Systemarkitektur

Indholdsfortegnelse

[INTRODUKTION 2](#_Toc448137531)

[Formål og omfang 2](#_Toc448137532)

[Definitioner og forkortelser 2](#_Toc448137533)

[Dokumentstruktur og læsevejledning 2](#_Toc448137534)

[Dokumentets rolle i en iterativ udviklingsproces 2](#_Toc448137535)

[SYSTEM OVERSIGT 2](#_Toc448137536)

[System kontekst 2](#_Toc448137537)

[System introduktion 2](#_Toc448137538)

[Domæne model 2](#_Toc448137539)

[SYSTEMETS GRÆNSEFLADER 3](#_Toc448137540)

[Grænseflader til person aktører 3](#_Toc448137541)

[Grænseflader til eksterne system aktører 3](#_Toc448137542)

[USERSTORIES 3](#_Toc448137543)

[LOGISK VIEW 3](#_Toc448137544)

[DEVELOPMENT View 3](#_Toc448137545)

[PROCESS VIEW 4](#_Toc448137546)

[DEPLOYMENT VIEW 5](#_Toc448137547)

[DATA VIEW 5](#_Toc448137548)

[Valg til opsætning af database: 5](#_Toc448137549)

[Generelle designbeslutninger 6](#_Toc448137550)

[Arkitektur mål og begrænsninger 6](#_Toc448137551)

[Arkitektur mønstre 6](#_Toc448137552)

[Generelle brugergrænsefladeregler 6](#_Toc448137553)

[Exception og fejlhåndtering 6](#_Toc448137554)

[Implementeringssprog og værktøjer 6](#_Toc448137555)

[Implementeringsbiblioteker 6](#_Toc448137556)

[KVALITET 6](#_Toc448137557)

[REFERENCER 7](#_Toc448137558)

# INTRODUKTION

## Formål og omfang

## Definitioner og forkortelser

## Dokumentstruktur og læsevejledning

## Dokumentets rolle i en iterativ udviklingsproces

# SYSTEM OVERSIGT

Rigtbillede

## System kontekst

Aktør kontekst diagram

## System introduktion

Introduktion til systemet

### Domæne model



Figur 1: Domæne model af Pristjek220

Figur 1 viser en domæne model over PrisTjek220, hvor der kan ses hvordan de forskellige blokke interagere med hinanden. Modellen indeholde to forskellige databaser, en Log-in database som indeholder, administrator og forretningsmanager logins. Derudover indeholder den også en varedatabase, som indeholder de forskellige vare, hvor man kan købe dem og hvad deres pris er. Forbrugeren kan lave en indkøbsliste, og ved hjælp af indkøbslisteindstillinger, beslutte forskellige kriterier som den generede detaljeret indkøbsliste skal overholde, som fx antal af butikker der må handles i. Den detaljeret indkøbsliste genereres så ud fra indkøbslisten ved at tjekke efter hvor varerne kan fås billigst i database samtidig med at indstillingerne overholdes.

# SYSTEMETS GRÆNSEFLADER

## Grænseflader til person aktører

Beskrivelse af aktører

## Grænseflader til eksterne system aktører

DB server

# USERSTORIES

Henvis til Kravspecifikation

# LOGISK VIEW

Klasse diagrammer og sekvensdiagrammer

## Tilføj vare til forretning



Figur : Sekvens diagram for Tilføj en vare

## Finde den billigste forretning for en vare i Pristjek220

## Indtast indkøbsliste

## Find ud af hvor varerne fra indkøbslisten kan købes billigst

## Finde hvilke forretninger der har en vare

## Sammenligning af billigste indkøb og indkøb i én forretning

## Tilføj en forretning til Pristjek220

## Autofuldførelse



Figur 3: Sekvens diagram for autofuldførelse

På Figur 1, ses sekvensdiagrammet for autofuldførelse, som viser hvad der sker når en User begynder at indtaste i en autofuldførelsesboks, der er så 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 både i forbruger GUI’en og i forretnings GUI’en.

## Send indkøbsliste på mail

## Fjern en vare fra en bestemt forretning

## Fjern en vare fra Pristjek220

## Fjern en forretning fra Pristjek220

## Find åbningstider for en forretning

## Indstillinger for indkøbsliste

## Juster hvor vare skal købes efter Pristjek220 er kommet med et forslag

## Kunne bestemme afstanden der skal tilbagelægges for at købe varerne fra forslaget

## Kunne vise en kørselsvejledning mellem de forskellige forretninger, som der skal handles i

## Bekræftelse af oprettelse/sletning af vare

# 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.

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.



Figur 4: Package diagram for Pristjek220

Pristjek220 er opdelt i nogle subsystemer, som kan ses på package diagrammet på Figur 1. Det er opdelt i en Consumer (Forbruger) og en Store Manager (Forretningsmanager), som begge har deres eget interface og business logic lag. Interfacet er det som de to brugere interagerer med, mens business logic (forretningslogikken) er selve funktionaliteten, som ligger bag ved interfacet og eksekverer afhængig af brugernes handlinger. De to brugere deles om et data layer (data lag), hvor al information omkring varerne, forretningerne og logins ligger.

# PROCESS VIEW

Sløjfet

# DEPLOYMENT VIEW

Deployment diagram



Figur 5: Deployment diagram for Pristjek220

Figur 1 viser deployment diagrammet for Pristjek220. Diagrammet viser på hvilke hardware elementer, som de forskellige software implementeringer skal implementeres. Kommunikationen mellem de forskellige enheder og DB Serveren, foregår ved brug af Named Pipes, som er den protocol der overføres til og fra databasen med. De forskellige artifact’s på de to forskellige Devices, er main programmet, som kører på dem.

# DATA VIEW

Diagrammer over databasen

### Valg til opsætning af database:

Planen med den første udgave af databasen var at den skulle indeholde en enkel tabel hvor man kunne se varens navn i rækkerne og de forskellige forretninger hen ad kolonerne og på den måde finde prisen for varen i den ene forretning. Dette viste sig at have nogle problemer da vi skulle lave klassen ”product” med en variable for hver forretning som vi kunne gemme prisen i. Det resulterede i at vores database ikke var åben for udvidelser; hvilket er ineffektivt når forretninger kan åbne og lukke og forretningerne får nye vare. Vi valgte derfor at ombygge stukturen af klassen ”product”, så vi i stedet fik tre klasser: ”product”, ”store” og ”storeproduct”. Disse tre klasser gjorde at vores database kunne håndtere at en forretning ville begynder og sælge en vare, og en ny forretning åbnede. Måden det blev implementeret på var ved at ”product” indeholdte en liste af ”storeproduct” hvor prisen for vare i en enkle forretning lægger i, denne klasse indeholder også en reference til produktet og til forretningen. Dette betyder at for varerne og forretningerne har vi kun én instans af for hver vare eller forretning der er, også har vi en ”storeproduct” imellem hver af de forretninger som sælger én vare, og kobler dem sammen på den måde. På samme måde indeholder ”store” også en liste af ”storeproduct” for at kunne finde ud af om én vare sælges i den forretning.

Vores første udkast af databasen blev lavet med en masse variabler som var navn givet forretningerne for at danne et enkelt diagram hvor man kunne se produktets navn og forretningen og hvad prisen er for det i diagrammet. Men på grund af der ikke kunne udvides med det design, blev det lavet om til 3 klasser, ”product”, ”store” og ”storeproduct”. ”storeproduct” er relationen mellem forretningen og varen denne klasse indeholder prisen af varen, da varen ikke har samme pris i alle forretninger. Denne beslutning gjorde at vi gennem ”product” klassen kunne have en liste af ”storeproduct” hvilket betyder at vi var i stand til at simulere at varen kunne sælges i flere forretninger med forskellig pris. Derudover havde vi så også mulighed til at gøre det samme ved forretninger så der var flere vare i en forretning.

# 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

## Design mønstre

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

## Generelle brugergrænsefladeregler

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.

## Exception og fejlhåndtering

Dette afsnit beskriver de generelle regler for exception og fejlhåndtering.

## Implementeringssprog og værktøjer

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

## Frameworks og packages

# KVALITET

Pålidelighed:

Vedligeholdelsesvenlighed:

Brugervenlighed:

Integritet:

Effektivitet:

# REFERENCER

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