# 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, og nemt vise hvilke klasser der har afhængigheder til en anden pakke. Package diagrammerne, er blevet placeret i Logisk View, pga. den tætte sammenkobling med klasse diagrammer.

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 samt de forskellige interfaces der er og forventes herimellem, samt forholdene mellem dem. Diagrammet bliver typisk brugt til Component-Based Development (CBD)[[1]](#footnote-1), 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.

# PROCESS VIEW

Er ikke brugt, fordi at gruppen følte at logical view beskrev det, det samme men mere grundigt.

# DEPLOYMENT VIEW

Deployment view beskæftiger sig med vise bindingen mellem software og hardware, altså på hvilke software moduler der skal ligge hvilken hardware.



Figur 1: Deployment diagram for Pristjek220

Figur 1 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 protocol, der overføres til og fra databasen med. De forskellige executable’s er applikationer, som kører på devicesene.

# DATA VIEW



Figur 2 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, var det i Pristjek220 nødvendigt at have en property til produktets pris på relationen mellem forretningen og produktet. Denne property skal indeholde, hvad prisen for produktet er i lige præcis den forretning, det tilhører.

# Generelle designbeslutninger

Dette afsnit fastholder de generelle designbeslutninger, der tages under arktiekturdesignet eller som er givet som ultimative krav.

Dette afsnit beskriver de ikke funktionelle krav der har betydning for arkitekturen. Det er krav som f.eks. at der skal anvendes standardprodukter, krav til portering til forskellige platforme, distribuering og genbrug.

Der kan også være krav til design og implementeringsstrategi, udviklingsværktøjer, team struktur, anvendelse af legacy (gammel) kode.

## Arkitektur mønstre

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

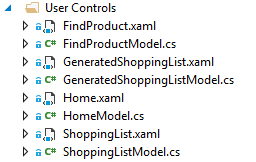


Figur 3: Package diagram for Pristjek220

## Design mønstre

### Mvvm pattern

Gennem Logical view, kan der ses at der i programmet er brugt Mvvm, ved at de forskellige view’s er en tilhørende viewmodel. Det kan også ses på Figur 4, som viser de forskellige views (xaml fil) med deres viewmodel (cs fil). Der kan findes yderligere eksempler på det i source koden.



Figur 4: Screenshot af opbygning af view og viewmodel

### Repository pattern



Figur 5: IMPLEMENTERING AF REPOSITORY PATTERN I PRISTJEK220

På Figur 5 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 en 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 der kan tilføjes eller fjernes mange ting på én gang, uden at gemme efter hver tilføjelse.

Her opremses de standard arkitektur mønstre, der er anvendt i systemet sammen med en reference til dokumentationen for mønstrene.

## Generelle brugergrænsefladeregler

Kan ses i kravspecifikationen under kvalitetskrav.

Her beskrives de generelle regler og principper, der skal overholdes for design af systemets brugergrænse-flader. Alternativt kan der her indsættes en reference til et eksternt dokument.

Dette afsnit beskriver de generelle regler for exception og fejlhåndtering.

## Udviklingsværktøjer

### Visual Studio (Microsoft, 2016)

Applikationerne er programmet i c# i Visual Studio, som er Microsofts programmeringsværktøj, med tillægsværktøjet Resharper og forskellige NuGet packages.

### Microsoft Visio (Microsoft, 2016)

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

### Microsoft Word (Microsoft, 2016)

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

### Scrumwise (Scrumwise, 2016)

Er et værktøj til at styrer et Scrum board, som kan tilgås online.

### Github (Github, 2016)

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

### TortoiseGit (TortoiseGit, 2016)

TortoiseGit er den Git klient, der er benyttet til at pull og push til GitHub repositoriet.

Her angives det valgte implementeringssprog og de værktøjer, der anvendes ved udviklingen.

## Frameworks og packages

Kort beskrivelse af hvilke frameworks og packages der er brugt og hvorfor de er brugt.

### Generelle frameworks

**EntityFramwork**

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

**Newtonsoft.Json**

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 skrive til og fra en fil.

### Test Frameworks

**NUnit**

Er et framework til at automatisere test med, således at en applikation kan teste den funktionaliteter, ved at opsætte test cases, som framworket, så kører.

**NSubstitute**

Er et framework til at substituerer klasser ud med, når der laves unit test, og i nogen tilfælde integrationstest, det gøres for at kunne isolere problemet til hvad der ønskes testet.

### Gui frameworks

**WPFToolkit**

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

**MvvmLight & MvvmLightLibs & CommonServiceLocator**

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.

# REFERENCER

1. <https://www.techopedia.com/definition/31002/component-based-development-cbd> [↑](#footnote-ref-1)