bliver udviklet ved brug af teknologien ASP.NET [15], som også er en del af .NET frameworket.

ASP.NET har gode fordele for projektet, da:

- ASP.NET applikationer kan hostes på Microsoft Azure, hvilket vil sige at det kan betales via førnævnte Developer Program Benefit
- Da ASP.NET også er en del af Microsofts økosystem med Azure kan ASP.NET applikationer deployes fra Visual Studio
- Der er mulighed for at designe ASP.NET applikationer ved brug af GUI arkitekturmønstre såsom MVC, hvilket betyder at det kan deles om en fælles kodebase med WPF desktop applikationen
- ASP.NET kan bruges til at implementere en dynamisk hjemmeside der kan skalere brugergrænsefladen efter opløsningen på brugerens enhed

7.1.5 Programmeringssprog

Til implementering af desktop applikation og web applikationen gøres der brug af C#. Til web applikationen gøres der yderligere brug af JavaScript til front-end funktionalitet.

For projektet gav det god mening at bruge C#, da:

- Det er et garbage collected sprog, hvilket simplificerer memory management i stor grad. Dette minimerer risikoen for hukommelsesfejl i systemet og kan derfor være med til at effektivisere udviklingstiden for at opnå et stabilt produkt
- Det er en integreret del af .NET frameworket. Desktop applikationen og web applikationen gør brug af teknologierne WPF og ASP.NET, hvilket primært gør brug af C#.
- Der kan skrives klassebiblioteker, såsom database API, som let kan deles mellem kodebaserne for desktop applikationen og web applikationen

Overordnet set er det forsøgt at skabe et velintegreret miljø af arbejdsværktøjer som er med til at skabe et effektivt workflow for holdet samtidig med at produktet kan realiseres.

7.1.6 HTTP

Til kommunikation mellem web applikationen hostet på den centrale server og klienter, bruges der bagomliggende HTTP protokol.

7.1.7 Bootstrap

Til udvikling af web applikationens brugergrænseflade bruges Bootstrap [16]. Bootstrap gav nogle gode fordele for projektet, da:

- Det indeholder grid layout funktionalitet, hvilket betyder at projektets web applikation kan skalere ordenligt mellem mobile enheder og større skærme
- Det indeholder prædefinerede kontroller, så der spares tid på at opsætte brugergrænsefladen

7.1.8 jQuery

Til udvikling af web applikationen bruges jQuery [17]. jQuery har mange funktionaliteter, dog blev det i projektet primært brugt til at lette JavaScript programmering af brugergrænsefladen, da man med jQuery har mulighed for at tilgå web applikationens Document Object Model (DOM) på en mere effektiv måde.

7.2 Test

Under kvalitetssikringen og test af produktet er der anvendt frameworks til at automatisere unit- og integrationstest. Følgende afsnit beskriver de to frameworks der er blevet brugt under projektforsøget til at teste projektets kode. Disse blev valgt på baggrund af, at de understøtter test af sourcekode skrevet på .NET platformen samt, at de er blevet introduceret i kurset I4SWT.

7.2.1 NUnit

Til opsætning af tests er der blevet gjort brug af NUnit frameworket [18]. NUnit gør det nemt at implementere automatiserede unit tests i C#. Yderligere indeholder det også en test runner, så alle tests i en solution kan blive eksekveret og resultaterne vist.

7.2.2 NSubstitute

For at kunne verificere adfærd og tilstande efter en bestemt handling var foretaget, er der blevet gjort brug af NSubstitute [19]. Med dette isolationsframework er det muligt at se på et objekts tilstand efter en handling og verificere resultatet. Frameworket gjorde det muligt at fjerne eksterne afhængigheder under testene for at sikre at testene blev udført under en kendt tilstand.

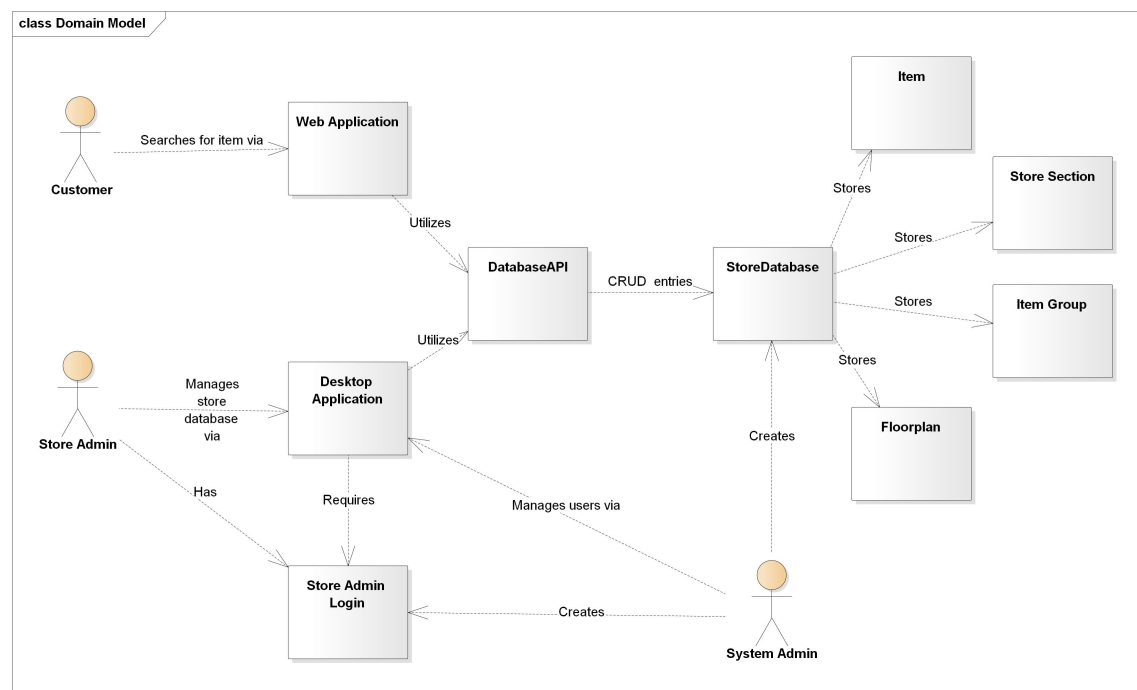
NSubstitute blev brugt til unit tests og integrationstests i løbet af projektet, for at have fuldt kontrol over “unit under test” og “integration under test.”

8 Systemarkitektur

På baggrund af kravsspecifikationen og analysen er der udarbejdet en arkitektur der muliggør forståelsen for systemets komponenter og deres sammenhæng. De fastsatte krav og valgte teknologier danner grundlag for systemets arkitektur.

8.1 Domænemodel

Domænemodelen, på figur 5, viser aktørernes interaktioner med systemet, interaktioner mellem elementerne i systemet samt systemets grænseflader.



Figur 5: Domænemodel for Locatiles

Følgende er beskrivelser til domænemodellens aktører og elementer:

System Admin	Systemadministratoren står for at oprette en butiksadministrators login samt oprette tilhørende butiksdatabase.
Store Admin	Butiksadministratoren kan administrere varer, varegrupper, plantegninger og sektioner på butikkens database.
Customer	En kunde kan via web applikationen søge efter varer med henblik på at få vist vares placering i butikken.
Web Application	Web applikationen bruges af kunden til at søge efter en vares placering i butikken.
Desktop Application	Desktop applikationen bruges af systemadministratoren og butiksadministratoren. Systemadministratoren bruger applikationen til at oprette brugere til systemet. Butiksadministratoren bruger applikationen til at administrere varer, varegrupper, plantegninger og sektioner.
Admin Login	En brugerkonto for butiksadministratorer lagret på brugerdatabase.

Database API	En API der bliver brugt af web applikationen og desktop applikationen til at tilgå databasen. API'en afskærmer databasen fra klienten, for at opnå en lavere kobling mellem klientens kode og potentielle datakilder.
Store Database	Databasen hvor i alt data om varer, sektioner, varegrupper og plantegninger, tilknyttet en enkelt butik, er gemt.
Item	Data gemt i databasen, indeholder oplysninger om en enkelt vare i butikken.
Store Section	Data gemt i databasen. En sektion er et punkt på plantegningen som kan tildeles varer. En sektion repræsenterer en fysisk reol, køledisk eller vare-ø.
Item Group	Data gemt i databasen, en varegruppe er en inddeling af lignende varer under et overordnet navn.
Floorplan	Data gemt i databasen, indeholder en plantegning for butikken, hvorpå sektioner kan placeres.

Gruppens medlemmer har fra tidligere semesterprojekter erfaret at der under opstartsfasen kan opstå mange diskussioner relateret til problemområdet, da gruppemedlemmer nemt kan have forskellige forståelser for de samme koncepter. Derfor er det yderst vigtigt at der dannes en fælles forståelse før udviklingen af produktet starter. Domænemodellen var et resultat af dette, og var et meget brugtbart værktøj. Da domænemodellen blev udviklet i fællesskab, var den med til at afklare vigtige misforståelser tidligt i projekts forløb. Under udviklingsforløbet kunne domænemodellen yderligere bruges som reference.

8.2 Database

Til systemet er det tiltænkt at der skal gøres brug af to typer af databaser; brugerdatabase og butiksdatabase. Følgende afsnit beskriver deres formål samt opbygning.

8.2.1 Brugerdatabase



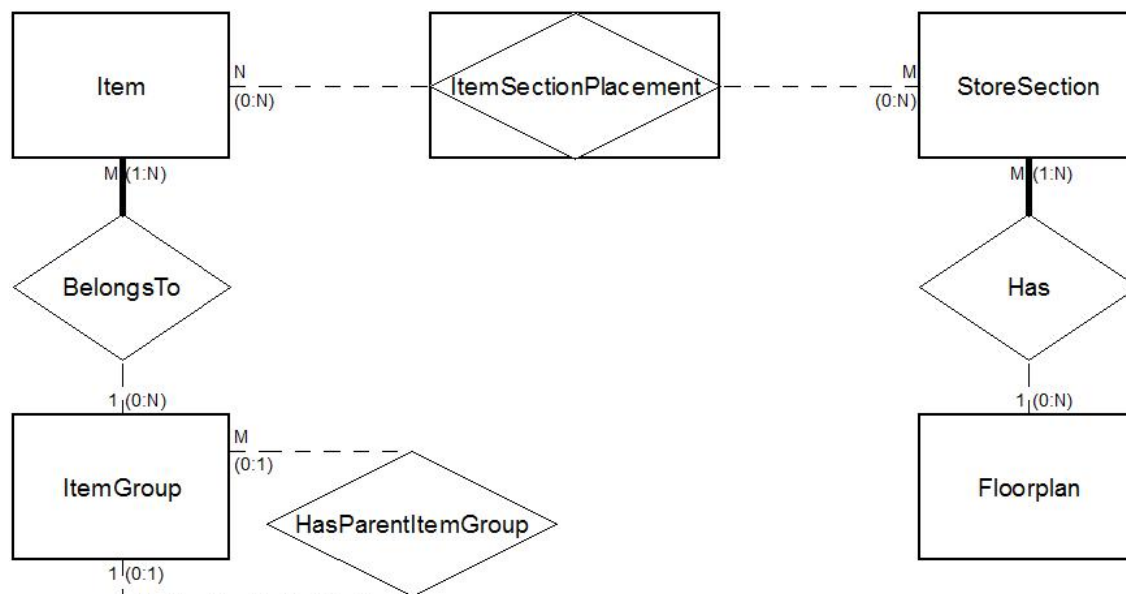
Brugerdatabase er til projektets prototype ikke implementeret. Databases design samt brug er dog fremtænkt. For yderligere beskrivelse af dette, henvises der til dokumentationen afsnit 3.4.1, s. 21.

8.2.2 Butiksdatabase



På figur 6 ses **et diagram der beskriver** arkitekturen for butiksdatabase. Diagrammet indeholder information om relationerne mellem entiteterne i databasen. For en mere detaljeret beskrivelse af entiteterne samt reglerne mellem deres relationer henvises til dokumentationen afsnit 3.4.2, s. 22.





Figur 6: ERD for butiksdatabasen

Hver butik har dets egen instans af denne butiksdatabase. Her kan det ses at en butiksdatabase består af:

- **Item**: Dette er en tabel af butikkens varer
- **ItemGroup**: Dette er en tabel af butikkens varegrupper
- **StoreSection**: Dette er en tabel af butikkens sektioner placeret på en plantegning
- **Floorplan**: Dette er en tabel af butikkens plantegning
- **ItemSectionPlacement**: Dette er en tabel over hvilke varer en sektion har fået tildelt

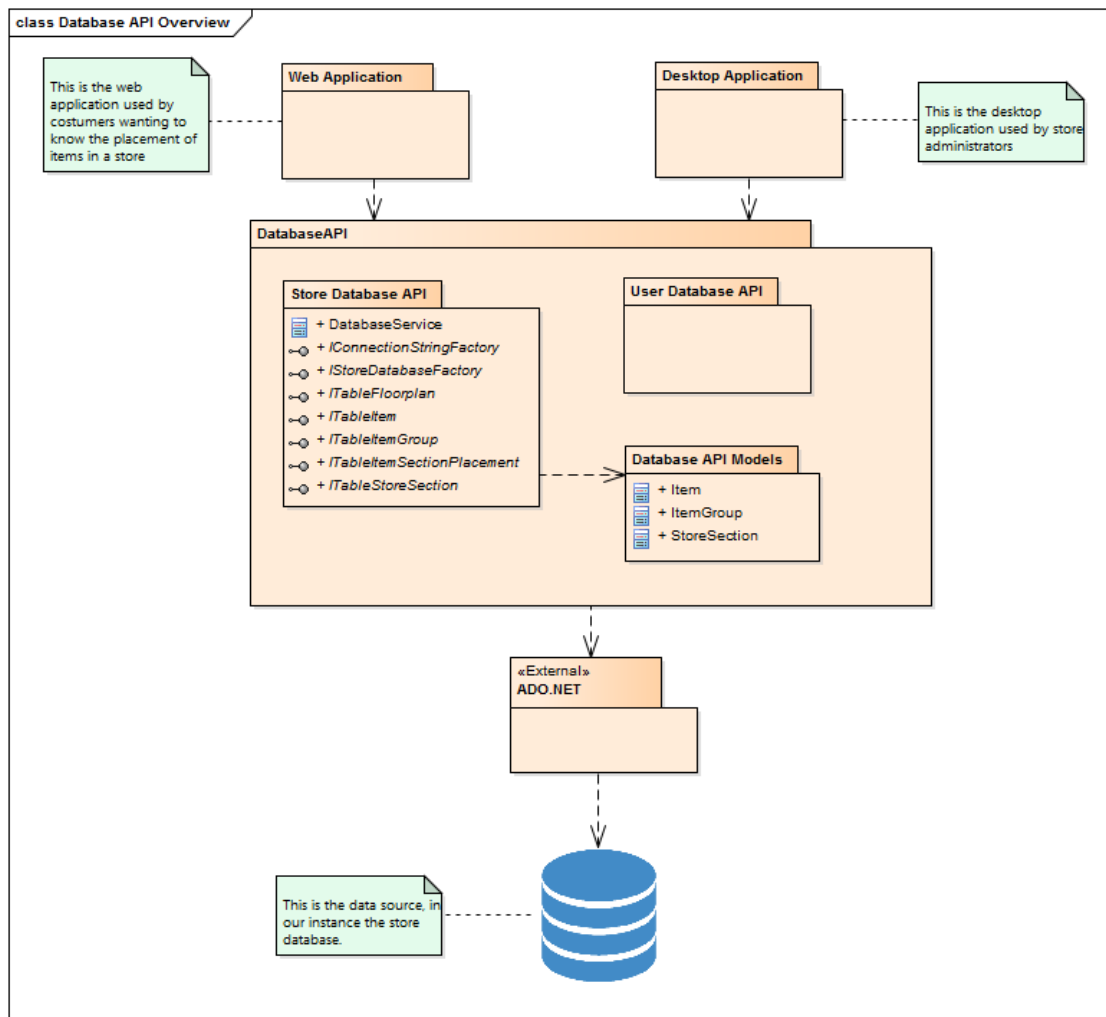
Butiksdatabasen er udtænkt ved brugen af konceptuel design [20]. Her blev problemdomænet, en butik, analyseret ved at identificere entiteter relevante for systemet - såsom varer og varegrupper. Disse entiteter samt deres relationer til hinanden blev derefter oprettet som tilsvarende tabeller i en relationel database.

Fordelen ved at have udviklet databasen ud fra fremgangsmåden konceptuel design er, at databasen har større chance for at overholde normaliseringsformerne N0 til N4 fra starten. Dette var et fordelagtigt værktøj for systemet, da det på denne måde er med til at sikre en database som ikke har problemer med redundans og dermed også konsistent lagring af en butiks data.

8.3 Database API

Database API er i systemet et mellemlag mellem de konkrete datakilder, såsom butikkens database, og de forretningslogik lag som eksisterer i systemet. Det konkrete design samt designvalg af Database API kan læses om i dokumentationen afsnit 4.2.1, s. 29.

På figur 7 er det illustreret at Database API underlæggende gør brug af ADO.NET til tilgang af databaser.



Figur 7: Overblik af Database API

På figuren ses det at Database API er et modul som består af flere delmoduler; Store Database API, User Database API og Database API Models. Disse delmoduler har det til fælles at de bruges af øvre lag til at tilgå konkrete datakilder.

Dette mellemlag blev introduceret som et Data Access Layer, for at fjerne koblingen mellem den øvre forretningslogik og konkrete datakilder. Ved at fjerne denne kobling, bliver forretningslogik ikke forurennet med teknologi-specifikt kode til tilgang af særlige datakilder som vil være svært at vedligeholde. Forretningslogikken har ikke kendskab til hvordan dataen, der skal bruges, er lagret. Den får det istedet data tilbage som objekt-orienterede modeller via Database API. Disse modeller er indeholdt i delmodulet Database API Models.

User Database API er et delmodul som ikke er implementeret i systemets prototype.

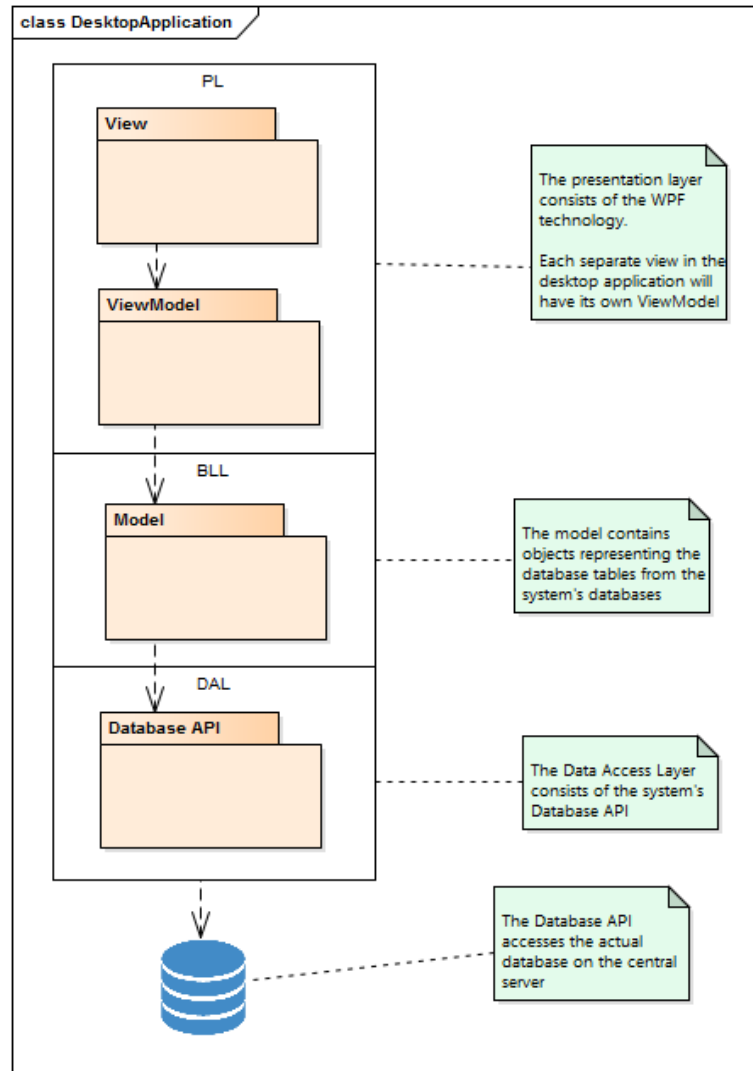


8.4 Brugergrænseflade arkitektur

Locatiles har to primære applikationer der gør brug af en grafisk brugergrænseflade: desktop applikationen og web applikationen. Disse applikationer er på det arkitektoniske niveau opsat efter forskellige brugergrænseflade arkitekturer. Dette afsnit vil beskrive arkitekturen af applikationerne samt de fordele det har givet.

8.4.1 Desktop applikation

Til opbygning af desktop applikationen er der gjort brug af et Model-View-ViewModel (MVVM) pattern, se bilag 1 - MVVM. På figur 8 ses arkitekturen for desktop applikationen.



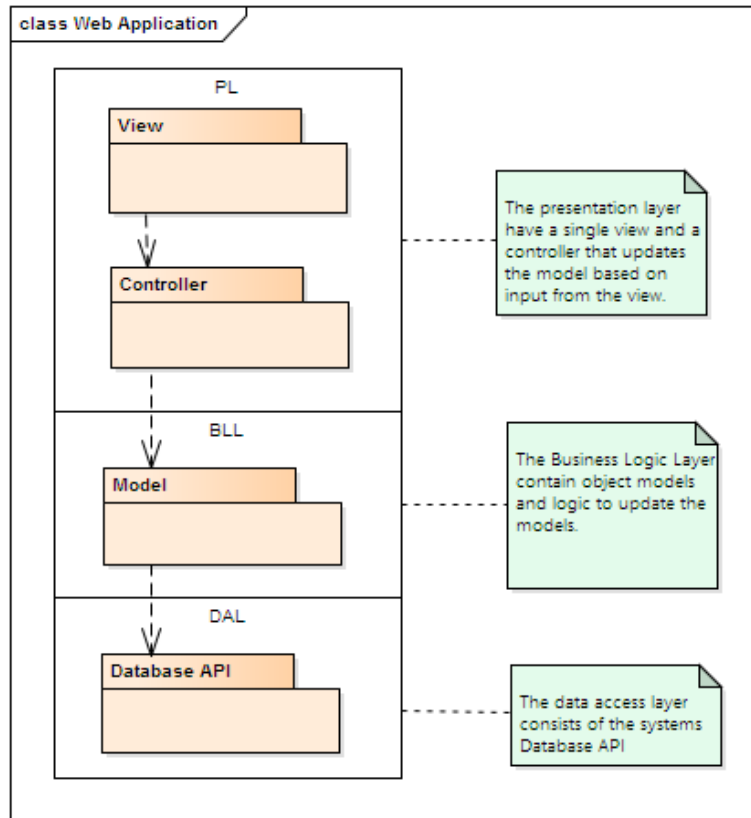
Figur 8: Desktop applikationens arkitektur baseret på MVVM

MVVM er valgt som arkitektur pattern til desktop applikationens arkitektur da denne adskiller programmets forretningslogik fra præsentrationslogikken. Dette medfører at det er betydeligt lettere at automatisere tests af applikationen, idét at forretningsloggikken kan testes separat fra præsentrationslogikken. Ved at separere disse to lag er det muligt at genbruge forretningslaget og datalaget, hvis applikationen skal gøre brug af en anden platform såsom UWP, da disse ikke er teknologispecifikke. Dette betyder også at Locatiles ved senere lejlighed kan integreres med andre butikapplikationer. Dette specificeres yderligere i afsnit 12.4, s. 37.

Desktop applikationen indeholder et tyndt forretningslag, da denne består af de objekt-orienterede repræsentationer af databasens tabeller og da applikationen blot indsætter i og trækker data fra databasen. Dermed er der ikke meget forretningslogik og derfor vil viewmodeller ofte gå udenom forretningslaget og kalde direkte ned til datalaget. Dette betyder at der ikke overholdes et "ideelt" MVVM pattern, så applikationens MVVM arkitektur, minder mere om arkitekturen fremvist i bilag 1.

8.4.2 Web applikation

Web applikationen gør brug af en arkitektur baseret på Model-View-Controller (MVC) arkitekturen, se bilag 2 - MVC. På figur 9 ses den de forskellige lag som udgør web applikationen.



Figur 9: Web applikationens arkitektur baseret på MVC

Ved at bruge MVC arkitekturen opnås en separation af præsentationslaget, forretningslogiklaget og data access laget. Dette gør systemets lag løst koblede, hvilket betyder at lagene kan uskriftes efter behov. Hvis koblingen mellem forretningslaget og data access laget er løs nok, kunne et eksempel være at udskifte data access laget med et nyt lag, der gør brug af Entity Framework frem for ADO.NET.

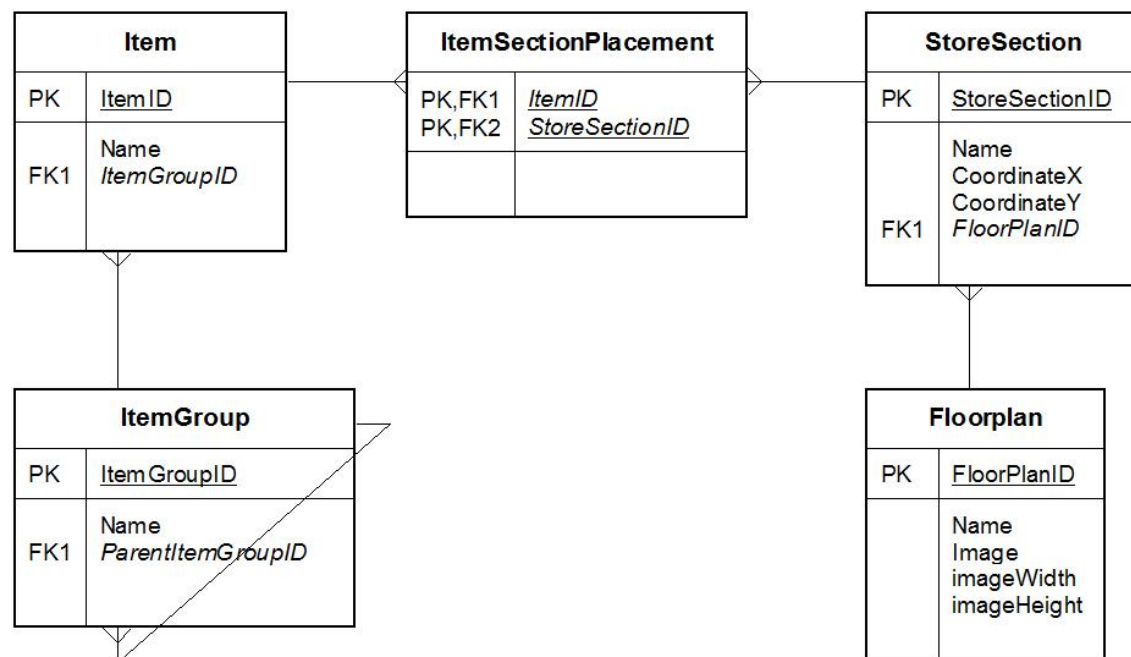
Alt interaktion mellem den grafiske brugergrænseflade (view) og forretningslogikken (BLL) går gennem controller, som sørger for at aktivere de rigtige handlinger i forhold til det der aktiveres i viewet og derefter returnere et view med opdateret data.

9 Design & Implementering

Design og implementering af systemets komponenter er et resultat af de valg der blevet taget i planlægningen af systemets arkitektur. Tilsammen udgør disse resultater implementeringen af semesterprojektets produkt. Følgende afsnit præsenterer disse designede samt implementerede komponenter.

9.1 Butiksdatabase

På figur 10 ses et entity-relationship diagram af databasen med attributter. For en detaljeret beskrivelse af hver attribut henvises der til dokumentationen afsnit 4.1.2, s. 25.



Figur 10: ERD med attributter for butiksdatabasen

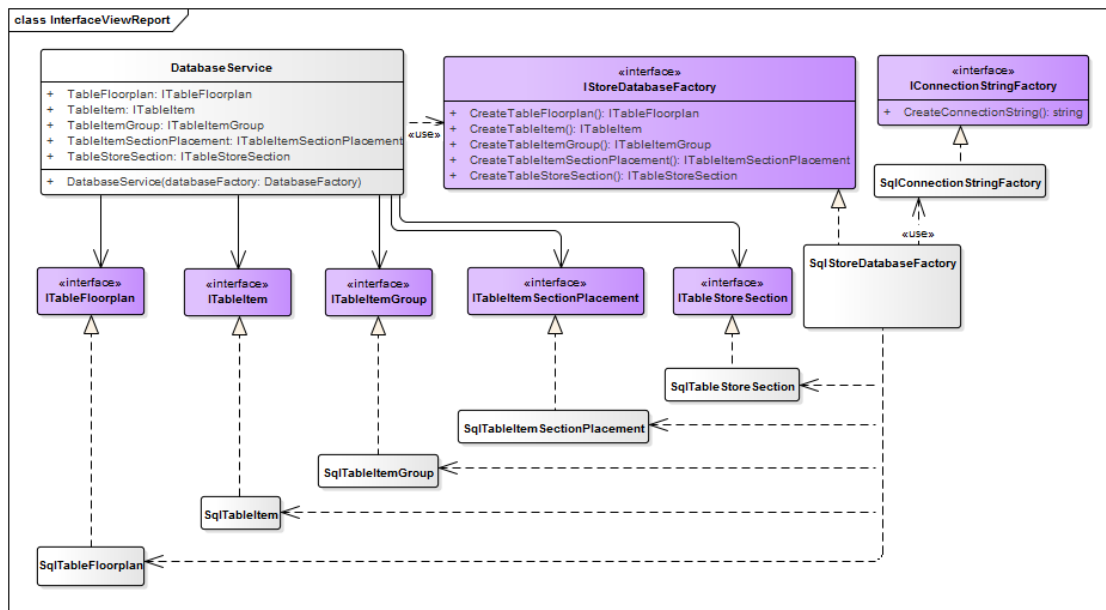
På diagrammet ses det at tabellen ItemGroup har en foreign key til sig selv. Dette design blev valgt, da det skulle være muligt for en varegruppe at have øvre varegrupper.

Yderligere ses det at tabellen Floorplan indeholder attributterne imageWidth og imageHeight. Disse bruges ikke aktiv produktets nuværende implementering. Der henvises til afsnit 12.2, s. 37 for mere information.

9.2 Database API

Dette afsnit beskriver software designet for systemets database API, herunder hvorfor de specifikke designvalg for denne blev truffet. For en mere detaljeret modellering af Database API klasserne, samt mere dybdegående beskrivelser af hver, henvises der til dokumentationen afsnit 4.2.1, s. 29.

På figur 11 ses Database API designet. Yderligere ses de konkrete implementeringer som gøres brug af i projektet, hvor Database API er implementeret til at gøre brug af en MS SQL Server. Dette er alle klasserne som starter med "Sql".



Figur 11: Overblik af Database API interfaces samt deres relationer med hinanden

Til design af Database API er der gjort brug af to design patterns. Disse design patterns er:

- Repository pattern [21]
- Abstract factory pattern [22]

På figur 11 repræsenterer klassen DatabaseService et repository til øvre forretningslogik. Det vil sige, for at forretningslogik kan udtrække data fra butikkens database, skal det bruge de eksponerede interfaces igennem DatabaseService.

Der er flere fordele som systemet drager nytte af ved at bruge repository pattern:

- Det centraliserer koden til tilgang af data til specifikke klasser. Dette er en stærk fordel for projektet da datatilgang skal bruges både af web applikationen samt desktop applikationen. Ved at centralisere det kan koden genbruges
- Kode til datatilgang bliver separeret fra forretningslogik, hvilket opdeler systemets kodebase i flere veldefinerede ansvar som forbedrer vedligeholdelsesmulighederne for koden
- Det bliver lettere at udvikle automatiserede test til integrationstest mellem DatabaseService og deres associerede datakilder
- Det bliver muligt at mappe rå data fra datakilderne (f.eks. rækker i tabeller) til objekt-orienterede modeller som forretningslogikken i stedet kan bruge. Dette tilfører et yderligere abstraktionslag mellem datakilder og forretningslogikken

Som eksempel ville en klient, for at oprette en ny vare i butikkens database, oprette en ny instans af "DatabaseService" med en factory såsom "SqlStoreDatabaseFactory". Herefter ville klienten bruge "TableItem" property'en til at kalde en metode for at oprette en ny vare. For yderligere beskrivelse omkring disse metoder henvises der til dokumentationen afsnit 4.2.1, s. 33.

Ud fra figur 11 kan det også ses at services for "DatabaseService" er opdelt i flere mindre interfaces, som implementeres af flere små klasser. Altså, for at udføre operationer relevante til butikkens plantegning, skal klienten gøre brug af "ITableFloorplan". Hvis klienten vil udføre operationer på varer, skal dette ske igennem "ITableItem" interfacet. Her er SOLID [23] principperne Single Responsibility [24] samt Interface Segregation [25] fulgt.

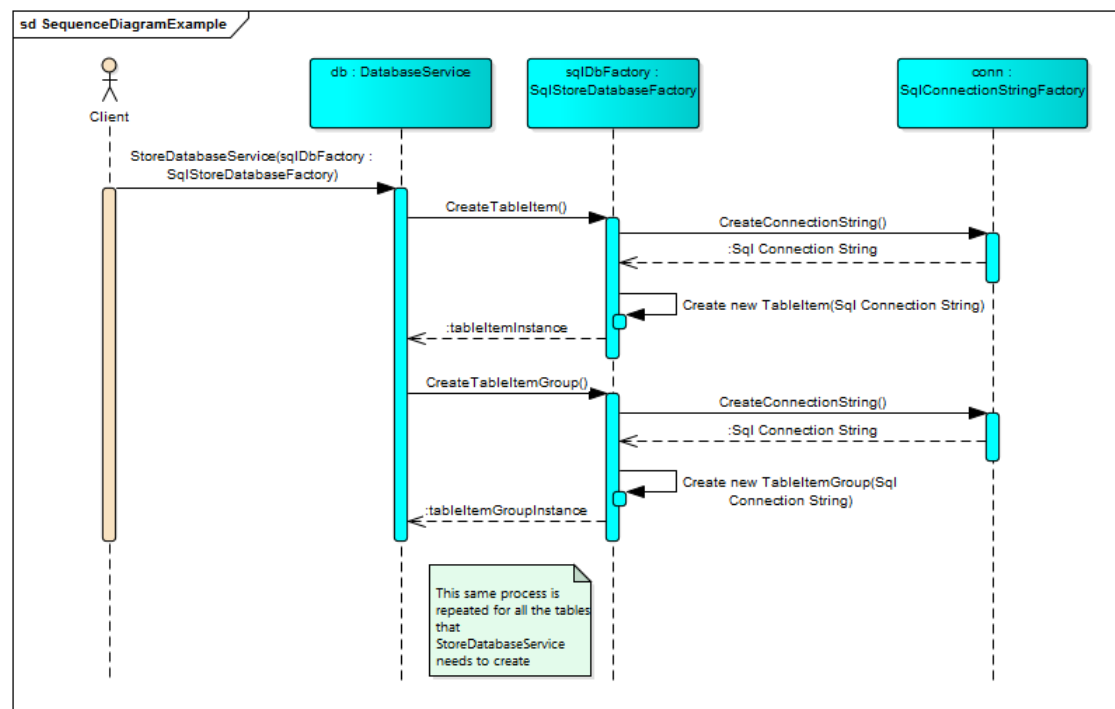
Single Responsibility Principle blev fulgt for at give hver klasse i relation til datakilden ét veldefineret ansvar. Yderligere blev Interface Segregation princippet fulgt, så “DatabaseService” klassen ikke ville ende med ét enkelt stort interface til manipulering af alle dele af butikkens datakilde. Dette ville påtvinge klienter til at tage stilling til metoder de i realiteten ikke ville have brug for.

Som eksempel, hvis “DatabaseService” var blevet designet som ét stort interface, skulle en klient til at filtrere igennem metoder til at oprette, slette samt redigere butikkens sektioner samt varegrupper, selvom klienten muligvis kun ville oprette varer.

Som det også kan observeres fra figur 11, findes der desuden to abstract factories. “IStoreDatabaseFactory” samt “IConnectionStringFactory”.

En konkret implementering af “IStoreDatabaseFactory” har til ansvar at konstruere konkrete typer af alle tabeller i “DatabaseService”. I dette projekt er det som sagt lavet en Database API implementering som tilgår en MS SQL Server gennem ADO.NET, og derfor er den konkrete implementering “SqlStoreDatabaseFactory”.

Grunden til at abstract factory pattern blev brugt i projektet, var for at simplificere konstruktionen af “DatabaseService” fra klientens synspunkt. Der er relativt mange properties der skal sættes korrekt i klassen afhængigt af den underliggende datakilde. Det er derfor nemmere fra klientens synspunkt at give en konkret implementering af en “IStoreDatabaseFactory” med, så konstrueringsansvaret er uddelegeret. På figur 12 er dette konstruktionsforløb illustreret.

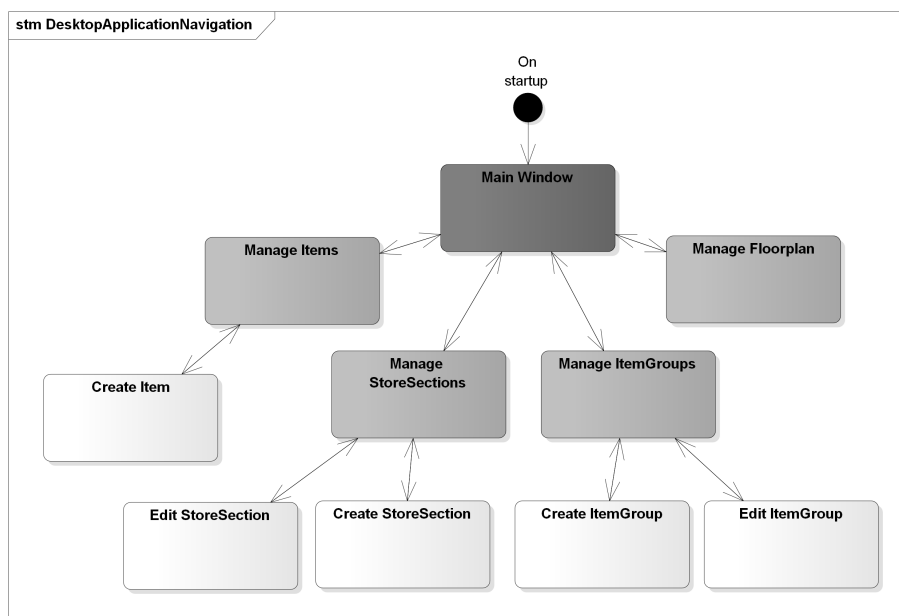


Figur 12: Sekvens af Database API konstruktion

Navngivningen af klasserne samt modulerne kunne dog forbedres. Hensigten i begyndelsen af projektet var at systemet, set fra et arkitektur-perspektiv, kun skulle bruge databaser til prototypen, hvilket også er sådan implementeringen er blevet udført. Dog er designet af softwaren opbygget på sådan en måde (ved brug af f.eks. repository pattern), at det ikke *behøves* at være en database som forretningslogikken skal tilgå. Konkrete implementeringer af Database API (ud over *SQL* implementeringen brugt i prototypen), kunne lige så godt have gjort brug af hvilken som helst anden datakilde, derfor synes navngivningen at være misvisende og for teknologi-specifik. Database API'en er også lidt ineffektiv, da den ikke gør brug af alle de værktøjer *SQL* stiller til rådighed. Eksempelvis bliver der ikke gjort brug af joins, hvilket gør koden mindre effektiv, og i et endeligt produkt skal API refaktoreres til at inkludere dette.

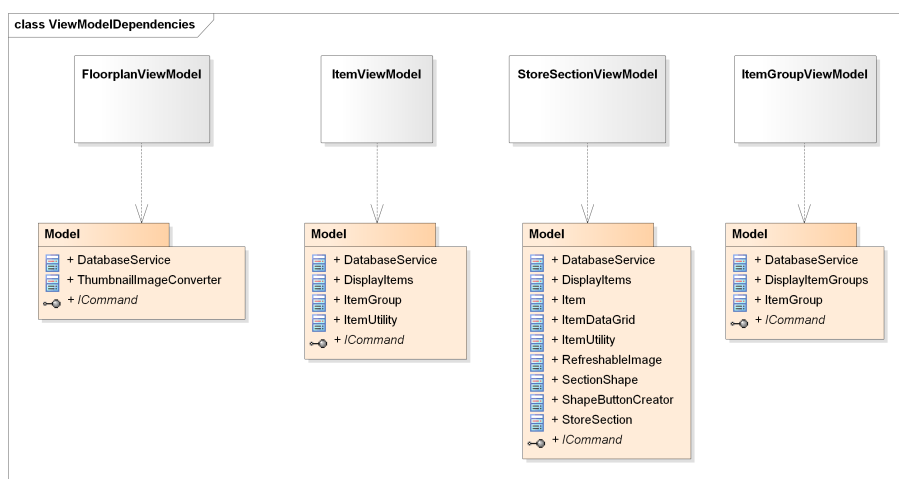
9.3 Desktop applikation

Desktop applikationen er en brugergrænseflade med flere forskellige menuer der kan navigeres imellem. De fire menuer, Manage Items, Manage StoreSections, Manage Itemgroups og Manage Floorplan, har hvert et singulært ansvar. Figur 13 ses et tilstandsmaskinediagram, der viser vinduernes relationer til hinanden. Boksene markeret med hvid, er dialogbokse der bliver åbnet af deres “parent” menu.



Figur 13: Tilstands maskine for desktop applikationen

Ifølge MVVM pattern, skal der være en viewmodel mellem et view og en model. Derfor har hver af de fire menuer deres egen viewmodel, som de deler med dialogboksene de kan åbne. Figur 14 viser desktop applikationens viewmodeller. Som vist på figuren, gør disse hver især brug af flere forskellige modeller afhængig af viewmodellens ansvar.



Figur 14: Modellag for desktop applikationen

Menuerne og deres dialoger gør brug af den samme viewmodel. Dette medfører at viewmodeller er meget omfattende da de indeholder alle properties og commands for deres givne vindue. Dette

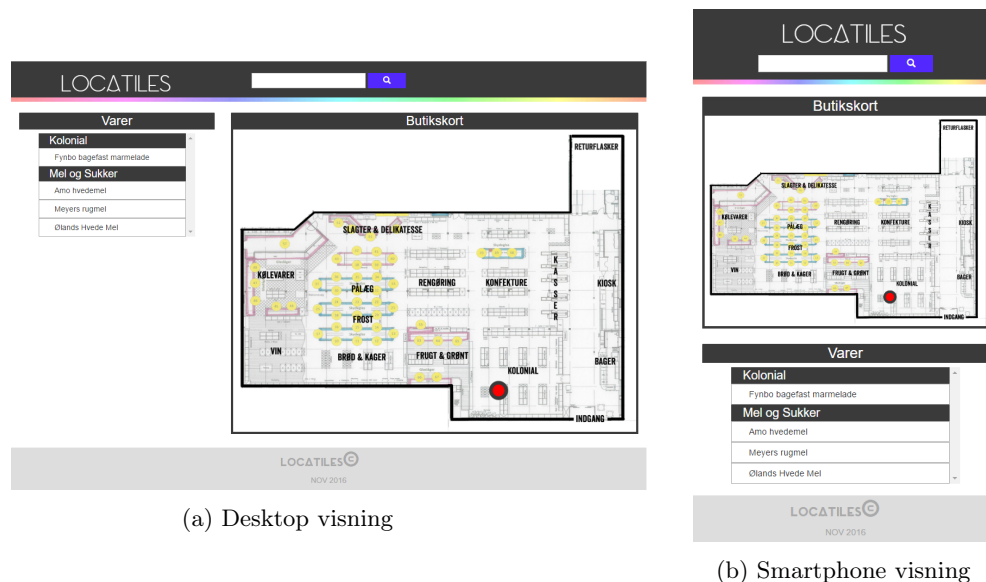
lettere gør dog implementeringen af viewmodellerne da commandhandlerne har adgang til alle relevante properties for det område som viewmodellen dækker over.

En detaljeret beskrivelse af både views, modeller og viewmodeller kan findes i dokumentationen afsnit 4.3, s. 37.

9.4 Web applikation

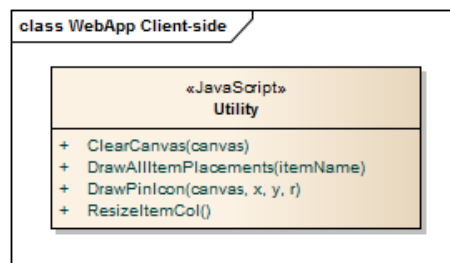
Web applikationen har kun et vindue og når dette åbnes, vil man kunne se en tom vareliste, et kort og et søgefelt med tilhørende søgeknop. Ved at søge efter en vare, vil varer med matchende varenavn blive vist i varelisten. Følgende afsnit beskriver designet af web applikationen. For yderligere beskrivelse af designet henvises der til dokumentationen afsnit 4.4, s. 51.

På figur 15 ses den grafiske brugerflade for web applikationen. For at give en god brugeroplevelse på forskellige skærmstørrelser er web applikationen implementeret med bootstrap hvilket gør at sidens elementer kan skalere og arrangere sig på en passende måde i forhold til skærmstørrelsen. Vi har valgt at placere kortet over varelisten hvis skærmstørrelsen svarer til en smartphone skærm.



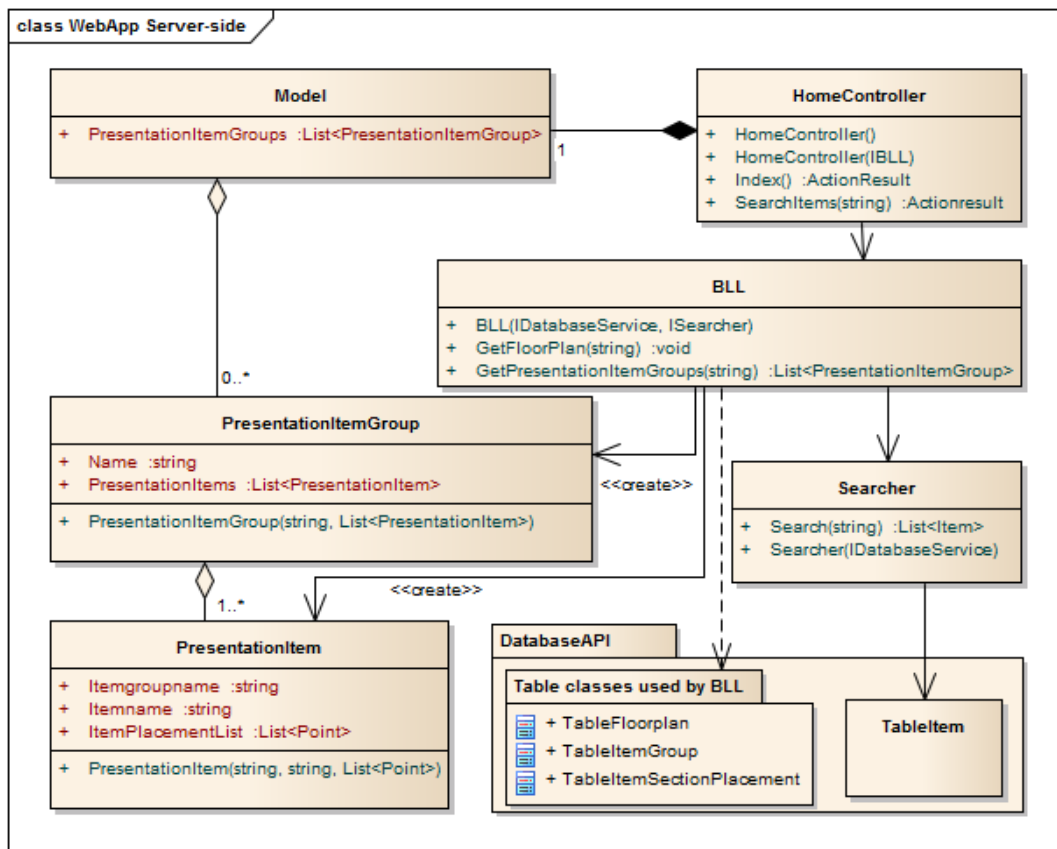
Figur 15: Responsivt design med Bootstrap

For at give klient-siden (browseren) funktionalitet er der fremstillet en funktionspakke som kan ses på figur 16.



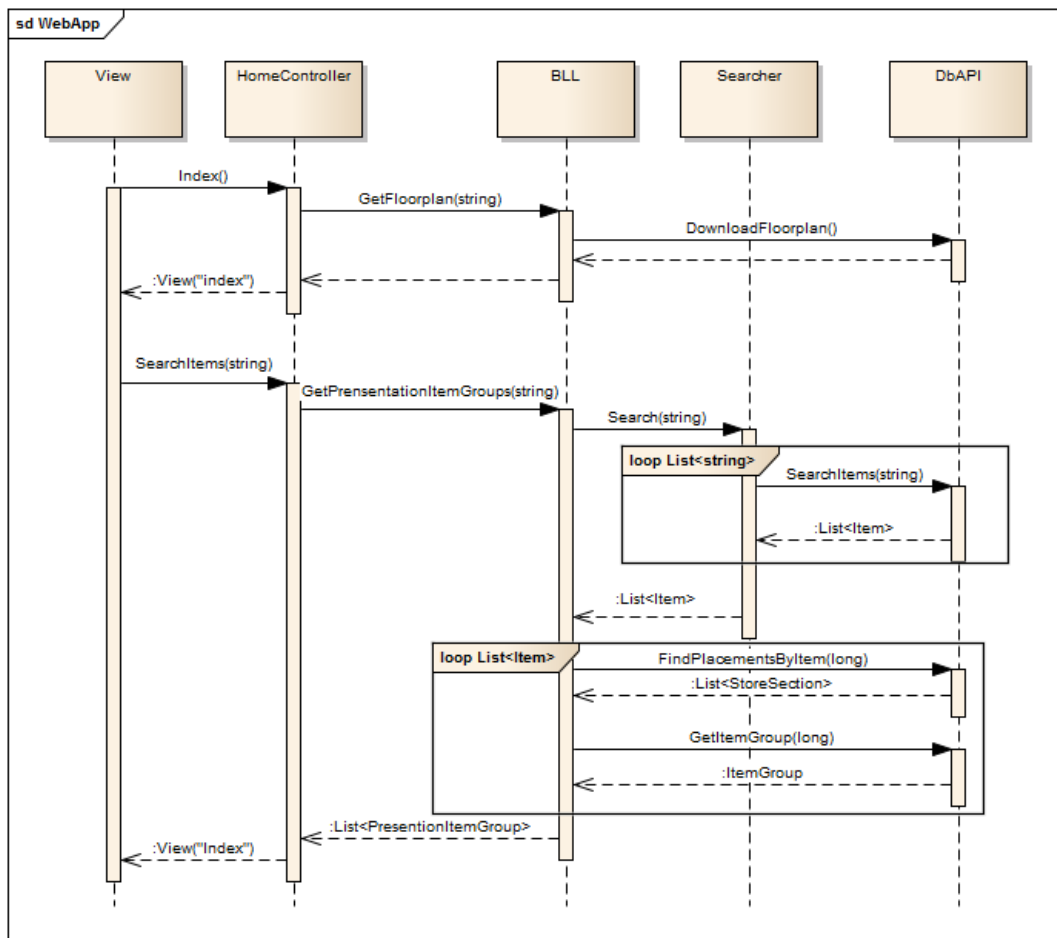
Figur 16: Diagram af den funktionspakke til web applikationen som bliver brugt på klientsiden

I web applikationen er der brugt en samling klasser der bruges til at samle og præsentere vareinformation på den grafiske brugergrænseflade. På figur 17 ses klasserne og relationerne mellem dem.



Figur 17: Klassesdiagram af de klasser fra web applikationen som bliver brugt på server-siden

På figur 18 ses et sekvensdiagram med interaktionerne web applikationens klasser. “View” og “DbAPI” er ikke egentlige klasser, men repræsenterer istedet henholdsvis den grafiske brugergrænseflade og sql table klasserne i data access laget.



Figur 18: Sekvensdiagram af webapplikationen

10 Test

Dele af projektets software er blevet testet ved brug af unit tests samt integrationsstest. Dette er blevet gjort for at øge pålideligheden til koden, verificere at implementerede funktionaliteter har den forventede adfærd, samt for at gøre potentielle fejl nemmere at lokalisere. Dette afsnit omhandler fremgangsmåden for tests af projektets forskellige software moduler.

10.1 Unit test

I projektforløbet er klasser, hvor det har givet mening, blevet unit testet. Disse klasser følger derfor et testbart design, hvor de kan isoleres fra deres afhængigheder igennem dependency injection. Dette afsnit beskriver hvilke klasser der er blevet unit testet og nævner yderligere klasser som ikke er blevet unit testet samt den tilhørende grund.

10.1.1 Database API

Adfæren for den konkrete implementering af Database API til en MS SQL Server, brugt i projektet, er tæt knyttet til den fysiske database som den manipulerer. Derfor blev det valgt at denne ikke ville give meget værdi at unit teste, da man i det tilfælde ville blive nødt til at isolere den fra databasen som bruges.

På grund af dette blev Database API i stedet integrationstestet med databasen. Integrationstesten er implementeret som fase 1, afsnit 10.2, s. 34.

10.1.2 Desktop applikation

Desktop applikationen implementerer et MVVM pattern, se afsnit 8.4.1 s. 24, for at gøre det muligt at teste brugergrænsefladens funktionalitet. Dette gøres ved at teste vinduernes viewmodeller. Det har dog kun været muligt at skrive automatiske tests til tre ud af fire view modeller, da StoreSectionViewModel gør stort brug af WPF funktionaliteter og afhænger derfor af viewet. Dette betyder at denne klasse og dens funktionaliteter skal testes manuelt.

For at teste vinduernes funktionalitet og samspil med viewmodellerne, er disse testet tildels gennem visuelle tests og tildels igennem produktets accepttests.

Unit under test	ItemGroup-ViewModel	ItemViewModel	Floorplan-Viewmodel	StoreSection-Viewmodel
Status	Gennemført	Gennemført	Gennemført	Ikke gennemført

Tabel 4: Unittests i desktop applikationen

Tabel 4 viser resultaterne for unit tests af desktop applikationens viewmodeller. En detaljeret oversigt over de udførte unit tests findes i dokumentationen afsnit 5.3.1 s. 57.

10.1.3 Web applikation

Alle klasser i webapplikationen der indeholder adfærd er blevet unittestet. Disse klasser ses i tabel 5. En mere detaljeret beskrivelse af web applikationens unit tests kan findes i dokumentationen afsnit 5.3.2 s. 57.

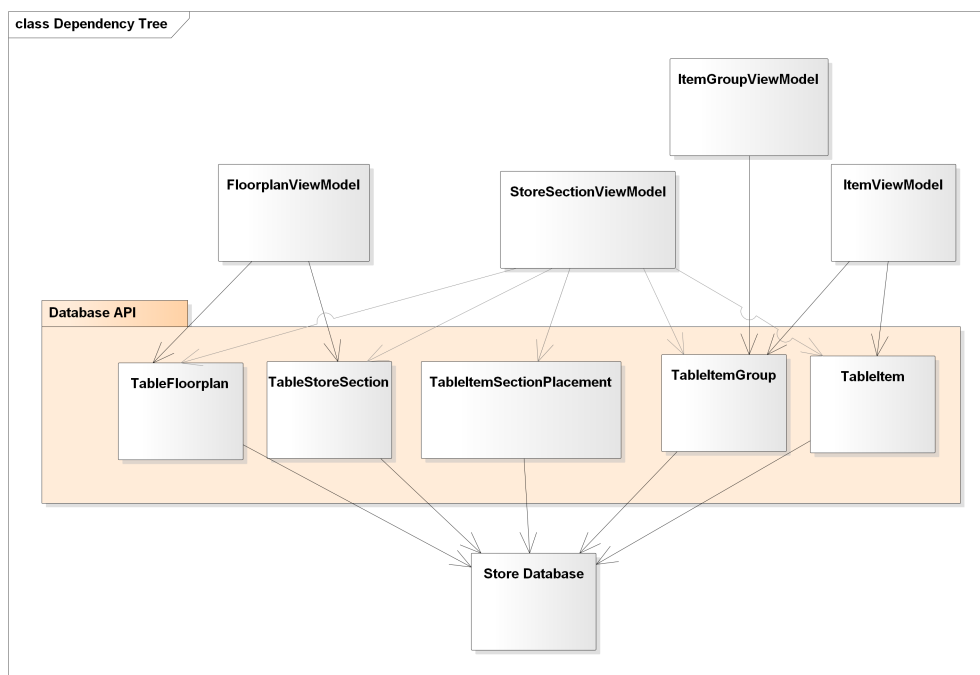
Unit under test	HomeController	BLL	Searcher
Status	Gennemført	Gennemført	Gennemført

Tabel 5: Unittests i webapplikationen

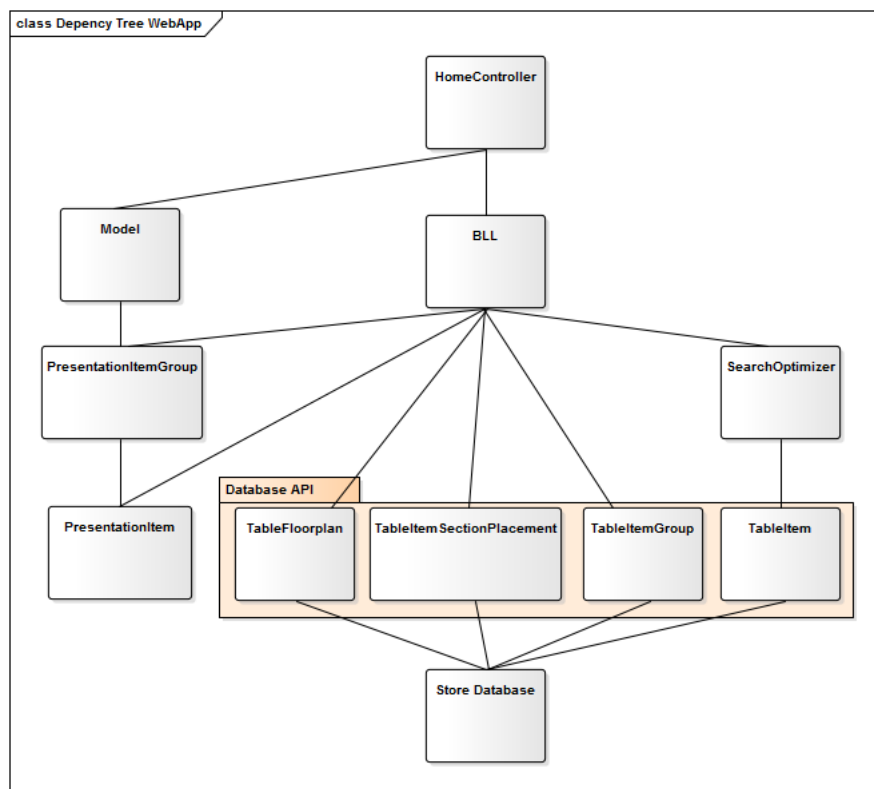
10.2 Integrationstest

For at teste interaktionerne mellem modulerne som udgør systemet, er der blevet foretaget integrationstests. Test strategien for integrationstesten er bottom-up metoden [26]. For hver applikation er der lavet et afhængighedstræ som visualiserer afhængighederne mellem de enheder som testes. Ud fra afhængighedstræerne og den valgte teststrategi er der defineret en række steps til udførelse af integrationstesten. For de definerede steps henvises der til dokumentationen afsnit 5.4 s. 58.

Integrationstesten er opdelt i tre faser, hvor den første tester database API og de sidste to tester applikationerne. Afhængighedstræet for integrationstestens første to faser ses på figur 19. Afhængighedstræet for integrationens tredje fase ses på figur 20.



Figur 19: Afhængighedstræ for desktop applikation



Figur 20: Afhængighedstræ for tredje fase

Fase 1 - integration af butiksdatabase og database API

I første fase indgår der to hovedkomponenter; butiksdatabasen og database API. Denne fase tester interaktionerne mellem butiksdatabasen og database API.

Fase 2 - integration af desktop applikation

I integrationstesten for anden fase indgår der tre hovedkomponenter; butiksdatabasen, database API og viewmodellerne. I denne fase testes interaktionerne mellem databasen og viewmodellerne via database API.

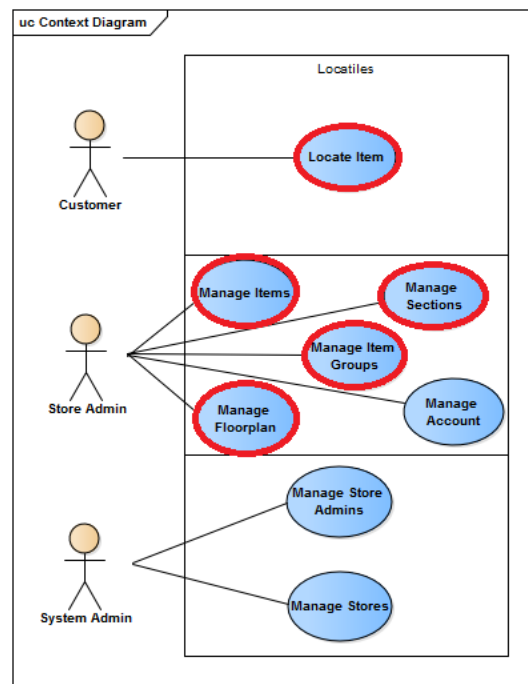
På figur 19 ses at StoreSectionViewModel indgår i afhængighedstræet. Denne indgår dog ikke i integrationstesten, da koblingen mellem viewmodellen og dets view er for stor.

Fase 3 - integration af webapplikation

I denne fase testes interaktionerne mellem klasserne HomeController og Searcher i webapplikationen, og butiksdatabasen via database API.

11 Resultater

Under den indledende fase af projektet blev der udarbejdet nogle epics, som definerede hoved-funktionaliteter der skulle indgå i produktet og lagde til grund for specificerede userstories. I projektafgrænsningen, afsnit 5, s. 12, blev der udvalgt visse userstories til at indgå i produktet. Dette resulterede i at der ved slutningen af projektforsløbet er blevet udviklet en web og en desktop applikation, som tilgår data fra en butiksdatabase. Figur 21 viser kontekstdiagrammet for Locatiles, hvor implementerede funktionaliteter er markeret med rødt.



Figur 21: Overblik over implementerede funktionaliteter

Igennem projektet er web applikationen blevet udviklet så en kunde kan søge på varer, som en butiksadministrator har placeret på butikkens plantegning via desktop applikationen. Derudover har butiksadministratoren mulighed for at administrere hvilke varegruppe en vare hører til samt opdatere butikkensplantegning.

Ydermere er der defineret accepttests til at verificere at produktet opfylder de stillede krav til funktionaliteter. Disse tests er blevet gennemgået for web og desktop applikationerne og godkendt af gruppen. For den udfyldte accepttest specifikation henvises der til dokumentationen afsnit 6, s. 61.

Dette betyder at alle "must have" kravene stillet under projektafgrænsningen er blevet fuldt implementeret.

12 Fremtidig arbejde

Da projektforløbet strækker sig over en relativ kort tidsperiode, hvor det ikke er muligt at implementere produktet som et leveringklart produkt, er der en række tilføjelser og ændringer som kunne ønskes til produktet. Overvejelser omkring dette vil blive beskrevet i følgende afsnit.

12.1 Ikke implementeret funktionaliteter

De mest oplagte tilføjelser til systemet er de funktionaliteter, der blev specificeret i kravspecifikationen og senere sorteret fra i projektafgrænsningen, som endnu ikke er blevet implementeret. Dette indebærer at brugerdatabase oprettes og et login system til butiksadministratorene implementeres. Med disse tilføjelser vil Locatiles kunne supportere flere brugere og butikker og anvendes i større sammenhæng.

12.2 Brugergrænseflader

Som nævnt i afsnit 9.1, s. 26 er der i butiksdatabase nogle attributter som ikke bruges i Locatiles' nuværende implementering. Disse attributter beskriver en plantegnings højde og vidde. Ændringer til brugergrænsefladerne omfatter at disse værdier tages i brug, så plantegninger ikke længere tvinges til at tilpasse bestemte mål, men kan skalere deres faktiske dimensioner. Da database API og databasen allerede tager højde for disse ændringer, skal der kun laves ændringer i brugergrænsefladerne.

12.3 Plantegninger

I den nuværende implementering af desktop applikationen kan en butik kun have en plantegning. Dette er uhensigtsmæssigt for butikker der består af flere etager. Ved at udvide database API og brugergrænsefladerne vil der kunne tages højde for denne type butik.

12.4 Integrering med eksisterende applikationer

De fleste butikker har i forvejen en applikation til deres butikker. Disse kunne inkorporere funktionaliteter fra Locatiles, så disse også kunne bruges til vare søgning. På denne måde kan kunder have en applikation til alle funktionaliteter og butikkerne slipper for at holde to separate applikationer opdateret.

13 Konklusion

I starten af semesterprojektet, blev idéen bag produktet Locatiles undfanget, og en kravspecifikation udarbejdet. Gruppen har igennem projektforsløbet arbejdet på at realisere produktet på baggrund af projektafgrænsningen. Som resultat af dette, udmundede projektforsløbet i en web applikation og en desktop applikation, som opfylder projektafgrænsningens krav.

Igennem projektforsløbet har gruppen gjort sig mange erfaringer. Gruppen gjort brug af den iterative udviklingsproces scrum, men i en modificeret udgave, der tog højde for de erfaringer, som gruppen havde fra tidligere semesterprojekter. Dette ledte til en mere effektiv arbejdsproces for gruppen. Under design og implementeringen af produktets software, har gruppen forsøgt at følge principperne for testbart design. Det blev erfaret at testbart design er både svært og tidskrævende at implementere, især i sammenkobling med grafiske brugergrænseflader. Til gengæld har det den positive effekt at tilliden til koden stiger, da man kan teste den på systematisk vis. Gruppen har også erfaret at godt design, selvom at design patterns samt arkitektoniske patterns bliver fulgt, i realiteten kan være svært at implementere. Der kan opstå særtilfælde specielt til det projekt man arbejder på, som gør at patterns ikke kan følges slavisk. Dette gør at man mange gange skal overveje fordele og ulemper relativt til egne behov.

Locatiles er resultat af et godt samarbejde mellem gruppemedlemmer med varierende faglig baggrund, interesser og kompetencer. Under produktudviklingen har hvert medlem haft mulighed for at udfolde sig indenfor deres interesseområder. Derudover var der mulighed for at komplementere hinanden, når der opstod forhindringer. Alt i alt har projektet bidraget til en erfaringsmæssig udvikling, som kan anvendes i fremtiden.

14 Bilagsliste

- Bilag 1 - MVVM: Uddrag “The Model-View-Viewmodel Design Pattern” af Poul Ejnar Røvsing.
- Bilag 2 - MVC: Uddrag “The Model-View-Viewmodel Design Pattern” af Poul Ejnar Røvsing.

Litteratur

- [1] Google. Google maps. <http://google.com/maps/>. Tilgået: 21-11-2016.
- [2] Dan Radigan. Epics, stories, versions, and sprints. <https://www.atlassian.com/agile/delivery-vehicles>). Tilgået: 01-12-2016.
- [3] DSDM Consortium. Moscow prioritisation. <https://www.dsdm.org/content/moscow-prioritisation>. Tilgået: 21-11-2016.
- [4] Inc. Object Management Group. Uml. <http://uml.org/>. Tilgået: 21-11-2016.
- [5] The scrum guide. J. sutherland and k. schwaber. <http://www.scrumguides.org/scrum-guide.html>. Tilgået: 16-11-2016.
- [6] Pivotal Software Inc. Pivotal tracker. <https://www.pivotaltracker.com/>. Tilgået: 29-11-2016.
- [7] Trello Inc. Trello. <https://trello.com/>. Tilgået: 16-11-2016.
- [8] GitHub. Github. <https://github.com/>. Tilgået: 16-11-2016.
- [9] Semesterprojekt gruppe 1. Semesterprojekt4-ikt. <https://github.com/CodingBeagle/SemesterProjekt4-IKT>. Tilgået: 16-11-2016.
- [10] Microsoft. .net documentation. <https://docs.microsoft.com/en-us/dotnet/>. Tilgået: 17-11-2016.
- [11] MSDN. Windows presentation foundation. [https://msdn.microsoft.com/en-us/library/ms754130\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx). Tilgået: 17-11-2016.
- [12] Microsoft. Sql server 2012. [https://technet.microsoft.com/en-us/library/ms143287\(v=sql.110\).aspx](https://technet.microsoft.com/en-us/library/ms143287(v=sql.110).aspx). Tilgået: 17-11-2016.
- [13] Microsoft. Microsoft azure. <https://azure.microsoft.com/en-us/>. Tilgået: 17-11-2016.
- [14] MSDN. Ado.net. [https://msdn.microsoft.com/en-us/library/e80y5yhx\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx). Tilgået: 17-11-2016.
- [15] Microsoft. Asp.net. <https://www.asp.net/>. Tilgået: 17-11-2016.
- [16] Bootstrap. Bootstrap. <http://getbootstrap.com/>. Tilgået: 08-12-2016.
- [17] jQuery. jquery. <https://jquery.com/>. Tilgået: 08-12-2016.
- [18] NUnit.org. Nunit. <https://www.nunit.org/>. Tilgået: 17-11-2016.
- [19] NSubstitute. Nsubstitute. <http://nsubstitute.github.io/>. Tilgået: 17-11-2016.
- [20] MariaDB. Database design phase 2: Conceptual design. <https://mariadb.com/kb/en/mariadb/database-design-phase-2-conceptual-design/>. Tilgået: 23-11-2016.
- [21] MSDN. Repository pattern. <https://msdn.microsoft.com/en-us/library/ff649690.aspx>. Tilgået: 16-11-2016.

- [22] dofactory. Abstract factory. <http://www.dofactory.com/net/abstract-factory-design-pattern>. Tilg et: 16-11-2016.
- [23] Robert C. Martin. Solid. [https://en.wikipedia.org/wiki/SOLID_\(object-oriented_design\)](https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)). Tilg et: 28-11-2016.
- [24] Robert C. Martin. Single responsibility principle. https://en.wikipedia.org/wiki/Single_responsibility_principle. Tilg et: 28-11-2016.
- [25] Robert C. Martin. Interface segregation principle. https://en.wikipedia.org/wiki/Interface_segregation_principle. Tilg et: 28-11-2016.
- [26] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 1999.