# 02324 - Videregåendeprogrammering - CDIO Final

## Gruppe: 14

David s147197

Johan s195491

Mohammad s184174

Christian s195480

William s195490

**Dato: 25. Juni 2020**

Github Repository

## Timeregnskab og Arbejdsfordeling

Billedet viser antallet af timer vi hver har brugt på projektet og hver persons nummer.

| Christian **1** | Mohammad **2** | David **3** | William **4** | Johan **5** |
|---|---|---|---|---|
| 90 timer | 90 timer | 90 timer | 90 timer | 90 timer |

Arbejdsfordelingen for rapporten kan ses med et nummer sidst på linjen. Hvert nummer hører til en person og går fra 1 til 5. Dette vil sige at alt overstående op til forrige nummer er skrevet af den person angivet af nummeret. Som eksempel er dette afsnit skrevet af Johan.$_5$ Og dette er skrevet af William.$_4$

## Opstart af software

For at kunne anvende denne webservice, så skal man selvfølgelig have internetforbindelse, og derudover have Google Chrome, Microsoft Edge eller Safari installeret. Det er disse browsere vi har testet projektet i, og dermed dem vi kan garanterer projektet understøttes af. Dernæst skal nedenstående URL anvendes og så er der tilgang til vores projekt.[5]

http://420green.sells-for-u.com:8080/CDIO_Final_war_exploded/Frontpage.html

Herefter kan der logges ind med følgende bruger ID (CPR), for de forskellige brugerroller:

| Navn | Bruger ID (cpr) | Rolle |
|---|---|---|
| Karsten | 1212993113 | Admin |
| Knud | 0000000000 | Produktionsleder |
| Kenny | 1234567890 | Farmaceut |
| Kirsten | 2004891337 | Laborant |

Hvis der skulle opstå problemmer med at komm

# Abstract

This paper shows the full scope of the process used to create a webservice that allows users in a medicinal corporation to store and receive data. The paper includes analysis, design, implementation and testing of the steps used to satisfy the customer and fulfill the requirements, from the start of the project until the end. The main focus of this paper is the design, functionality and implementation of the webapp.[1]

# Forord

Denne rapport viser et overblik over processen for udarbejdelsen af en webservice for en medicinal virksomhed, så de kan afveje deres produkter, samt hente/gemme data. Der vil blive beskrevet nærmere om analyse, design, implementering samt testning af webapplikationens enkelte dele, heraf kundens krav samt hvilke krav der opfyldes, og hvordan kravene opfyldes. Denne rapport vil primært fokusere på design, funktionalitet og implementeringen af webappen.[4]

# Indholdsfortegnelse

# 1 Problemformulering

Vi har fået stillet til opgave af en medicinalvirksomhed, der ønsker at få udviklet software til brug af afvejning, samt dokumentation af afvejning og råvare forbrug. Opgaven stiller krav til at der skal være 4 forskellige bruger-typer; Admin, Farmaceut, Produktionsleder, Laborant. Det skal endvidere være muligt for en admin at oprette, redigere, se alle og slette brugere.[3]
En farmaceut skal kunne oprette, rette og se alle råvare, samt oprette og se recepter. En farmaceut skal også kunne gøre alt en produktionsleder kan. Produktionslederen skal kunne oprette og vise råvarebatches, samt oprette og vise produktbatches. En produktionsleder skal også kunne gøre alt en laborant kan. Til sidst skal en laborant kunne fortage afvejninger af de forskellige råvarebatches der skal bruges til produktbatches. Systemet skal også være i stand til at give passende fejlbeskeder.[2]

# 2 Indledning

## 2.1 Projektplanlægning

I den indledende fase af projektet, stod det klart at en velstruktureret og velovervejet planlægning ville bære frugt. Til dette anvendes et Gantt-kort, som beriger os med et holistisk overblik over forløbet.[3]

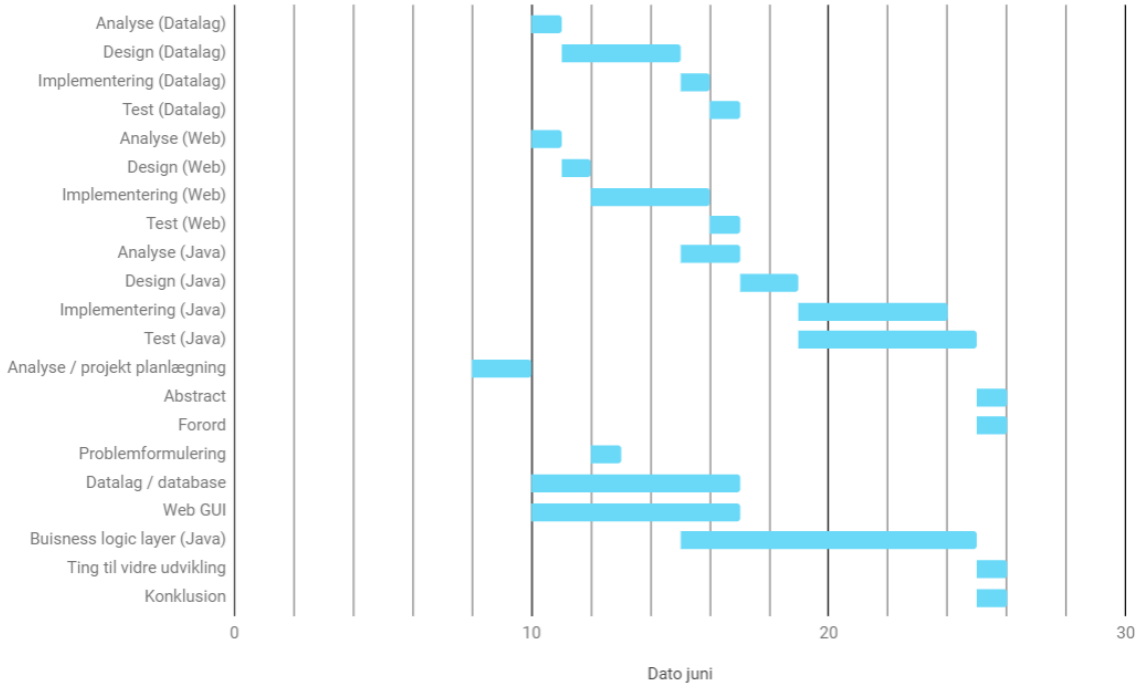


Figure 2.1: Gantt chart[2]

Som supplering til Gantt-kortet, har vi oprettet et Trello-board, hvor store opgaveområder splittes op i mindre arbejdsopgaver. Dette fremmer overskueligheden over arbejdsfordeligen og projektplanlægningen som helhed. Derudover har vi holdt daglige statusmøder, samt modtaget feedback fra hjælpelærerne, for at sikre en iterativ udviklingstilgang til projektet.[1]

# 3 Server & Database

## 3.1 Server

For at kunne hoste vores projekt har vi valgt at sætte en webserver op, og derefter indlejre en Tomcat-server i denne for at hoste vores REST API. Da webserveren ikke har en statisk offentlig IP-adresse, har vi sat dynamisk DNS op, således at vores domæne oversættes til webserverens aktuelle IP-adresse.[5]

Figure 3.1: Webserver [4]

Vi har sat Tomcat-serveren op på port 8080, hvilket vi har åbnet for på webserveren. Således ligger hele vores projekt på serveren, dvs. webappen kan tilgås offentligt via den givede URL.[2]

## 3.2 Database

For at kunne understøtte håndtering af data i systemet, har vi gjort brug af en MySQL database. Databasen er blevet lavet ved først at analysere os frem til hvilke variabler der ønskes at kunne gemmes og hentes frem, ud fra opgavebeskrivelsen.[5] Herefter lavede vi et Extended Entity Diagram (Se figur 3.2) over den fuldstændige database, som vi bagefter kunne bruge til at "forward engineer" en MySQL script fil, der gør det muligt at oprette hele databasen.[3] Udover de restriktioner databasen indeholder i forhold til variablernes type og størrelse, så indeholder den også "triggers", der automatisk opdatere når en bestemt produktbatch er under oprettelse og når den er færdig. Herefter bruger vi Java Database Connectivity (JDBC) til at indsætte og hente data til og fra databasen. Dette vil blive dækket under implementeringen af DataAccess.[1]

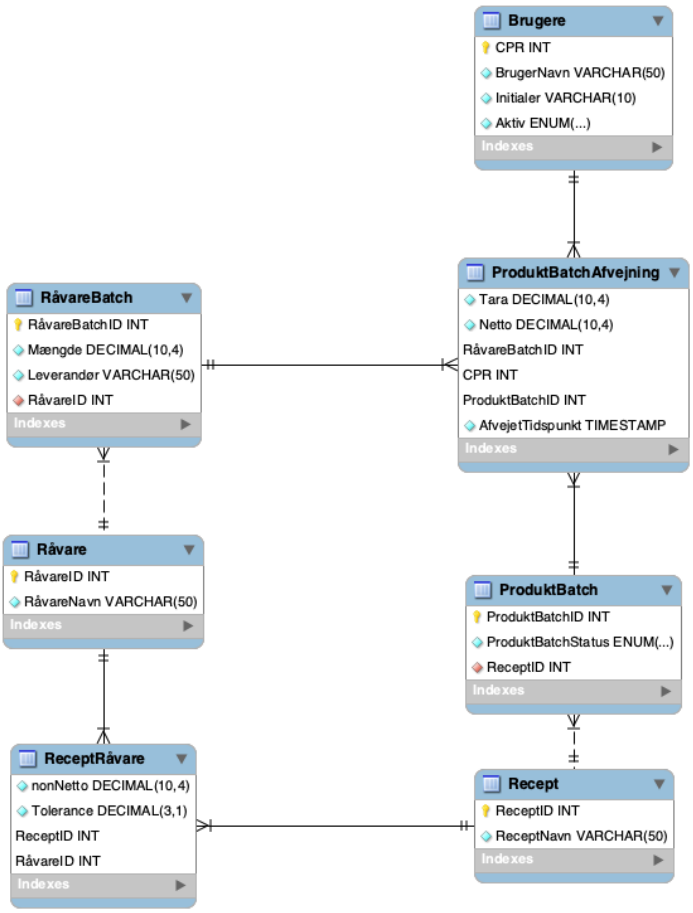


Figure 3.2: EER diagram[4]

# 4 Analyse

## 4.1 Use case model

I den indledende del af analysen startede vi med en usecase model over systemet , for at kunne specificere kravene yderligere. Til dette startede vi med et usecase diagram for at danne et overblik over systemets aktører og use cases. Systemet har i alt 4 aktører, da der kun er 4 forskellige roller en bruger af systemet kan have. Nedenstående diagram viser de 4 aktører, samt tilhørende use cases.[1]

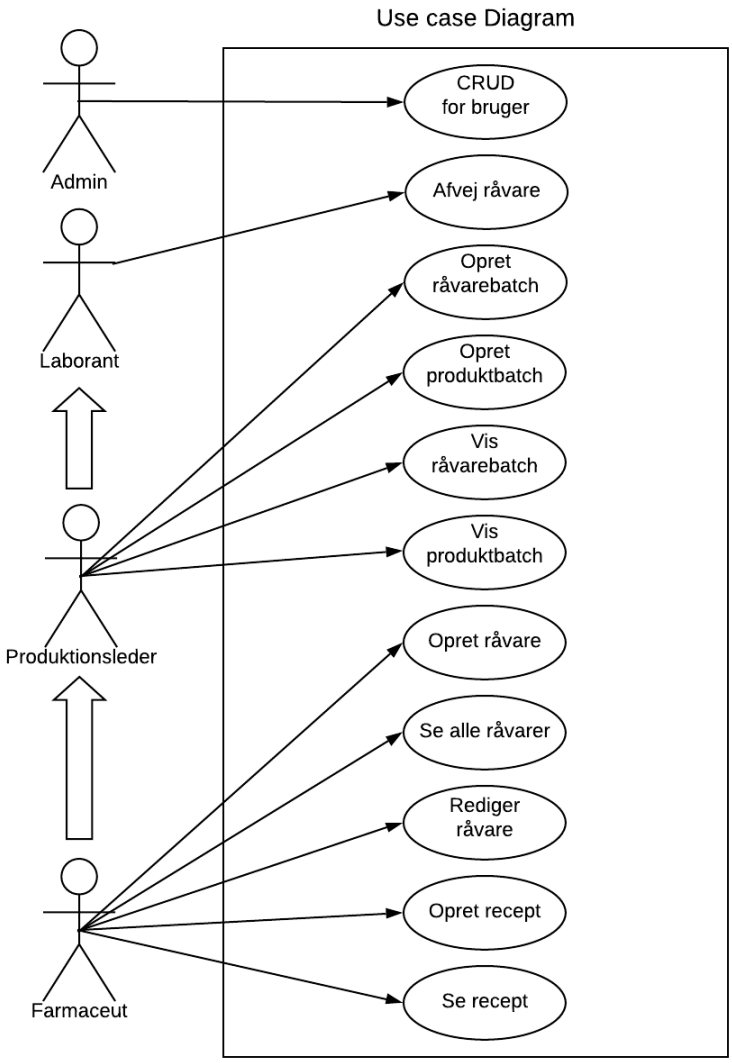


Figure 4.1: Use case Diagram[4]

Ud fra dette diagram har vi valgt at beskrive nogle af de mindre vigtige use cases på brief format, samt de mere kritiske use cases på fully dressed.[2]

| Use cases | Brief description |
|---|---|
| Create users | Aktøren (admin) trykker på "opret bruger"-knappen i menu og udfylder de påkrævede informationer og trykker gem. |
| Read users | Aktøren (admin) trykker på "vis brugere"-knappen i menuen og en liste over systemets brugere fremvises. |
| Update users | Aktøren (admin) trykker på "rediger bruger"-knappen i menuen og indtaster brugerID på brugeren der skal redigeres, udfylder de påkrævede informationer og trykker gem. |
| Delete users | Aktøren (admin) trykker på "slet bruger"-knappen i menuen og indtaster brugerID på brugeren der skal slettes, hvorefter den pågældende brugers "Status" vil bliv sat til "Ikke aktiv. |
| Opret råvarebatch | Aktøren (Produktionsleder) trykker på "Opret råvarebatch"-knappen i menuen og indtaster de påkrævede informationer og trykker på "Gem"-knappen |
| Opretproduktbatch | Aktøren (Produktionsleder) trykker på "Opret produktbatch"-knappen i menuen og indtaster de påkrævede informationer og trykker på "Gem"-knappen |
| Vis råvarebatch | Aktøren (Produktionsleder) trykker på "vis råvarebatch"-knappen i menuen og en liste over systemets råvarebatches fremvises. |
| Vis produktbatch | Aktøren (Produktionsleder) trykker på "Vis produktbatch"-knappen i menuen og en oversigt over alle produktbatches fremvises. Aktøren indtaster et specifikt produktbatchID i søgefeltet, trykker på "søg"-knappen, hvorefter detaljerede informationer om dette produktbatch fremvises. |
| Se alle råvare | Aktøren (Farmaceut) trykker på "Se alle råvarer"-knappen i menuen og en liste over systemets råvarer fremvises. |
| Rediger råvare | Aktøren (Farmaceut) trykker på "Ret råvare"-knappen i menuen og indtaster ID på råvaren, hvis navn ønskes redigeret og indtaster det nye navn. |
| Opret recept | Aktøren (Farmaceut) trykker på "Opret råvare"-knappen i menuen og indtaster de påkrævede informationer og trykker på "Gem"-knappen |
| vis recept/recepter | Aktøren (Farmaceut) trykker på "Se recepter"-knappen i menuen og en oversigt over alle recepter fremvises. Aktøren indtaster et specifikt receptID i søgefeltet, trykker på "søg"-knappen, hvorefter detaljerede informationer om dette recept fremvises. |

Figure 4.2: Brief beskrivelse af use cases [3]

| **Use case navn** | Afvej råvare |
|---|---|
| **Scope** | Web app CDIO |
| **Primær aktør** | Laborant |
| **Preconditions** | At aktøren er logget ind som Laborant |
| **Postcondition** | Produktions-batchet er afsluttet |
| **Main success scenario** | 1. Tryk på "Afvejning"-knappen i menuen<br>2. Angiv produktionsbatch ID der skal afvejes til<br>3. Udfyld felterne med de påkrævede informationer<br>4. Tryk "Gem" |
| **Extentions** | 2a. Det indtastede produktionsbatch er ikke oprettet<br>1. Systemet fremviser en tom liste over afvejninger for det pågældende produktionsbatch<br><br>3a. Der er forskel mellem værdierne for "Tara" og "kontrol vægt"<br>1. Systemet fremviser en besked om at der er opstået en fejl i afvejningen |

Figure 4.3: Use case "Afvej råvare $_5$"

| **Use case navn** | Opret Råvare |
|---|---|
| **Scope** | Web app CDIO |
| **Primær aktør** | farmaceut |
| **Preconditions** | At aktøren er logget ind som farmaceut |
| **Postcondition** | Den indtastede råvare er registreret og gemt i system databasen |
| **Main success scenario** | 1. Aktøren trykker på "Opret råvare"-knappen i menuen<br>2. Aktøren udfylder de påkrævede oplysninger i oprettelsesskemaet og trykker på "Gem"-knappen<br>3. Aktøren modtager en bekræftelses-besked om at råvaren er oprettet og den bliver tilføjet til databasen. |
| **Extentions** | 2a. Aktøren indtaster en allerede eksisterende råvare og trykker på gem knappen.<br>1. Aktøren modtager en besked om at der er opstået en fejl under oprettelsen af råvaren |

Figure 4.4: Use case "Opret råvare$_1$"

| | |
|---|---|
| **Use case navn** | (CRUD) - Rediger bruger |
| **Scope** | Web app CDIO |
| **Primær aktør** | Admin |
| **Preconditions** | At aktøren er logget ind som admin |
| **Postcondition** | Den valgte bruger, brugeroplysninger er opdateret i databasen. |
| **Main success scenario** | 1. Aktøren trykker på “rediger bruger”-knappen i menuen<br>2. Aktøren udfylder de påkrævede informationer i redigeringsformen<br>3. Aktøren modtager en besked om at brugeren er redigeret og de nye oplysninger gemmes i systemets database |
| **Extentions** | 2a. Aktøren indtaster ugyldigt bruger ID<br>1. Aktøren modtager en besked om at der er opstået en fejl ved redigeringen af brugeren |

Figure 4.5: Use case ”Rediger $\text{bruger}_2$”

## 4.2 Kravsliste

Ud fra problemformuleringen og use case modellen, er vi kommet frem til nedenstående kravliste med funktionelle krav til systemet.[3]

```
Krav 1: Der skal være states så man ikke skal skrive al info igen

Krav 2: Systemet skal understøtte 4 forskellige roller: Administrator,
        Farmaceut, Produktionsleder og Laborant.

Krav 3: Systemet skal vise en brugergrænseflade afhængig af valgte rolle på startsiden

Krav 4: Systemet skal understøtte 5 brugsscenarier: Brugeradministration,
        råvareadministration, råvarebatchadministration, produktionbatchadministration og afvejning

Krav 5: Brugeradministration
        Krav 5.1: En administrator skal kunne oprette en bruger
        Krav 5.2: En administrator skal kunne rette en bruger
        Krav 5.3: En administrator skal kunne fjerne en bruger (inaktivere)
        Krav 5.4: En administrator skal kunne se alle brugere
        Krav 5.5: En bruger defineres ved BrugerID, Navn og initialer

Krav 6: Råvareadministration
        Krav 6.1: En farmaceut skal kunne oprette en råvare
        Krav 6.2: En farmaceut skal kunne rette en råvare
        Krav 6.3: En farmaceut skal kunne vise alle råvare i systemet
        Krav 6.4: En råvare defineres ved råvare, samt har et navn
        Krav 6.5: Man skal kunne oprette en recept
                Krav 6.5.1: En recept består af receptNr, navn og sekvens af receptkomponenter
                            (råvare (type), en mængde samt en tolerance)
        Krav 6.6: En farmaceut skal kunne gøre det samme som en produktionsleder


Krav 7: Batchadministration
        Krav 7.1: En produktionsleder skal kunne oprette råvarebatches
        Krav 7.2: En produktionsleder skal kunne vise råvarebatches
        Krav 7.3: En produktionsleder skal kunne oprette produktionsbatches
        Krav 7.4: En produktionsleder skal kunne vise prodkutionsbatches
        Krav 7.5: En produktionsleder skal kunne gøre det samme som en laborant


Krav 8: Afvejning
        Krav 8.1: Systemet skal have en brugergrænseflade hvor en laborant
                  kan logge ind ved at indtaste laborantNr
        Krav 8.2: Systemet skal kunne understøtte og validerer varierende tolerancer
                  for forskellige råvarer
        Krav 8.3: En laborant skal kunne indtaste afvejet vægt for en råvare.
        Krav 8.4: En laborant skal kunne indtaste afvejet vægt for tara.
        Krav 8.5: En laborant skal kunne indtaste et batchnummer for en råvare
        Krav 8.6: En laborant skal kunne indtaste en kontrolvægt.
```

Figure 4.6: Kravliste[5]

## 4.3 Moscow

Kravene fra kravlisten, har vi valgt at sorterer efter MOSCOW, hvilket inddeler kravene i forskellige grader for prioritering. Dette giver et overblik over hvor arbejdsbyrden skal lægges og kan ses nedenfor.[4]

| | |
|---|---|
| Must have | krav 2, krav 4, krav 5, krav 6, krav 7, krav 8 |
| Should have | krav 3 |
| Could have | krav 1 |
| Wont have | |

Figure 4.7: Kravliste[1]

## 4.4 Features

Kravene kan yderligere samles i features, som udvikles og tilføjes til systemet. Disse features lægger fundamentet for systemplanlægningen, da disse er skrevet i prioriteret rækkefølge, samt omfavner alle kravene fra kravlisten.[5]

| Feature | beskrivelse | relevante krav |
|---|---|---|
| Login | Brugeren kan logge ind på systemet alt efter rolle | krav 2, 4, 8.1 |
| Brugeradministration | Admin brugeren kan udfører CRUD for andre brugere. | krav 5 |
| Råvareadministration | Farmaceut kan oprette, redigere råvare, samt oprette og redigere recepter. | krav 6.1, 6.2, 6.3, 6.4, 6.5, 6.5.1 |
| Batchadministation | Produktionsledere rollen kan oprette og se råvarebatches, samt produktionsbatches | krav 7.1, 7.2, 7.3, 7.4 |
| Afvejning | Laborant kan lave afvejninger til bestemte produktionsbatches. | krav 8.3, 8.4, 8.5, 8.6 |
| Bruger nedarvning | Farmaceut skal kunne gøre det samme som produktionsleder, og en produktionsleder skal kunne gøre det samme som en laborant. | krav 6.6, 7.5 |
| Afvejnings tolerance | Systemet kan validerer varierende tolerancer for forskellige råvarer. | krav 8.2 |

Figure 4.8: Kravliste[3]

## 4.5 Domænemodel

For at få et overblik over domænerne i systemet, samt forstå sammenspillet mellem domænerne, er der konstrueret en domænemodel. Som beskrevet tidligere, så kan Farmaceuten udføre alle opgaver (med undtagelse for administration af brugere) og produktionslederen kan administrere råvare-/produktbatches, samt afveje råvarer.[2] Vi har dog valgt at vise hovedopgaverne for de forskellige roller, da farmaceuten og produktionslederen nedarver de pågældende opgaver de har adgang til, og skal ikke kunne udføre en ny funktionalitet, som kun de har adgang til.[4]

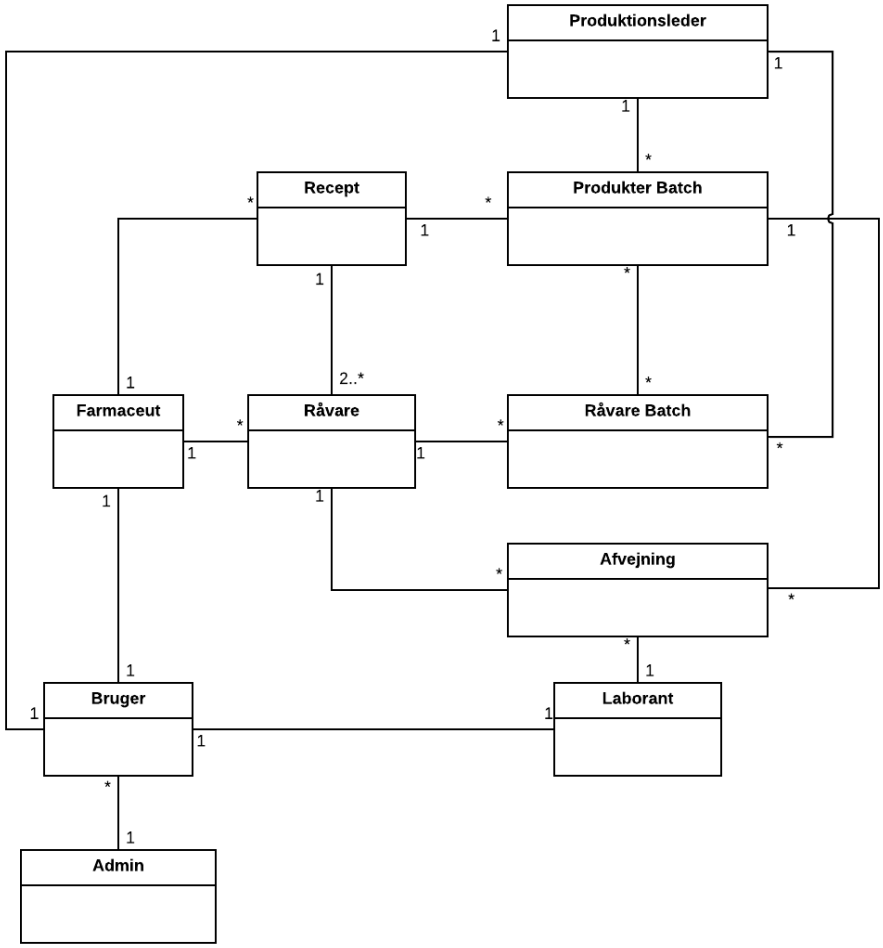


Figure 4.9: Domænemodel[1]

# 5 Design

## 5.1 Pakke Diagram

For at illustrerer systemets overordnede arkitektur, har vi konstrueret et pakkediagram.

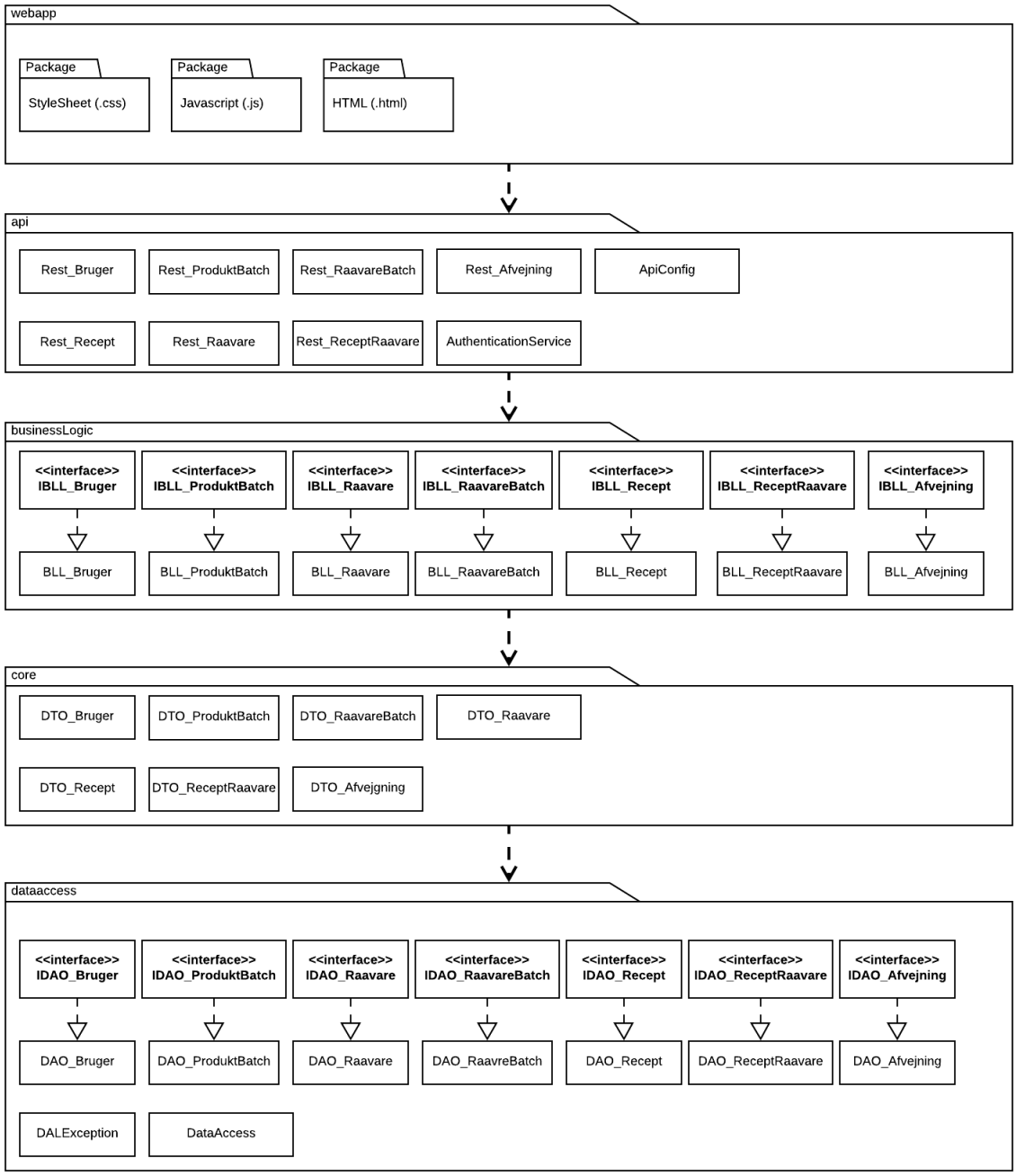


Figure 5.1: Pakke Diagram[4]

Vi har valgt at designe systemet efter en 3 lags arkitektur, hvor systemet indeles i et præsentationslag, i form af en brugergrænseflade, et logik lag, samt et datalag. I hvert af disse tre lag, har vi yderligere opdelt systemet i mindre logiske dele. I ovenstående pakkediagram udgør pakkerne "dataacess" og "core" datalaget, businessLogic udgør logik-laget og præsentationslaget består af "webapp" og "api" pakkerne.[3]

I webapp-laget ligger html-siderne, stylesheets og javascripts. Dette har vi valgt gøre for at adskille selve hjemmesidens udsende og dets funktionalitet, såsom knapper, fra det resterende java kode. I api pakken har vi valgt at have en klar separation mellem den del af koden der henter informationer fra hjemmesiden og den resterende funktionalitet.[5]
BusinessLogic pakken indeholder klasser der gør det muligt at tilføje forskellige former for funktionalitet, og er med til at forbinde front-end og back-end delen af vores kode. Core pakken indeholder klasser der kan bruges til at oprette objekter, der gør det nemmere at transportere data mellem de forskellige lag i form af data transfer objekter(DTO).[2] Til sidst har vi dataccess med klasserne DAO som bruges til at hente og indsætte data i databasen. Derudover indeholder den også en "DataAccess" klasse der bruges til at oprette java database adgang direkte til mySQL serveren vi har tilknyttet.[5]

## 5.2 Klasse Diagram

Vi har valgt at udarbejde et design klasse diagram over vores backend system. Vi har dog valgt kun at vise Raavare og Bruger domænerne, da det ellers ville blive en gentagelse af den samme struktur. Vi har i alt 7 domæner, der hvert har deres egne DTO, IDAO, DAO, I_BLL, BLL og Rest klasser.[1] Vi har valgt at vise Raavare, fordi det viser den generelle opbygning af hvert enkel klasse i alle domænerne, dog med undtagelse af Bruger domænet, der har en ekstra klasse "AuthenticationService".[4]

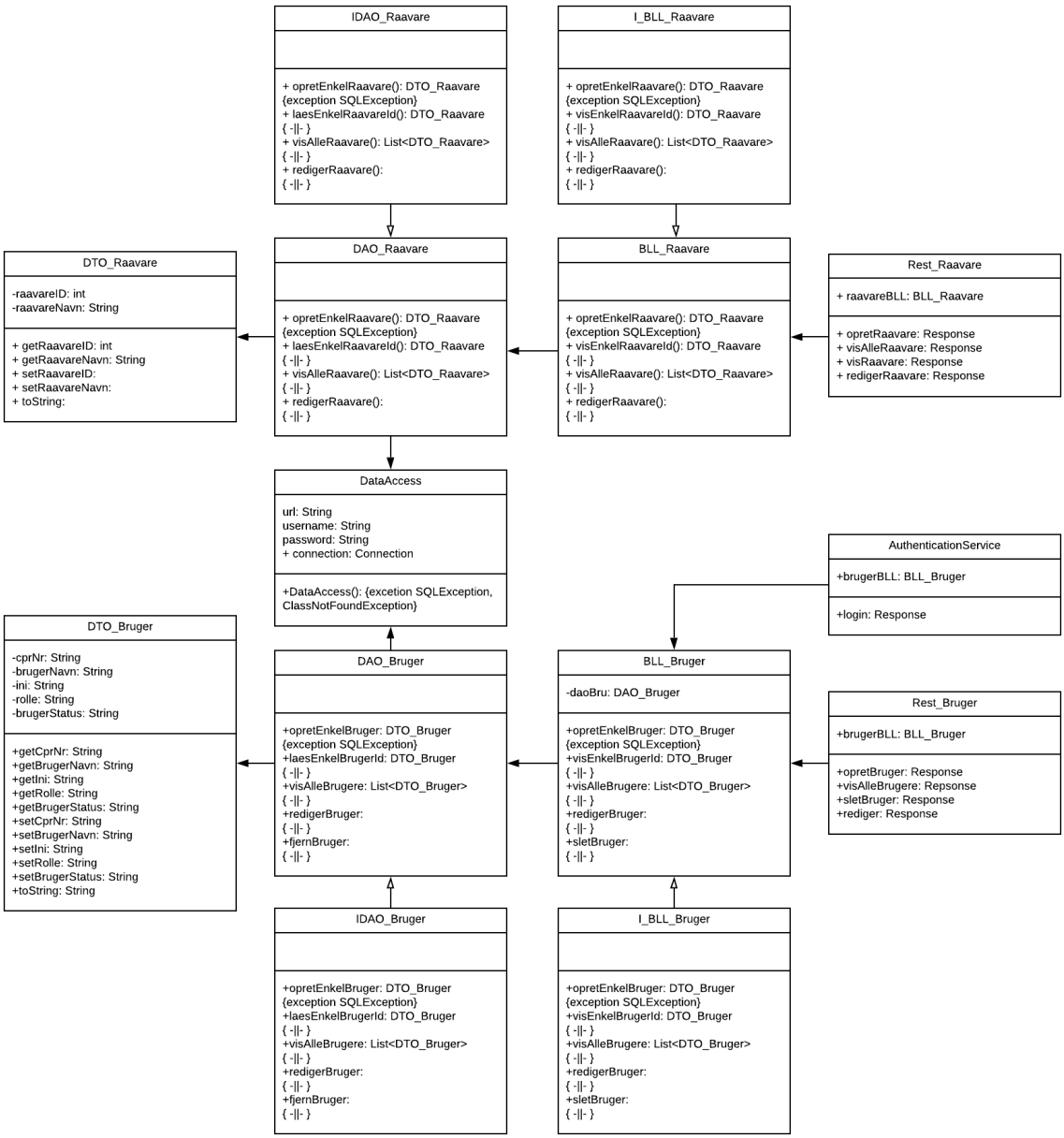


Figure 5.2: Design klassediagram over Råvare og Brugere[2]

## 5.3 Sekvens Diagram

Vi har valgt at lave et sekvens diagram over "vis brugere funktionen" i backenden af web applikationen, hvilken afspejler en af kernefunktionaliteterne, som går igen i systemet. Nedenstående diagram viser sammenspillet mellem de forskellige klasser, fra at REST API'en modtager forespørgslen, til at data hentes ud af databasen og retuneres til frontend.[3]

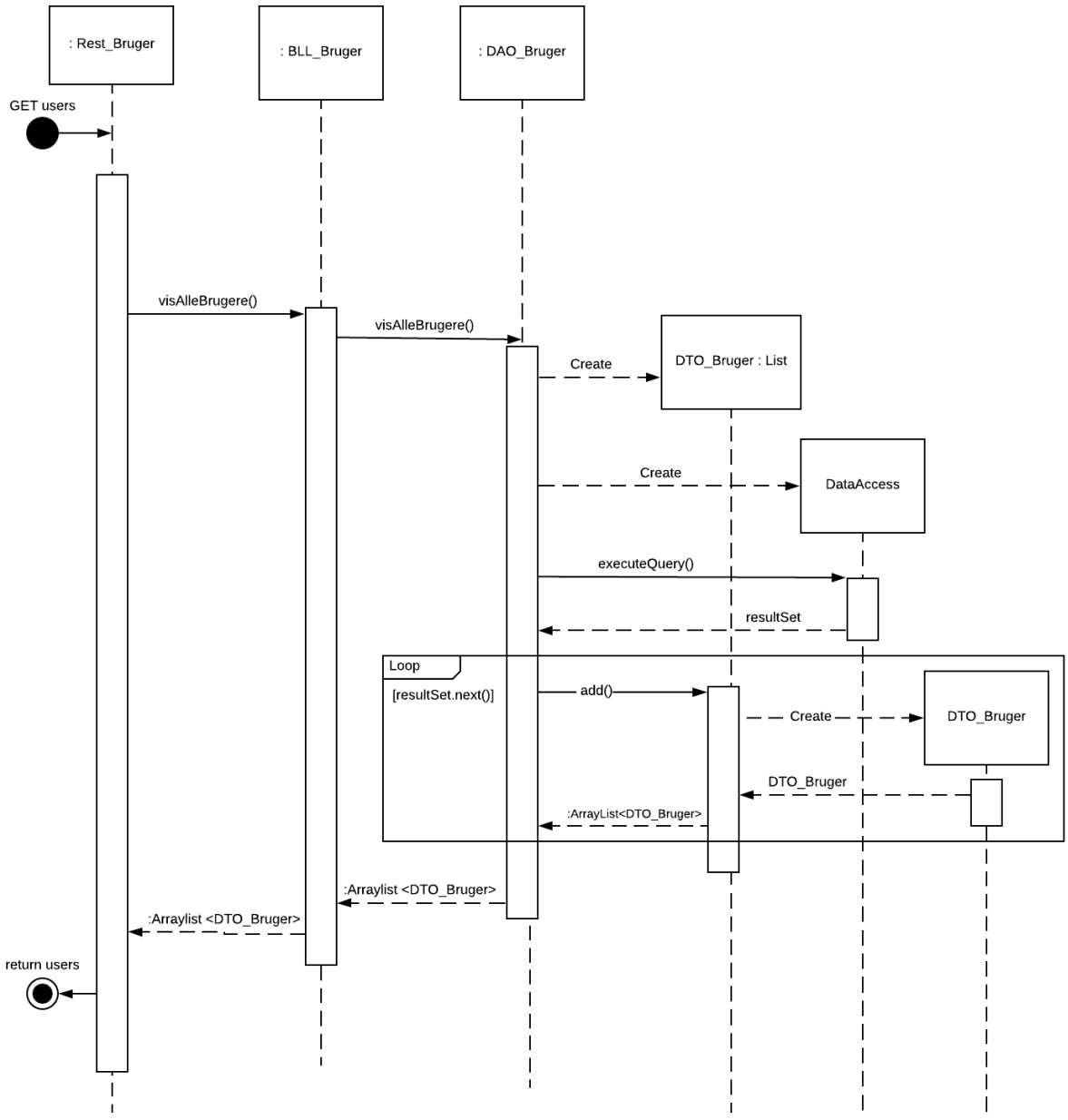


Figure 5.3: Sekvens Diagram[1]

## 5.4 Webapp

Som tidligere nævnt, så indeholder webapp-laget bl.a. HTML-sider og stylesheets, hvor HTML-siderne omfatter sidernes sektioner, elementer samt struktur, og stylesheets omfatter stylingen af disse. Det vigtigste er at have overblikket over hvordan siderne er forbundet, det kan man bruge et Site-Map til.[4]

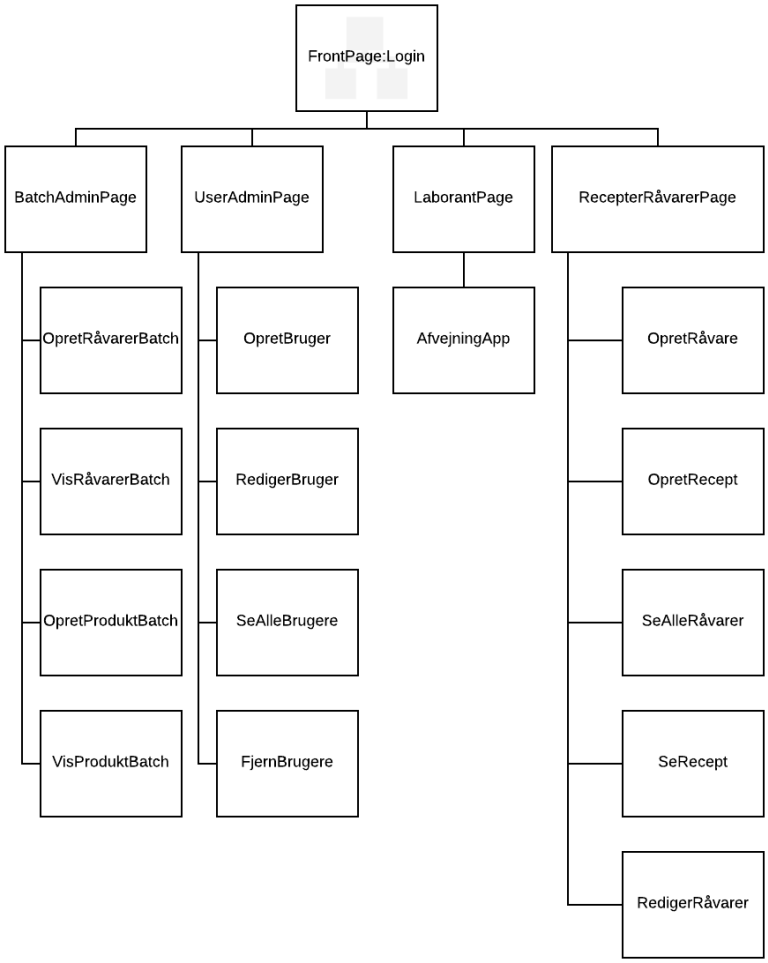


Figure 5.4: SiteMap[5]

Ovenstående Site-Map illustrerer forbindelsen imellem siderne i webapp laget. Fra FrontPage logger man ind, og ud fra sin brugerrolle, skal man kunne vælge at tilgå enten BatchAdminPage, UserAdminPage, LaborantPage, eller RecepterRåvarerPage. Hvilket lagde op til at vi fik ideen om at lave singlepages til hver rolle. I stedet for at have én side med alt indhold, så valgte vi at forbinde en login-side til 4 forskellige sider. På de sider skulle det være muligt at navigere sig rundt, uden at skifte side. Alt sidens funktionaliteter kan ses under den ønskede side på site-mappet.[3]
Når kortet over siderne er lagt med et SiteMap, begav vi os ud i design af sideindhold. Bemærk at der ikke menes layout styling, nærmere bestemmelse af hvad funktionaliteterne på siderne indeholder mht. knapper og information. På figur 10.1 i bilag ser man hvordan vi har tænkt os at vores frontpage skal se ud. Her illustrerer vi hvordan et login som produktionsleder fører videre til administration af

råvarebatches/produktionsbatches siden. Derudover har vi også designet det indhold som singlepage siderne skal kunne vise, når menuerne interageres med (vis/opret råvarebatch siderne i bilaget).[5]

### 5.4.1 HTML & CSS

Når vi ved hvad der som grundlag skal være på siden, så kan vi tilføje styling. Vi bygger først skelettet for webappen med HTML, og tilføjer det æstetiske med CSS. Men før vi gør det, så laver vi en mock-up, i form af nogle illustrationer i et storyboard, som visualiserer vores hjemmesides indhold. Derefter er det nemmere at observere hvilke HTML elementer vi skal bruge, og hvordan de skal formes med CSS.[2]

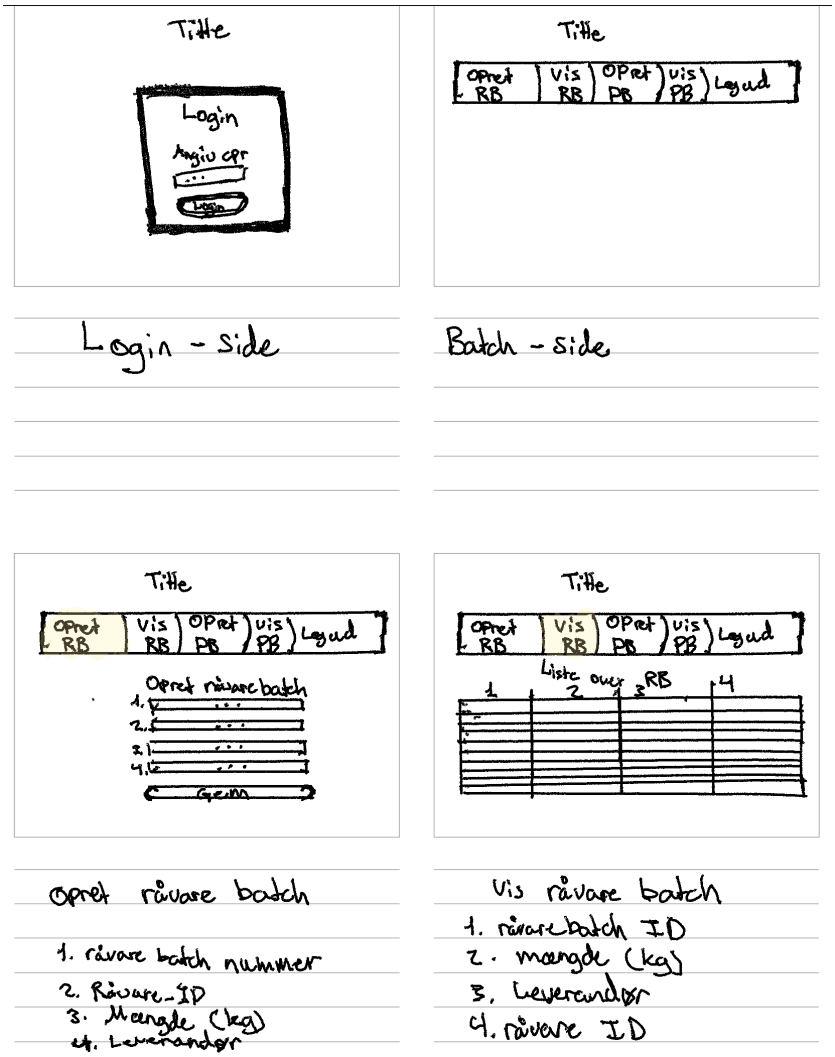


Figure 5.5: Storyboard med Weblayout-design[4]

Ovenstående storyboard illustrerer først sammenkoblingen fra login siden til vores batch-side. Vores overvejelser er at lave en sektion (div) med en login form. Login-sektionen skal fremtræde tydeligt vha. farver eller skygge som er forskellige fra baggrunden.[1]
Vores top-menu på batch-siden skal indeholde de muligheder som firmaet kræver en produktionsleder skal kunne gøre, udover at kunne logge ud. Til at oprette et råvare-batch har vi igen en form, som

indeholder nogle input elementer. Tilsidst i formen har vi en submit-knap, som skal være en input-type.[3] Til at vise råvarer, kunne man lave liste elementer. Dog ville det være mere praktisk med en table, da man ville kunne sætte nogle faste headers for kolonnerne. Så slipper man for at de forskellige attributter fremtræder i hvert liste-element, hvilket ville være uoverskueligt. I en tabel ville vi også kunne sortere de forskellige kolonner, hvis ønsket.[5]

### 5.4.2 HTML & jQuery

Når vi har fået det visuelle analyseret og gjort klar til design, så skal det funktionelle overvejes og analyseres. Til at designe vores ønskede topmenu, har vi jQuery biblioteket, som giver os evnerne til f.eks. at skjule og vise forskellige sektioner.[4] Det kan i vores tilfælde bruges til at skifte imellem de forskellige muligheder i vores topmenu. Så når produktionslederen klikker på 'opret råvarebatch', så skal den tilhørende form vises med jQuerys ".show()" metode. Når produktionslederen klikker på 'vis råvarebatch', så skal den tidligere form skjules med ".hide()" metoden, og tabellen for råvarebatches skal vises med ".show()" metoden.[3]

# 6 Implementering

## 6.1 Webapp

### 6.1.1 HTML & CSS

**Login-siden**
Vi opsætter login-siden ved at bruge en form i en div, som vores overvejelser også bar præg af. I denne div, som vi har navngivet "FrontPageDiv", har vi oprettet en form. Denne form har nogle forskellige elementer, som overskriften (h3), input feltet, som skal vise hvad der skal indtastes vha. en placeholder attribut, og tilsidst en knap som skal fungere som en submit knap, der sender det indtastede tekst input. Vi har valgt at give knappen typen "button", standardproceduren ville nok have været at give den "submit" typen, da det er en attribut-værdi for at sende form-data.(Det bruger vi AJAX til)[1]

```
<body>
<h1 class="center">Gruppe 14</h1>
<div class="FrontPagediv form-popup">
    <h1>Login</h1> <br>
    <form method="GET" action="api/authentication/login">
        <h3 class="center" style="...">Angiv venligst cpr</h3>
        <input id="cpr" class="center inputStyler" type="text" placeholder="Indtast cpr"><br>
        <button id="submit" type="button" class="center ButtonStyle" style="...">Login</button>
    </form>
</div>
</body>
```

Figure 6.1: Forsidens HTML-body

En mindre CSS-feature som vi har implementeret for at få login-formen til at virke interaktiv, er en hover-funkion. Når brugeren holder sin musemarkør over enten inputfeltet eller loginknappen, så skifter baggrundsfarven.[2]

```
input:hover{
    background-color: aliceblue;
}
```

Figure 6.2: Input hover-funktion

Funktionen er meget simpel, men er essentiel at have med på hele websiden, om det er topmenuen, lister, inputfelter, da brugeren får et indtryk af at hjemmesiden skelner mellem tekst og links eller inputs.
For at få vores div til at være centreret i midten af hjemmesiden, uanset hvad højden og bredden er på diven, så bruger vi transform/translate metoden.[5] (Figur 6.4)

```
.FrontPagediv{
    width: 500px;
    position: fixed;
    top: 45%;
    left: 50%;
    transform: translate(-50%, -50%);
}
```

Figure 6.3: Frontpagediv styling

Metoden bliver typisk brugt i responsivt design, og kræver ikke at man definerer marginerne, som hvis man bruger absolute-positioning metoden.[4]

**Topmenuen**

Vi har valgt at implementere vores topmenu som en unordered list(<ul>). Mulighederne på menubaren er liste elementer.[3]

```
<div style="..." >
    <ul>
        <li onclick="toggleDivContent('opretRaavareBatch')"><a>Opret råvarebatch</a></li>
        <li id="visRaavareBatches" onclick="toggleDivContent('visRaavareBatch')"><a>Vis råvarebatch</a></li>
        <li onclick="toggleDivContent('opretProduktBatch')"><a>Opret produktbatch</a></li>
        <li id="visProduktBatches" onclick="toggleDivContent('visProduktBatch')"><a>Vis produktbatch</a></li>
        <li class="right" onclick="location.href='Frontpage.html'"><a>Log ud</a></li>
    </ul>
</div>
```

Figure 6.4: Topmenuens HTML kode

Som det kan ses på figur 6.4, så benytter vi os af <a>-tagget på hver tekst. Normalt ville man gøre brug af "href" i a-tagget, da det fører brugeren videre til en URL, som er specificeret.(Det bruger vi jQuery til). Uden en "href"-attribut, virker tagget bare som en placeholder for et hyperlink.[2] Placeholder links er til tilfælde, hvor man vil bruge et anker-element, uden at navigere til et specifikt sted. Det er især praktisk til markering af den aktuelle side i en navigationsmenu eller til breadcrumb-trails.[1]


Figure 6.5: Topmenu med styling

Semantikken af en liste får det til at ligne en grupperet sektion, altså en liste af navigationsmuligheder, fremfor knapper. Hvis vi brugte knapper til en menu-bar skulle vi finde på vores egen måde at neste elementerne på, det gør listen for os.[4]

### 6.1.2 HTML & jQuery(AJAX)

Den måde hvorpå vi har opsat siden til at udrette de forskellige operationer brugerne kan udføre, er ved at opsætte divs ud fra hver operation og kun synliggøre dem, på det tidspunkt brugeren klikker på en af knapperne i topmenuen. Figur 6.5 viser opsætningen af produkt- og råvarebatch-siden, hvor operationerne er opdelt i forskellige divs.[5]

```
<div style="..." >
    <ul>
        <li onclick="toggleDivContent('opretRaavareBatch')"><a>Opret råvarebatch</a></li>
        <li id="visRaavareBatches" onclick="toggleDivContent('visRaavareBatch')"><a>Vis råvarebatch</a></li>
        <li onclick="toggleDivContent('opretProduktBatch')"><a>Opret produktbatch</a></li>
        <li id="visProduktBatches" onclick="toggleDivContent('visProduktBatch')"><a>Vis produktbatch</a></li>
        <li class="right" onclick="location.href='Frontpage.html'"><a>Log ud</a></li>
    </ul>
</div>


<div id="opretRaavareBatch" class="center form" style="...">
    <h2>Opret råvarebatch</h2>
    <form>
        <input id="raavarebatchNummer" class="center" type="number" placeholder="Råvarebatch nummer"><br>
        <input id="raavarebatchRaavareID" class = "center" type="number" placeholder="Råvare-ID"><br>
        <input id="raavarebatchMaengde" class = "center" type="number" step="0.001" placeholder="Mængde (Angivet i kilo)"><br>
        <input id="raavarebatchLeverandoer" class = "center" type="text" placeholder="Leverandør"><br>
        <button id="opret-raavarebatch" class = "center " type="button">Gem</button>
    </form>
</div>

<div id="visRaavareBatch" style="..." class="form">
    <h2>Liste over alle råvarebatches</h2>
    <form method="GET" action="api/raavarebatch/vis">
        <table id="raavareBatchInfoTable" style="...">
            <thead id="raavareBatchThead">
            <tr class="raavareBatchThead">
                <th class="raavareBatchThead">Råvarebatch ID</th>
                <th class="raavareBatchThead">Råvare mængde</th>
                <th class="raavareBatchThead">Leverandør</th>
                <th class="raavareBatchThead">Råvare ID</th>
            </tr>
            </thead>
            <tbody id="raavareBatchesliste"></tbody>
        </table>
    </form>
</div>
```

Figure 6.6: Batch-sidens HTML med div

Som det kan ses så gør vi brug af en toggleDivContent-metoden, når der bliver trykket på en af topmenu-knapperne. ToggleDivContent-metoden kan ses på figur 6.7. Ved hjælp af den her metode kan vi oprette alle nødvendige tabeller og input felter som vi skal bruge uden at sende brugeren videre til en ny side.[1]
Da vi har med dynamiske informationer, så kan vi ikke bare sætte en fast størrelse på tabellerne. Derfor har vi i vores JavaScript lavet en pointer til vores tabeller, som vi kan appende nogle tabelbodies til, når vi gør brug af vores API's GET og POST requests. Hermed kan vi, hver gang vi indlæser data lave en dynamisk tabel, så vi kan vise al dataen, som bliver hentet ind fra databasen.[3]
Figur 6.8 viser den måde hvorpå vi genererer tabellen for at vise produktbatches. Vi definerer først et objekt som vi tildeler de attributter som hele JavaScript-klassen skal bruge. Når brugeren trykker på topmenu knappen for at vise produktbatches så bliver funktionen udført, hvor vi i success scenariet appender en tabel for hver dataobject vi får ind gennem API'en.[2]

```
function toggleDivContent(id) {
    $('.form').hide();
    $(document.getElementById(id)).show();
    $('button').on('click', function(){
        $('button').removeClass('selected');
        $(this).addClass('selected');
    });
}
```

Figure 6.7: ToggleDiv funktion

```
var produktbatch = {
    produktbatchID: $produktbatchNummer.val(),
    produktbatchReceptNummer: $produktbatchReceptNummer.val(),
    status:'',
    cpr:'',
    raavareBatchID:'',
    tara:'',
    netto:''
};


$('#visProduktBatches').on('click', function () {
    $('#produktBatchInfoTable').show();
    $('#produktBatchRaavareInfoTable').hide();
    $('#produktBatchesliste').html('');
    $('#produktbatchesRaavare').html('');
    document.getElementById( elementId: "produktbatchIDRaavare").value = '';

    $.ajax({
        type: 'POST',
        url: 'api/produktbatch/vis',
        contentType: "application/json; charset=utf-8",
        data: produktbatch,
        success: function (data) {
            $.each(data, function (i, produktbatch) {
                $produktBatchesliste.append('<tr><td>' + produktbatch.produktBatchID + '</td>' +
                    '<td>' + produktbatch.receptID + '</td>' +
                    '<td>' + produktbatch.status + '</td></tr>');
            });
        },
        error: function () {
            alert('Fejl ved indlæsning af produktbatch');
        }
    });
});
```

Figure 6.8: Generering af tabeller

## 6.2 API

### 6.2.1 REST

Når brugeren på websiden skal indtaste informationer, som skal interagere med vores database, så er vi nødt til at sende dataen videre gennem en API. I vores program har vi implementeret 8 REST-klasser der hver har sine egne funktioner i forhold til hvilke tabeller de skal hente informationer fra eller indsætte værdier i.[3]
Billedet på figur 6.9 viser implementeringen af REST-klassen for produktbatches. Hver metode har en "Path", som vi kan referere til gennem vores JavaScript. API'en vil tage imod den information vi har sendt videre gennem JavaScript'en og vi konverterer det til DTO-objekter, som vi derefter sender videre gennem de nedre lag i vores program.[4]
Når vi kalder videre ned af lagene, så bliver der i alle metoder og klasser kastet en SQLexception tilbage til API'en, hvis der skulle gå noget galt med informationerne der er blevet sendt videre. Exceptionen bruger vi til at kommunikere tilbage til JavaScripten om at der er opstået en fejl, da vi returnerer en response statuskode 400. Hvis der ikke bliver fanget en SQLException så sender vi en status 200 tilbage for at give JavaScripten lov til at arbejde med transfer objekterne vi kreerer.[5]
Grunden til at vi har lavet en REST API er først for at kunne sende data fra vores web-applikation videre til vores Java-klasser for at arbejde med dataen, og også for at kunne splitte GUI'en fra selve logikken, for lettere at kunne genbruge den samme kode til senere brug i andre applikationer eller i nye JavaScript filer.[1]

```java
package api;
import ...

@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("produktbatch")
public class Rest_ProduktBatch{
    I_BLL_ProduktBatch proBLL = new BLL_ProduktBatch();

    @Path("opret")
    @POST
    public Response opretProduktBatch(String inputs) throws SQLException {
        String[] strarray = inputs.split( regex: "&");
        //Lav JavaScripten først
        DTO_ProduktBatch produktBatchDTO = new DTO_ProduktBatch(Integer.parseInt(strarray[0].substrin
        try{
            proBLL.opretProduktBatch(produktBatchDTO);
        } catch(SQLException e) {
            return Response.status(400).entity("SQLException: " + e.getMessage()).build();
        }
        return Response.ok().build();
    }

    @Path("vis")
    @POST
    public Response visAlleProduktBatches() throws SQLException {
        List<DTO_ProduktBatch> returnRaa;

        try{
            returnRaa = proBLL.visAlleProduktBatches();
        } catch(SQLException e) {
            return Response.status(400).entity("SQLException: " + e.getMessage()).build();
        }
        return Response.ok(returnRaa).build();
    }
```

Figure 6.9: Produktbatch REST klasse

### 6.2.2 Authentication service

Når brugeren først åbner hjemmesiden op, så skal han indtaste et CPR-nummer, som vi vil bruge som en slags godkendelsesmetode for at brugeren må komme ind på siden, og også for at viderestille personen til den rigtige HTML-side. For eksempel hvis brugeren indtaster for Kenny (se "opstart af software" afsnittet), så vil han blive omdirigeret til en side, hvor han kan vælge mellem hvilken rolles arbejde han vil udføre, men derimod hvis man logger ind som Karsten så bliver man viderestillet automatisk, da en admin kun skal udføre én slags opgave.[2]
For at godkende brugerens rolle og cpr-nummer laver vi et POST kald med cpr-nummeret som parameter og videresender det til vores DAO. I DAO'en laver vi en SQL søgning på cpr-nummeret og returner bruger-DTO'en tilbage til vores JavaScript, som vi så henter rollen fra.[4]
Hvis brugeren indtaster et cpr-nummer der ikke er registreret i databasen, så vil DAO'en sende en SQL-exception tilbage til API'en, som derefter returnerer en fejl 400 til JavaScripten.[3]

## 6.3 BusinessLogic

Som set i designklassediagram (figur 5.2), så har vi BLL-klasser hvilket styrer business logic for websitet. Dette ville sige, at man bruger dette lag til at håndtere generelle regler ift. når man indtaster informationer som fx et cpr nummer (til at oprette en bruger) der allerede eksisterer i systemet.
Vi tager et nærmere kig på "BLL_Bruger"-klassen, der håndterer netop det ovennævnte.[2]

```
@Override
public DTO_Bruger opretEnkelBruger(DTO_Bruger enkelBruger) throws SQLException {
    int måned = Integer.parseInt(enkelBruger.getCprNr().substring(2,4));
    int dag = Integer.parseInt(enkelBruger.getCprNr().substring(0,2));
    int år = Integer.parseInt(enkelBruger.getCprNr().substring(4,6));
    if (enkelBruger.getIni().indexOf('%') != -1) {
        throw new SQLException();
    } else {
        if (måned == 1 | måned == 3 | måned == 5 | måned == 7 | måned == 8 | måned == 10 | måned == 12) {
            if (dag >= 1 && dag <= 31) {
                return daoBru.opretEnkelBruger(enkelBruger);
            }
        } else if (måned == 4 | måned == 6 | måned == 9 | måned == 11) {
            if (dag >= 1 && dag <= 30) {
                return daoBru.opretEnkelBruger(enkelBruger);
            }
        } else if (måned == 2) {
            if ((år % 4) == 0) {
                if (dag >= 1 && dag <= 29) {
                    return daoBru.opretEnkelBruger(enkelBruger);
                }
            } else {
                if (dag >= 1 && dag <= 28) {
                    return daoBru.opretEnkelBruger(enkelBruger);
                }
            }
        }
    }
    throw  new SQLException();
}
```

Figure 6.10: Kode for "opretEnkelBruger()"

Denne metode tager imod en bruger af typen "DTO_Bruger", som den modtager fra Rest_Bruger-klassen. I denne metode tjekker vi for forkert indtastning af initialer og cpr-nummer. I vores DTO_Bruger-klasse, så har vi defineret cpr-nummeret til at være en String, som vi så parser til int. Udfra disse integers, så laves if-statements for at tjekke om der tastet den rigtige måned og den rigtige dato for den pågældende måned.[5]
Der bliver ikke tjekket for de sidste 4 cifre, da det er et løbenummer, som man ikke kan tjekke for uden adgang til folkeregistreret.

En væsentlig ting, der blev opdaget ved oprettelsen af en bruger, var at når man indtaster initialer, der har mellemrum (K L, P O eller lign.) så får bliver det encoded til fx K%20L, P%20P eller lign. Vi har så taget højde for det, ved at tjekke for hvilket index i den pågældende string "%" opstår. Hvis "%" opstår, altså index er !=-1, så bliver en fejlbesked sendt.[1]

Der var dog nogle enkelte logik elementer, som vi ikke kunne håndtere i BLL. Et eksempel er ved redigerBruger() metoden i DAO_Bruger klassen. Ved redigering af bruger, så opdagede vi at MySQL kan godt udføre update-statements selvom den angivet bruger ikke findes i databasen. Der vil dog ikke ske en ændring i databasen men der vil hellere ikke blive sendt en error. Vi indså derfor at der skulle sendes nogle bestemte SQL-statements, hvis vi vil have fejlbeskeder ved forkert indtastning af bruger. Disse statements udføres i DAO, da vores forbindelse til MySQL sker i det lag. Her er et billede redigerBruger() metoden i DAO_Bruger:[2]

```
@Override
public void redigerBruger(DTO_Bruger nyBruger) throws SQLException {
    try {
        DataAccess dataAccess = new DataAccess();
        Connection conn = dataAccess.connection;
        PreparedStatement preSt = conn.prepareStatement( sql: "UPDATE brugere SET BrugerNavn = ?, Initialer = ?, Rolle = ? WHERE Cpr = ?;");

        preSt.setString( parameterIndex: 1, nyBruger.getBrugerNavn());
        preSt.setString( parameterIndex: 2, nyBruger.getIni());
        preSt.setString( parameterIndex: 3, nyBruger.getRolle());
        preSt.setString( parameterIndex: 4, nyBruger.getCprNr());

        PreparedStatement preSt2 = conn.prepareStatement( sql: "SELECT * from brugere WHERE Cpr = ?;");
        preSt2.setString( parameterIndex: 1, nyBruger.getCprNr());
        ResultSet resultSet = preSt2.executeQuery();
        resultSet.beforeFirst();

        if (!resultSet.next()) {
            throw new SQLException();
        } else {
            preSt.executeUpdate();
        }
        conn.close();

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

Figure 6.11: Der tjekkes for om brugeren overhovedet findes i systemet

For at tjekke om at brugeren findes i systemet, så tjekker vi resultsettet. Udfra resultsettet, så kan vi se hvad vi får returneret tilbage. Hvis vi får det indtastede cpr nummer tilbage, så ved vi at der findes en bruger med pågældende cpr nummer og kan derfor opdatere hans oplysninger.[3]

## 6.4 Core

"DTO"-klasserne bruges til at holde på den data der udvekles mellem de forskellige lag i programmet. Klasserne indeholder alle atrributter som er relevant for den entitet de repræsenterer. En af disse er "DTO_Raavre"-klassen, hvilken holder på data relevant for råvare.[4]

```
public class DTO_Raavare {
    private int raavareID;
    private String raavareNavn;

    public DTO_Raavare(int raavareID, String raavareNavn){
        this.raavareID = raavareID;
        this.raavareNavn = raavareNavn;
    }
    public int getRaavareID() { return raavareID; }
    public String getRaavareNavn() { return raavareNavn; }
    public void setRaavareID(int raavareID) { this.raavareID = raavareID; }
    public void setRaavareNavn(String raavareNavn) { this.raavareNavn = raavareNavn; }
    @Override
    public String toString() {
        return "RaavareDTO{" +
                "raavareID=" + raavareID +
                ", raavareNavn='" + raavareNavn + '\'' +
                '}';
    }
}
```

Figure 6.12: DTO_Raavare klasse

Denne klasse indeholder attributter for råvare-id, råvare-navn, samt passende setter og getter metoder.

## 6.5 Dataaccess

For at tilgå data fra databasen har vi implmenteret klassen "Dataacess", hvilken udgør fundamentet for hele datalaget. Klassen formål er at oprette en forbindelse til MySQL-databasen, hvorefter det er muligt at sende og modtage data. nedenstående billed viser koden.[5]

```
public class DataAccess {
    private final String url = "jdbc:mysql://" + "localhost:3306" +
            "/webshop" + "?characterEncoding=utf8&serverTimezone=UTC";
    private final String username = "root";
    private final String password = "1234";
    public Connection connection;

    public DataAccess() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        this.connection = DriverManager.getConnection(url, username, password);

    }
}
```

Figure 6.13: DataAccess klasse

Måden vi har implementeret klassen på, er ved at oprette variabler af String type, hvilke indeholder

URL, brugernavn samt kodeordet til databasen. Disse parametre anvendes til at lave et "connection"-object, der opretter selve forbindelsen. Dette har vi gjort for at have en generel klasse til oprettelse af forbindelsen, hvorefter "statements" mm. kan specificeres yderligere af de DAO-klasser der anvender "dataaccess"-objektet.[1]

Implementeringen af DAO-klasserne er sket ud fra interfaces, som klasserrne derefter implementerer. Disse klasser er alle navngivet med "IDAO". Grunden til at vi har valgt interfaces, er for at gøre evt vedligeholdelse af systemet lettere, hvis nu en af DAO-klasserne skulle være defekt, eller skulle udskiftes i fremtiden. Desuden udgør disse en kontrakt for hvilke metoder der kobler sig til klassen, samt hvordan de opererer. nedenstående billede er et eksempel på et af disse interfaces, nemlig "IDAO_Raavare".[3]

```
public interface IDAO_Raavare {
    DTO_Raavare opretEnkelRaavare(DTO_Raavare enkelRaavare) throws SQLException;
    DTO_Raavare laesEnkelRaavareId(int raavareID) throws SQLException;
    List<DTO_Raavare> visAlleRaavare() throws SQLException;
    void redigerRaavare(DTO_Raavare nyRaavare) throws SQLException;
}
```

Figure 6.14: IDAO_Raavare klasse

Dette interface indeholder fire abstrakte metoder, der henholdvis kan oprette en raavare, indlæse en eller alle råvarer, samt redigering af en råvare. Interfaces bliver implementeret af klasserne navngivet "DAO". En af disse er "DAO_Raavare" der implementerer overstående "IDAO_Raavare"-interface.[2]

```
public DTO_Raavare opretEnkelRaavare(DTO_Raavare enkelRaavare) throws SQLException {
    try {
        DataAccess dataAccess = new DataAccess();
        Connection conn = dataAccess.connection;
        PreparedStatement preSt = conn.prepareStatement( sql: "INSERT INTO raavare VALUES(?,?)");
        preSt.setInt( parameterIndex: 1, enkelRaavare.getRaavareID());
        preSt.setString( parameterIndex: 2,enkelRaavare.getRaavareNavn());
        preSt.executeUpdate();
        conn.close();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return laesEnkelRaavareId(enkelRaavare.getRaavareID());
}
```

Figure 6.15: OpretEnkelRaavare metoden

Overstående billede viser implementatioen af DAO_Raavare klassens metode "opretEnkelRaavare". Denne tager et "DTO_Raavare"-objekt som argument,og opretter et DataAccess-objekt, hvilket bruges til at oprette en forbindelse til databasen, samt et Insert statement, til indsættelse af data om en bestemt raavare i databasen.[1] Denne data bestemmes af "preSt"-objektet af typen PreparedStatement, der sætter første værdi i statementet til at være en råvares ID og det næste til at være råvarens navn, som fås ved kald af get metoder på "DTO_Raavare"-objektet. Efter denne data er bestemt, bliver Insert-statementet eksikveret og forbindelsen til databasen afbrydes. Koden er sat ind i et "try-catch"-statement, der håndterer exceptions, hvis nogle af de staements der er i try-blokkken kaster nogen af typen "ClassNotFoundException". Hvis der opstår en SQLException, kaster klassen denne videre, så den kan håndteres et andet sted i programmet.[5]
Dog kan det ske ved multiple indsætninger af værdier, at der opstår en deadlock fejl. Denne bliver vi nødt til at håndtere i metoden, da vi gerne vil prøve at indsætte værdierne igen uden at skulle helt

tilbage til JavaScripten. Derfor har vi i DAO_Afvejning indsat endnu et catch-statement der fanger SQL exceptions og tjekker om det er en deadlock fejl. Hvis det er venter vi et stykke tid for ikke at overbelaste databasen. Dette gentages 10 gange og hvis der stadig opstår fejl sender vi en exception tilbage til API'en.[4]

```
@Override
public void opretEnkeltAfvejning(DTO_Afvejning afvejning) throws SQLException {
    boolean failed = false;
    int retries = 10;

    do {
        failed = false;

        try {
            DataAccess dataAccess = new DataAccess();
            Connection conn = dataAccess.connection;
            PreparedStatement preSt = conn.prepareStatement( sql: "INSERT INTO produktbatchafvejning VALUES(?,?,?,?,?,now())");
            setCreatePreparedStatement(preSt, afvejning);
            preSt.executeUpdate();


            conn.close();
        } catch (SQLException e) {
            failed = true;
            if (e.getErrorCode() == MysqlErrorNumbers.ER_LOCK_DEADLOCK){
                try {
                    Thread.sleep( millis: 1000);
                } catch (InterruptedException interruptedException) {
                    interruptedException.printStackTrace();
                }
            }
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
            throw new SQLException();
        }
    }
    while (failed && retries-- > 0);
    if (failed && retries == 0){
        throw new SQLException();
    }
}
```

Figure 6.16: Deadlock håndtering

# 7 Test

Den afgørende faktor i vores valg af test-strategi, bygger primært på det faktum, at mange af de implementerede features afhænger af store dele af systemet. Derfor har vi valgt at fokuserer mere på brede system-tests, hvor features testes som en helhed. Dog har vi også valgt at have unit-tests af nogle kristiske metoder i systemet. Nedenstående test-cases repræsenterer et lille udsnit af de tests, som er blevet udført på systemet.[3]

| **Item** | TC01 |
|---|---|
| **Type** | System test |
| **Title** | Opret bruger |
| **Priority** | High |
| **Status** | Godkendt |
| **Initial Configuration** | Logget ind som admin rolle |
| **Software Configuration** | En af følgende browsere skal være downloaded:<br>- Google Chrome<br>- Microsoft Edge<br>- Safari |
| **Steps** | 1. Tryk på "Opret bruger"-knappen i topmenuen.<br>2. Indtast et ikke eksisterende cpr-nummer, navn, samt initialer og vælg en rolle.<br>3. Tryk "Gem" |
| **Expected behaviour** | 1. Brugeren er oprettet i systemet<br>2. Brugeren kan ses, hvis man trykker på "Se alle brugere"-knappen i topmenuen. |

Test case $1_4$

| **Item** | TC02 |
|---|---|
| **Title** | Login som farmaceut |
| **Type** | System test |
| **Priority** | High |
| **Status** | Godkendt |
| **Initial Configuration** | Startsiden er indlæst i browseren |
| **Software Configuration** | En af følgende browsere skal være downloaded:<br>- Google Chrome<br>- Microsoft Edge<br>- Safari |
| **Steps** | 1. Indtast et cpr-nummer for en eksisterende farmaceut i systemet<br>2. tryk "Login"<br>3. Vælg herefter muligheden "Login som farmaceut" |
| **Expected behaviour** | 1. Siden for recept og råvare administration fremvises i browseren |

Test case $2_1$

# 8 Ting til videreudvikling

Som det kan ses på figur 8.1, så er der nogle ting vi ikke har nået at implementere i vores program. Det der ikke er blevet nået at blive implementeret er i forhold til kontrolleringen af afvejningerne, som en laborant laver. Når laboranten indtaster nogle vægte han har afvejet skal det gerne stemme overens med de vægte som skal bruges til recepten.[2]

| Feature | beskrivelse | relevante krav |
|---|---|---|
| Login | Brugeren kan logge ind på systemet alt efter rolle | krav 2, 4, 8.1 |
| Brugeradministration | Admin brugeren kan udfører CRUD for andre brugere. | krav 5 |
| Råvareadministration | Farmaceut kan oprette, redigere råvare, samt oprette og redigere recepter. | krav 6.1, 6.2, 6.3, 6.4, 6.5, 6.5.1 |
| Batchadministation | Produktionsledere rollen kan oprette og se råvarebatches, samt produktionsbatches | krav 7.1, 7.2, 7.3, 7.4 |
| Afvejning | Laborant kan lave afvejninger til bestemte produktionsbatches. | krav 8.3, 8.4, 8.5, 8.6 |
| Bruger nedarvning | Farmaceut skal kunne gøre det samme som produktionsleder, og en produktionsleder skal kunne gøre det samme som en laborant. | krav 6.6, 7.5 |
| Afvejnings tolerance | Systemet kan validerer varierende tolerancer for forskellige råvarer. | krav 8.2 |

Figure 8.1: Feature liste med hvad der er færdigt[5]

**Exceptions**
Håndteringen af diverse fejl og bruger-input, kunne være mere detaljeret, hvis vi havde lavet mere specifikke exceptions. Dette kunne blandt andet have givet mulighed for understøttelse fejlbeskeder ud fra de input-felter hvor der er indtastet noget forkert, hvilket ville fremme brugervenligheden markant.[1]

**Mindre redundans i stylesheets**
Vores topmenu, hover-funktioner og table-views bliver stylet i hver sides tilhørende CSS-fil. Dvs. vi har meget redundans i stylesheetsne. Først og fremmest har vi tables, der har fået tildelt nogle unikke ID'er og classes, som skal bruges i javascript, dem har vi valgt at style ud fra individuelt på hver side. For at forhindre redundansen dér, kunne vi have tildelt dem en class fra en CSS-fil som definerer hvordan f.eks. alt i et table skal styles.
Derudover kunne vi have gjort det samme for topmenuen, som også bliver stylet i hver CSS-fil, hvor topmenuen er relevant.[4]

**Display login besked**
Når brugeren har indtastet sit cpr og det er blevet godkendt i databasen, så ville det være meget godt, hvis der blev vist en besked med "velkommen navn" eller en anden besked i den form, hvorimod den lige nu ikke oplyser noget.[3]

**Sortering på table-views**
En anden feature til videreudvikling kunne være muligheden for at sortere i table-viewet. Pt. har vi kun en søge funktion som kun viser det specifikke man leder efter, men f.eks. at kunne sortere efter nyligt/ældste oprettede recepter, råvarer eller batches, kunne være relevant når listerne bliver større. Sortering efter hvilke råvarer man har i forhold til recepter ville også være en potentiel feature.[2]

# 9 Konklusion

Som helhed er vi tilfredse med det endelige resultat af udviklingsforløbet. De fleste af de features vi havde sat os for at implementere, er blevet implementeret og gennemtestet. Dog fik vi ikke realiseret krav 8.2 fuldendt i systemets afvejnings-feature, da systemet ikke validere på tolerancerne, men blot registrerer dem. Derfor er det op til laboranten at aflæse disse og selv kontrollerer at tolerancerne overholdes.$_5$
Dernæst fandt vi vores tidsestimering en kende urealistisk, men da vi ikke har lavet projekter af dette omfang før, havde vi svært ved at tidsetimerer ordenligt. I forlængelse af dette, havde vi mange uforudsete problemer med Tomcat og vores REST API, hvilket vi burde have taget højde for.$_2$
Alt dette har resulteret i et tidspres, hvilket har frataget os muligheden for at finpudse styling og små fejl. Disse faktorer har dog ingen betydning for systemets overordnede funktionalitet, hvilket også er derfor vi overordnede set er tilfredse med udviklingsprocessen og det resulterede produkt.$_1$

# 10 Bilag

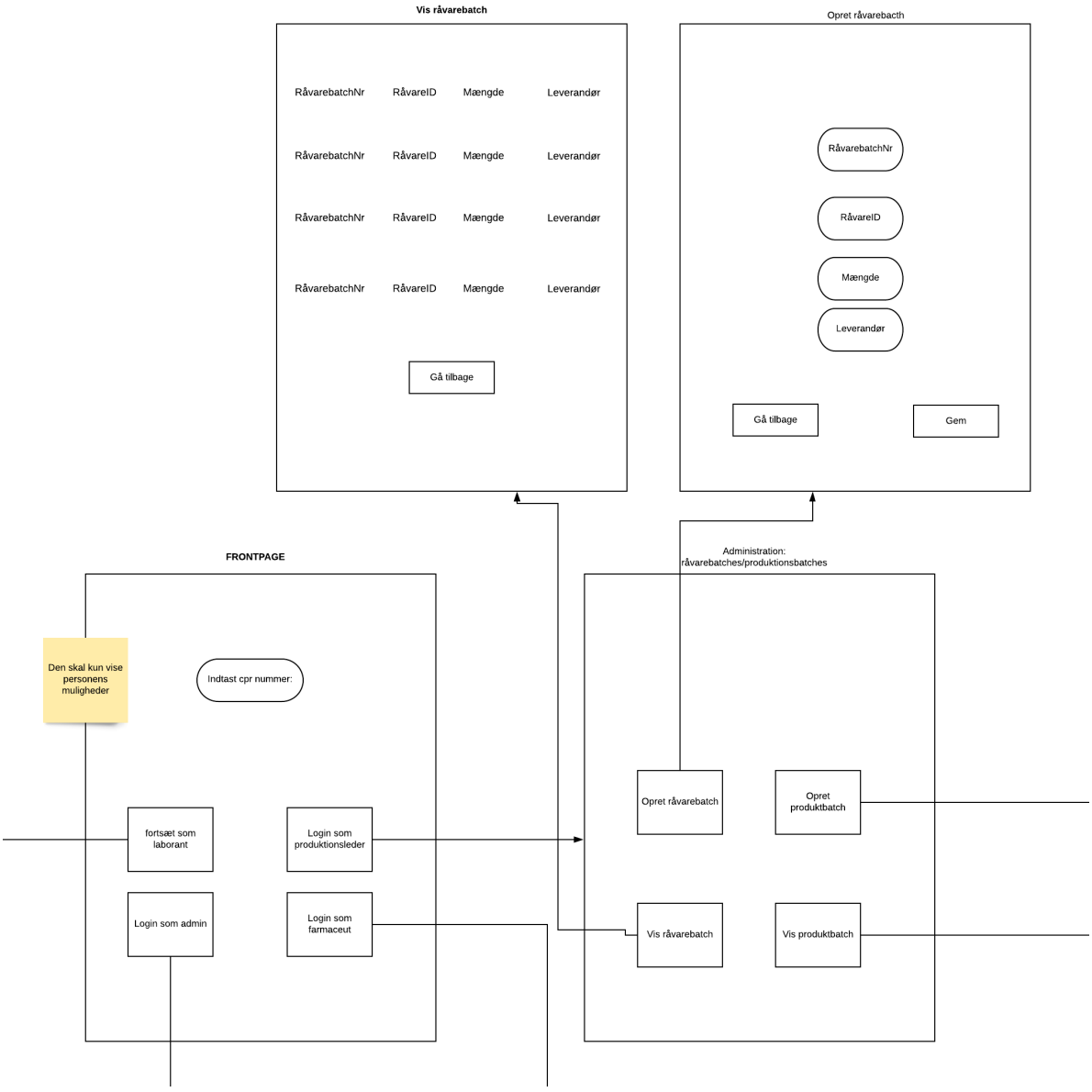


Figure 10.1: Weblayout$_3$