CDIO FINAL – Gruppe 18

Fadl Matar: s195846

Khaled Zamzam: s195487

Sid Ali Mounib: s195484

Bashar Khaled Bdewi: s183356

Git repo: https://github.com/ZamzamGit/CDIO_FINAL

Indhold

1. Indledning

En medicinalvirksomhed ønsker at få udviklet et softwaresystem, der skal bruges til dokumentering af råvareafvejning, kvalitetskontrol og lagerstyring.

For at løse denne opgave stillet til os af medicinalvirksomhed, vil vi lave kravspecifikationer til de forskellige funktioner af vores system. Her analysere vi og designer programmet ved brug af forskellige artifacts, for at få en bedre forståelse af hvordan programmet skal implementeres og hvilke funktioner det skal indeholde. I rapporten benytter vi os af værktøjet Lucidchart, for at lave UML-diagrammer til nogle af de forskellige artifacts der befinder sig i analyse og design delen. Til sidst skal dette system også kunne vises i form af en hjemmeside.

2. Kravspecifikation

2.1 Ikke funktionelle krav (Bashar s183356)

ID	Beskrivelse
KIF1	Systemet skal have en hurtig responstid.
KIF2	Systemet skal være let for brugeren at bruge.
KIF3	Programmet skal være nemt at vedligeholde.

2.2 Funktionelle krav (Bashar s183356)

ID	Beskrivelse
KF 1	Systemet skal indeholde 4 roller (Administrator, Farmaceut, Produktionsleder og Laborant).
KF 2	Systemet skal kunne validere den pågældende bruger og kun give brugeren adgang til de rettigheder han/hun har fået givet
KF 3	Administrator skal kunne administrere brugerne (oprette, rette, fjerne og vise brugere i systemet).
KF 4	Farmaceut skal kunne administrere råvarer, recepter, råvarebatches og produktbatches i systemet.
KF 5	Produktionsleder skal kunne administrere råvarebatches og produktbatches og har tillige laborantens rettigheder.

KF 6	Laborant skal kunne foretage selve afvejningen
KF 7	Systemet skal indeholde et web interface (GUI) til administrator, farmaceuten, produktionsleder og laborant

KF 8	En database med oplysninger om brugere, råvarer, råvarebatches, recepter, samt planlagte og færdigproducerede produktbatches.

2.3 MoSCoW (Fadl s195846)

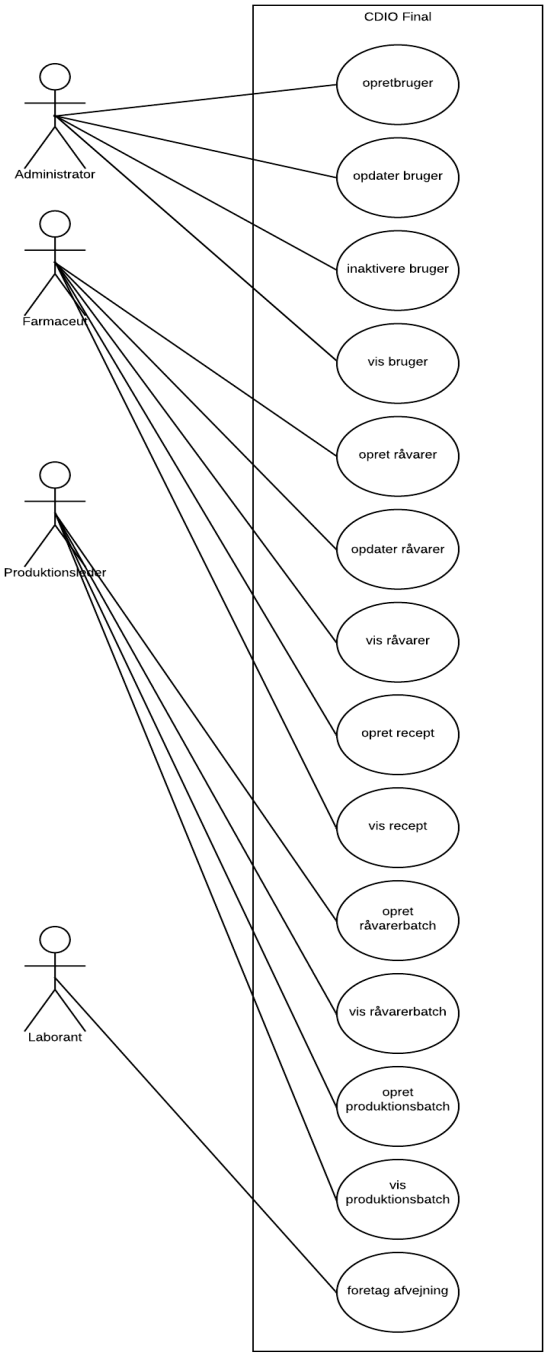
I denne tabel har vi opdelt vores funktionelle krav i en MoSCoW.

1. Administrator skal kunne administrere brugerne (oprette, rette, fjerne og vise brugere i systemet). (Must have)
2. Farmaceut skal kunne administrere råvarer, recepter, råvarebatches og produktbatches i systemet. (Must have)
3. Produktionsleder skal kunne administrere råvarebatches og produktbatches og har tillige laborantens rettigheder. (Must have)
4. Brugerens skal have 4 roller at vælge imellem (administrator, farmaceut, produktionsleder, laborant). (Must have)
5. Laborant skal kunne foretage selve afvejningen. (Must have)
6. Systemet skal indeholde 4 roller (Administrator, Farmaceut, Produktionsleder og Laborant). (Must have)
7. Systemet skal indeholde et web interface (GUI) til administrator, farmaceuten, produktionsleder og laborant. (Must have)
8. Systemet skal kunne validere en bruger af systemet før han kan få adgang til siderne. (Should have)
9. Systemet skal indeholde et web interface (GUI) til administrator, farmaceuten, produktionsleder og laborant. (Must have)
10. Lav en løsning som gemmes i en SQL database. (Should have)

3. Analyse

3.1 Use cases

3.1.1 Use case diagram (Fadl s195846)



3.1.2 Use case tabel med beskrivelse (Bashar s183356)

ID	Beskrivelse
UC1	Oprette bruger foretages af *administrator* aktøren. Denne skal kunne oprette brugere i systemet.
UC2	Rette(opdatere) bruger foretages af *administrator* aktøren. Denne skal kunne rette i en bruger.
UC3	Inaktivere bruger foretages af *administrator* aktøren. En bruger, der én gang er oprettet i systemet, kan ikke slettes men kan inaktiveres.
UC4	Vise bruger foretages af *administrator* aktøren. Denne skal kunne vise alle brugere i systemet
UC5	Oprette råvare Foretages af farmaceut aktøren. Denne skal kunne oprette råvarer i systemet. En råvare defineres ved et råvareNr (brugervalgt og entydigt) og navn. (d.v.s. råvareNr skal IKKE autogenereres.)
UC6	Opdatere råvare Foretages af farmaceut aktøren. Denne skal kunne rette råvarer i systemet.
UC7	Vis råvarer Foretages af farmaceut aktøren. Denne skal kunne vise den pågældende råvarer i systemet.

UC8	Oprette recept Foretages af farmaceut aktøren. Denne skal kunne oprette recepter i systemet. En recept defineres ved et receptNr (brugervalgt og entydigt), navn samt en sekvens af receptkomponenter.
UC9	Vise recept foretages af farmaceut aktøren. Denne skal kunne vise recepter i systemet.
UC10	Oprette råvarebatch Foretages af produktionslederen. Denne skal kunne oprette råvarebatches i systemet. En råvarebatch defineres ved et råvarebatchNr (brugervalgt og entydigt) samt mængde og leverandør.
UC11	Vise råvarbatch Foretages af produktionslederen. Denne skal kunne vise råvarebatches i systemet.
UC12	Oprette produktbatch Foretages af produktionslederen. Denne skal kunne oprette produktbatches i systemet. En produktbatch defineres ved et produktbatchNr (entydigt), nummeret på den recept produktbatchen skal produceres udfra, dato for oprettelse, samt oplysning om status for batchen. Status kan være: oprettet / under produktion / afsluttet. Afvejningsresultater for de enkelte *receptkomponenter* gemmes som en *produktbatchkomponent* i *produktbatchen* i takt med at produktet fremstilles.

	Når produktionslederen har oprettet et nyt *produktbatch* ***udprintes på papir*** *denne* og uddeles til en udvalgt laborant. Laboranten har herefter ansvaret for produktionen af batchet
UC13	Vise produktbatch Foretages af produktionslederen. Denne skal kunne vise produktbatches i systemet.
UC14	Foretage afvejning Foretages af laboranten. Denne skal kunne afveje de pågældende råvare.

3.1.3 Aktør liste (Bashar s183356)

Aktør	**Mål**
Administrator	Administrerer brugerne i systemet.
Farmaceut	Varetager administrationen af *råvarer* og *recepter* i systemet og har tillige produktionslederensrettigheder.
Produktionsleder	Varetager administrationen af *råvarebatches* og *produktbatches* og har tillige laborantens rettigheder.
Laborant	Foretager selve afvejningen.

3.1.4 Brief-dressed use case beskrivelser (Fadl s195846)

Brief use case af "Opret råvare":

Farmaceuten logger ind via startsiden. Personen vælger så her at oprette en råvare ved at trykke på "opret/opdater råvare" knappen, hvor han/hun så her skal angive den nye råvarers navn samt ID. For at gemme disse ændringer trykker han/hun på "opret råvare" knappen.

Brief use case af "Opdater råvare":

Farmaceuten logger ind via startside. Personen vælger så her at opdatere en råvare ved at trykke på "opret/opdater råvare knappen", hvor han/hun så her skal trykke på knappen "opdater råvare", for så at indtaste det id på den råvare der skal opdateres, her skifter farmaceuten navnet på råvaren og bekræfter denne ændringer ved at trykke på "bekræft" knappen.

Brief use case af "Foretag afvejning":

Laboranten identificerer sig i systemet. Herefter svarer systemet tilbage med laborantens navn. Laboranten indtaster på den produktbatch der skal udfyldes. Beholderens vægt registreres i produktbatchkomponenten under tara. Systemet oplyser herefter hvilken råvare der skal afvejes.

3.1.5 Fully-dressed use case beskrivelser (Fadl s195846)

Use case: Opret bruger

Scope: Medicinalvirksomhed

Niveau: Brugermål

Primær aktør: Administrator

Interessenter:

- Administrator: ønsker et administrationsmodul.

Forudsætninger/preconditions: Brugeren skal ikke have oprettet sig i systemet før.

Successkritier/postconditions: Brugeren er blevet oprettet.

Vigtigste succes-scenarie:

1. Åbner siden i en web-browser.
2. Vælger knappen hvorpå der står administrator.

3. Tryk på opret bruger.
4. Indtast ID
5. Indtast brugernavn
6. Indtast initialer
7. Indtast CPR
8. Vælg rolle.
9. Tryk på gem ændringer knappen.
10. Brugeren er nu tilføjet.

Udvidelser:

1. Administratoren indtaster ID.

   1.1 Hvis id'et ikke er i intervallet 1-999, bedes han skrive det om igen.
2. Administratoren indtaster brugernavn.

   2.1 Hvis brugernavnet ikke er i intervallet fra 2-20 tegn langt, bedes han at skrive det om igen.
3. Administratoren indtaster initialer.

   3.1 Hvis initialerne ikke er i intervallet 2-4 tegn langt bedes han skrive det om igen.

Teknologi og dataliste: Enheder, såsom computer der kan åbne en hjemmeside, og kører html.

Frekvens: Use casen bliver taget i brug hver gang en ny bruger bliver oprettes

Use case: Inaktiver bruger

Scope: Medicinalvirksomhed

Niveau: Brugermål

Primær aktør: Administrator

Interessenter:

- Administrator: ønsker et administrationsmodul.

Forudsætninger/preconditions: Brugeren skal være oprettet i systemet.

Successkritier/postconditions: Brugeren er blevet inaktiveret.

Vigtigste succes-scenarie:

1. Åbner siden i en web-browser.
2. Vælger knappen hvorpå der står administrator.
3. Tryk på rediger.
4. Indtaster ID på bruger der skal redigeres
5. Tryk på rediger
6. Skift status fra aktiv til inaktiv.
7. Tryk på bekræft knappen.
8. Brugeren er nu inaktiveret og kan ikke få adgang til systemet.

Udvidelser:

1. Administratoren indtaster ID.

    1.1 Hvis id'et ikke er befinder sig i systemet, bedes han skrive det om igen.

Teknologi og dataliste: Enheder, såsom computer der kan åbne en hjemmeside, og kører html.

Frekvens: Use casen bliver taget i brug hver gang en ny bruger skal inaktiveres.

3.2 Navneanalyse og nødvendige klasser (Bashar s183356)

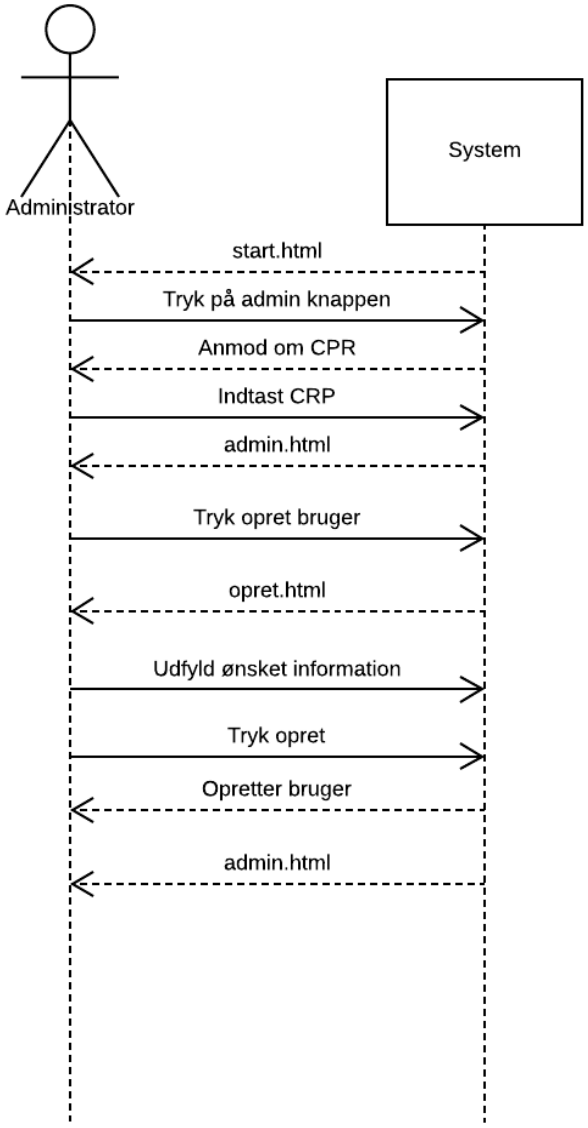
Udefra vores use case beskrivelser kan vi analysere os frem til nogle nøgleord vi skal bruge fremadrettet i rapporten. De forskellige navneord er følgende:

- *Administrator*
- *Farmaceut*
- *Produktionsleder*
- *Laborant*
- *Bruger*
- *Råvare*
- *Recept*
- *Råvarebatch*
- *Produktbatch*
- *Afvejning*

Ud fra analysen af vores use cases kan vi konkluderet at vi skal bruge følgende klasser:

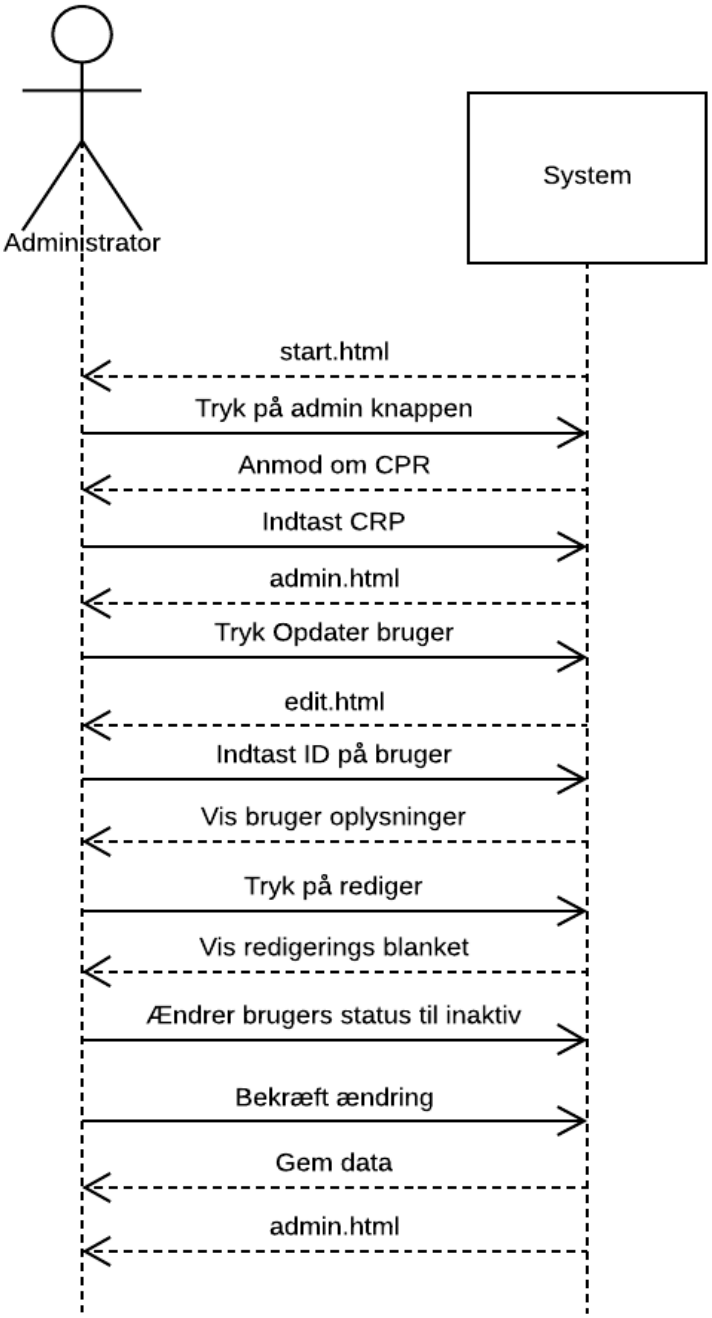
- *Administrator*
- *Farmaceut*
- *Produktionsleder*
- *Laborant*
- *Råvare*
- *Recept*
- *Råvarebatch*
- *Produktbatch*

3.3 Systemsekvensdiagram (Sid s195484)



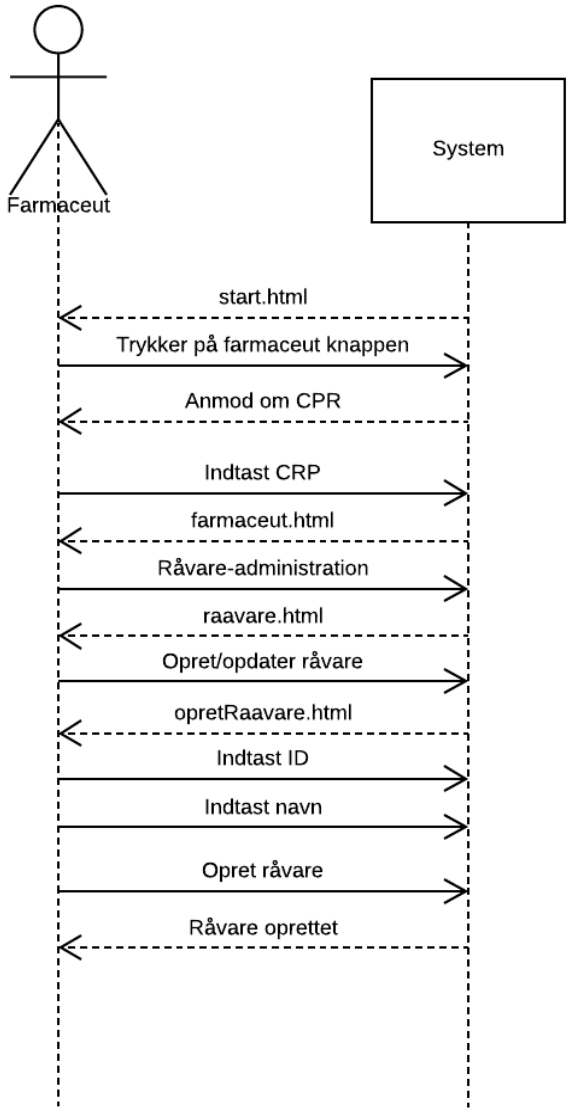
Diagrammet ovenfor er et systemsekvensdiagram mellem administratoren og systemet formålet med diagrammet er at vise hvordan administratoren interagere med systemet. Diagrammet ovenfor er specielt designet til at repræsentere administratorens opret bruger use case. I denne use case kan vi se at systemet starter med at vise start.html siden. Dernæst vil administratoren trykke på admin knappen, hvor systemet derfor anmoder om CPR-nummeret. Efter administratoren indtaster sit CRP-nummer vil admin.html siden blive vist. Administratoren kan herefter trykke på opret bruger knappen hvilket henter opret.html siden. Efter administratoren

har udfyldt informationerne om den pågældende bruger og trykket på opret, vil systemet oprette brugeren og videresende administratoren til admin.html.



Illustrationen ovenfor er det andet systemsekvensdiagram vi har konstrueret, hvilket er designet ud fra use casen inaktiver bruger. Ovenfor kan vi igen se interaktionen mellem administratoren og systemet. De første trin er meget ens til tidligere diagram, hvor administratoren logger ind med sit

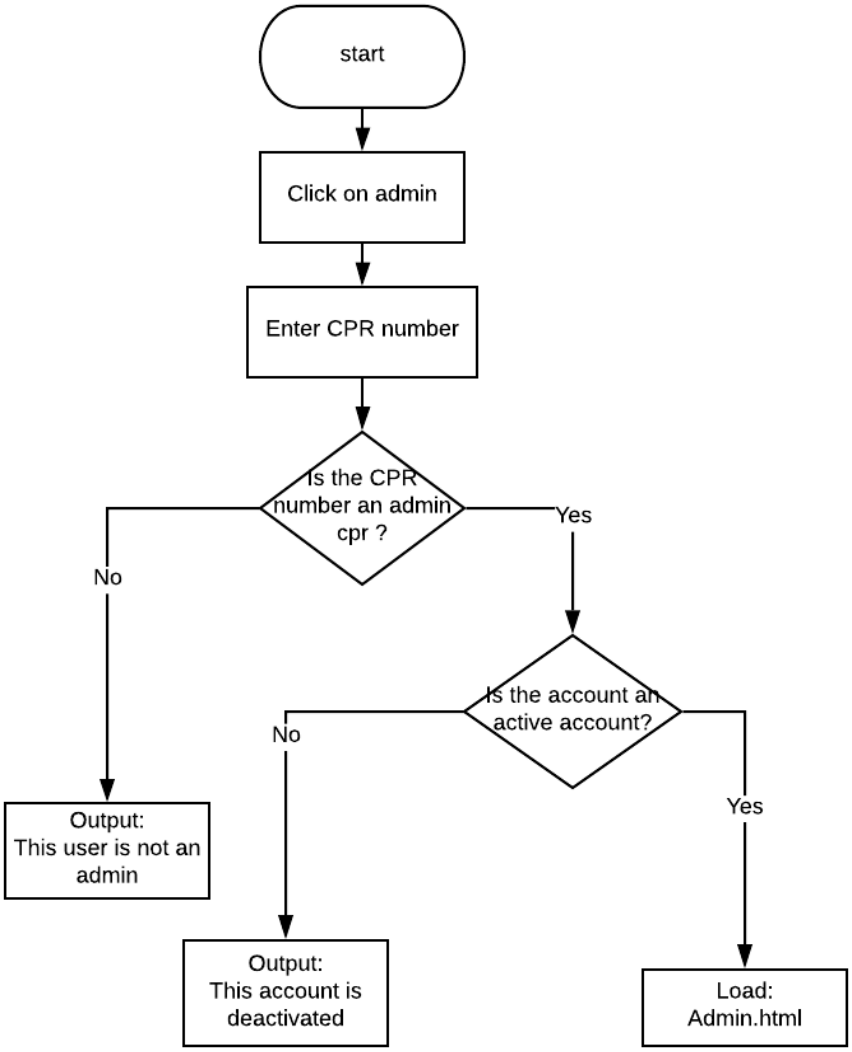
CPR-nummer og kommer ind på admin.html siden. I dette tilfælde vil administratoren trykke på opdater bruger, som systemet besvare ved at hente edit.html siden. Administratoren kan nu indtaste id'et på brugeren som skal inaktiveres. Systemet vil dernæst hente bruger oplysningerne til administratoren. Administratoren kan nu trykke på rediger, hvor systemet vil bringe en redigerings blanket. Ved at ændre brugerens status til inaktiv kan administratoren inaktivere brugeren. Efter administratoren bekræfter ændringen vil systemet gemme dataet og hente admin.html siden.



Det sidste systemsekvensdiagram repræsenterer farmaceut use casen opret råvare. På samme måde som administratoren skal farmaceuten indtaste sit CRP for at komme ind på farmaceut.html

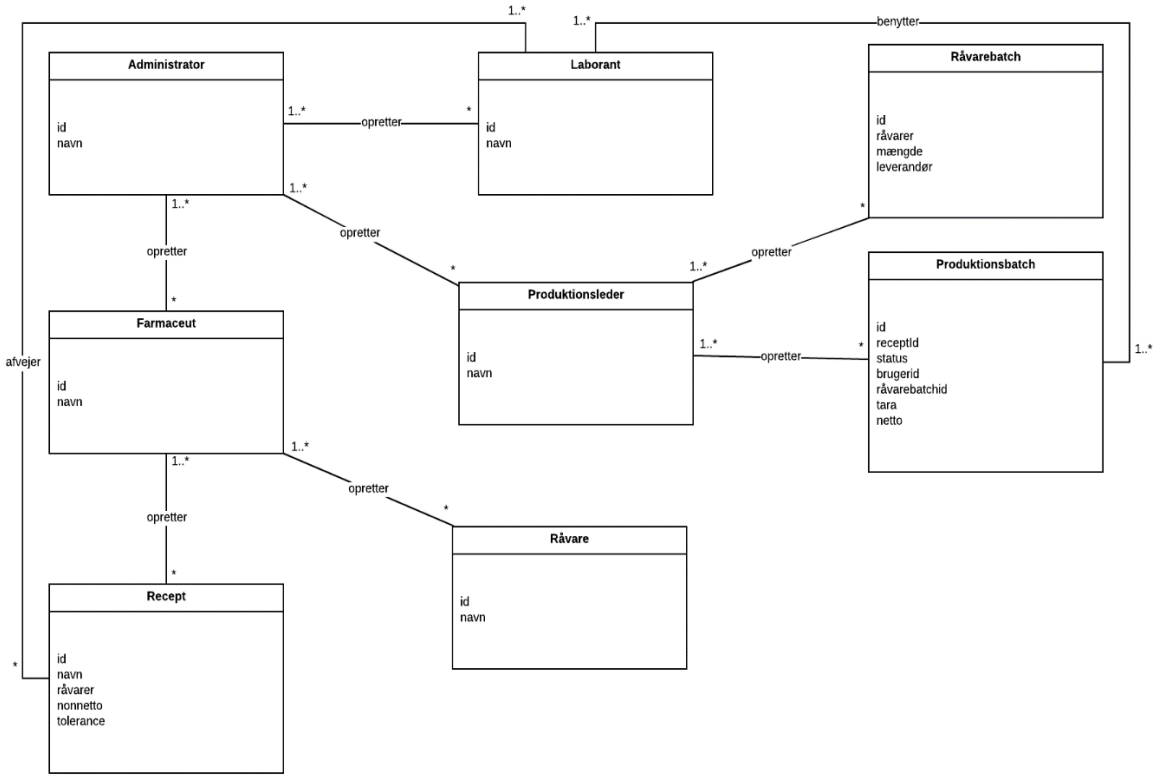
siden. Dernæst kan farmaceuten trykke på råvare administration, hvilket forårsager at systemet henter raavare.html siden. Herefter kan farmaceuten trykke på opret/opdater råvare som frembringer opretRaavare.html siden. Farmaceuten kan nu indtaste ID'et og navet på råvaren. Til sidst kan farmaceuten trykke på opret råvare og systemet vil oprette råvaren.

3.4 Flowchart diagram (Sid s195484)



Diagrammet ovenfor repræsentere et flow chart diagram over admin login handlingen. Her kan vi se de forskellige scenarier der kan opstå. Efter administratoren har indtastet sit CRP, kan CRP'en matche en admin brugers CRP. Hvis dette CPR ikke matcher vil den give et output/alert som fortæller, at brugeren ikke er en admin. Hvis den derimod matcher vil den nu tjekke om brugeren er aktiv eller inaktiv, hvor den kun loader admin siden, hvis brugeren er aktiv.

3.5 Domæne model (Khaled s195487)



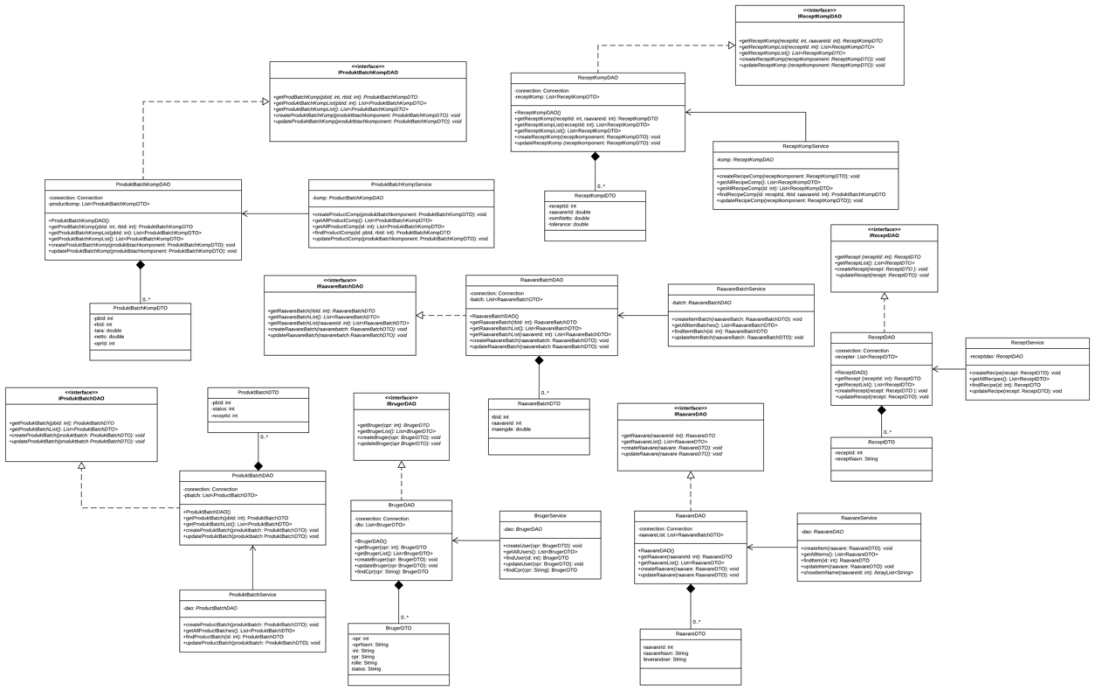
Her har vi en domænemodel for vores system. En domænemodel er den vigtigste model i en objektorienterende analyse, som illustrerer vigtige objekter fra den virkelige verden. Her ses relationer mellem objekter. Hvor en administrator har ansvaret for at oprette en bruger (såsom en laborant, farmaceut eller produktionsleder). En farmaceut har ansvaret for at oprette recepter og råvare. En produktionsleder har ansvaret for at oprette produktbatches og råvarebatches. En laborant har ansvaret for at afveje råvarer, som er vist i en recept.

En domænemodel består af objekter, associeringer mellem objekter med multipliciteter og attributter i et objekt. Her ses der fx at en laborant har et navn og et id. En association er en relation mellem to klasser, der angiver en meningsfuld sammenhæng. Her ses der fx at en farmaceut opretter en råvare, og hvor flere råvare kan blive oprettet af en farmaceut.

4. Design

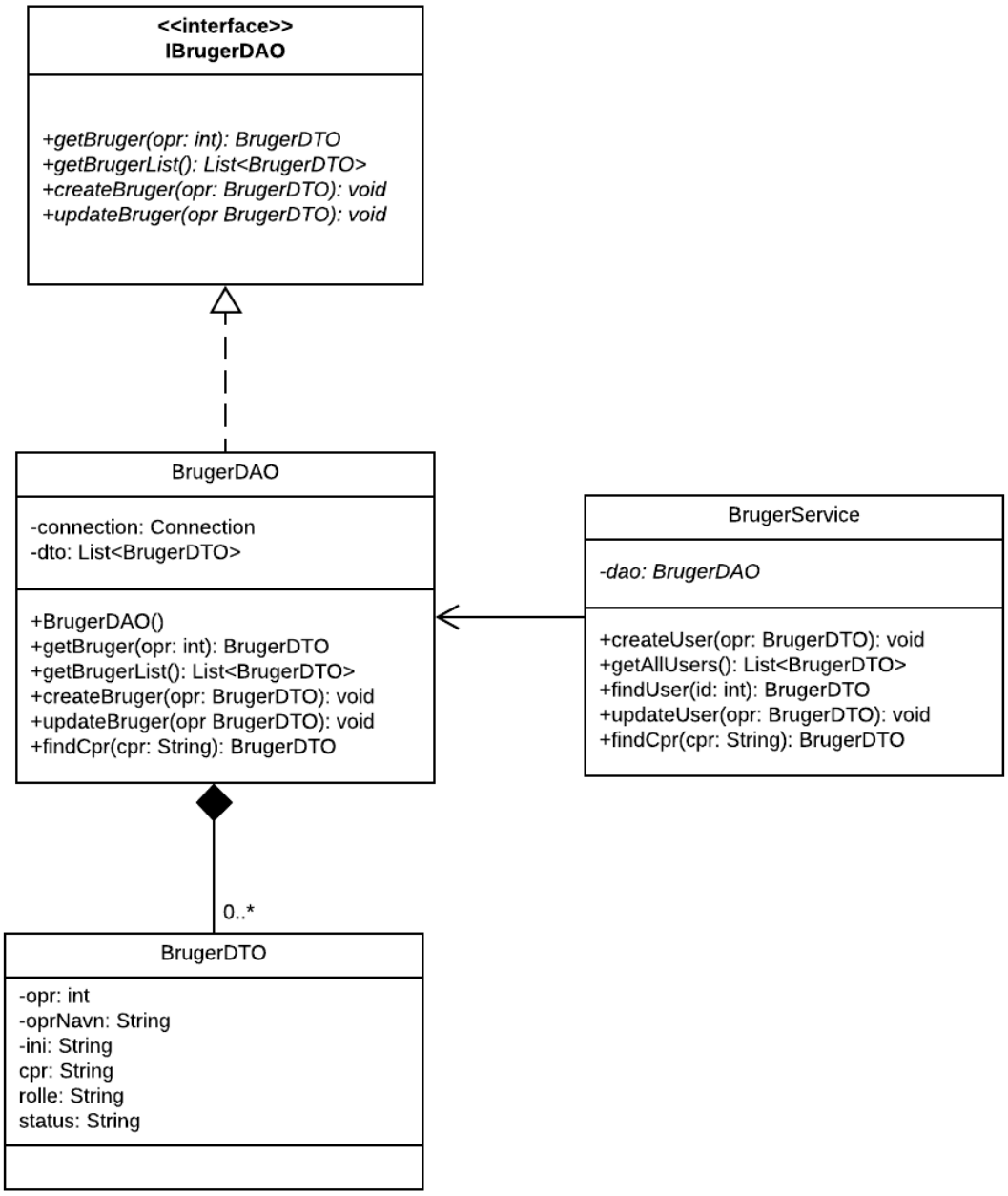
4.1 Designklassediagram (Khaled s195487)

Billede af hele klassediagrammet (zoom ind for at se den tidligere) eller find diagrammet via dette link: https://app.lucidchart.com/documents/edit/ddafd7cb-46c3-4c3a-b6c9-25fba5953d72/0_0?beaconFlowId=D56542FD982BBB4C



I modsætning til en domænemodel viser et designklassediagram software klasser og deres relationer. Et designklassediagram, illustrerer klassernes attributter og metoder.

For at få et bedre overblik over klassediagrammet, nu hvor det er stort, vil vi beskrive den nedenstående figur.



Den type for relation mellem klasserne BrugerDTO og BrugerDAO er komposition (den sorte diamant). Her kan klassen BrugerDTO kun være en del af BrugerDAO. Altså BrugerDTO kan ikke eksistere uden BrugerDAO, da det er BrugerDAO der har ansvaret for at oprette, vise og opdatere

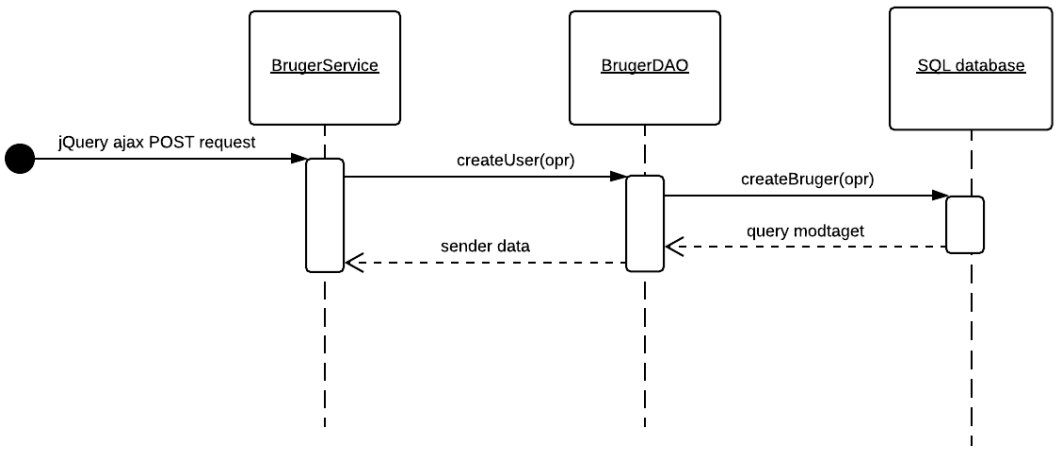
brugere. Som der ses, er der også blevet sat en multiplicitet. Her bliver der vist at BrugerDAO, kan have 0 eller flere brugere.

Relationen mellem BrugerDAO og IBrugerDAO, viser at BrugerDAO implementere IBrugerDAO interfacet. Hvert interface er et sæt af defineret abstrakte metoder med offentlig synlighed. BrugerDAO refererer til interfacet IBrugerDAO, men ikke den konkrete implementering.

I relationen mellem BrugerDAO og BrugerService, ses der, at BrugerService-klassen har en attribut som refererer til BrugerDAO klassen, hvor den via attributten vil benytte sig af metoderne i BrugerDAO.

4.2 Sekvensdiagrammer (Khaled s195487)

Sekvensdiagram til use case: Opret bruger

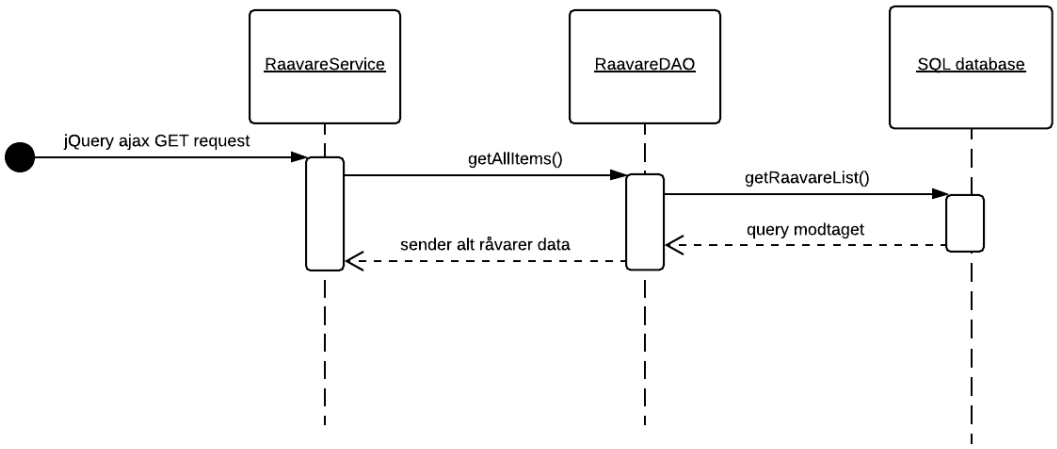


Overstående sekvensdiagram viser processen for når en administrator skal oprette en bruger. Lige så snart en administrator har indtastet nogle oplysninger på en ny bruger, og trykker på knappen, for at bekræfte oprettelse af brugeren, vil der blive kaldt en jQuery ajax POST request. POST bliver brugt til at oprette en ny ressource til serveren. Når ajax kaldet bliver udført, vil det blive sendt til BrugerService-klassen, som er en ressource klasse. Da det er en POST request, er det createUser(opr) metoden, som vil blive udført, hvor den derefter vil udføre metoden createBruger(opr) i BrugerDAO-klassen, som laver et SQL INSERT query til SQL databasen. Her vil dette query blive modtaget, hvor data bliver sendt til BrugerService klassen. Det der ikke bliver vist

i sekvensdiagrammet er, at data sendt til BrugerService klassen vil blive generet til JSON, hvor javaScript modtager det, og opdatere DOM'en. Her vil administratoren modtage beskeden "Bruger oprettet".

Når en administrator vil opdatere en bruger (her refererer vi til use casen *Opdatere bruger*), vil processen være præcis det samme. Her vil det være en PUT request i stedet for en POST, hvor updateUser(opr) metoden vil blive kaldt, derefter updateBruger(opr), som vil lave et SQL UPDATE query, og opdatere dataet.

Sekvensdiagram til use case: Vis råvarer



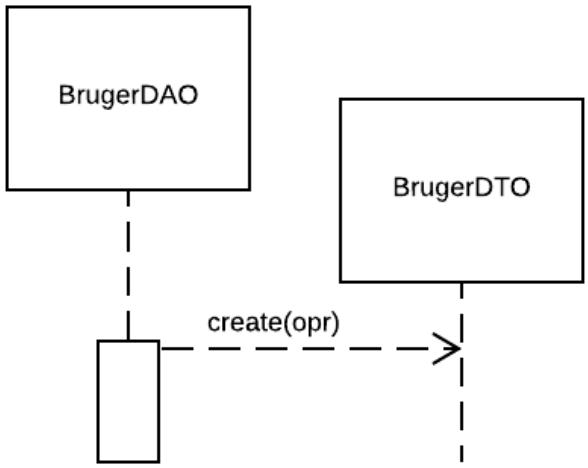
Her viser vi et sidste sekvensdiagram til use casen *Vis råvarer*, hvor oplysninger om alle råvare bliver vist automatisk i en tabel til farmaceuten. Dette sekvensdiagram minder om det tidligere viste sekvensdiagram. Da der i stedet skal læses data fra serveren, bruges der en GET metode. Her vil den sendes til RaavareService-klassen, som vil udføre getAllItems(), som vil udføre getRaavareList(), som laver en SQL SELECT query.

4.3 GRASP (Khaled s195487)

Ved at forstå GRASP kan man opnå et godt objekt-orienteret design. Med GRASP får man instruktioner for hvordan ansvaret skal fordeles mellem klasserne.
GRASP består af ni principper, men vi vil her kun beskæftige os med fem af dem.

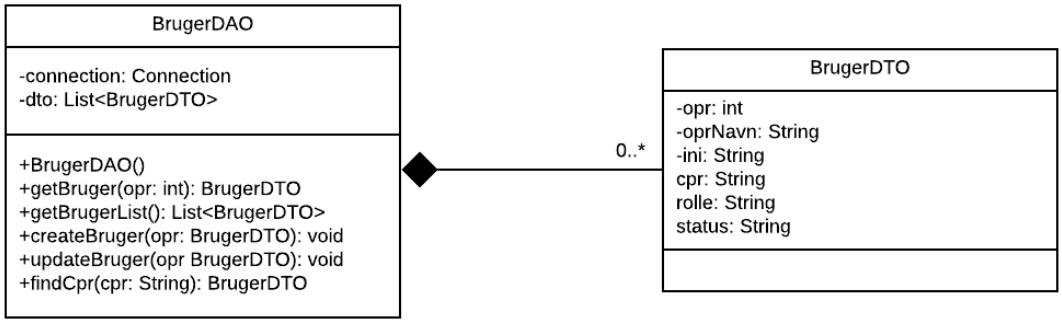
Creator

Dette GRASP-mønstre går ud på hvem der har ansvar for at oprette en instans af en anden klasse. I vores system er det alle DAO-klasser creators, da de har ansvaret for at oprette objekter af DTO-klasser.



Overstående figur viser et godt eksempel på en DAO-klasse (BrugerDAO), som opretter et objekt af en DTO-klasse (BrugerDTO).

Den nedstående figur (som også er vist i designklassediagrammet), viser også at BrugerDAO, kan oprette flere objekter af BrugerDTO. Ansvar bliver opfyldt af metoder, og her har createBruger(opr) ansvaret for at oprette objekter af BrugerDTO.



Information Expert

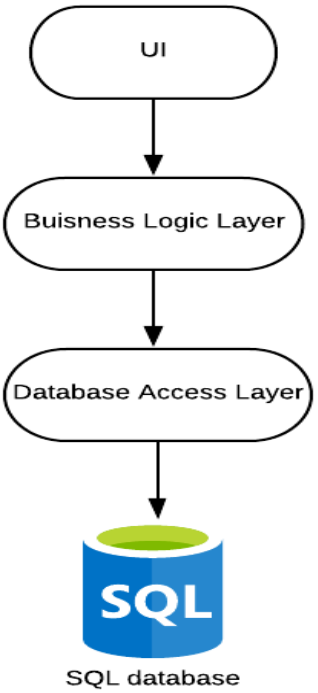
Under Information Expert, tildeles der et ansvar til en klasse, der har informationer til at opfylde ansvaret. Et objekt oprettet af en Creator er typisk en Information Expert. Her er en DTO-klasse en god kandidat til en Information Expert.

Overstående figur viser et eksempel. BrugerDAO har metoden updateBruger. I metoden skal der findes et ID på en bruger, for at tjekke om brugeren eksisterer. Når updateBruger metoden bliver udført, vil getter-metoden getOprId i BrugerDTO blive udført, for at tjekke om der er en bruger med denne ID. Her er det BrugerDTO der har alle informationer om brugerne.

Lav kobling

Det er vigtigt at man har lav kobling i sin kode. Lav kobling opnås når der er lav afhængighed mellem klasser. Altså påvirkninger ved ændringer reduceres og det bliver nemmere at genbruge samt lettere at forstå. Ansvar skal tildeles for at opnå en lav kobling.

I vores kode gør vi brug af interfaces, som er en vigtig mekanisme til at opnå lav kobling. Her kan to forskellige klasser gør brug af det samme interface, uden det påvirkes ved ændringer, da klasserne ikke kender til den konkrete implementering.



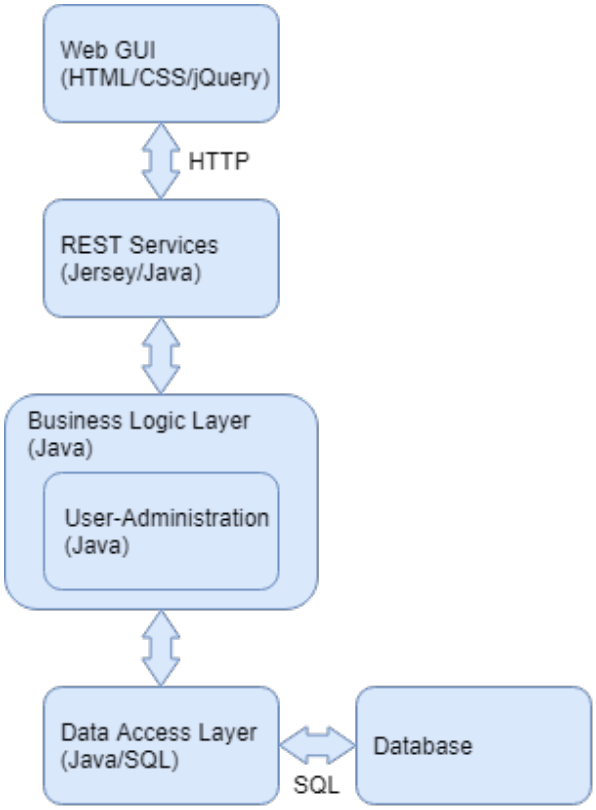
For også at opnå lav kobling, gør vi også brug af tre-lags-modellen (se overstående figur). Her har vi vores UI (User Interface), som er browseren, hvor brugeren foretager sig nogle aktioner. Vi har Buisness Logic Layer, her har vi vores Java kode, hvor logikken ligger. Til sidst har vi vores Database Access Layer, hvor vi bruger Java og SQL til at hente eller indsætte data i vores SQL-database.

Høj binding

Udover over lav kobling, er det også vigtigt at have høj binding i koden. Man opnår høj binding ved at tildele ansvar. Altså det øges ved fordeling af opgaver. Her har DTO-klasser ansvar for at have informationer om brugere, råvarer m.m., DAO-klasser har ansvar for at udføre nogle funktioner til DTO-klasserne, såsom opret, vis og opdatere. Service-klasserne har ansvar for at udføre metoderne i DAO-klasserne i GUI'en. Med sådan en fordeling mellem klasserne opnår vi høj binding.

Controller

Dette GRASP mønstre går ud på hvilket objekt der befinder sig under brugergrænsefladen, og er ansvarlig for at modtage og koordinere inddata.



Overstående figur viser arkitekturen for hele det overordnet system. Udover GUI, Buisness Logic Layer og Data Access Layer (som blev vist i tidligere figur), har vi også REST Services. I vores implementering er vores Java Service-klaser REST Services. Når en bruger derfor foretager sig noget i GUI'en, bliver en jQuery ajax funktion udført, hvor den kontakter en af Service-klasserne, som udfører en metode i en af DAO-klasserne. Udefra denne forklaring kan vi komme frem til at Service-klasserne er vores Controllers, og har ansvaret for at sende og modtage data.

5. Implementering

5.1 Implementation af oprettelse af bruger (Sid s195484)

Under implementeringen vil forskellige essentielle dele af koden blive forklaret. Herunder vil der blive dannet en dybere forklaring af koden og hvordan de forskellige klasser og metoder giver programmet mulighed for at tilfredsstille de forskellige mål fra de forskellige bruger.
Den første del af forklaringen vil tage udgangspunkt i implementeringen af usecasen opret bruger.

```
<div id = "opret">
    <form id = "create-form">
        <input type = "text" class = "id login-box" name = "oprId" placeholder="ID"
minlength="1" maxlength="3">
        <br>
        <input type = "text"  class = "navn login-box" name = "oprNavn"
placeholder="Navn" minlength="2" maxlength="20">
        <br>
        <input type = "text" class = "ini login-box" name = "ini" placeholder="Initial"
minlength="2" maxlength="4">
        <br>
        <input type = "text" class = "cpr login-box" name = "cpr" placeholder="CPR"
maxlength="11">
        <br>
        <select name = "rolle">
            <option disabled selected hidden>Vælg rolle</option>
            <option>Administrator</option>
            <option>Farmaceut</option>
            <option>Produktionsleder</option>
            <option>Laborant</option>
        </select>
        <br>
        <select name = "status">
            <option disabled selected hidden>Vælg status</option>
            <option>Aktiv</option>
            <option>Inaktiv</option>
        </select>
        <br>
        <button id ="add-user">Opret</button>
    </form>
</div>
```

Denne kode repræsentere fundamentet for vores opret side. Der er blevet dannet en div som indeholder 4 input bar og 2 select bar. De 4 inputs giver os mulighed for at angive personens ID, Navn, Initial og CRP mens de 2 selects giver os muligheder for at angive brugerens rolle og status. Vha. javascript og json library kan vi nu bruge disse inputs.

```
function createUser() {
    $('#add-user').on('click', function (e) {
        e.preventDefault();
        var user = $('#create-form').serializeJSON();
        $.ajax({
            url: "system/bruger",
            method: 'POST',
            contentType: "application/json",
            data: user,
            success: function () {
                    alert("Bruger oprettet");
                    loadPage('admin.html');

            },
            error: function () {
                alert("Allerede en person med dette ID");
            }
        });
    });
}
```

De tidligere inputs bliver brugt ved admin.js javascript filen. Under denne fil findes en funktion som hedder createUser(). Første bliver der tjekket om knappen med id'en add-user bliver klikket. Derefter er der en preventDefault metode som bliver kørt på funktionen som forhindre den i at udføre dets standard funktion. Senere vil dataet fra html filen gemt under var user. Dernæst bliver en ajax funktion oprettet hvor url'en af hvor dataet skal blive sendt bliver angivet, samt hvilken metode den skal bruge og datatypen. Sidst bliver en succes og error funktion brugt som enten fortæller at brugeren blev oprettet og sender brugeren af hjemmesiden tilbage til admin siden. Ellers fortæller den at der allerede er en bruger med den angivet id.

```
package service;
import ...

@Path("bruger")
public class BrugerService {
    private static BrugerDAO dao = new BrugerDAO();

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void createUser(BrugerDTO opr) throws DALException {
        dao.createBruger(opr);
    }
```

Dataet fra ajax funktionen bliver nu sendt til java koden. Vi kan se at denne kode sidder i system pakken og BrugerService klassen har en path som hedder bruger. Server og bruger er det samme url som vi angav ved url linjen af ajax koden. Vi angav også at det var en post metode hvilket bringer os createUser metoden. Denne metoder bruger en anden metode som hedder CreateBruger som findes under BrugerDao.

```
public void createBruger(BrugerDTO opr) throws DALException {
    String sql = "INSERT INTO user VALUES(?,?,?,?,?,?)";

    for (BrugerDTO b: dto) {
        if(b.getOprId() == opr.getOprId()) {
            throw new DALException("Allerede en bruger med denne ID");
        }
    }
    try {

        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt(1, opr.getOprId());
        preparedStatement.setString(2, opr.getOprNavn());
        preparedStatement.setString(3, opr.getIni());
        preparedStatement.setString(4, opr.getCpr());
        preparedStatement.setString(5, opr.getRolle());
        preparedStatement.setString(6, opr.getStatus());
        preparedStatement.executeUpdate();
```

Under denne metode definerer vi en string sql som bliver brugt som in sql query. I denne string indeholder vores sql input hvor spørgsmålstegnene repræsenterer dataet der bliver inputtet. Der er også vigtigt at man ikke indsætter 2 bruger med samme id da det er en primary key. Derfor er der blevet lavet et for loop som går igennem alle brugere i databasen og ser om det angivet id allerede er i brugt. Sidst er der en try catch som prøver at indsætte vores input i sql koden. Dette giver os mulighed for at indsætte hjemmesidens brugere i en database så de kan hentes selv ever tomcat serveren er genstartet.

Vha. de øvre metoder kunne vi modtage data fra hjemmesiden ved brug af html, hvor vi senere brugte javascript til at sende dataet videre til java, hvor vi kunne connecte til en mysql database, hvilket giver os mulighed for at oprette brugere.

5.2 Implementation af opdatering af en råvare (Fadl s195846)

Billedet ovenfor er en illustration af vores html kode over vores opret/opdater side for en råvare. Dette er fundamentet for denne hjemmeside. Som der kan ses på billedet, har vi tre div'er nemlig en for oprettelsen for af en råvare, en div der er en del af opdateringen hvor man kan skrive hvilken råvare der skal opdateres og til sidst en div for opdateringen af en råvare, hvor man her så kan gå ind og ændre på f.eks. navnet af en råvare. Vi har til at starte med givet de to div'er der dækker over opdateringen af en råvare en display attribut der angivet som værende "none". Dette kan vi ændre på når der bliver klikket på en knap ved hjælp af Javascript.

```html
<a href="raavare.html">Gå tilbage</a>

<h1>Opret din nye råvare</h1>
    <div id = "raavare">
        <form id = "create-item-form">
            <input id = "a" class = "id opret-box" maxlength="3" minlength="1" name = "raavareId" placeholder="ID" type = "text">
            <br>
            <input id = "b" class = "navn opret-box"  maxlength="20" minlength="2" name = "raavareNavn" placeholder="Navn" type = "text">
            <br>
        </form>
        <button id ="opret-raavare">Opret råvare</button>
        <button id ="update-raavare">Opdater råvare</button>
        <div id="identifier">
            <form>
            <input id = "identifier-input" class = "id opret-box" maxlength="3" minlength="1" name = "raavareId" placeholder="Indtast ID" type = "text">
            </form>
            <button id ="identifier-btn">Enter</button>
        </div>
    </div>


    <div id="update">
        <form id = "update-item-form">
            <input id = "raavare-id" class = "id opret-box" maxlength="3" minlength="1" name = "raavareId" placeholder="ID" type = "hidden">
            <input id = "raavare-navn"class = "navn opret-box"  maxlength="20" minlength="2" name = "raavareNavn" placeholder="Navn" type = "text">
        </form>
        <button id ="bekræft-raavare">Bekræft</button>
    </div>
```

Vi benytter os af 5 inputs for både at kunne oprette og opdatere den pågældende råvare der er tale om. Derudover benytter vi os af 4 knapper til at kunne manipulere siden med.
Som sagt benytter vi os af javascript til at kunne manipulere hjemmesiden og kunne opdatere vores råvare. Måden vi blandt andet benytter det på er ved at fremvise vores opdaterings div frem på følgende måde:

```javascript
$("#update-raavare").click(function () {
    $("#identifier").slideToggle();
});
```

Det dette stykke kode her viser at når der trykkes på opdater knappen ville den div med id'et "identifier" blive vist ved hjælp af en slideToggle metode.

```
function updateItem() {
    $('#bekræft-raavare').on('click', function (e) {
        e.preventDefault();
        var item = $('#update-item-form').serializeJSON();
        $.ajax({
            url: "system/raavare",
            method: 'PUT',
            contentType: "application/json",
            data: item,
            success: function () {
                alert("Råvare opdateret");
            },
            error: function () {
                alert("Du kan ikke ændre et ID");
            }
        });
    });
}
```

Illustrationen ovenfor er et billede af metoden der sørger for at opdatere en råvare. Vi starter først med at oprette en click event på knappen med id'et "bekræft-raavare". Derefter lagre vi alt den information vi får givet ud fra de input vi har i formen med id'et "update-item-form" i en var ved navn item. Informationen i denne var har vi så omdannet til et JSon format.

Derefter opretter vi en ajax funktion hvor url'en som peger hen på den jersey vi har for råvaren som har en PUT metode der kører java metoden for at kunne opdatere en råvare. Derefter bliver slags metode den skal benytte samt hvilken datatype dette er. Til sidst angiver vi to forskellige funktioner nemlig en success funktion og en error funktion. Hvor den her giver farmaceuten besked om råvaren enten er blevet opdateret eller når man har ændret på noget som ikke skulle ændres.

```
@PUT
public void updateItem(RaavareDTO raavare) throws DALException {
    dao.updateRaavare(raavare);
}
```

Som nævnt tidligere giver vi ajax funktionen url'en til vores jersey for råvare services. Ind under dette har vi en række metoder der kan hjælpe os med at manipulere data for en råvare vi er dog blot intereseret i at kunne opdatere en råvare i dette eksempel, og da vi allerede har angivet hvilken metode ajax funktionen skal benytte fra denne jersey klasse, som er PUT metoden. Denne metode kører så java metoden for at kunne opdatere en råvare.

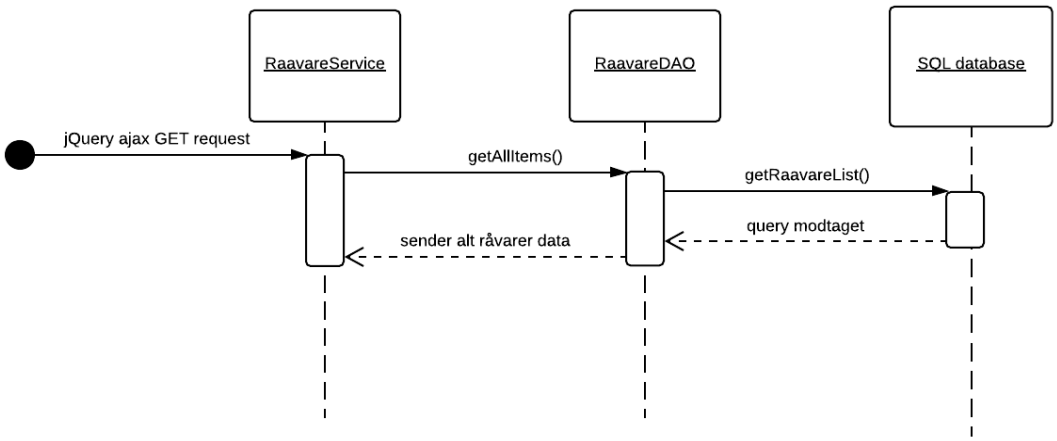
```
@Override
public void updateRaavare(RaavareDTO raavare) throws DALException {
    RaavareDTO r = getRaavare(raavare.getRaavareId());

    if(r == null) {
        throw new DALException("Ingen råvare med denne ID");
    }

    String sql = "UPDATE item SET ItemName = ? WHERE ItemID = ?";

    try {
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, raavare.getRaavareNavn());
        preparedStatement.setInt( parameterIndex: 2, raavare.getRaavareId());
        preparedStatement.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Først gemmer vi id'et for råvaren i r som er af datatypen RaavareDTO. Derefter tjekker vi om denne råvare eksisterer ellers kaster vi blot en exception istedet.

Derefter opretter vi et sql statement som vi benytter til at kunne opdatere den pågældende råvare i databasen. De spørgsmålstegn vi har i vores sql statement bliver udfyldt af henholdsvis parameter index 1 og parameter index 2, dette bekræfter vi så med en executeUpdate metode.

5.3 Implementation af visning af råvarer (Khaled s195487)

Her vil jeg tage udgangspunkt i sekvensdiagrammet med use casen *Vis råvarer*:



Når alle råvarer hentes fra SQL-databasen, vil de blive indsat i en HTML tabel:

```
<table id = "item-table-form">
  <thead>
  <tr>
    <th>Råvare ID</th>
    <th>Råvare Navn</th>
  </tr>
  </thead>
  <tbody id = "item-data">
  </tbody>
</table>
```

Når en bruger kommer ind på html-siden, vil en jQuery ajax funktion bliver udført automatisk. Nedestående billede viser funktionen for ajax kaldet.

```
function getAllItems() {
  $('#item-table-form tbody').empty();
```

```
  $.ajax({
    url: "system/raavare",
    method: "GET",
    contentType: "application/json",
    success: function (items) {
      $.each(items, function (i, item) {
        addItem(item);
      });
    },
    error: function () {
      alert("No items to load");
    }
  });
}
```

Det er funktionen getAllItems(), der bliver automatisk kørt. Metoden starter ud med, at tabellen tømmes, og derefter bliver det lavet en GET-metode til url'en: system/raavare, hvor "raavare" er path navnet for klassen RaavareService.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public List<RaavareDTO> getAllItems() throws DALException {
  return dao.getRaavareList();
}
```

Da det er en GET-metode, er det overstående funktion som bliver kørt, også som der er vist i sekvensdiagrammet, når der bliver lavet en jQuery ajax request.
Denne funktion kører metoden getRaavareList(), som befinder sig i klassen RaavareDAO.

```
public List<RaavareDTO> getRaavareList() throws DALException {
  raavareList = new ArrayList<>();
```

```
String sql = "SELECT * FROM item";

try {
    Statement st = connection.createStatement();
    ResultSet rs = st.executeQuery(sql);
    while (rs.next()) {
        RaavareDTO raavare = new RaavareDTO();
        raavare.setRaavareId(rs.getInt(1));
        raavare.setRaavareNavn(rs.getString(2));
        raavareList.add(raavare);
    }
} catch (SQLException e) {
    e.printStackTrace();
}

if(raavareList.isEmpty()) {
    throw new DALException("Ingen brugere");
}
return raavareList;
}
```

Overstående billede viser hele metoden for getRaavareList(). Her bliver der startet ud med at en ArrayList instantieres. Derefter laves der en String, som indeholder en SQL SELECT statement, som viser alle råvarer og eksekverer denne query.
Derefter laves der en ny instans af RaavareDTO inde i et while loop, og tildeler den værdier fra databasen, som derefter bliver tilføjet til ArrayListen. Denne while loop bliver ved, indtil tabellen for råvarer i databasen ikke har flere rækker at gå videre til.
Til sidst returner metoden ArrayListen.

I JavaScript metoden getAllItems(), vil denne metode blive kørt, hvor alle råvarer vil blive indsat i tabellen.

```
function addItem(item) {
    var $item = $('#item-data');
    $item.append('<tr>' +
        '<td>' + item.raavareId + '</td>' +
        '<td>' + item.raavareNavn + '</td>' +
        '</tr>');
}
```

6. Test

6.1 Test cases (Fadl s195846)

Test case ID	TC01
Resume	Tjekker om en farmaceut kan oprette en råvare.
Krav	KF 4
Forudsætninger	Råvaren skal ikke være kendt af systemet
Succeskriterier	Råvaren kan oprettes
Test procedure	1. Åben farmaceut.html 2. Klik på knappen råvare-administration 3. Klik på knappen opret/opdater råvare 4. Indtast ID og navn på råvare 5. Klik på knappen opret råvare
Test data	ID: 99 Navn: Natriumhydroxid
Forventet resultat	Råvare oprettet
Faktiske resultat	Råvare oprettet
Status	Lykkes
Testet af	Fadl Matar
Dato	24-06-2020
Test miljø	IntelliJ IDEA 2019

Test case ID	TC 02
Resume	Tjekker om en administrator kan rette i en bruger
Krav	KF 3
Forudsætninger	Brugeren skal allerede være kendt af systemet
Succeskriterier	Brugeren kan rettes i
Test procedure	1. Åben admin.html 2. Tryk på rediger knappen 3. Indtast ID på den bruger der skal rettes i 4. Tryk på knappen rediger 5. Efter endt opdatering tryk på knappen bekræft
Test data	CPR = 280399-3333
Forventet resultat	CPR = 280399-3333
Faktiske resultat	CPR = 280399-3333
Status	Lykkes
Testet af	Fadl Matar
Dato	24-06-2020
Test miljø	IntelliJ IDEA 2019

Test case ID	TC 03
Resume	Systemet skal kunne validere den pågældende bruger og kun give ham/hende adgang til de rettigheder han har fået givet
Krav	KF 2
Forudsætninger	Brugeren skal være kendt af systemet
Succeskriterier	Brugeren kan kun få adgang til de rettigheder han besidder.
Test procedure	1. Åben start.html 2. Indtast CPR der hører til den pågældende rolle. 3. Tryk på log-in knappen
Test data	Rolle = Admin CPR = 280399-3333
Forventet resultat	Administratoren har kun adgang til admin siden og intet andet.
Faktiske resultat	Administratoren har kun adgang til admin siden. Hvis det angivet cpr bliver brugt under andre roller ville han/hun ikke få adgang til de pågældende sider.
Status	Lykkes
Testet af	Fadl Matar
Dato	24-06-2020
Test miljø	IntelliJ IDEA 2019

6.2 JUnit test (Sid s195484)

For yderligere at teste forskellige funktioner af vores program har vi konstrueret en JUnit test. Under denne test vil der være specielt fokus på funktion som indebære at vi kan hente bruger data, indsætte bruger data og oprette en bruger. Dette er blevet gjort ved hjælp af 3 metoder. checkUserUpdate(), checkUserInformation() og checkUserCreation() samt andet findes nedenunder.

```
@Test
void checkUserUpdate() throws DALException {

    BrugerDAO b = new BrugerDAO();
    BrugerDTO a = b.getBruger(15);

    String beforeNameChange = b.getBruger(15).getOprNavn();
    a.setOprNavn("Ændret navn");
    b.updateBruger(a);
    String afterNameChange = b.getBruger(15).getOprNavn();

    a.setOprNavn("Sid Mounib");
    b.updateBruger(a);

    assertNotEquals(beforeNameChange, afterNameChange);
}
```

Som sagt sætter checkUserUpdate() metoden primært fokus på at tjekke om en bruger ændring kan blive udført. Dette bliver testet ved koden ovenfor. Først bliver der oprettet en ny instans af BrugerDAO og dernæst bliver en bruger loadet ind til BrugerDTO. Senere bliver en String beforeNameChange oprettet hvor navnet af en bruger bliver gemt. Herefter bliver navnet på brugeren ændret og opdateret, hvor det nye navn bliver gemt under String afterNameChange. Navnet bliver derefter ændret tilbage sit originale navn. Sidst tester vi om beforeNameChange og afterNameChange er anderledes, og hvis de er, betyder det at en ændring er sket og at vi har opdateret en user med succes.

```
@Test
    void checkUserInformation() throws DALException {
        BrugerDAO b = new BrugerDAO();

        assertEquals("Sid Mounib", b.getBruger(15).getOprNavn());
        assertEquals("SM", b.getBruger(15).getIni());
        assertEquals("260799-4321", b.getBruger(15).getCpr());
        assertEquals("Farmaceut", b.getBruger(15).getRolle());
        assertEquals("Aktiv", b.getBruger(15).getStatus());
}
```

Den næste test hedder checkUserInformation som tester om dataet som er hentet, passer med det data som vi forventer bliver hentet. Dette blives gjort ved at instansiere BrugerDao og køre assertEquals metoder til alle attributter en bruger har.

```
@Test
    void checkUserCreation() throws DALException{

        BrugerDAO b = new BrugerDAO();
        BrugerDTO a = new BrugerDTO();

        boolean doesUserExist;
        boolean doesNewUserExist;

        try{
            System.out.println(b.getBruger(35).getOprNavn());
            doesUserExist = true;
        }catch(DALException e){

            doesUserExist = false;
        }

        a.setOprId(35);
        a.setOprNavn("TestSubject");
        a.setIni("TS");
        a.setCpr("123456-1234");
        a.setRolle("Farmaceut");
        a.setStatus("Aktiv");
        b.createBruger(a);

        try{
            b.getBruger(35).getOprNavn();
            doesNewUserExist = true;

        }catch(DALException e){
            doesNewUserExist = false;
        }
        assertNotEquals(doesUserExist,doesNewUserExist);
    }
}
```

Den sidste test er checkUserCreation(). Efter vi igen har institueret BrugerDAO og BrugerDTO laver vi en boolean doesUserExist og en boolean doesNewUserExist. Målet med dette er at tjekke om vi kan oprette en bruger. Først starter vi med en try catch som tager navnet af en bruger som ikke eksistere hvilket sender den videre til catch hvor den sætter doesUserExist til false. Dernæst oprettes og opdateres en bruger hvor vi igen bruger den samme try catch. Denne gang eksisterer brugeren og derfor bliver doesNewUserExist sat til true. Sidst bliver de to boolean sammenlignet, hvor vi forventer at de ikke er ens. Årsagen bag denne metode er at vi vil sikre os at en bruger ikke har eksisteret før testen, er blevet lavet.

6.3 Postman (Khaled s195487)

Vi har også testet vores metoder i vores REST Service klasser gennem Postman. Her skal vi en URL for en REST Service, og vælger en metode (GET, POST, PUT), hvor vi indsætter data ved brug af JSON. For at bruge JSON er det dog vigtigt at vi har inkluderet:
Content-Type: application/json.

7. Konklusion

Vi kan konkludere at vi igennem dette projekt har formået at opfylde de vigtigste krav, ved at implementere disse i det endelige system.

Nu når vi har formået at implementere de vigtigste krav som vi fik stillet i starten af dette projekt, er der dog også en række aspekter af dette projekt vi kunne tænke os at have arbejdet videre med hvis der var stillet mere tid til rådighed, såsom designet af hjemmesiden samt css'en.

Ændringer vi kunne have lavet i løbet af dette projekt, er blandt andet at få anskaffet os et domæne til denne hjemmeside til medicinalvirksomheden, så man ikke behøvede at have alle filer på en enhed for at kunne kører systemet.