# Design og implementering - Database

I dette afsnit vil designprocessen, implementering samt test af database-delen af systemet blive beskrevet. Her vil blive beskrevet Data Access Layer for både Fridge App og Web App, med alle de overvejelser der er blevet gjort i designprocessen og implementering af DAL for begge applikationer.

## Fridge app

I dette afsnit vil designprocessen, implementering samt test af database-delen for Fridge app’en blive beskrevet, samt de overvejelser der er blevet gjort for database-tilgang fra applikationen.

### Design

I dette afsnit vil designprocessen blive beskrevet, hvor de designovervejelser der er gjort i forhold til DAL for Fridge App.

#### Teknologi

Før at det var muligt at designe databasen, skulle der først vælges en teknologi, til Data Access Layer, hvor der ville blive brugt en relationel database. På dette tidspunkt, var det oplagte valg ADO.NET, da der var blevet undervist i Database-kurset omkring dette, den anden mulighed var Entity Framework. Entity Frameworket ville være klart lettere at arbejde med, men for læringens skyld blev det valgt at arbejde med ADO.NET og der ville igen blive kigget på Entity Framework i forhold til Web app’en.

#### Objekt model

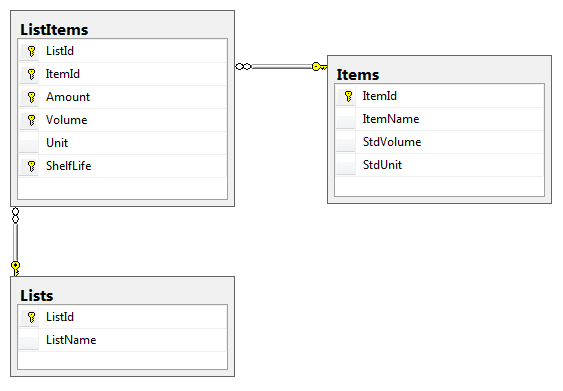
Der skulle findes en objekt model til systemet, hvorpå data kunne gemmes i den relationelle database. Til dette blev der anvendt DDS-Lite. Det var ønsket at man kunne opretholde indtil flere lister, hvorpå disse lister indeholdte varer. Udover det, var det ønsket at man kunne gemme varetyper som f.eks. mælk og æg i systemet, for at brugeren ikke kunne indskrive disse, hver gang der blev tilføjet en vare til en liste. På dette grundlag, blev der lavet en objekt model som ses i Figur 1.

C:\Users\Mathis\Desktop\SmartFridgeV1.jpg

Figur Objekt model

Her er der udnyttet et Mange-til-mange forhold mellem List (liste) og vores Item (varetype), hvori at ListItem, er selve den vare der er på en liste. I Figur 2, ses en illustration af objekt modellen med attributter. Her ses det at man har lagret en varetype i tabellen ’Items’, hvori man har standard værdier for mængde og unit. ’ListItems’ tabellen indeholder attributterne for selve varen, og man ser der er en composite key. Dette er for man kan f.eks. have mælk med forskellige holdbarhedsdatoer og forskellige mængde vare.

Da objektmodellen er fastlagt, er det nu muligt at overveje nogle designmønstre for ens DAL.



Figur Objekt model med attributter

#### Repository Pattern

I mange applikationer, er det forretningslogikken der tilgår data og databaser. Dog kan direkte tilgang introducere indtil flere problemer, som gentagne kode, højere risiko for programmeringsfejl, dårlig abstraktion og lav testbarhed.

For at kunne løse disse problemer, introduceres et Repository mønster til vores implementering. Repository mønstret separerer logikken mellem forretningslogikken og datakilden, som en database. En illustration af dette, ses i Figur 1. Repositoryet sørger derfor at tilgå datakilden for data og mapper data til forretningslogikken. Ved at logikken separeres, opnås der at alt data tilgang foregår igennem Repositoryet, der opnås en højere abstraktion. Herved har man opnået højere testbarhed af forretningslogikken (MSDN, 2015).



Figur Repository Pattern (MSDN, 2015)

Repository mønstret tilbyder et sted man har sin datatilgang, hvor der kan gøres brug af ADO.NET data providers, hvilket gør oplagt at anvende sammen med ADO.NET. Det vil også skabe højere abstraktion, samt at gøre DAL langt mere testbart.

Anvendelse af mønstret vil blive beskrevet i implementeringsafsnittet for databasen.

#### Unit of Work

Unit Of Work er et design mønster, hvorpå man holder styr på hvilke transaktioner forretningslogikken laver til databasen. I et Unit Of Work, gøres alle de transaktioner som forretningslogikken ønsker at gøre, og derefter vil Unit of Work holde styr på hvad der skal ske på database-siden, og commit og rollback de ændringer, som forretningslogikken har gjort. På den måde opnås der noget logik, kan opretholde en liste af ændringer der skal ske og give det videre til databasen (Fowler, 2015).

Unit of Work er oplagt at anvende sammen med Repository mønstret, hvorpå det er muligt at have et Unit of Work, hvor Repositoryet anvendes til database-transaktioner, hvorefter transaktionerne kan blive commitet.

Anvendelse af mønstret vil blive beskrevet i implementeringsafsnittet for databasen.

#### Synkronisering

For at muliggøre at der både er Fridge app’en og Web app’en i systemet, og sørge for at Fridge app er funktionel uden internetforbindelse, var det blevet en nødvendighed at implementere synkronisering af en lokal og en ekstern database.

Til dette formål, er Micsoft Sync Framework blevet valgt, da det er løsning der er mulig selv at implemente, uafhængigt af hvilken data provider der anvendes. En illustration af synkronisering mellem lokal database og ekstern database på en webserver ses på Figur 4.



Figur Illustration af database synkronisering

#### Endeligt design

Efter de designovervejelser der er blevet gjort i de forrige afsnit, er der blevet udarbejdet følgende klassediagram. Klassediagrammet kan ses i bilag XX.

### Implementering

### Test

## Database - Webapp

### Design

#### Façade mønster

Façade er et design mønster, hvorpå man skaber et simpelt interface til et kompleks subsystem.

### Implementering

### Test