

DET DANSKE VACCINATIONS REGISTER



Indholdsfortegnelse

Indledning	3
------------	---

Systemudviklingsdelen

Brugsscenarie	3
---------------	---

FURPS+	4
--------	---

Use Case Diagram	5
------------------	---

Success- og fejlscenarier	5
---------------------------	---

Domænemodel	6
-------------	---

Programmeringsdelen

Grafisk User Interface	7
------------------------	---

Konklusion

Fejlkilder ved systemet	8
-------------------------	---

Litteraturliste

Kilder	8
--------	---

Bilag

Java kode	9
-----------	---

Indledning

Vi arbejder med en case for Det Danske Vaccinationsregister, som skal bruges til at registrere vaccinationer i en fælles database. Vi analyserer hvem der skal bruge systemet, hvordan og hvad de forskellige aktører skal bruge det til. Der udarbejdes gennem en iterativ process en prototype for den grafiske brugerflade som skal bruges til at registrere oplysninger, behandle dem og for borgeren til at blive oplyst om deres vaccinations historik. I denne iteration vil vi fokusere på registreringen af vaccination, og vi laver en grafisk brugerflade til dette formål.

Brugsscenarie

Patienten kommer til enten klinik eller personlig læge med enten sygdom eller hhv. informationer omkring kommende rejse, der skulle resultere i en foretrukken vaccination. Patientens oplysninger bliver sat ind i systemet, hvorefter man kan påbegynde besøget. Informationer om patienten er CPR nummer efterfølgende bekræftelse på at der er patienten vi har at gøre med, som regel med angivelse af navn. Herefter bekræftes betaling af vaccinen, hvorefter lægen tager over resten. Her registrerer lægen vaccinationen, hvilket gøres med følgende oplysninger: hvem som har ordineret vaccinen, hvem der har indgivet vaccinen, hvem der modtager den, dato samt sted hvor vaccinen skal udføres, hvilken dosis som skal gives, aktive stoffer som vaccinen indeholder, gyldighedstid, ATC kode, batchkode og hvilken sygdom der vaccineres imod.

Alle disse oplysninger er essentielle da man kan få indgivet vaccinationer adskillige steder, såsom hos egen praktiserende læge, på et hospital eller hvis vaccinen er en rejsevaccination, så kan den blive indgivet på klinikker med speciale på rejsevaccinationer. Vaccinen kan også indgives af forskellige individer, som ikke nødvendigvis skal være din læge. Disse personer skal have autoritet på dette felt og kan være en lægesekretær, en sygeplejerske, en klinikassistent eller anden sundhedsfaglig person. Disse oplysninger bliver derefter videregivet til Sundhedsplatformen, som ekstra data om personen, hvor det kun er ens personlige læge eller andre autoriteter, som den enkelte person har givet tilladelse til at se bestemte data, kan se. dvs. ved kommune autoriteter, bestemte lægebesøg mm.

FURPS+ kriterierne består af functional og nonfunctional krav for et system. Functional krav beskriver ting som et system skal kunne. Non-functional krav beskriver hvordan et system fungerer. I FURPS+ kriterierne er F der står for Functional de funktionelle krav og resten er non-functional.

FURPS+ – kriterierne som er kravene for et system, består af kategorierne:

Functional—features, capabilities, security.

Hver bruger skal have login. Systemet skal kunne genkende den person der bruger systemet om man er fra sundhedsvæsenet eller patient.

Hvis det er en fra sundhedsvæsenet skal brugeren efter login udfylde de nødvendige oplysninger om vaccinationen, som er oplysninger om hvem der har ordineret vaccinationen, hvem der har indgivet vaccinationen, hvem der har modtaget den, betaler, dato, sted, dosis, aktive stoffer, gyldighedstid, ATC kode, batchnummer, og sygdom der vaccineres mod.

Ved registrering skal det være muligt at notere om borgeren selv betaler, eller om det er sygesikringen, der skal dække udgiften.

Hvis det er en patient skal den gå videre til alle relevante oplysninger om patientens vaccination, og de oplysninger skal være tilgængelige til at udtrække for patienten i et struktureret format.

Usability—human factors, help, documentation.

De felter som man skal udfylde information skal være dokumenteret således at brugeren forstår hvad der skal udfyldes i forskellige felter.

Systemet skal være let at læse og der skal være en brugervejledning så uanset niveau af viden om IT skal brugeren let kunne bruge systemet. Brugeren skal ikke være overvældet af unødvendige oplysninger, brugerens opmærksomhed skal være direkte på de nødvendige oplysninger.

Reliability—recoverability, predictability.

Systemet skal være tilgængeligt hele tiden af døgnet undtagen fra søndag morgen kl 5. til kl. 12 som skal bruges til at opdatere systemet.

Systemet skal gendannes ved tilfælde af at den lukker ned.

Alt data skal kunne være lagret efter hver registrering og der skal også skabes backups hver 12 timer. Hver uge skal backups synkroniseres til en sky og et eksternt system og det data skal bekræftes manuelt så at det er det rigtige data der er blevet gemt korrekt.

Performance—response times, throughput, accuracy, availability, resource usage.

Det meste af tiden skal systemet svarer på brugernes forespørgsler på maksimum 0.3 sekunder. Der må ikke være variation og responstiden skal være den samme for alle funktioner. Systemet skal kunne virke med den samme responstid selv når 1000 personer trykker på de samme forespørgsler på samme tid. Data skal være tilgængeligt max 5 minutter efter at det er blevet registreret af en fra sundhedsvæsenet.

Supportability—adaptability, maintainability, internationalization, configurability.

It supportere skal kunne få feedback fra brugerne og kunne komme ind og lave analyser, finde fejl i systemet og rette dem.

Implementation—resource limitations, languages and tools, hardware.

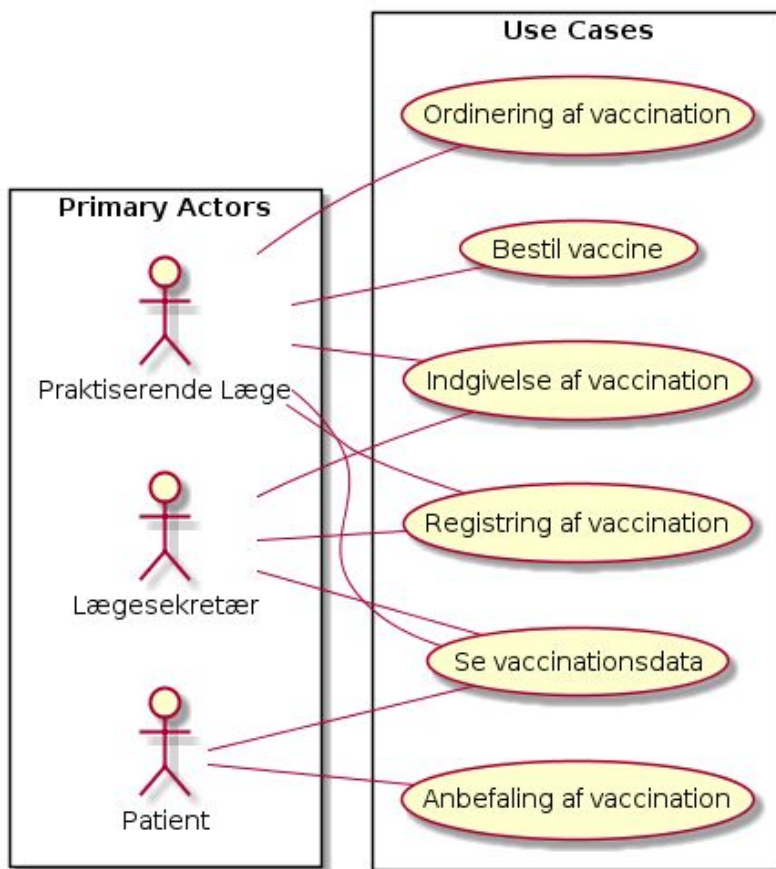
Systemet skal virke på alle browsere og computere, nyere end 2010.

Interface—constraints imposed by interfacing with external systems.

Man skal kunne udtrække og lagre filer på det system man bruger dette system på. F.eks. kan alle de basale Windows, MacOS, Linux og Unix operativsystemer godt gemme i de fleste filformater. Det skal være muligt i et system der bruger dette system at kunne gemme data der har med afregning af vaccinationer at gøre og eksportere det til en XML-fil og samtidig også kunne have andre systemer kørende der sørger for afregning med lægerne.

Andre systemer må ikke selv kunne trække data ud, og sundhedsvæsenet og IT-supporter burde få notifikation om det hvis det sker.

Use Case Diagram



Success- og fejlscenarier

Success scenarie: Data er i de rigtige formater og korrekte.

Fejl scenarie: At der er mangler i data og at den derfor ikke går igennem eller de informationer som er udfyldt er ukorrekte.

Vaccinationsdata

Success scenarie: At både patienten og indgiveren er i stand til og se vaccinationsdata.

Fejl scenarie: De er ikke i stand til og se data eller at det kun er patienten som ikke kan se data.

Anbefaling

Success scenarie: At man giver den rigtige anbefaling ud fra den data man har.

Fejl scenarie: Man får en forkert anbefaling. Dette kunne være at der anbefales at man bliver vaccineret imod noget bestemt, men at man bliver vaccineret imod noget som kun børn bliver vaccineret imod.

Use case 1 name	Registrering af vaccination
Actors	<ul style="list-style-type: none"> - Praktiserende læge - Lægesekretær
Use case description	Et af de to actors registrere patientes vaccinationer til vaccinationsdata
Succes scenarie	At registrering er skrevet rigtigt ind, og at det bliver gemt sikkert i vaccinations data
Fejlscenarie	At et af de to actors indskrifer data forkert, at der er fejl i systemet og at det derfor ikke kan gemmes

Use case 2 name	Indgivelse af vaccination
Actors	<ul style="list-style-type: none"> - Praktiserende læge - Lægesekretær
Use case description	Lægen og lægesekretæren indgiver vaccinationen til patienterne
Succes scenarie	At lægen indgiver den rigtige vaccination til patienten
Fejlscenarie	At lægen indgiver den forkerte vaccination til patienten i forhold til fejl i bestilling af vaccine

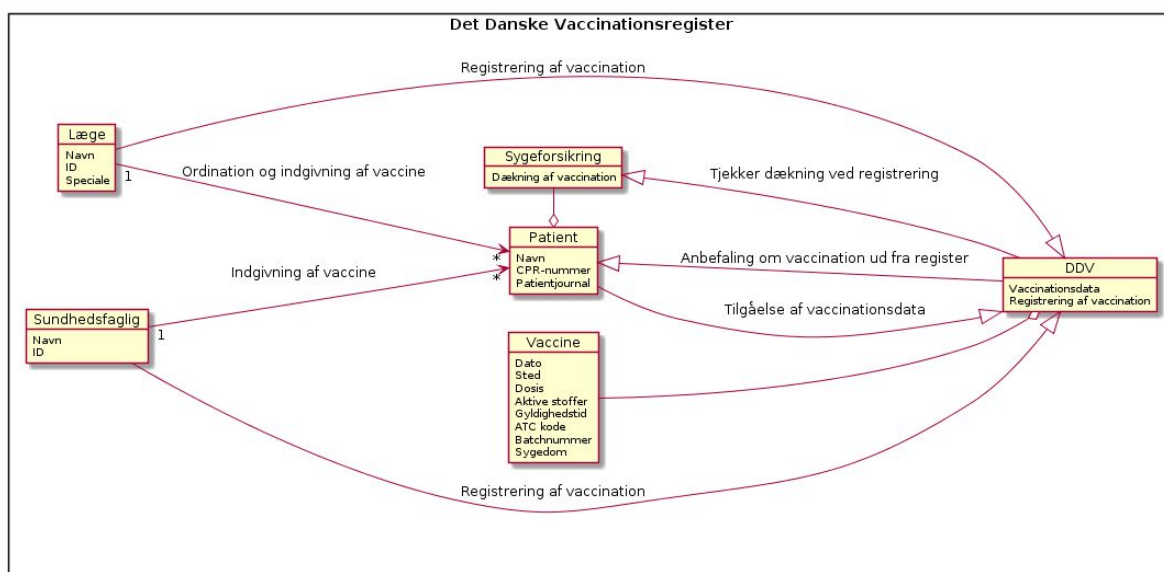
Use case 3 name	Se vaccinationsdata
Actors	<ul style="list-style-type: none"> - Praktiserende læge - Lægesekretær - Patient
Use case description	Her kan alle tre actors se hvilken vaccination patienten har taget
Succes scenarie	At alle de tre actors kan se patientens data
Fejlscenarie	At actors, praktiserende læge og lægesekretær kun kan se data eller, at det data patienten kan se er begrænset

Domænemodel

Vi har lavet en model som illustrere relationerne mellem de forskellige objekter i vores system. Objekterne i systemet er læger, sundhedsfaglige personale, vacciner, patienter, patienternes sygeforsikring og Det Danske Vaccinationsregister.^[1]

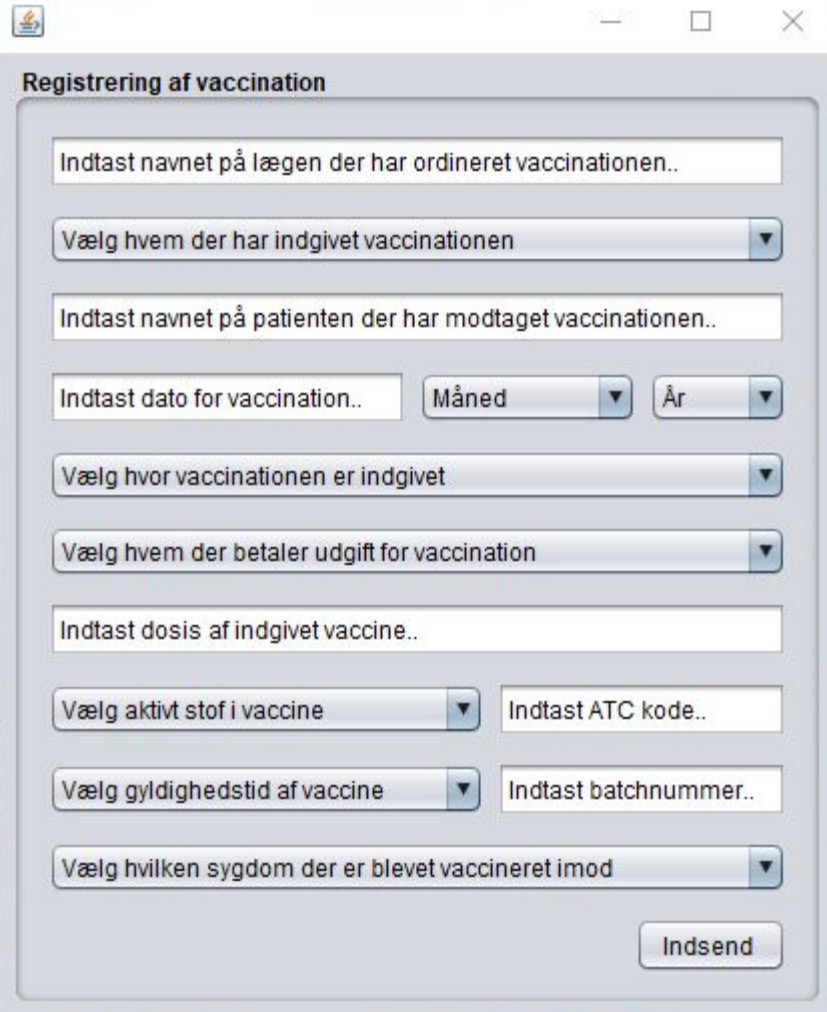
Objekternes beskrivelse fortæller hvilke informationer som de giver til de andre objekter, og pilene beskriver relationen mellem informationerne.

I vores domænemodel har vi læger og sundhedsfaglige, som begge kan indgive og registrere vaccination, men kun læger kan ordinere vaccination. Sygeforsikringen bruges til at forsyne registreringen med information om hvem der skal betale for patientens vaccination, og Det Danske Vaccinationsregister har informationer om de forskellige vacciner, og hvilke patienter har fået hvilke patienter. Disse informationer bruge til at give patienter anbefalinger om vacciner.



Grafisk User Interface

Vi har lavet en grafisk brugerflade til use case: *registrer borgervaccination* i DDV. Brugerfladen er en blanding af tekstfelter med vejledning, og drop-down menuer som gør det muligt at kategorisere informationer i databasen, hvilket blandt andet er relevant for at adskille borger- og sygesikring betalte vaccinationer.



The screenshot shows a web-based form titled "Registrering af vaccination". The form contains the following fields and controls:

- Text input: "Indtast navnet på lægen der har ordineret vaccinationen.."
- Drop-down menu: "Vælg hvem der har indgivet vaccinationen"
- Text input: "Indtast navnet på patienten der har modtaget vaccinationen.."
- Date selection: "Indtast dato for vaccination.." with separate drop-downs for "Måned" and "År".
- Drop-down menu: "Vælg hvor vaccinationen er indgivet"
- Drop-down menu: "Vælg hvem der betaler udgift for vaccination"
- Text input: "Indtast dosis af indgivet vaccine.."
- Drop-down menu: "Vælg aktivt stof i vaccine"
- Text input: "Indtast ATC kode.."
- Drop-down menu: "Vælg gyldighedstid af vaccine"
- Text input: "Indtast batchnummer.."
- Drop-down menu: "Vælg hvilken sygdom der er blevet vaccineret imod"
- Submit button: "Indsend"

Konklusion

Vi kan ud fra vores analyse af systemet konkludere at der er adskillige fejlkilder, men det er fejlkilder som kan forhindres ved at kode systemet til at fange dem. Ud fra produktet har vi lavet use cases, som beskriver processen og samarbejdet mellem patient, læge og vaccinationsprogrammet.

Når man kigger på at gøre patienter mere empowered og med en beskrivelse af DDV er det nu muligt ved at følge vores beskrivelse af hvordan systemet kommer til at se ud og hvilke omgivelser det er bedst at implementere systemet i. Ud fra FURPS+ kriterieliste og use cases kan man se på de kategorier som systemet skal kunne så man er sikker på at man ikke har udeladt en vigtig del.

Vi ved også allerede nu hvilke andre dele af systemet som skal udarbejdes, blandt andet den grafiske brugerflade som borgerne skal bruge til at se deres vaccinationer.

Litteraturliste

1. "Object Diagram" | <http://plantuml.com/object-diagram>
2. "Applying uml and patterns s. 42" Craig Larman

Bilag

package DDV;

```
public class NewJFrame extends javax.swing.JFrame {
    public NewJFrame() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    private void jTextField3ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    private void jTextField4ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    private void jTextField5ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    private void jComboBox2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new NewJFrame().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JComboBox<String> jComboBox2;
    private javax.swing.JComboBox<String> jComboBox3;
    private javax.swing.JComboBox<String> jComboBox4;
    private javax.swing.JComboBox<String> jComboBox5;
    private javax.swing.JComboBox<String> jComboBox6;
    private javax.swing.JComboBox<String> jComboBox7;
    private javax.swing.JComboBox<String> jComboBox8;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    // End of variables declaration
}
```

DET DANSKE VACCINATIONS REGISTER

20:04

≡ **Ordinering af vaccination**

CPR-nummer

Indtast for at finde patient

Vælg vaccine

Tryk for at vælge

Indholdsfortegnelse

Indledning	3
------------	---

Systemudviklingsdelen

Aktivitetsdiagrammer	3
----------------------	---

System sekvensdiagrammer	5
--------------------------	---

Design klassediagram	8
----------------------	---

Mockup	9
--------	---

Programmeringsdelen

Brugergrænseflade	10
-------------------	----

Bilag

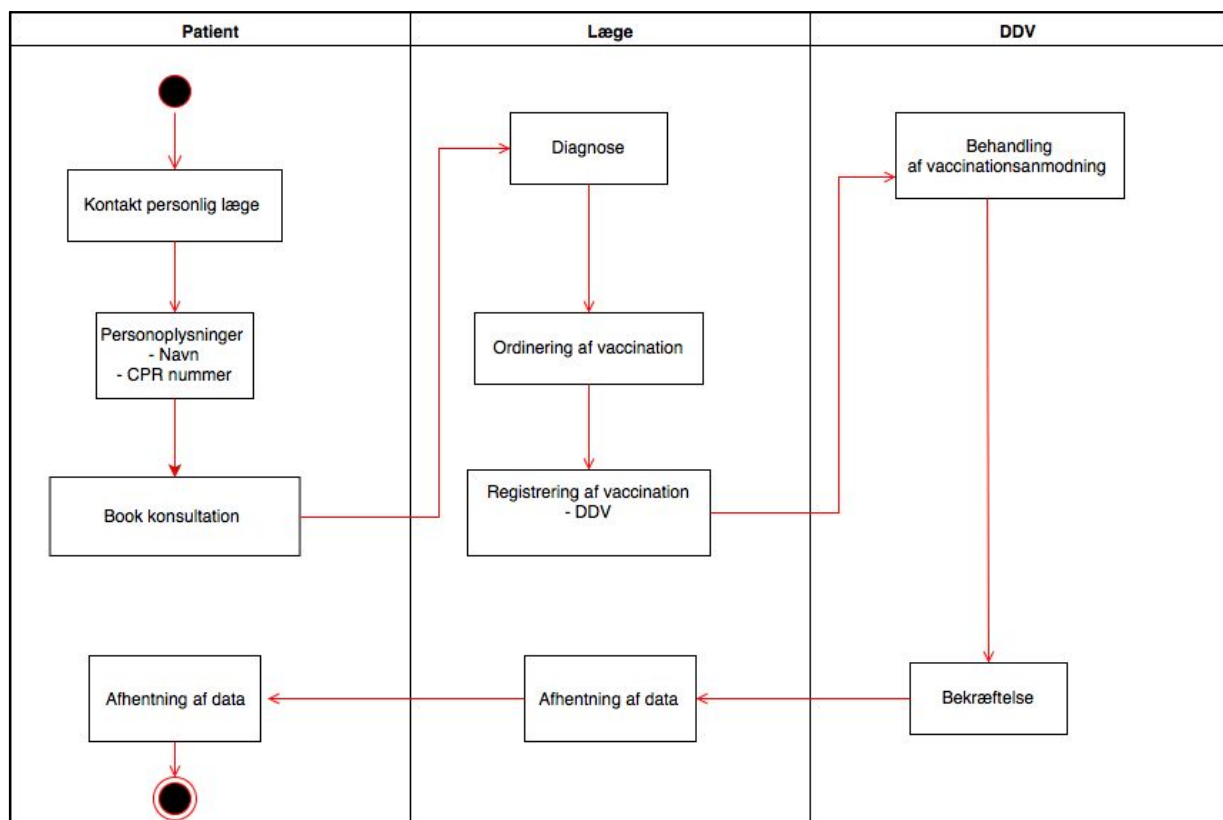
Java kode	10
-----------	----

Use cases fra iteration 1	12
---------------------------	----

Indledning

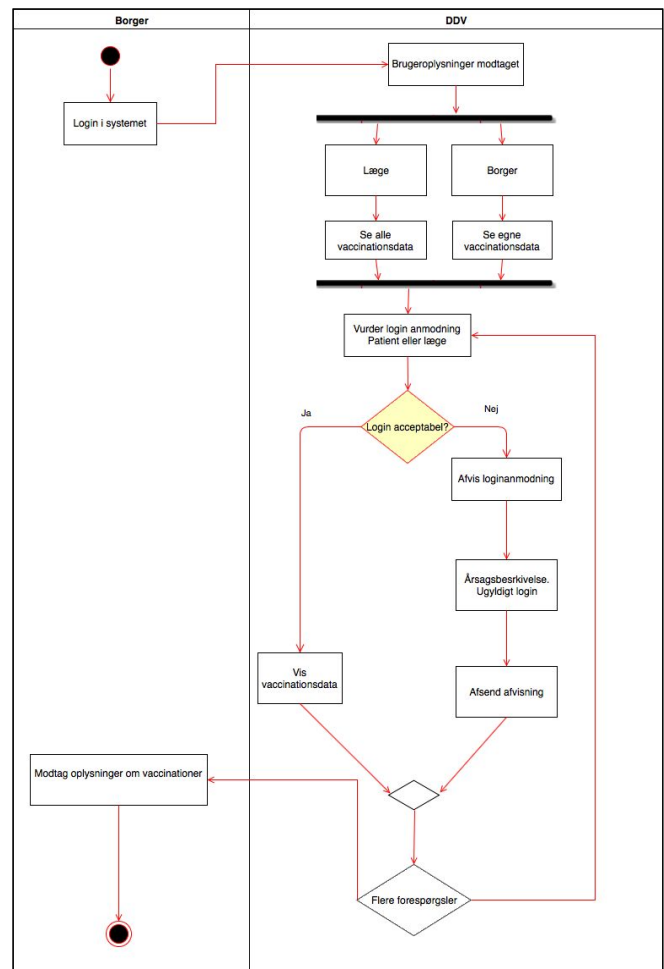
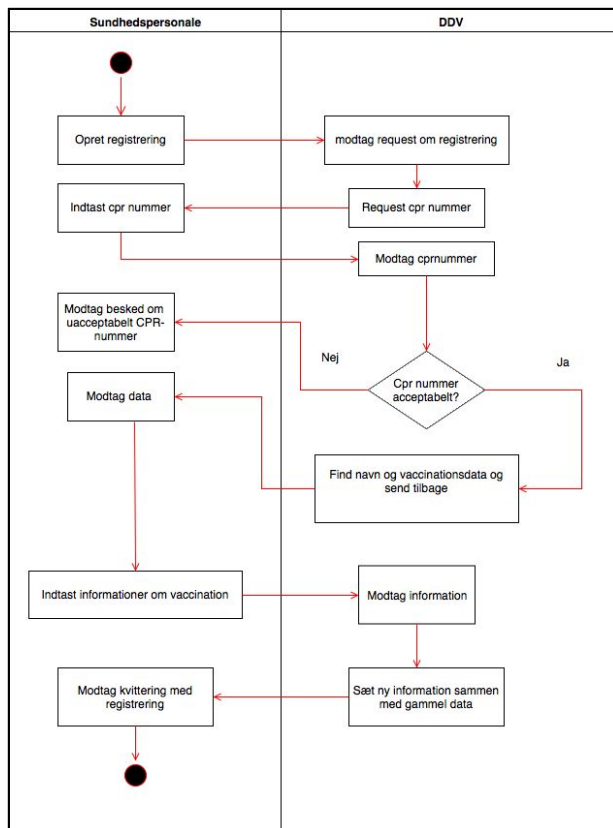
Vi arbejder med en case for Det Danske Vaccinationsregister, som skal bruges til at registrere vaccinationer i en fælles database. Vi analyserer hvem der skal bruge systemet, hvordan og hvad de forskellige aktører skal bruge det til. Der udarbejdes gennem en iterativ process en prototype for den grafiske brugerflade som skal bruges til at registrere oplysninger, behandle dem og for borgeren til at blive oplyst om deres vaccinations historik. I denne iteration vil der fokuseres på systemudvikling indenfor aktivitetsdiagrammer, system sekvensdiagrammer, design klassediagram og mockups.

Aktivitetsdiagrammer



Det her aktivitetsdiagram viser ud fra vores brugsscenarie hvordan en behandling ville foregå, hvor man starter med at kontakte lægen og booke konsultation, hvorefter lægen vurderer hvilken vaccination man har brug for og registrere det i DDV som så vil være synligt og muligt til afhentning for patienten.

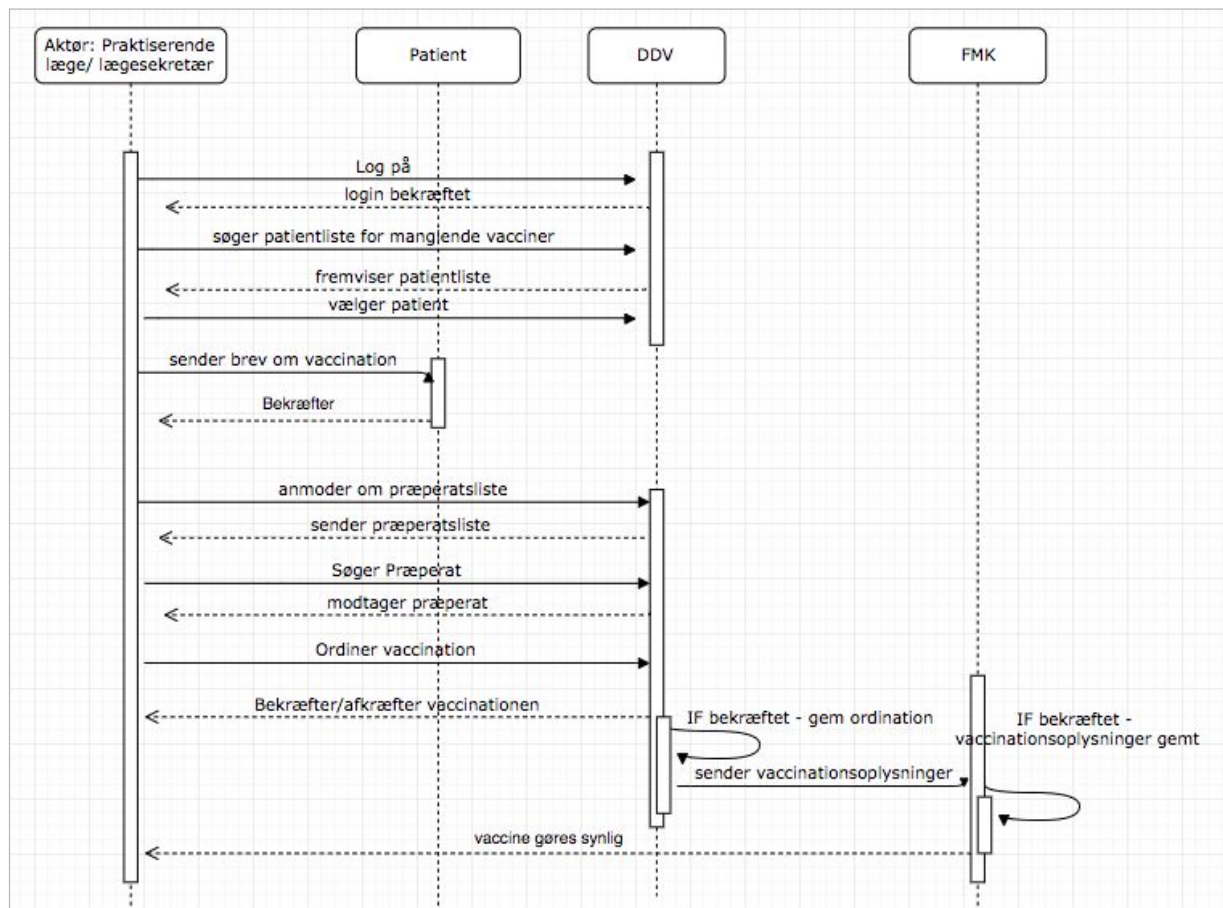
De her to aktivitetsdiagrammer viser detaljeret for hvad der sker i systemet hvis det er lægen eller patienten der logger ind.



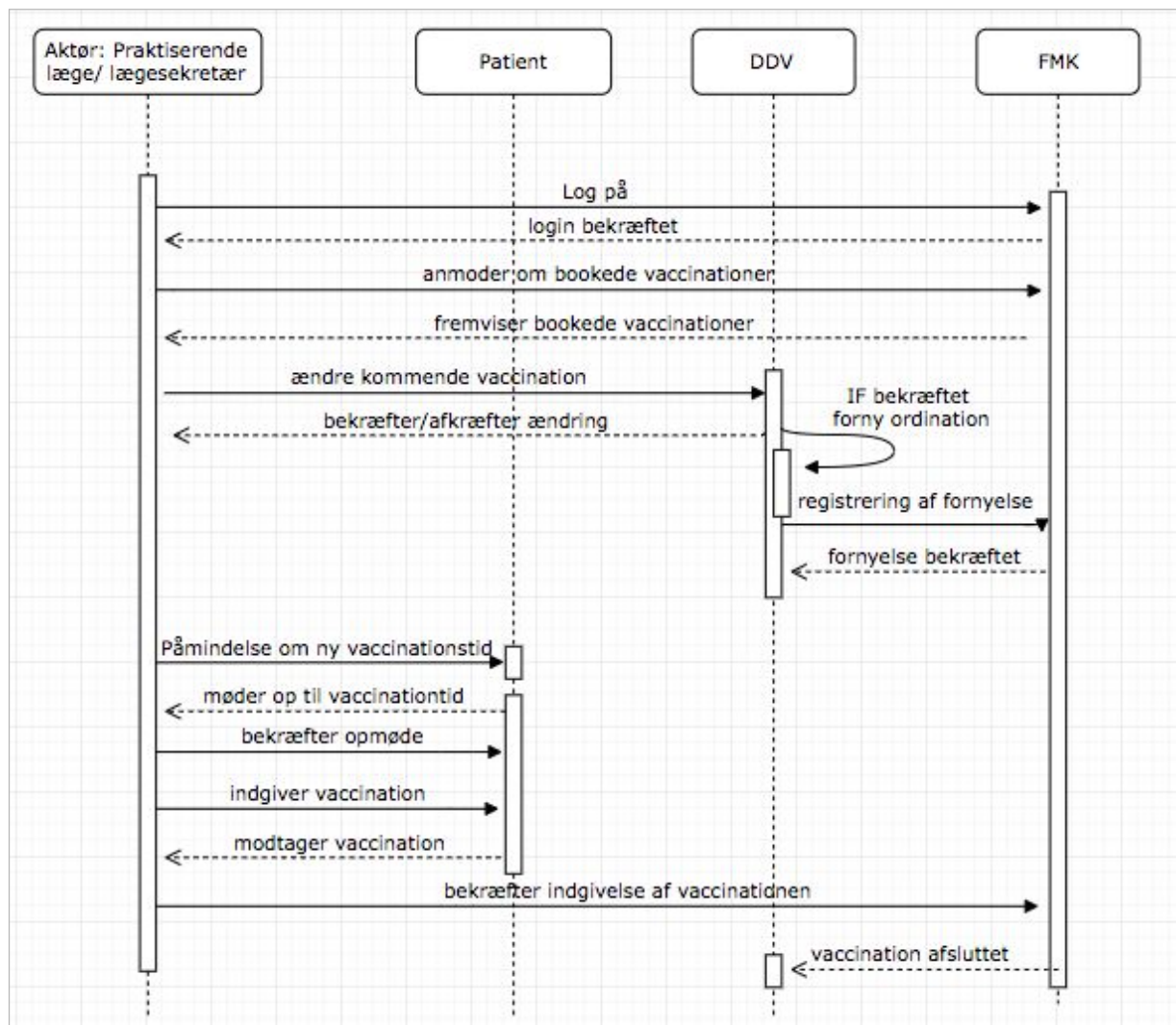
System sekvensdiagrammer

På baggrund af vores tre use cases, har vi lavet tre sekvensdiagrammer ud fra hver enkelte use case. her kan man se forløbet i hvert usecase mellem vores aktører samt det Fælles Medicinkort (FMK) og Det Danske Vaccinationsregister(DDV).

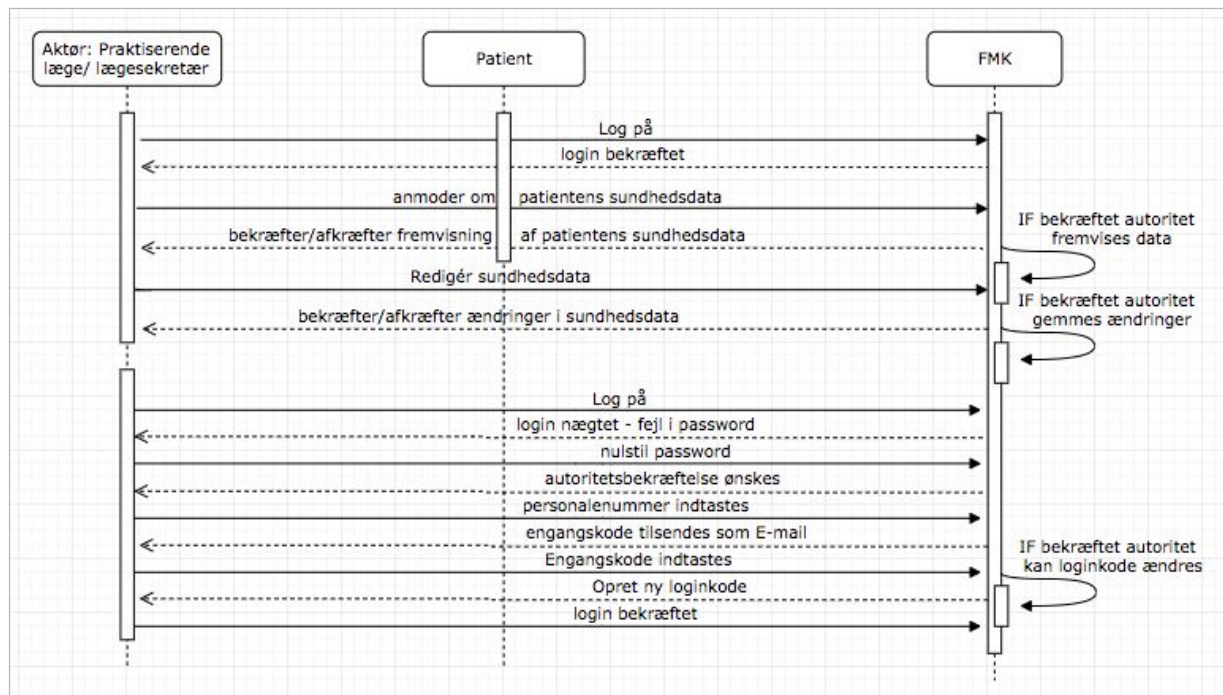
Use case name: Registrering af vaccination



Use case name: *Indgivelse af vaccination*



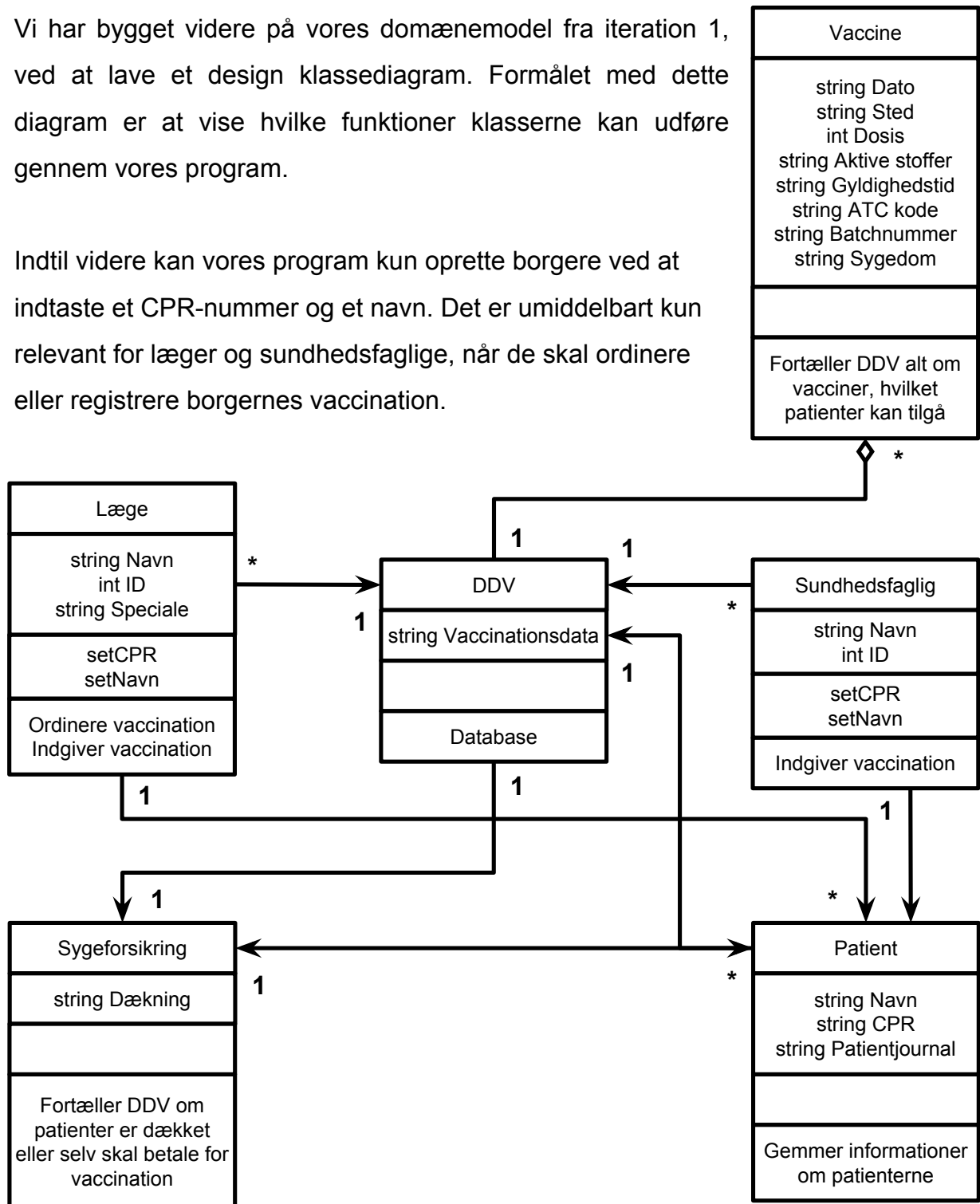
Use case name: Se vaccinationsdata



Design klassediagram

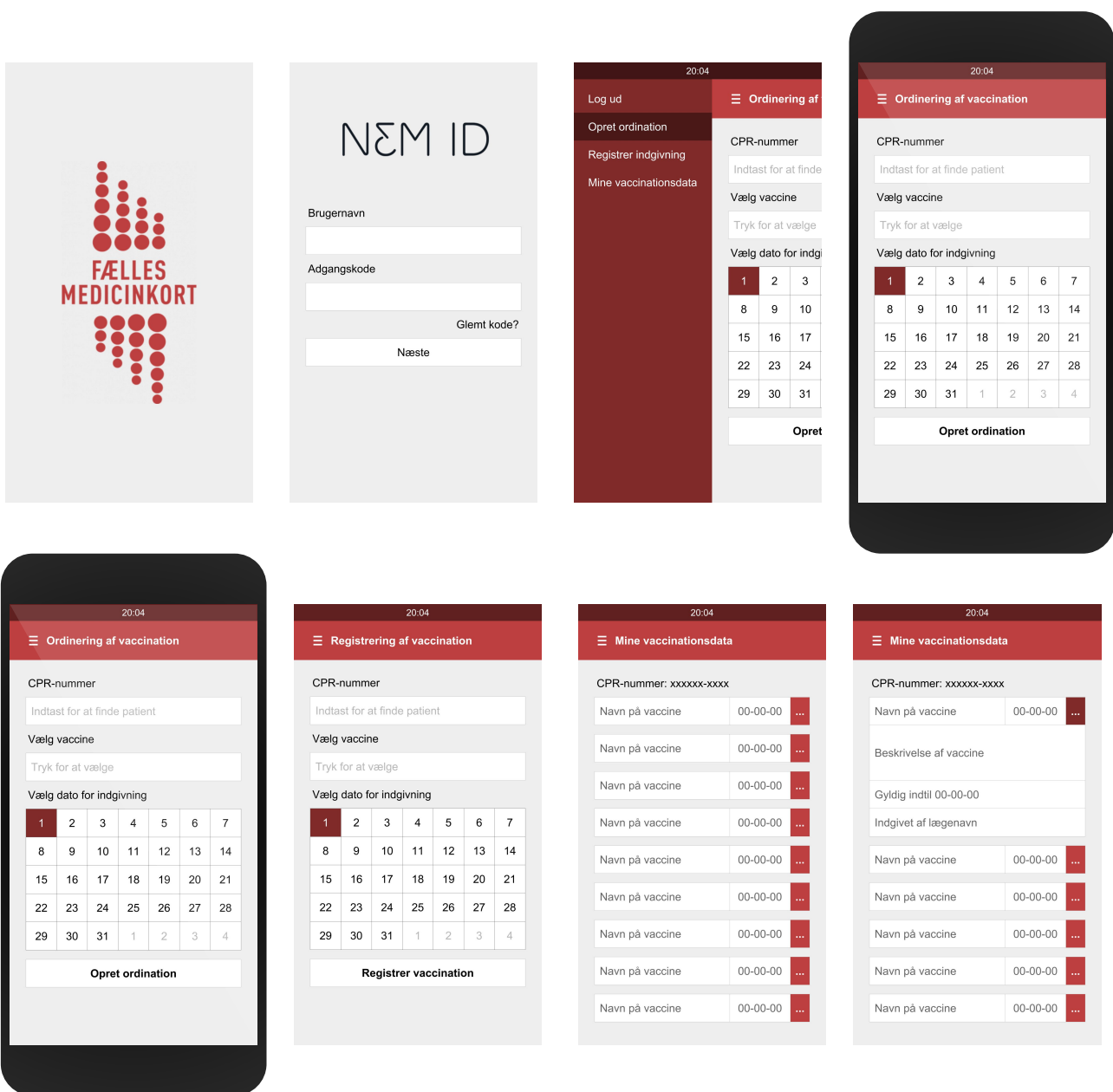
Vi har bygget videre på vores domænemodel fra iteration 1, ved at lave et design klassediagram. Formålet med dette diagram er at vise hvilke funktioner klasserne kan udføre gennem vores program.

Indtil videre kan vores program kun oprette borgere ved at indtaste et CPR-nummer og et navn. Det er umiddelbart kun relevant for læger og sundhedsfaglige, når de skal ordinere eller registrere borgernes vaccination.

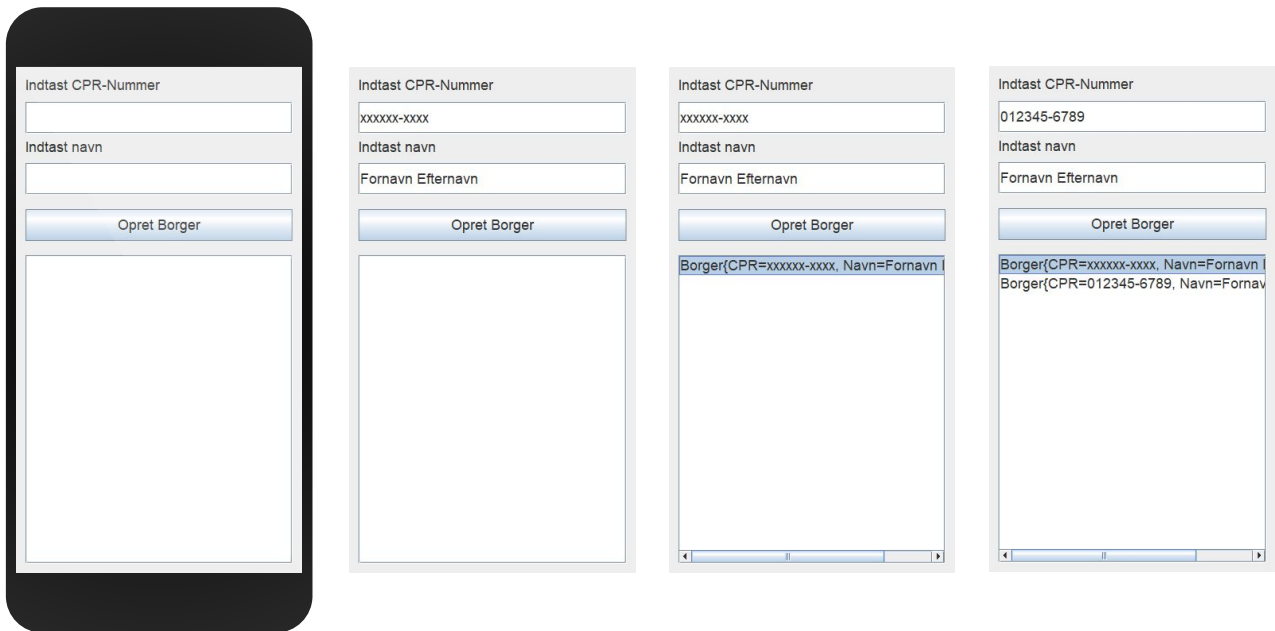


Mockup

Vi har lavet en illustration af vores program i form af et mockup. Formålet med en skitse er at bestemme hvilke funktioner skal vises i programmet, men et mockup går mere i dybden med hvordan funktionerne skal se ud. Vores program har 3 grupper af brugere: læger, sundhedsfaglige og borgere. Vores mockup er fra lægernes synspunkt, fordi de har adgang til alle funktionerne. Sundhedsfaglige har ikke adgang til oprettelse af ordination, og borgere har kun adgang til deres personlige vaccinationsdata. Vi har valgt at lave programmet i et format som passer til en smartphone, fordi de er mere tilgængelige og lettere at illustrere.



Brugergrænseflade



Java kode

Main klasse

```
public class Main {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame();  
        frame.setVisible(true);  
    }  
}
```

JFrame

```
private void buttonOpretBorgerActionPerformed(java.awt.event.ActionEvent evt)  
{  
    Borger b = new Borger();  
    b.setCPR(textCPR.getText());  
    b.setNavn(textNavn.getText());  
    model.addElement(b);  
}
```

Borger klasse

```
public class Borger {  
    private String CPR;  
    private String Navn;  
    public Borger() {  
    }  
    public Borger(String CPR, String Navn) {  
        this.CPR = CPR;  
        this.Navn = Navn;  
    }  
    public String getCPR() {  
        return CPR;  
    }  
    public void setCPR(String CPR) {  
        this.CPR = CPR;  
    }  
    public String getNavn() {  
        return Navn;  
    }  
    public void setNavn(String Navn) {  
        this.Navn = Navn;  
    }  
  
    @Override  
    public String toString() {  
        return "Borger{" + "CPR=" + CPR + ", Navn=" + Navn + '}';  
    }  
}
```

Use cases fra Iteration 1

Use case name	Registrering af vaccination
Scope	DDV
Primary actors	Praktiserende læge Lægesekretær
Pre-conditions	1. Server skal være aktiv således at personale kan logge på 2. Felterne til udfyldning af registrering skal stå klart 3. Skal være i stand til og gemme registreringer
Success guarantees	Patienten er blevet registreret.
Main success scenario	1. De er i stand til og logge på 2. Personale kan finde vaccination 3. Lægen kan påbegynde registreringen 4. Registreringen bliver registreret i DDV og er synlig for personale.
Extensions	1a. fejl ved login af personale 1. Systemet skal informere hvorfor fejl opstod. Dette er i form af forkert navn og kode eller hvis serveren er nede. 1b. kan ikke finde vaccination. 1. Fortæller hvor fejlen ligger i forhold til stavefejl osv.
Frequency of occurrence	Hver gang borgeren modtager en vaccination.

Use case name	Indgivelse af vaccination
Scope	DDV
Primary actors	Praktiserende læge Lægesekretær
Pre-conditions	At den rigtige vaccination står klar til lægen før patienten får indgivet vaccinationen.
Success guarantees	Patienten får indgivet sin vaccination
Main success scenario	<ol style="list-style-type: none"> 1. Personale bestiller vaccination, ud fra registrering af vaccinationer til patientens behov. 2. Personale indgiver vaccination til patient.
Extensions	<ol style="list-style-type: none"> 1a. Der er blevet fremstillet en forkert vaccination. <ol style="list-style-type: none"> 1. Personale er i stand til at bestille en ny vaccination omgående. 1b. Personale informerer patient omkring fejlen. <ol style="list-style-type: none"> 1. Patienten får en ny tid til indgivelse af vaccination
Frequency of occurrence	Hver gang patient skal have indgivet en vaccination

Use case name	Se vaccinationsdata
Scope	FMK
Primary actors	Praktiserende læge Lægesekretær Patient
Pre-conditions	Serveren er oppe og klar til at forbinde sundhedspersonale og patient til data
Success guarantees	Patienten og autoriteret sundhedspersonale er i stand til at se data
Main success scenario	<ol style="list-style-type: none"> 1. Patienten og sundhedspersonale kan logge på systemet og se data for patienten 2. Patienten kan se alle sine data. 3. Autoriteret sundhedspersonale er i stand til at redigere data i form af usete fejl. 4. Autoriteret sundhedspersonale kan gemme rettelser.
Extensions	<p>1a. Fejl ved login til data.</p> <ol style="list-style-type: none"> 1. login-systemet fortæller at der er fejl. <p>1b. Fortæller hvad præcis fejlen er, og hvordan man kan rette fejlen. Dette er i forhold til glemt login eller udelukkelse af system i form af internetforbindelse eller uautoriseret adgang.</p> <ol style="list-style-type: none"> 1. Patient eller personale retter fejlen i forhold til forkert brugernavn eller kode og logger på.
Frequency of occurrence	Hver gang borgeren har behov for og se vaccinationsdata eller personale.

DET DANSKE VACCINATIONS REGISTER

20:04

≡ Ordinerer af vaccination

CPR-nummer

Indtast for at finde patient

Vælg vaccine

Tryk for at vælge

Vælg dato for indgivning

1

2

3

4

5

6

7

Indholdsfortegnelse

Indledning	3
------------	---

Systemudviklingsdelen

Peer Reviews	3
--------------	---

Success- og fejlscenarier	4
---------------------------	---

Bruger- og accepttest	5
-----------------------	---

Programmeringsdelen

Anvendt DDL og DML	6
--------------------	---

ER diagram	6
------------	---

Java program	7
--------------	---

SQL kode	8
----------	---

Bilag

Java kode	9
-----------	---

Genaflevering

Aktivitetsdiagrammer	12
----------------------	----

Klassediagram	13
---------------	----

Indledning

I denne her iteration har vi som opgave at virkeliggøre vores mockup som vi lavede i forrige iteration. Her skal vi som udgangspunkt lave en program, der er tilknyttet til en database. Her skal programmet være i stand til og oprette borgere i databasen via vores program samt hente og modificere borgernes oplysninger. For og styrke vores kvalitet vil vi også gennemgå en peer review med en anden gruppe, hvor vi giver hinanden feedback. Som afslutning vil vi lave en bruger og accepttest, samt identificere de 3 vigtigste success og fejlscenarier ud fra vores use cases.

Peer Reviews

I denne opgave har vi med en anden gruppe sat os ned, og givet hinanden konstruktiv feedback. Her fokuserede vi på og kommentere på hinandens use cases, da vi mente det var noget vi kunne tage hånd om. Vi startede med at fortælle den anden gruppe at de havde punkter i use casen, som ikke tilhørte under en casual use case men en fully dressed use case. Udover det, så mente den anden gruppe man også kunne tilføje nogle flere punkter til vores main success scenario. Dette var inden for *Registrering af vaccination* og *indgivelse af vaccination*. I *Registrering af vaccination* tilføjede vi et ekstra punkt som har nummer 5, da den anden gruppe mente det var en væsentlig mangel.

1. De er i stand til og logge på
2. Personale kan finde vaccination
3. Lægen kan påbegynde registreringen
4. Registreringen bliver registreret i DDV og er synlig for personale.
5. Registrering bliver gemt.

Det er vigtigt at man kan se gemte registreringer, hvis der nu skulle opstå en fejl. Så er det godt man kan gå tilbage og finde ud af hvor fejlen er opstået, som måske kunne være forkert registrering af vaccination.

Her ser vi en liste af succes scenarier samt fejlscenarier, som vi mener har størst betydning. Sundhedspersonale skal være i stand til og logge på. Hvis de ikke kan komme på, så kan de heller ikke ordinere medicinen, og det samme gælder med og finde vaccinen. Ved forkert stavning af vaccination, så er det vigtigt at man er i stand til og rette efter sig. Dette er i forhold til patientens sikkerhed.

Success- og fejlscenarier

Et af vores successscenarier er at lægen er i stand til at finde vaccinationen for at ordinere én til patienten. Det betyder at systemet skal kunne vise hvilken vaccination en patient skal have samt se en liste af vaccinationer.

Vores andet scenarie er, at den pågældende patient samt sundhedspersonalet skal kunne logge på systemet og se patientdata, hvilket betyder, at systemet skal kunne give adgang til autoriseret sundhedspersonale samt for den patient hvis sundhedsdata, der er søgt på. Der gives adgang til forskellige slags data, for henholdsvis sundhedspersonale og patient.

Vores tredje scenarie er at når sundhedspersonale indgiver den ordinerede vaccination til patienten, så gemmes dataen for ordineringen efter indgivelsen i DDV. Dette betyder, at systemet skal kunne registrere patienten og vaccinationen, og kunne gemme det i en database til senere afhentning.

Extensions

1. Når lægen skal finde en person i databasen til at ordinere en vaccination og personen ikke eksisterer, så skal lægen registrerer patienten i databasen, hvorefter lægen kan ordinere den vaccination som patienten skal have indgivet.
2. Hvis den vaccination der skal indgives er skrevet forkert ved en fejl så skal man kunne være i stand til at redigere data samtidig være i stand til og se hvor fejlen er opstået.
3. Når lægen registrerer en patient i databasen og systemet ikke gemmer patienten senere på dagen skal der så komme en notifikation til lægen at patienten ikke er gemt, og man skal kunne kontakte support.

Bruger- og accepttest

Peter K. Christiansen CPR-nummer(1206902115) har fået en allergi og skal oprettes i databasen og ordineres.

Accepttest

Spørgsmål	Svar (sæt ét kryds)	
	Ja	nej
Oprette en borger? Lykkedes det? (30 sek.)		
Hvis nej, hvad gik galt?		
Henter data fra databasen? Lykkedes det? (30 sek.)		
Hvis nej, hvad gik galt?		
Redigere i databasen? Lykkedes det? (40 sek.)		
Hvis nej, hvad gik galt?		
Gemmer ny-redigeret data? Lykkedes det? (60 sek.)		
Hvis nej, hvad gik galt?		

Brugertest

Spørgsmål	Svar (sæt ét kryds)			
	Utilfreds	Mindre tilfreds	Tilfreds	Meget tilfreds
Hvor hurtigt hentes patienterne databasen?				
Hvor overskueligt er designet?				
Fremkommer det tydeligt hvilke knapper man skal trykke?				

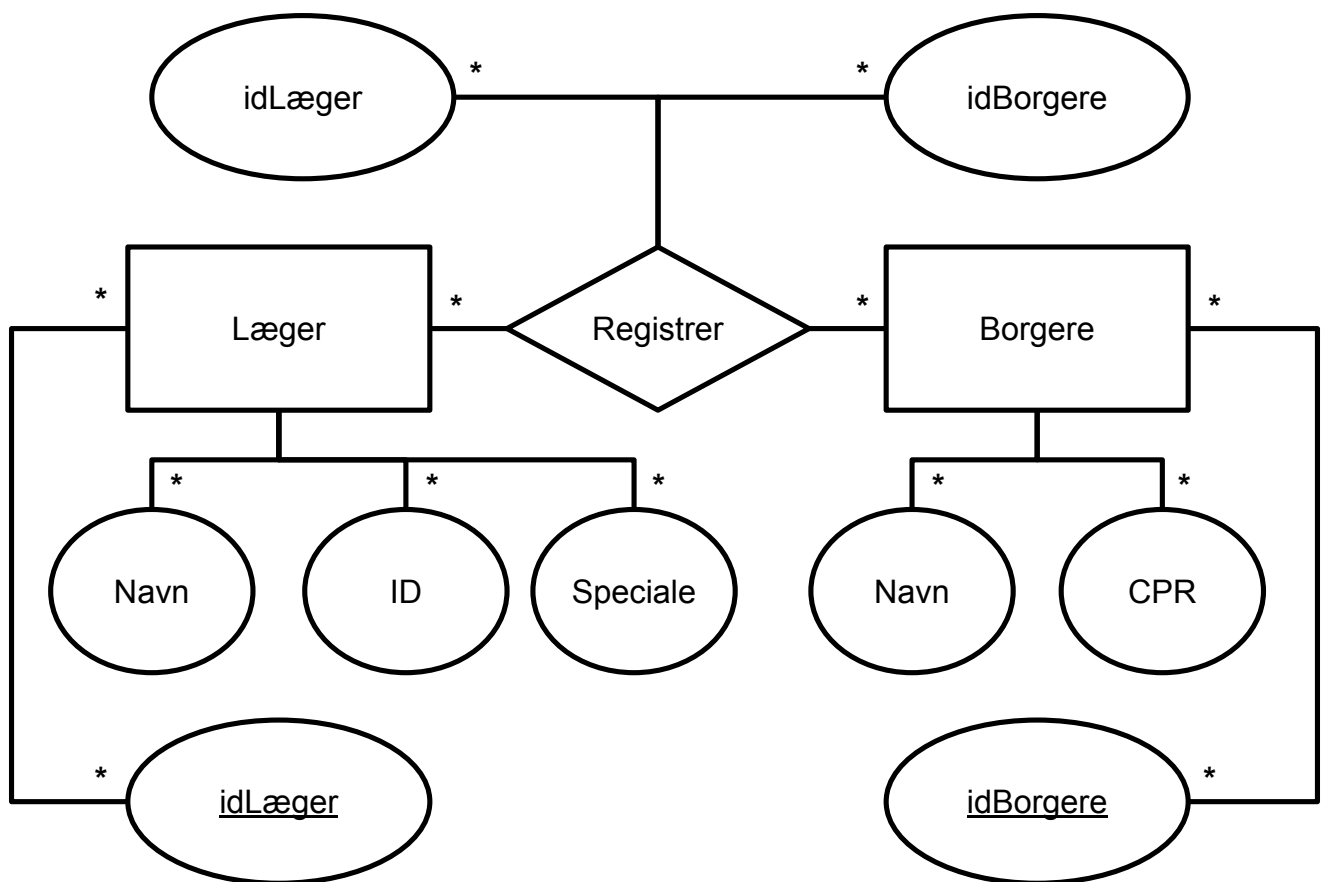
Anvendt DDL og DML

Vi brugte SQLite til at lave en database og i den brugte vi først DDL til at oprette en tabel Borger hvor vi har angivet kollonerne CPRNummer, Navn og datatype.

Så bruger vi DML til at indsætte data ind i tabellen Borger, og det data vi har sat ind er: "1234567890" , "Navn Efternavn".

ER diagram

Physical data model



Java program

Vi henter data fra vores SQL database ind i vores program.

The screenshot shows a Java GUI window with a title bar containing a small icon and standard window controls (minimize, maximize, close). The window is divided into two main sections. On the left, there is a form with two input fields: 'Indtast CPR' and 'Indtast navn'. Below these fields are three buttons: 'Opret borger', 'Opdater borger', and 'Hent database'. The 'Hent database' button is highlighted with a blue border. On the right, there is a text area displaying the text 'Borger [CPR = 1234567890, Navn = Fornavn Efternavn]'.

Vi opretter en ny borger i vores database og viser borgeren på listen.

This screenshot shows the same Java GUI window after the 'Opret borger' button has been clicked. The 'Indtast CPR' field now contains the value '1234' and the 'Indtast navn' field contains 'Freja'. The text area on the right now displays two lines of text: 'Borger [CPR = 1234567890, Navn = Fornavn Efternavn]' and 'Borger [CPR = 1234, Navn = Freja]'.

Vi ændrer CPR-nummeret for den valgte borger i databasen og viser det på listen.

This screenshot shows the Java GUI window after the 'Opdater borger' button has been clicked. The 'Indtast CPR' field now contains '12345' and the 'Indtast navn' field still contains 'Freja'. The text area on the right now displays two lines of text: 'Borger [CPR = 1234567890, Navn = Fornavn Efternavn]' and 'Borger [CPR = 12345, Navn = Freja]'.

SQL kode

sqlite3 database.db

```
CREATE TABLE Læger(
```

```
Speciale NVARCHAR(70) NOT NULL,
```

```
ID CHARACTER(10) PRIMARY KEY NOT NULL,
```

```
Navn NVARCHAR(70) NOT NULL
```

```
);
```

```
INSERT INTO Læger (Speciale, ID, Navn) VALUES ('Praktiserende', 'xxxx-xxxx',
```

```
'Fornavn Efternavn');
```

```
CREATE TABLE Borger(
```

```
CPR CHARACTER(10) PRIMARY KEY NOT NULL,
```

```
Navn NVARCHAR(70) NOT NULL
```

```
);
```

```
INSERT INTO Borger (CPR, Navn) VALUES ('1234567890', 'Fornavn Efternavn');
```

Java kode

```
private void buttonOpretBorgerActionPerformed(java.awt.event.ActionEvent evt) {  
  
    Connection c = null;  
    Statement stmt = null;  
  
    try {  
        Class.forName("org.sqlite.JDBC");  
        c = DriverManager.getConnection("jdbc:sqlite:database.db");  
        c.setAutoCommit(true);  
  
        stmt = c.createStatement();  
  
        String stmtText = "INSERT INTO Borger (CPR, Navn) VALUES ('"+  
            textCPR.getText() + "', '" + textNavn.getText() + "');";  
  
        stmt.executeQuery(stmtText);  
  
        stmt.close();  
        c.close();  
  
        Borger b = new Borger();  
        b.setCPR(textCPR.getText());  
        b.setNavn(textNavn.getText());  
  
        model.addElement(b);  
  
    } catch (ClassNotFoundException e) {  
  
    } catch (SQLException e) {  
  
    } catch (Exception e) {  
    }  
}
```

```

private void buttonHentDatabaseActionPerformed(java.awt.event.ActionEvent evt) {

    Connection c = null;
    Statement stmt = null;

    model.clear();

    try {
        Class.forName("org.sqlite.JDBC");
        c = DriverManager.getConnection("jdbc:sqlite:database.db");
        c.setAutoCommit(true);

        stmt = c.createStatement();

        ResultSet rs = stmt.executeQuery("SELECT * FROM Borger;");

        while (rs.next()) {
            Borger b = new Borger();
            b.setCPR(rs.getString("CPR"));
            b.setNavn(rs.getString("Navn"));
            model.addElement(b);
        }

        rs.close();
        stmt.close();
        c.close();
    } catch (ClassNotFoundException e) {

    } catch (SQLException e) {

    } catch (Exception e) {
    }
}

```

```

private void buttonOpdaterBorgerActionPerformed(java.awt.event.ActionEvent evt){

    Connection c = null;
    Statement stmt = null;

    try {
        int indeks = listBorger.getSelectedIndex();
        Borger valgtElement = (Borger) model.getElementAt(indeks);

        Class.forName("org.sqlite.JDBC");
        c = DriverManager.getConnection("jdbc:sqlite:database.db");
        c.setAutoCommit(true);

        stmt = c.createStatement();

        String stmtText = "UPDATE Borger "
            + "SET CPR = '" + textCPR.getText() + "', Navn = '" + textNavn.getText()
            + " "
            + "WHERE CPR = '" + valgtElement.getCPR() + " AND Navn = '" +
            valgtElement.getNavn() + "';";

        stmt.executeQuery(stmtText);
        stmt.close();
        c.close();

        Borger b = new Borger();
        b.setCPR(textCPR.getText());
        b.setNavn(textNavn.getText());

        model.removeElementAt(indeks);
        model.addElement(b);

    } catch (ClassNotFoundException e) {

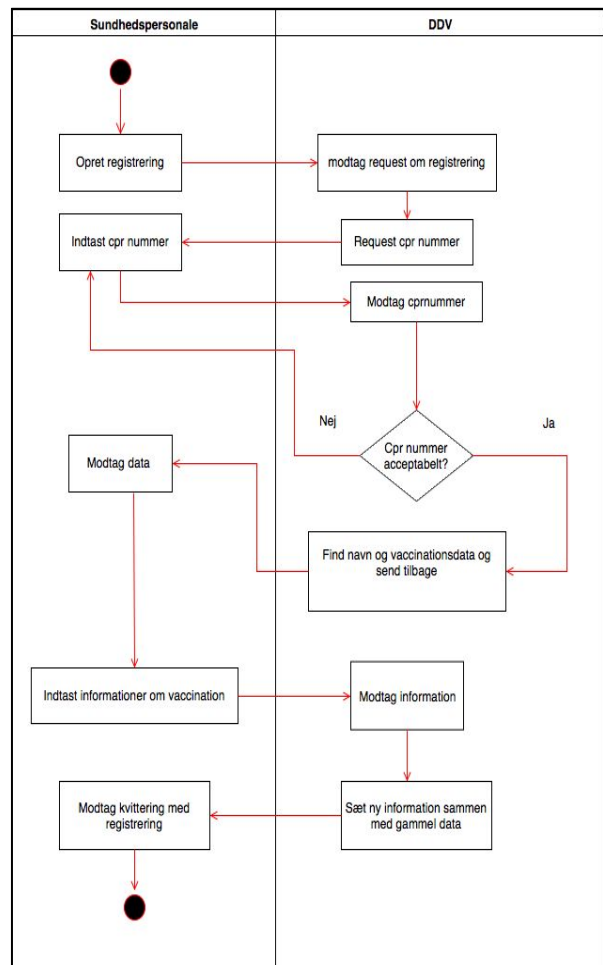
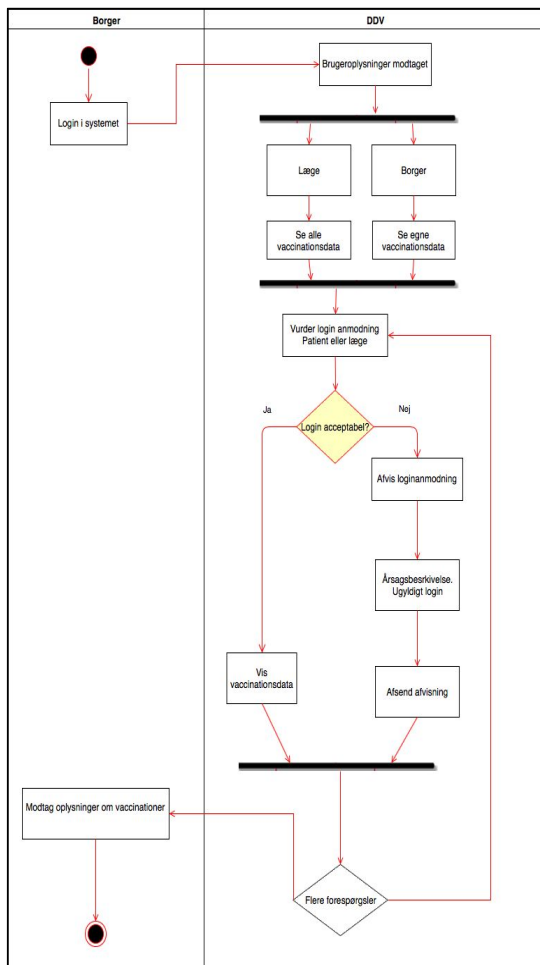
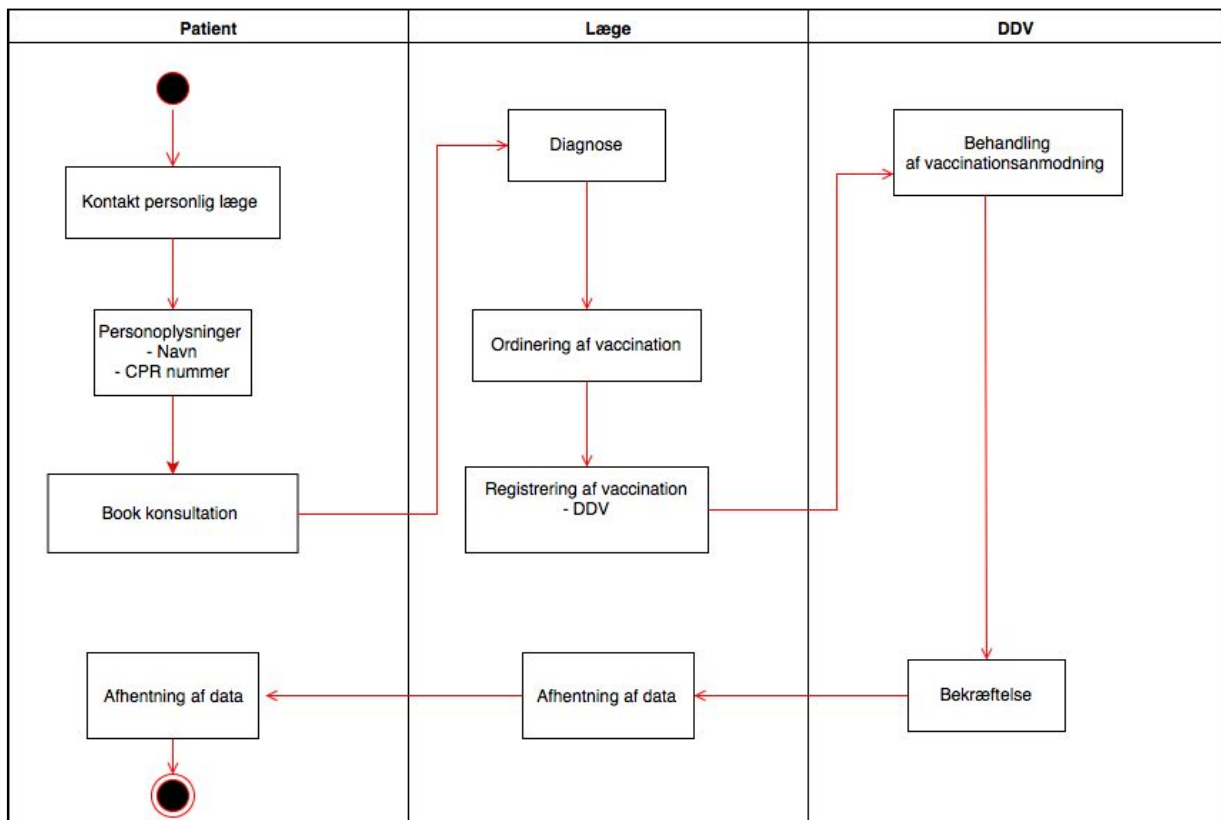
    } catch (SQLException e) {

    } catch (Exception e) {
    }

}

```

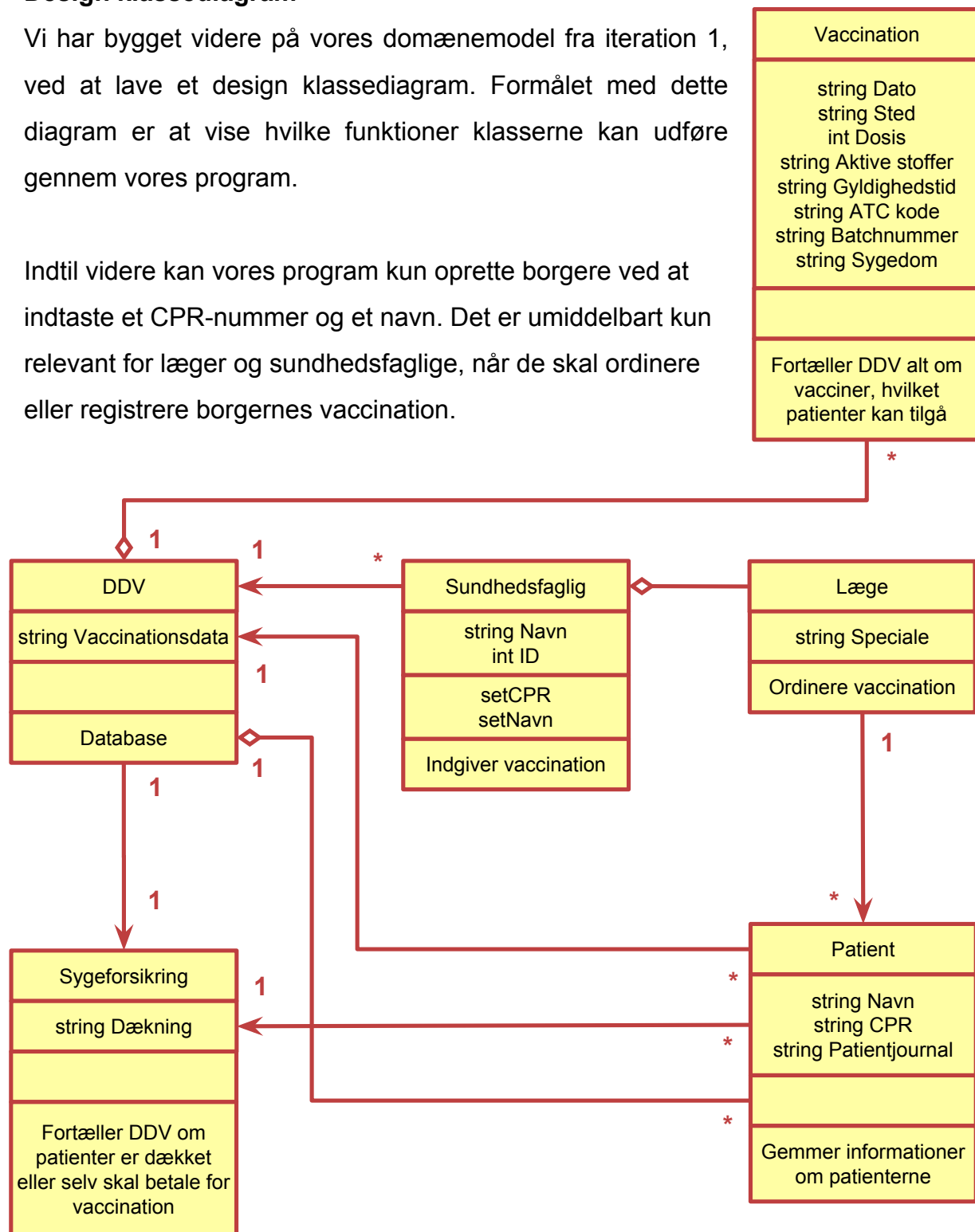
Aktivitetsdiagrammer



Design klassediagram

Vi har bygget videre på vores domænemodel fra iteration 1, ved at lave et design klassediagram. Formålet med dette diagram er at vise hvilke funktioner klasserne kan udføre gennem vores program.

Indtil videre kan vores program kun oprette borgere ved at indtaste et CPR-nummer og et navn. Det er umiddelbart kun relevant for læger og sundhedsfaglige, når de skal ordinere eller registrere borgernes vaccination.



DET DANSKE VACCINATIONS REGISTER

Indholdsfortegnelse

Indledning 3

Systemudviklingsdelen

Konklusion 3

Programmeringsdelen

Java applikation 4

Indledning

I denne iteration af vores system vil vi fokusere på resultaterne af vores accepttest og brugertest. Vi laver også den sidste iteration af vores java program, hvor vi transformere vores database fra Java til XML.

Accepttest Spørgsmål	Svar (sæt ét kryds)	
	Ja	Nej
Oprette en borger? Lykkedes det? (30 sek.)	X	
Hvis nej, hvad gik galt?		
Henter data fra databasen? Lykkedes det? (30 sek.)	X	
Hvis nej, hvad gik galt?		
Redigere i databasen? Lykkedes det? (40 sek.)	X	
Hvis nej, hvad gik galt?		
Gemmer ny-redigeret data? Lykkedes det? (60 sek.)	X	
Hvis nej, hvad gik galt?		

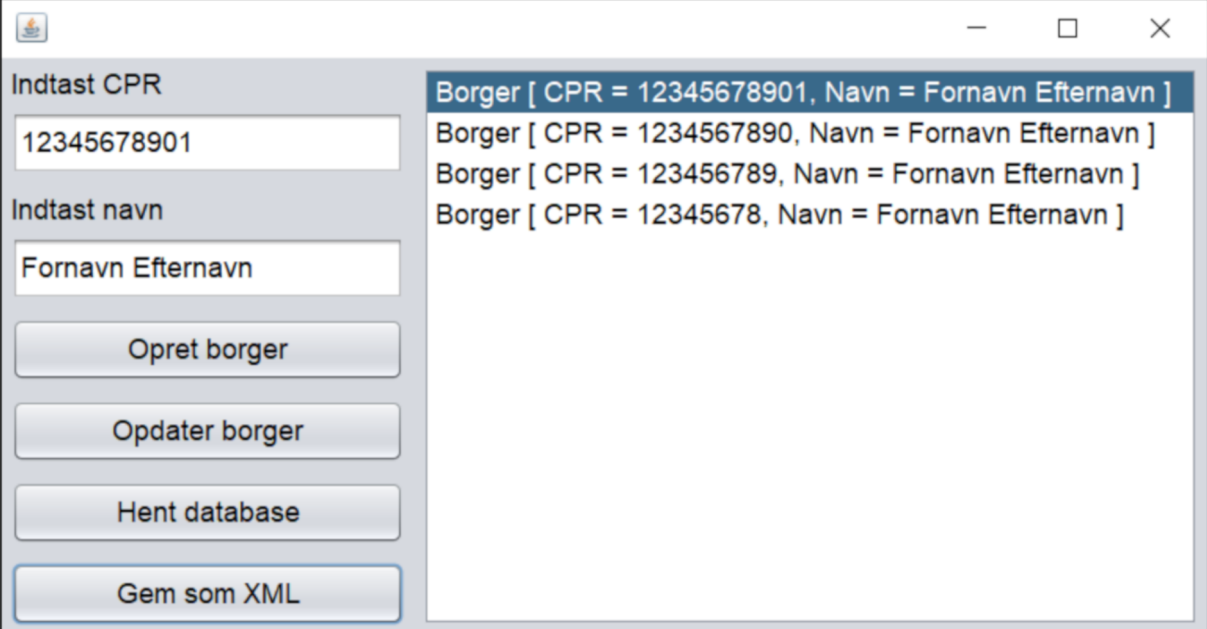
Brugertest Spørgsmål	Svar (sæt ét kryds)			
	Utilfreds	Mindre tilfreds	Tilfreds	Meget tilfreds
Hvor hurtigt hentes patienterne databasen?				X
Hvor overskueligt er designet?		X		
Fremkommer det tydeligt hvilke knapper man skal trykke?			X	
Overordnet bedømmelse af programmet			X	

Konklusion

Hvis vi ser på accepttesten, kan vi se at brugeren som testede vores program, fuldførte programmets funktioner som den skulle kunne. Vi kan se at det skete inden for den tidsbestemte tidsgrænse som vi satte op, hvilket var det vi sigtede efter. Hvis vi ser på brugertesten kan vi se integreringen mellem databasen og programmet fungere meget hurtig. Dog kan vi ud fra besvarelser også se at designet ikke var så brugervenligt som vi havde formået det ville være. Derudover kan vi se at placeringen af vores knapper har været rimelig overskuelige, samt den information som står på knapperne. Afluttende bedømmelse ser ud til og være tilfredstillende for brugeren, og hvad måske kunne score et ekstra point nok var designet.

Java applikation

Vi havde ikke mulighed for at kode vores program til at gemme databasen eller listen i programmet som en XML fil, så derfor har vi kun lavet en repræsentation af hvordan det ville se ud i programmet.



The screenshot shows a Java application window with a sidebar on the left and a main content area on the right. The sidebar contains two input fields: 'Indtast CPR' with the value '12345678901' and 'Indtast navn' with the value 'Fornavn Efternavn'. Below these fields are four buttons: 'Opret borger', 'Opdater borger', 'Hent database', and 'Gem som XML'. The main content area displays a list of 'Borger' objects, each represented as a string: 'Borger [CPR = 12345678901, Navn = Fornavn Efternavn]', 'Borger [CPR = 1234567890, Navn = Fornavn Efternavn]', 'Borger [CPR = 123456789, Navn = Fornavn Efternavn]', and 'Borger [CPR = 12345678, Navn = Fornavn Efternavn]'.

Vi viser også hvordan vores klasse af borgere ser ud i Java kode, i forhold til hvordan det ser ud når det bliver transformeret til XML kode.

Java kode

@XmlRootElement

```
public class Borger {  
    public String CPR;  
    public String Navn;  
}
```

XML kode

```
<Borger>  
    <CPR>12345678901</CPR>  
    <Navn>Fornavn Efternavn</Navn>  
</Borger>
```